Advanced Databases

Semistructured Data - XML

Dr. George Mertzios
Michaelmas Term

george.mertzios@durham.ac.uk
Room 2066, MCS Building
Tel: 42 429

Types of data organization

- Three characterizations of data:
  - *Structured* data
  - *Semi-structured* data (XML)
  - *Unstructured* data

- Structured data:
  - data represented in a *strict format* (i.e. schema)
    - relational data model (tables, tuples, attributes)
  - the DBMS checks to ensure that the data follows
    - the structures (table, attributes, domains)
    - the integrity & referential constraints (primary / foreign keys)

    that are specified in the schema.

Types of data organization

- Semi-structured data: data that
  - may be irregular or incomplete and
  - have a structure that may change rapidly / unpredictably

- This data may have some structure, but:
  - not all the parts have the same fixed structure
  - each data object may have *different attributes* that are not known in advance

- How do we end up with such data?
  - sometimes data is collected *ad-hoc*
    - i.e. no predefined structure
    - for instance: details of all research projects
  - not known in advance how it will be stored / managed

Types of data organization

- Semi-structured data: data that
  - – may be irregular or incomplete and
  - – have a structure that may change rapidly / unpredictably

- Also called *self-describing* (or *schema-less*) data:
  - – information that normally belongs to a schema, now is contained within the data itself

  ⇒ the schema information is mixed with the data values

- In some cases:
  - – there exists a separate schema (or a Document Type Definition – DTD)
  - – but only places *loose constraints* on data

Types of data organization

- Unstructured data:
  - very limited indication of the type / structure of data
- Typical examples:
  - a text document with *some* information within it
  - a web page in HTML that contains *some* data
- Example: a cooking recipe in an HTML Document

Flour: 80 cl

Yeast: 10 grams

Water: 80 cl (warm)

Salt: 1 teaspoon

Attention: Cook for 3 hours!

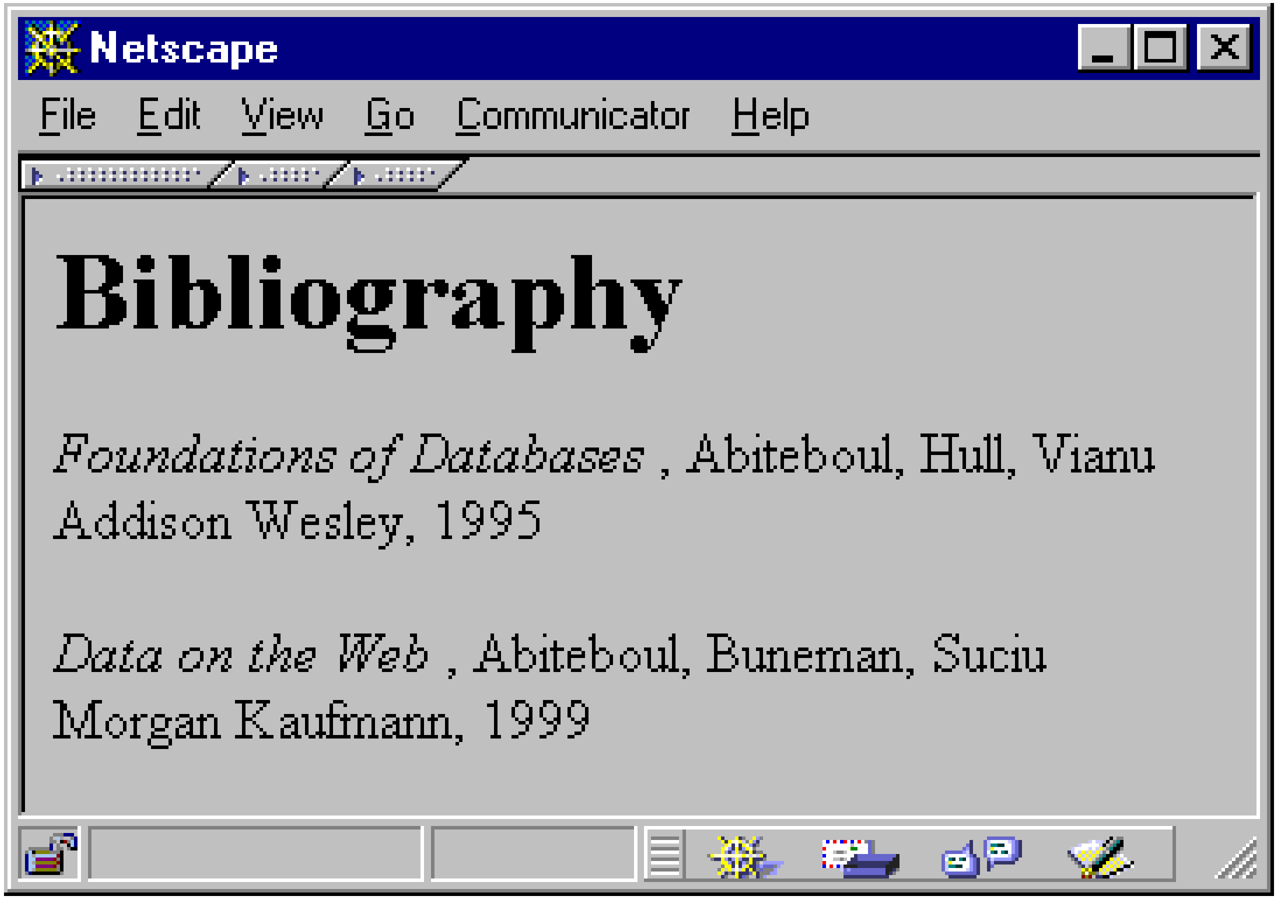
Semi-structured data

- Main language for semi-structured data:
  - XML (eXtended Markup Language)
  - a language for *structuring* and *exchanging* web data

