Semi-Structured Data: XML

Instructor: Shel Finkelstein

Reference:
A First Course in Database Systems, 3rd edition,
Chapter 11.1-11.3, some of 11.4. a little of Chapter 12
Slides from Prof. Jeffrey Ullman, Stanford University

Important Notices

- Final Exam is on Wednesday, March 22, noon-3pm in our usual classroom.
  - Final is Cumulative, with more focus on second half of quarter.
    - Final may include material in Lecture 14 (XML); no JSON or NOSQL.
  - Please bring a red Scantron sheet (ParSCORE form number f-1712) sold at the Bookstore, and #2 pencils. You'll answer multiple choice on Scantron.
    - Ink and #3 pencils don't work.
  - You may bring a single two-sided 8.5" x 11" sheet of paper with as much info written (or printed) on it as you can fit and read unassisted, just as for the Midterm.
    - No sharing of these sheets will be permitted.
  - You must show your UCSC id when you turn in your Final and Scantron.
  - The Final from Fall 2016 and Answers to that Final have been posted on Piazza (Resources → Exams).

More Important Notices

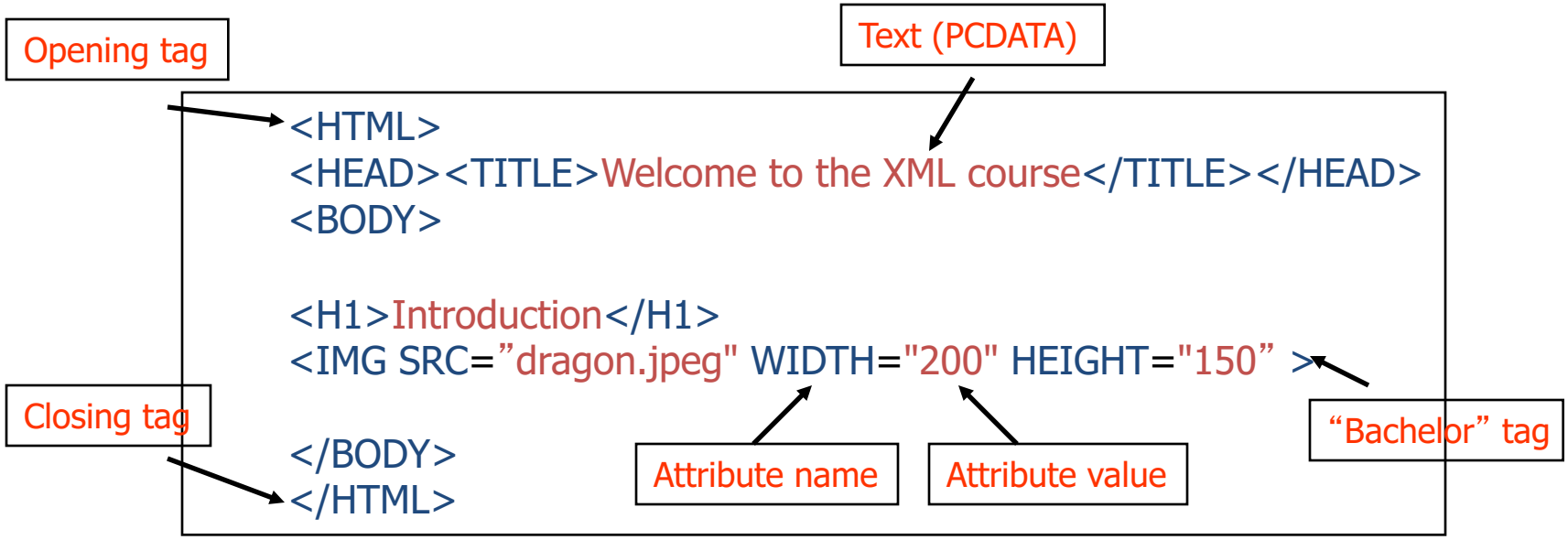
- Gradiance Assignment #5 (on Functional Dependencies and Normal Forms) is due by Friday, March 17, 11:59pm.
- There will be Lab Sections during the last week of classes.
  - These Lab Sections are an opportunity go over the answers to Lab4 and other Labs, or ask questions about other course material.
  - Solution to Lab4 was posted on Piazza on Monday, March 13.
- Online course evaluations began Monday, March 5, and run through Sunday, March 19 at 11:59pm.
  - Instructors are not able to identify individual responses.
  - Constructive responses help improve future courses.