- Similarities with HTML:
  - HyperText Markup Language
  - a language for *displaying* web pages

- Both XML and HTML are “tag” languages

HTML vs. XML



Bibliography

Foundations of Databases , Abiteboul, Hull, Vianu
Addison Wesley, 1995

Data on the Web , Abiteboul, Buneman, Suciu
Morgan Kaufmann, 1999

HTML describes the presentation of data

HTML vs. XML

HTML code for this output:

```
<h1> Bibliography </h1>
<p> <i> Foundations of Databases </i>
        Abiteboul, Hull, Vianu
        <br> Addison Wesley, 1995 </p>
<p> <i> Data on the Web </i>
        Abiteoul, Buneman, Suciu
        <br> Morgan Kaufmann, 1999 </p>
```

Tags describe the output format of data:

<i> *this text is in Italics* </i>

HTML vs. XML

XML describes the content of data (semantics)

```
<bibliography>
        <book>
                      <title> Foundations of Databases </title>
                      <author> Abiteboul </author>
                      <author> Hull </author>
                      <author> Vianu </author>
                      <publisher> Addison Wesley </publisher>
                      <year> 1995 </year>
        </book>
        …
</bibliography>
```

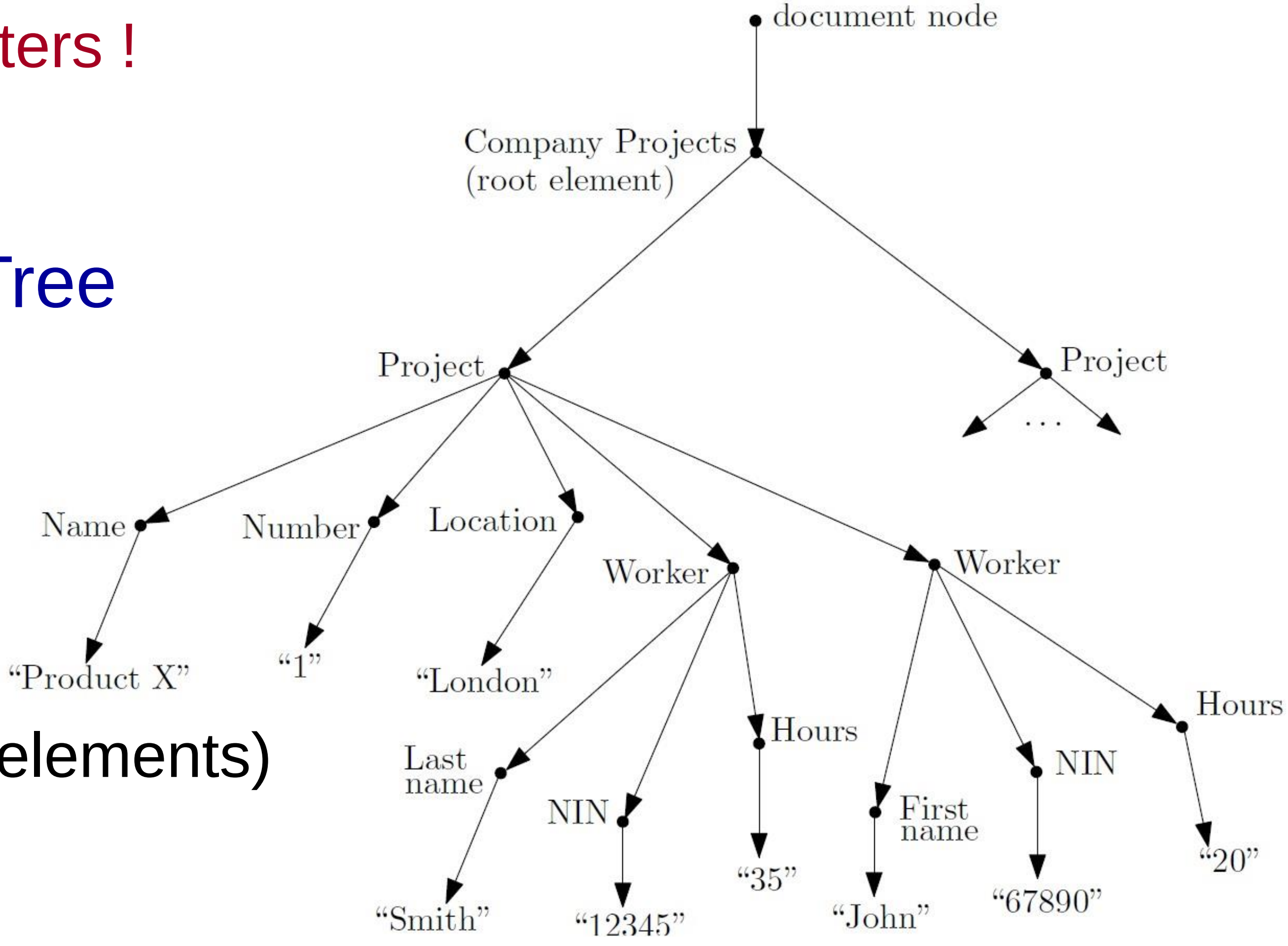
XML terminology

- tags: book, title, author, …
- start tag: <book>, end tag: </book>
- elements:
  - <book>…</book>
  - <author>…</author>
- elements are nested
- empty element:
  - <author></author> or: <author/>
- an XML document: single *root element*

- Attention: XML is case-sensitive !

XML data as a graph

- XML data have a (directed) tree structure:
  - ordering matters !



- Hierarchical Tree Data Model:

- internal nodes are elements
- leafs are raw data (base elements)
- document node
- root node

XML data as a graph

- XML data have a (directed) tree structure:
  - – ordering matters !

```
<name>
        <FName> John </FName>
        <LName> Smith </LName>
</name>
```

≠

```
<name>
        <LName> Smith </LName>
        <FName> John </FName>
</name>
```

- Query languages for XML:
  - – traverse the tree-labeled representation

Querying loses its “traditional” declarative nature

⇒ it becomes more “navigational”

XML data

- XML is self-describing

- Without a schema:
  - only the relative position of elements in the tree matters

- Schema now becomes part of the data
  - it is discovered from the data, not imposed apriori
  - Relational schema: person(name,phone)
  - In XML <person>, <name>, <phone> are:
    - part of the data
    - possibly repeated many times (semi-structured data)

⇒ XML is much more flexible

Why is XML interesting?

- XML is just syntax for data
  - Note: we have no syntax for relational data
  - But XML is not relational: *semi-structured*

- This is exciting because we:
  - can translate *any* data to XML
  - can ship XML over the Web
  - can input XML into any application

⇒ easy data sharing & exchange on the Web

Some advantages of XML

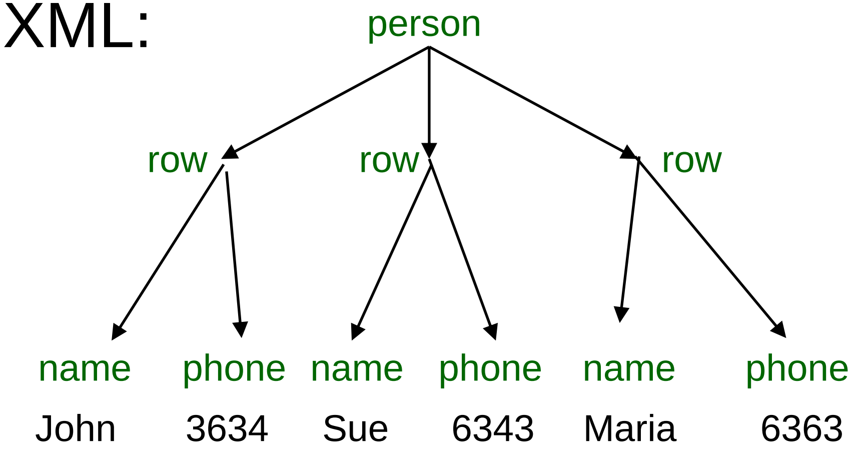
- Simplicity:
  - Relatively simple standard, human-legible language and reasonably clear
- Extensibility (unlike HTML):
  - allows users to define their own tags, for their own application requirements
- Platform- and vendor-independent:
  - works with every platform, supports all alphabets
- Separation of content & presentation:
  - a “write once – publish anywhere” language
  - allows customized view of data (e.g. in browser)

Relational data as XML

person

name	phone
John	3634
Sue	6343
Maria	6363

XML:



person

row row row

name phone name phone name phone

John 3634 Sue 6343 Maria 6363

```
<person>
      <row> <name> John </name>
                  <phone> 3634 </phone>
      </row>
      <row> <name> Sue </name>
                  <phone> 6343 </phone>
      </row>
      <row> <name> Maria </name>
                  <phone> 6363</phone>
      </row>
</person>
```

XML is Semi-structured data

- Missing attributes:

```
<person>   <name> John</name>
           <phone>1234</phone>
 </person>

<person>  <name>Joe</name>
</person>
```

← no phone !

- Could be represented in a table with NULLs:

name	phone
John	1234
Joe	NULL

XML is Semi-structured data

- Repeated attributes:

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

← two phones !

- Impossible in tables:

(possible only with
multi-valued attributes)

name	phone	
Mary	2345	3456

???

XML is Semi-structured data

- Different structure in different elements:

```
<person>
      <name>  <first> John </first>
              <last> Smith </last>
      </name>
</person>

<person>
      <name>  Chris N. Wilson </name>
</person>
```

← structured name !

← plain name !

- Heterogeneous collections:
  - <bookstore> contains both <book>s and <publisher>s

Attributes in XML

- XML Attributes:
  - a name-value pair with descriptive / identifying information about an element
  - placed inside the start tag of the element
  - attribute value enclosed in quotes “ ”

```
<STAFF branchNo = “B005”>
```

```
<SEX gender = “F”>
```

```
<book price = “95”, currency = “USD”>
      <title> Database System Concepts </title>
      <year> 2006 </year>
</book>
```

Attributes in XML

- XML Attributes:
  - an attribute can appear only once within a tag
  - but subelements can use the same attribute name !

```
<project name = “databases”>
     <researcher name = “John Smith”>
        …
     </researcher>
</book>
```

  - attributes in XML are not ordered (although elements are):