Semi-Structured Data Models

- In the relational database management system, a schema must be defined *before* data can be stored.
  - Schema is known to the query processor.
  - Exploited to derive efficient implementations to access and update data.
- In a semi-structured data model (e.g., XML and JSON), a schema need not be defined prior to “data creation”.
  - Flexible data model as the schema need not be defined ahead of time, and there may not be a structured schema associated with the data.
  - Semi-structured data tends to be “self-describing”.
  - Also tends to be hierarchical.

HyperText Markup Language (HTML)

- Lingua franca for publishing hypertext on the World Wide Web.
- Designed to describe how a Web browser should arrange text, images and push-buttons on a page.
- Easy to learn, but does not convey structure.
- Fixed tag set.



The Structure of XML

- XML consists of *tags* and *text*

- Tags come in pairs <date> ...</date>

- They must be properly nested

<date> <day> ... </day> ... </date> --- good

<date> <day> ... </date>... </day> --- bad

(You can't do ... ... in HTML)

Well-Formed XML

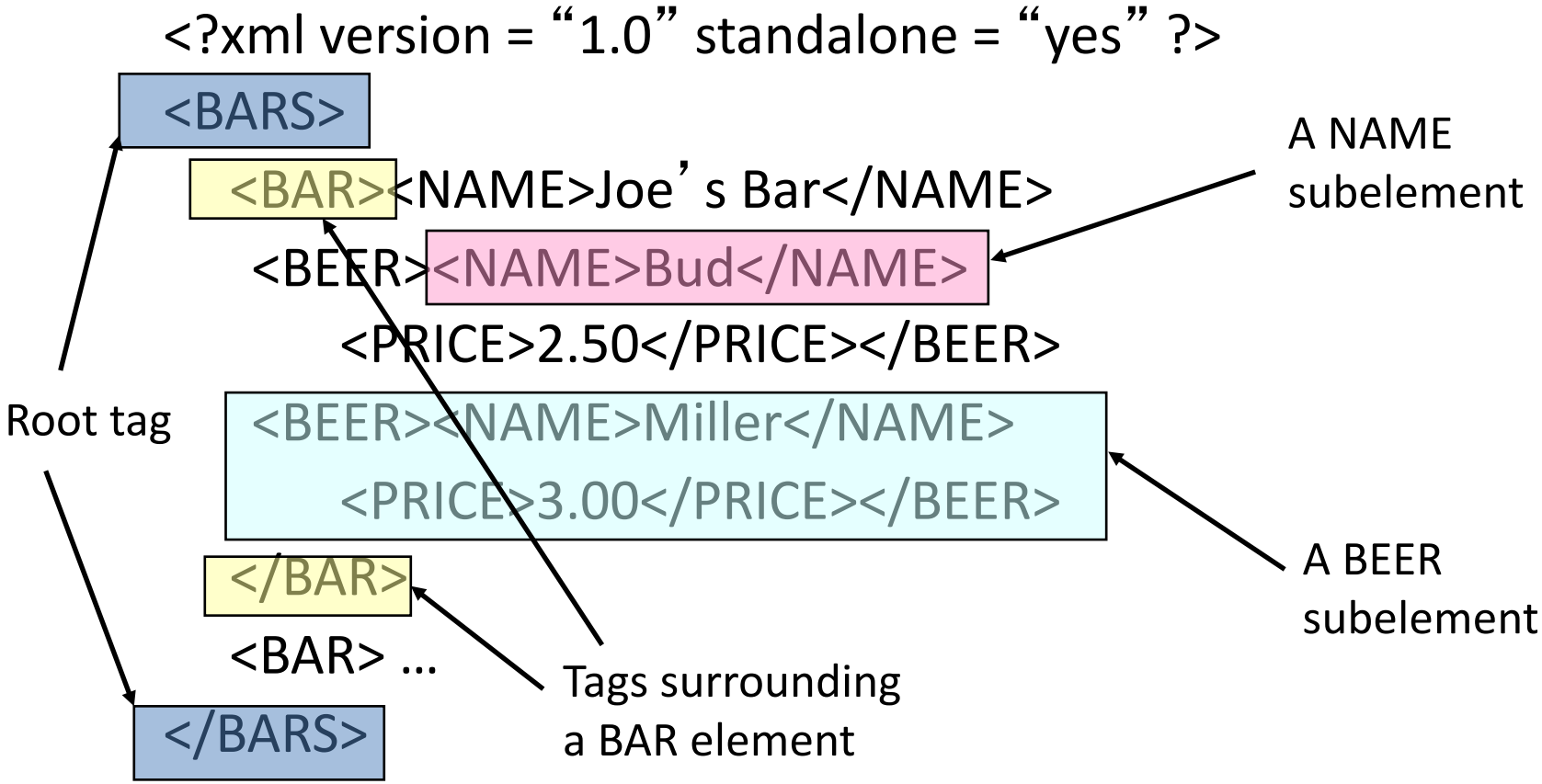
- Start the document with a *declaration*, surrounded by
  <?xml … ?> .
- Normal declaration is:
  <?xml version = ”1.0” standalone = ”yes” ?>
  - “standalone” = “no Data Type Definition (DTD) provided”

- The document starts with a *root tag* that surrounds nested tags.

<Tags>

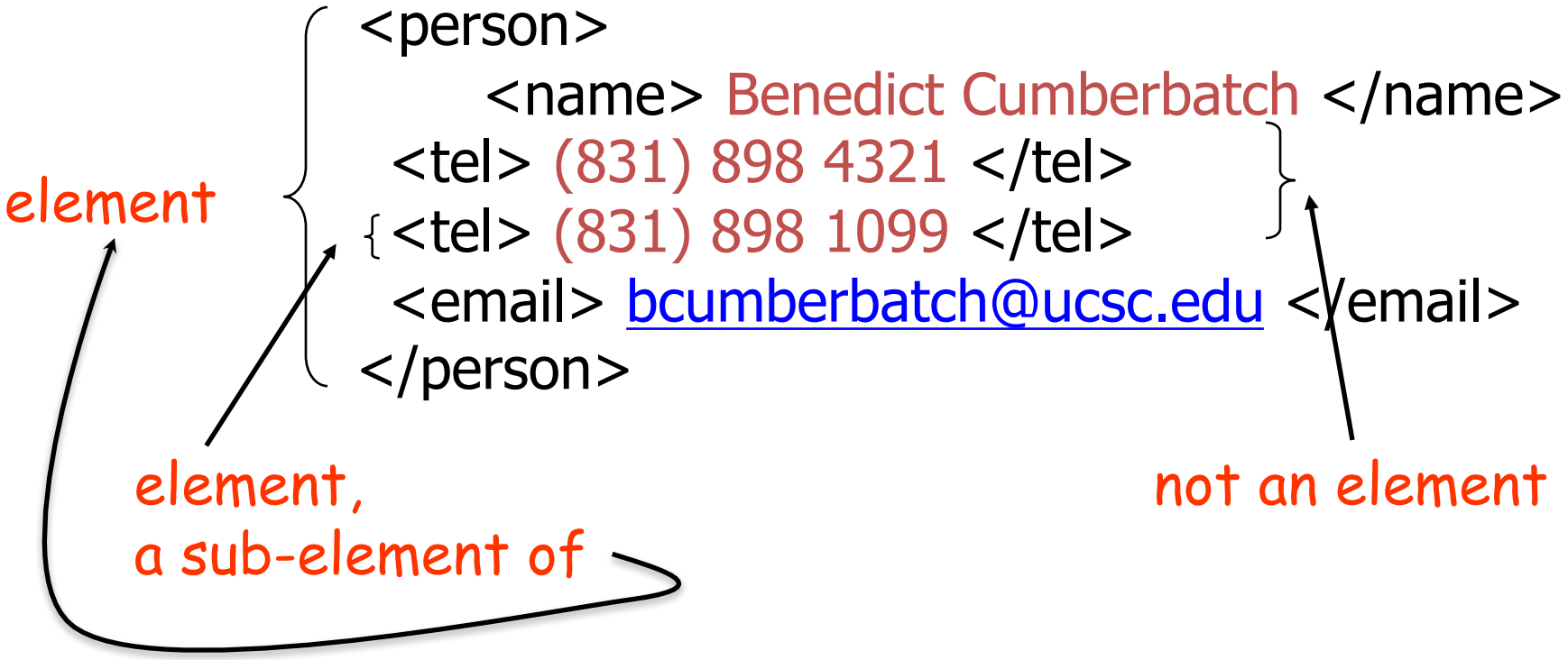
- Tags are normally matched pairs, as <FOO> … </FOO>.
- XML tags are case-sensitive.
  - E.g., <FOO> … </foo> does not match.
- Tags may be nested arbitrarily.
- XML has only one basic type, which is text.