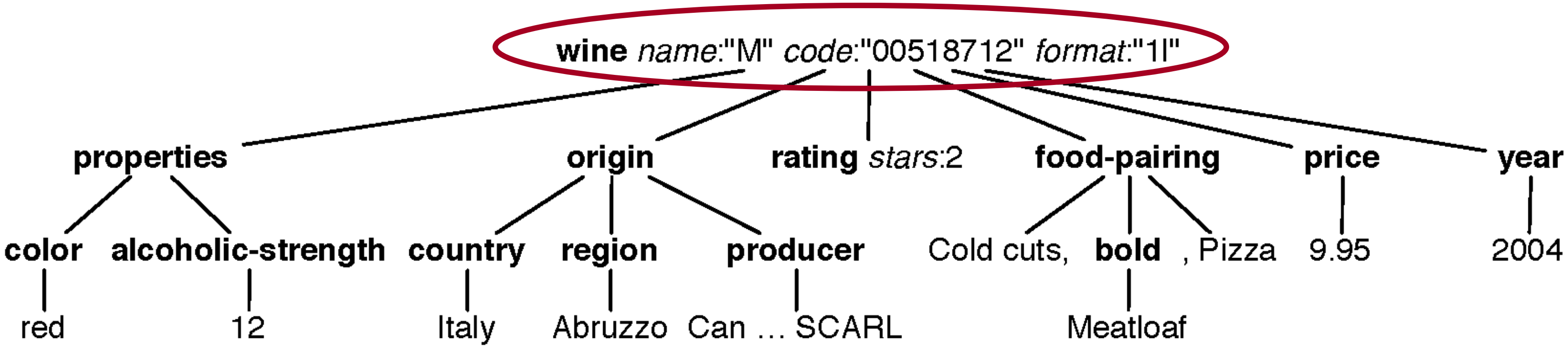
```
<name FName = “John”  LName = “Smith”>
```

=

```
<name LName = “Smith”  FName = “John”>
```

Attributes in XML

- In the Hierarchical Tree Data Model:
  - attributes extend the declarations of elements



wine *name*:"M" *code*:"00518712" *format*:"1l"

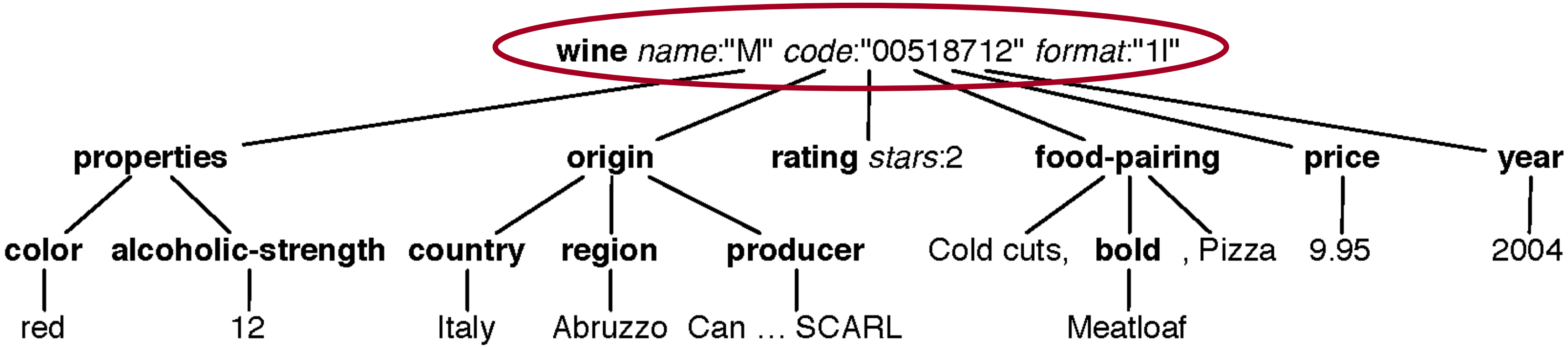
properties **origin** **rating** *stars*:2 **food-pairing** **price** **year**

color **alcoholic-strength** **country** **region** **producer** Cold cuts, **bold** , Pizza 9.95 2004

red 12 Italy Abruzzo Can … SCARL Meatloaf

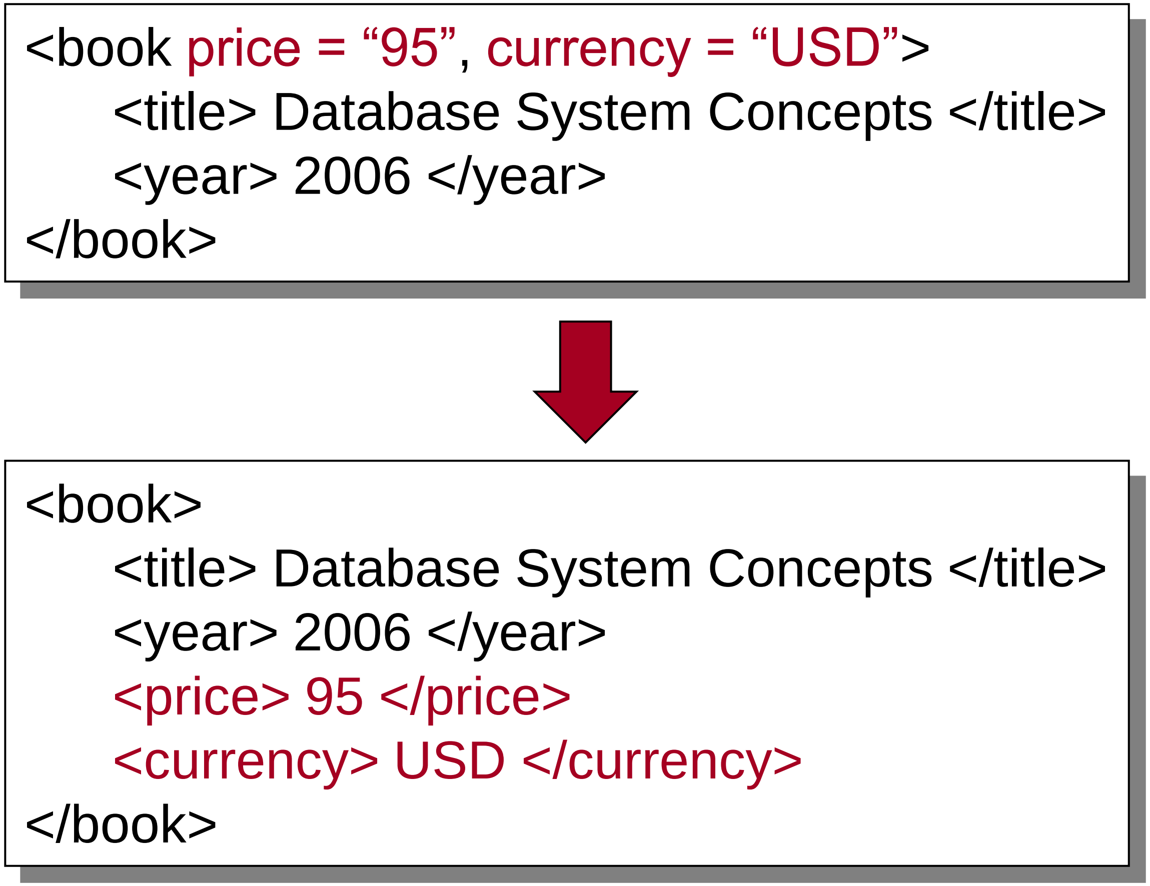
Attributes in XML

```
<wine   name="M"   code="00518712"    format="1l">
    <properties>
        <color>red</color>
        <alcoholic-strength>12</alcoholic-strength>
    </properties>
    <origin>
        <country>Italy</country>
        <region>Abruzzo</region>
        <producer>Cantina Miglianico SCARL</producer>
    </origin>
    <rating stars="2"/>
    <food-pairing>Cold cuts, <bold>Meatloaf</bold>, Pizza</food-pairing>
    <price>9.95</price>
    <year>2004</year>
</wine>
```