Example: Well-Formed XML



More Terminology

- The segment of an XML document between an opening and a corresponding closing tag is called an *element*.



Using XML to Specify a Tuple

```
<person>
  <name> Benedict Cumberbatch</name>
  <tel> (831) 898 4321 </tel>
  <email> bcumberbatch@ucsc.edu </email>
</person>
```

Using XML to Specify a List

- We can represent a list by using the *same* tag repeatedly:

```
<addresses>
   <person> ... </person>
   <person> ... </person>
   <person> ... </person>
   ...
</addresses>
```

Example:
Two Ways of Representing a DB

projects:

title	budget	managedBy

employees:

name	ssn	age

Project and Employee Relations in XML

```
<db>
  <project>
    <title> Pattern recognition </title>
    <budget> 10000 </budget>
    <managedBy> Joe </managedBy>
  </project>
  <employee>
    <name> Joe </name>
    <ssn> 344556 </ssn>
    <age> 34 < /age>
  </employee>
  <employee>
    <name> Sandra </name>
    <ssn> 2234 </ssn>
    <age> 35 </age>
  </employee>
  <project>
    <title> Auto guided vehicle </title>
    <budget> 70000 </budget>
    <managedBy> Sandra </managedBy>
  </project>
  :
</db>
```

Way 1: Projects and employees are intermixed.

Project and Employee Relations in XML (cont'd)

```
<db>
   <projects>
     <project>
          <title> Pattern recognition </title>
          <budget> 10000 </budget>
          <managedBy> Joe </managedBy>
     </project>
     <project>
          <title> Auto guided vehicles </title>
          <budget> 70000 </budget>
          <managedBy> Sandra </managedBy>
     </project>
    :
   </projects>
```

```
<employees>
  <employee>
     <name> Joe </name>
     <ssn> 344556 </ssn>
     <age> 34 </age>
  </employee>
  <employee>
     <name> Sandra </name>
     <ssn> 2234 </ssn>
     <age>35 </age>
  </employee>
 :
  <employees>
</db>
```

Way 2: Employees follow projects.