Attributes vs. Sub-elements

- Note the potential ambiguity:
  - attributes can be replaced by sub-elements



```
<book price = “95”, currency = “USD”>
     <title> Database System Concepts </title>
     <year> 2006 </year>
</book>
```

```
<book>
     <title> Database System Concepts </title>
     <year> 2006 </year>
     <price> 95 </price>
     <currency> USD </currency>
</book>
```

Attributes vs. Sub-elements

- A good rule in practice:
  - avoid attributes → prefer sub-elements
  - unless you need an attribute!

- Disadvantages of attributes:
  - cannot contain multiple values (child elements can)
  - cannot describe structure (child elements can)
  - more difficult to manipulate by program code
  - not easily expandable (for future changes)
  - not easy to test against a Document-Type-Definition

- Exception to this rule:
  - use attributes to describe metadata
    (i.e. data about data)

Attributes vs. Sub-elements

Example: attributes for metadata

- here “id” is just a counter
- only used to identify the different notes
- not part of the data

```
<messages>
<note id="p501">
      <to>Tove</to>
      <from>Jani</from>
      <heading>Reminder</heading>
      <body>Don't forget me this weekend!</body>
</note>

<note id="p502">
      <to>Jani</to>
      <from>Tove</from>
      <heading>Re: Reminder</heading>
      <body>I will not!</body>
</note>
</messages>
```

Attributes: ID and IDREF

- Some attributes can be declared as of type:
  - ID, used as an identifier for the element
  - IDREF, used as a pointer to an element with a specific ID

- If an attribute is explicitly declared as IDREF:
  - its value must be equal to an ID attribute !

⇒ ID and IDREF work in a similar way as primary keys and foreign keys in the relational data model

⇒ the *tree* representation becomes a *directed graph* (not necessarily acyclic !)

Attributes: ID and IDREF

- IDREFS:
  - an extension of the IDREF attribute type
  - the attribute value is a string consisting of a list of IDs, separated with a whitespace
  - it links the element to a set of elements (i.e. IDs)

```
<person id = “o555”>
     <name> Jane </name>
</person>

<person id = “o456”, children-idrefs = “o123 o555”>
     <name> Mary </name>
</person>

<person id = “o123”, mother-idref = “o456”>
     <name> John </name>
</person>
```

Attributes: ID and IDREF

Here *id*, *idref* and *idrefs* in XML are just syntax:

- referential constraints
  (i.e. ID / IDREF / IDREFS declarations)
  imposed by the Document Type Definition (DTD) !

```
<person id = “o555”>
    <name> Jane </name>
</person>

<person id = “o456”, children-idrefs = “o123 o555”>
    <name> Mary </name>
</person>

<person id = “o123”, mother-idref = “o456”>
    <name> John </name>
</person>
```

More XML

- Comments in XML:
  - enclosed in <!-- and --> tags
  - can contain any data except the string --
- CDATA in XML:
  - character data, containing any text
  - will not be parsed by any XML processor
- Entity reference in XML:
  - to refer to reserved symbols and special characters
  - begins with the ampersand character **&**
  - ends with the semicolon **;**
- When displayed in a browser:
  - the reference &...; will be replaced by its content ...

More XML

The UTF-8 code for © is #xA9

Examples of entity references:

to display in browser:	we write in XML:
X<1	X&lt;1
value>’a’&value<“z”	value&gt;&apos;a&apos;&amp;value&lt;&quot’z&quot;
©Durham University	&#xA9;Durham University

reserved symbol	mnemonic name in XML
&	amp
<	lt
>	gt
‘ (single quote)	apos
“ (double quote)	quot

XML validation

- Although XML is self-describing data:
  - we can still impose a more rigorous structure
  - e.g. for increased business integration between companies (in B2B solutions)

- In particular:
  - list the permissible element names,
  - which elements can appear in combination with which other ones,
  - how elements can be nested,
  - what attributes for which element type, …

XML validation

- Although XML is self-describing data:
  - we can still impose a more rigorous structure
- Document Type Definition (DTD):
  - concrete set of rules for elements and attributes
  - allows seamless data exchange between documents with the same DTD
  - appropriate for specific applications, e.g. protein structures, menus in a restaurant…
- XML schema:
  - more powerful than DTD
  - allows more complex structures

Types of XML validation

1. *Well formed* XML document (syntactic guidelines of the tree model):

- single root element
- matching tags (properly nested)
- initial XML declaration, e.g.:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

  - `standalone=“yes"` ⇒ no DTD
  - `standalone=“no"` ⇒ structure imposed by DTD

Types of XML validation

2. *Type-valid* XML document (stronger):

- must be well formed and
- the elements / attributes must follow the pre-defined structure, described in the DTD

- DTD (Document Type Definition):
  - in an *extra file*, or
  - *embedded* within the XML file

3. *Schema-valid* XML document (stronger):

- must be well formed and
- conforms to an XML schema

Document Type Definition (DTD)

General Structure of a DTD:

```
<!DOCTYPE root_name [
    <!ELEMENT elem_name (subElem1, subElem2, …)>
    … more elements …
]>
```

In DTD, an element is declared by:

- its name (i.e. its tag in the XML document)
- the sequence of the names of its sub-elements (placed in parentheses)

- Special case of a sub-element:
  `(#PCDATA)` means that the sub-element is plain text

Sub-elements of an element

- The sub-elements:
  - are declared by their names (i.e. its tags in XML)
  - appear nested within the element (in XML)
  - they appear in XML in the order specified in the DTD
  - if commas are omitted: arbitrary order
- Multiplicity of a sub-element is specified by:

  a) `*` = zero or more

  b) `+` = one or more

  c) `?` = zero or one
- In addition: “`|`” means “or”

Document Type Definition (DTD)

Example of a simple DTD:

```
<!DOCTYPE DurhamPUBS [
  <!ELEMENT DurhamPUBS (PUB*)>
  <!ELEMENT PUB (NAME,(BEER | VODKA)+, ADDRESS?)>
  <!ELEMENT NAME (#PCDATA)>
  <!ELEMENT BEER (NAME, PRICE)>
  <!ELEMENT VODKA (NAME, PRICE)>
  <!ELEMENT PRICE (#PCDATA)>
  <!ELEMENT ADDRESS (#PCDATA)>
]>
```

Using DTD

We can either embed the DTD within the XML file:

```
<?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>
     ...
</BARS>
```

Using DTD

Or we can specify the file where the DTD can be found:

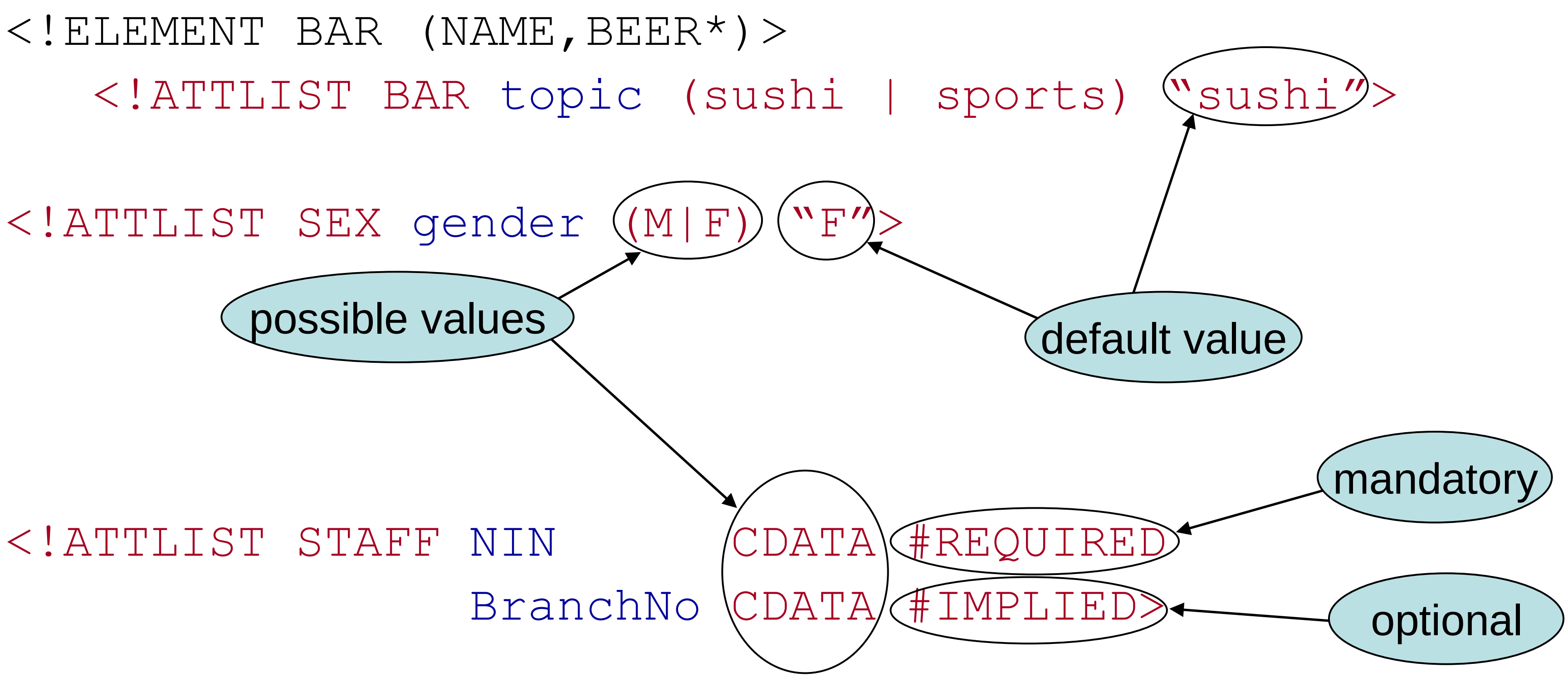
```
<?XML VERSION = "1.0" STANDALONE = "no"?>
<!DOCTYPE Bars SYSTEM "bar.dtd">
<BARS>
    <BAR> <NAME> Joe's Bar </NAME>
          <BEER> <NAME> Bud </NAME>
                 <PRICE> 2.50 </PRICE>
          </BEER>
          <BEER> <NAME> Miller </NAME>
                 <PRICE> 3.00 </PRICE>
          </BEER>
     </BAR>
       ...
</BARS>
```