Project and Employee Relations in XML (cont'd)

```
<db>
  <projects>
    <title> Pattern recognition </title>
    <budget> 10000 </budget>
    <managedBy> Joe </managedBy>
    <title> Auto guided vehicles </title>
    <budget> 70000 </budget>
    <managedBy> Sandra </managedBy>
    :
  </projects>
  <employees>
    <name> Joe </name>
    <ssn> 344556 </ssn>
    <age> 34 </age>
    <name> Sandra </name>
    <ssn> 2234 </ssn>
    <age> 35 </age>
    :
  </employees>
</db>
```

Or without "separator" tags <project>, <employee>

Attributes

- An (opening) tag may contain *attributes*. These are typically used to describe the content of an element.
- Attributes cannot be repeated within a tag.

```
<entry>
   <word language = “en”> cheese </word>
   <word language = “fr”> fromage </word>
   <word language = “ro”> branza </word>
   <meaning> A food made … </meaning>
</entry>
```

Attributes (cont'd)

- Another common use for attributes is to express dimension or type.

```
<picture>
   <height dim= "cm"> 2400 </height>
   <width dim= "in"> 96 </width>
   <data encoding = "gif" compression = "zip">
      M05-.+C$@02!G96YEFEC ...
   </data>
</picture>
```

- A document that obeys the "nested tags" rule and does not repeat an attribute within a tag is said to be *well-formed* .

When to Use Attributes

- It's not always clear when to use attributes vs. elements.

```
<person ssno= "123 45 6789">
  <name> F. MacNiel </name>
  <email>
     fmacn@dcs.barra.ac.sc
  </email>
  ...
</person>
```

```
<person>
  <ssno> 123 45 6789 </ssno>
  <name> F. MacNiel </name>
  <email>
     fmacn@dcs.barra.ac.sc
  </email>
  ...
</person>
```

Using IDs and IDRefs

```
<family>
    <person  id="jane" mother="mary" father="john">
        <name> Jane Doe </name>
    </person>
    <person id="john" children="jane jack">
        <name> John Doe </name>
    </person>
    <person id="mary" children="jane jack">
        <name> Mary Doe </name>
    </person>
        <person  id="jack" mother="mary" father="john">
        <name> Jack Doe </name>
    </person>
</family>
```

An Example

```
<db>
  <movie id="m1">
    <title>Waking Ned Divine</title>
    <director>Kirk Jones III</director>
    <cast idrefs="a1 a3"></cast>
    <budget>100,000</budget>
  </movie>
  <movie id="m2">
    <title>Dragonheart</title>
    <director>Rob Cohen</director>
    <cast idrefs="a2 a9 a21"></cast>
    <budget>110,000</budget>
  </movie>
  <movie id="m3">
    <title>Moondance</title>
    <director>Dagmar Hirtz</director>
    <cast idrefs="a1 a8"></cast>
    <budget>90,000</budget>
  </movie>
  :
  <actor id="a1">
    <name>David Kelly</name>
    <acted_In idrefs="m1 m3 m78" >
    </acted_In>
  </actor>
  <actor id="a2">
     <name>Sean Connery</name>
     <acted_In idrefs="m2 m9 m11">
     </acted_In>
     <age>68</age>
  </actor>
  <actor id="a3">
      <name>Ian Bannen</name>
      <acted_In idrefs="m1 m35">
      </acted_In>
  </actor>
   :
</db>
```

DTD Structure

<!DOCTYPE <root tag> [

    <!ELEMENT <name>(<components>)>

    . . . more elements . . .

]>

Document Type Descriptors

- Document Type Descriptors (DTDs) impose structure on an XML document, much like relation schemas impose a structure on relations.
- The DTD is just a *syntactic* specification.
  - Not a semantic specification

Example: Address Book

```
<person>
    <name> MacNiel, John </name>
    <greet> Dr. John MacNiel </greet>
    <addr>1234 Huron Street </addr>
    <addr> Rome, OH 98765 </addr>
    <tel> (321) 786 2543 </tel>
    <fax> (321) 786 2543 </fax>
    <tel> (321) 786 2543 </tel>
    <email> jm@abc.com </email>
</person>
```

Exactly one name

At most one greeting

As many address lines as needed (in order)

Mixed telephones and faxes

As many emails as needed

Specifying the Structure

The structure of a person entry can be specified by:

name, greet?, addr*, (tel | fax)*, email*

XML uses a form of Regular Expression (described later).