Declaring attributes in DTD

- Attribute list declarations identify:
  - which elements may have which attributes
  - what values the attributes can hold
  - whether they are of type ID, IDREF, or IDREFS
- Optionality:
  - #REQUIRED: the attribute must always be provided
  - #IMPLIED: the attribute is optional

Declaring attributes in DTD

- Examples of syntax:
  - a bar may be of a special topic: *shushi* bar / *sports* bar
  - default topic: *sushi*



```
<!ELEMENT BAR (NAME,BEER*)>
   <!ATTLIST BAR topic (sushi | sports) “sushi”>

<!ATTLIST SEX gender (M|F) “F”>

<!ATTLIST STAFF NIN      CDATA #REQUIRED
                BranchNo CDATA #IMPLIED>
```

Declaring attributes in DTD

- Attribute list declarations identify:
  - which elements may have which attributes
  - what values the attributes can hold
  - whether they are of type ID, IDREF, or IDREFS

- ID / IDREF / IDREFS type declaration in a DTD:

```
<!ATTLIST BRANCH  branchNo  ID>

<!ATTLIST STAFF  nin  ID>

<!ATTLIST STAFF  worksIn  IDREF>

<!ATTLIST BRANCH  employs  IDREFS>
```

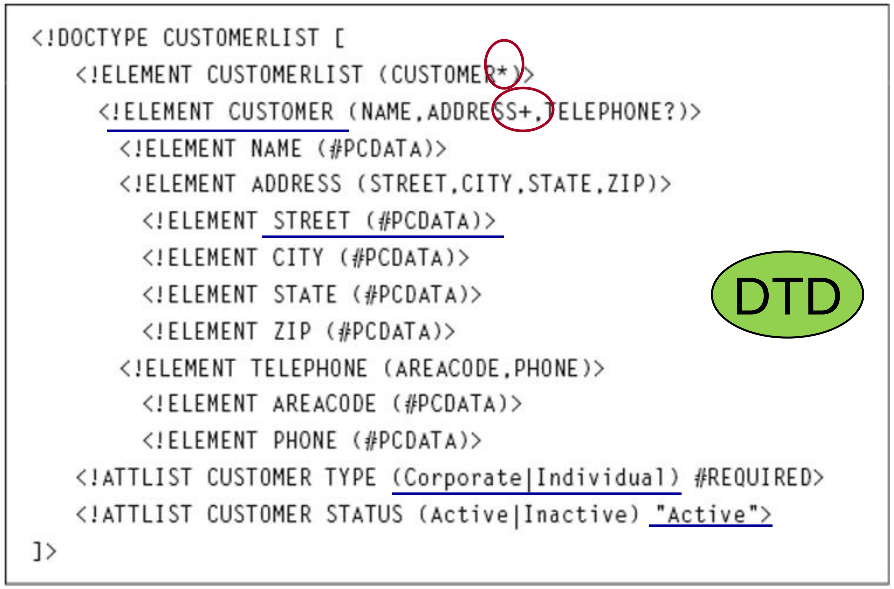
attributes
of type ID, IDREF
and IDREFS

Declaring attributes in DTD

Example: suppose we want to map each beer to its (unique) manufacturer:

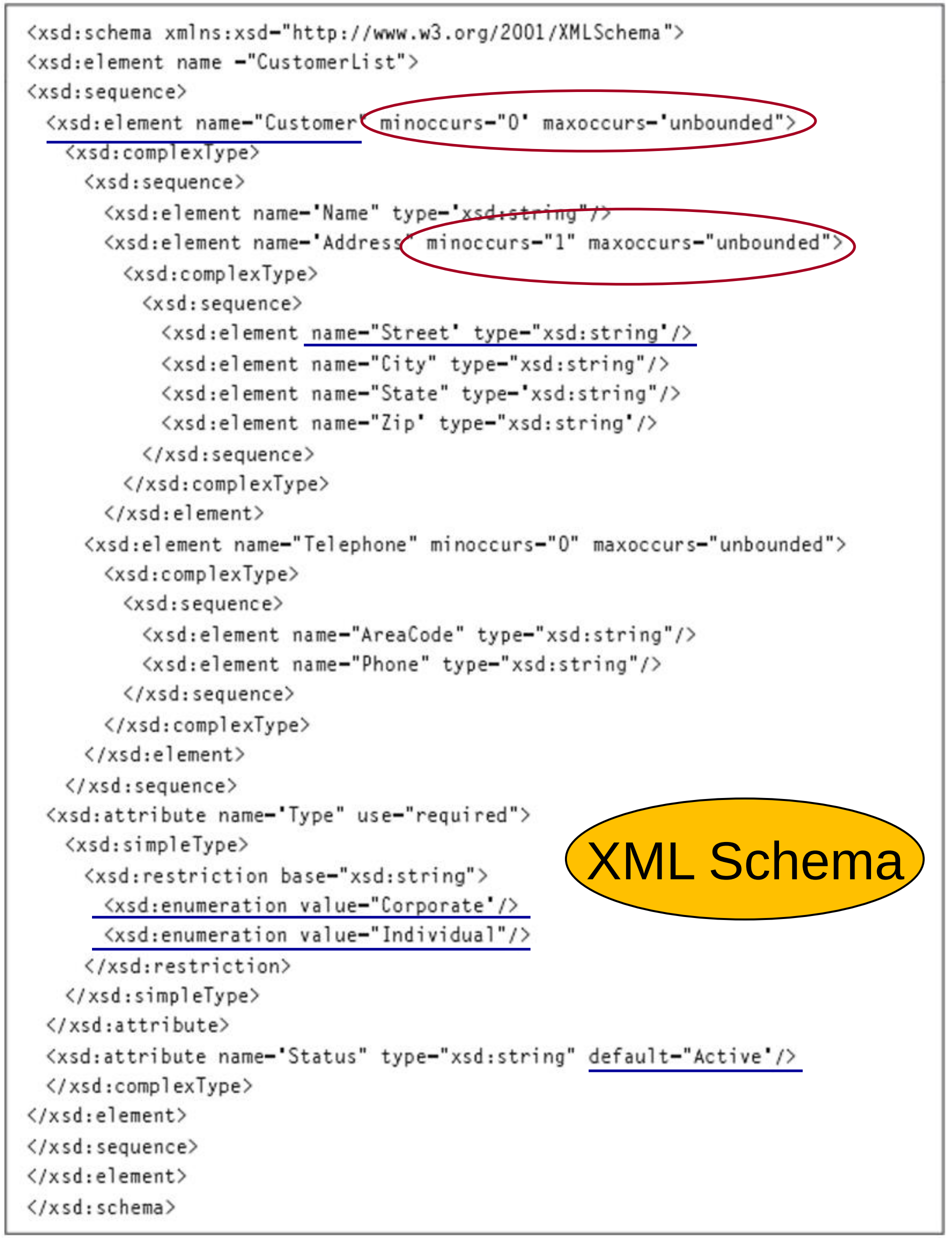
```
<!DOCTYPE Bars [
    <!ELEMENT BARS (BAR* MANUFACTURER*)>
    <!ELEMENT BAR (NAME, BEER+)>
    <!ELEMENT NAME (#PCDATA)>
    <!ELEMENT MANUFACTURER (ADDRESS)>
          <!ATTLIST MANUFACTURER name ID>
    <!ELEMENT ADDRESS (#PCDATA)>
    <!ELEMENT BEER (NAME, PRICE)>
          <!ATTLIST BEER manuf IDREF>
    <!ELEMENT PRICE (#PCDATA)>
 ]>
```

DTD vs. XML Schema



```
<!DOCTYPE CUSTOMERLIST [
   <!ELEMENT CUSTOMERLIST (CUSTOMER*)>
     <!ELEMENT CUSTOMER (NAME,ADDRESS+,TELEPHONE?)>
       <!ELEMENT NAME (#PCDATA)>
       <!ELEMENT ADDRESS (STREET,CITY,STATE,ZIP)>
         <!ELEMENT STREET (#PCDATA)>
         <!ELEMENT CITY (#PCDATA)>
         <!ELEMENT STATE (#PCDATA)>
         <!ELEMENT ZIP (#PCDATA)>
       <!ELEMENT TELEPHONE (AREACODE,PHONE)>
         <!ELEMENT AREACODE (#PCDATA)>
         <!ELEMENT PHONE (#PCDATA)>
   <!ATTLIST CUSTOMER TYPE (Corporate|Individual) #REQUIRED>
   <!ATTLIST CUSTOMER STATUS (Active|Inactive) "Active">
]>
```

DTD



```
<xsd:schema xmlns:xsd-"http://www.w3.org/2001/XMLSchema">
<xsd:element name -"CustomerList">
<xsd:sequence>
  <xsd:element name-"Customer" minoccurs-"0' maxoccurs-'unbounded">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name-'Name" type-'xsd:string"/>
        <xsd:element name-'Address" minoccurs-"1" maxoccurs-"unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name-"Street' type-"xsd:string'/>
              <xsd:element name-"City" type-"xsd:string"/>
              <xsd:element name-"State" type-'xsd:string"/>
              <xsd:element name-"Zip' type-"xsd:string'/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      <xsd:element name-"Telephone" minoccurs-"0" maxoccurs-"unbounded">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name-"AreaCode" type-"xsd:string"/>
            <xsd:element name-"Phone" type-"xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  <xsd:attribute name-'Type" use-"required">
    <xsd:simpleType>
      <xsd:restriction base-"xsd:string">
        <xsd:enumeration value-"Corporate'/>
        <xsd:enumeration value-"Individual"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
  <xsd:attribute name-'Status" type-"xsd:string" default-"Active'/>
  </xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:element>
</xsd:schema>
```

XML Schema

XML schema:

- an XML document
- uses elements / attributes to express the semantics
- more powerful than a DTD

Summary of the Lecture

- Structured / Semi-structured / Unstructured data
- HTML vs. XML (presentation vs. semantics)
- XML (semi-structured data):
  - relational data as XML
  - hierarchical tree data model
  - attributes / ID / IDREF / IDREFS
  - well formed / type valid / schema valid XML
- DTD (Document Type Definition)
  - declaring XML structure
  - declaring attributes / ID / IDREF / IDREFS
- XML Schema