A DTD for Address Book

```
<!DOCTYPE addressbook [
  <!ELEMENT addressbook (person*)>
  <!ELEMENT person
      (name, greet?, address*, (fax | tel)*, email*)>
  <!ELEMENT name        (#PCDATA)>
  <!ELEMENT greet       (#PCDATA)>
  <!ELEMENT address     (#PCDATA)>
  <!ELEMENT tel         (#PCDATA)>
  <!ELEMENT fax         (#PCDATA)>
  <!ELEMENT email       (#PCDATA)>
]>
```

"Parsed Character Data" (i.e., text)

Our Relational DB Revisited

projects:

title	budget	managedBy

employees:

name	ssn	age

Two Potential DTDs for the Relational DB

```
<!DOCTYPE db [
   <!ELEMENT db           (projects, employees)>
   <!ELEMENT projects     (project*)>
   <!ELEMENT employees (employee*)>
  <!ELEMENT  project      (title, budget, managedBy)>
   <!ELEMENT employee   (name, ssn, age)>
   ...
]>
```

```
 <!DOCTYPE db [
    <!ELEMENT db          (project | employee)*>
    <!ELEMENT project     (title, budget, managedBy)>
    <!ELEMENT employee (name, ssn, age)>
    ...
 ]>
```

Some Things are Hard to Specify

Each employee element is to contain name, age and ssn elements in some order.

```
<!ELEMENT employee
   ( (name, age, ssn) | (age, ssn, name) | (ssn, name, age) | ...
   )>
```

Suppose there were many more fields …

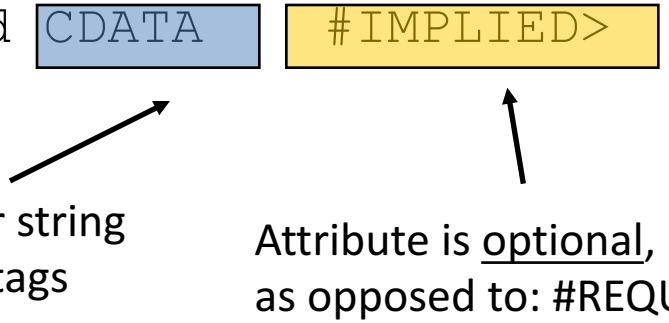
Summary of XML Regular Expressions

- A The tag A occurs
- e1,e2 The expression e1 followed by e2
- e* 0 or more occurrences of e
- e? Optional -- 0 or 1 occurrences
- e+ 1 or more occurrences
- e1 | e2 either e1 or e2
- (e) grouping, e.g.,
  <!ELEMENT Address Street, (City | Zip)

Specifying Attributes in the DTD

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

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



Example of Attribute Use

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

```
<BAR kind = ”sushi”>
   <NAME>Homma’s</NAME>
   <BEER><NAME>Sapporo</NAME>
       <PRICE>5.00</PRICE></BEER>
   ...
</BAR>
```

Specifying ID and IDREF Attributes in a DTD

```
<!DOCTYPE family [
  <!ELEMENT family   (person)*>
  <!ELEMENT person  (name)>
  <!ELEMENT name    (#PCDATA)>
  <!ATTLIST person
            id          ID      #REQUIRED
            mother   IDREF   #IMPLIED
            father    IDREF   #IMPLIED
            children  IDREFS  #IMPLIED>
]>
```

id is an ID attribute

An XML Document That Conforms to the DTD

```
<family>
    <person  id="jane"  mother="mary" father="john">
        <name> Jane Doe </name>
    </person>
    <person id="john" children="jane jack">
        <name> John Doe </name>
    </person>
    <person id="mary" children="jane  jack">
        <name> Mary Doe </name>
    </person>
        <person  id="jack"  mother=”mary" father="john">
        <name> Jack Doe </name>
    </person>
</family>
```

Consistency of ID and IDREF Attribute Values

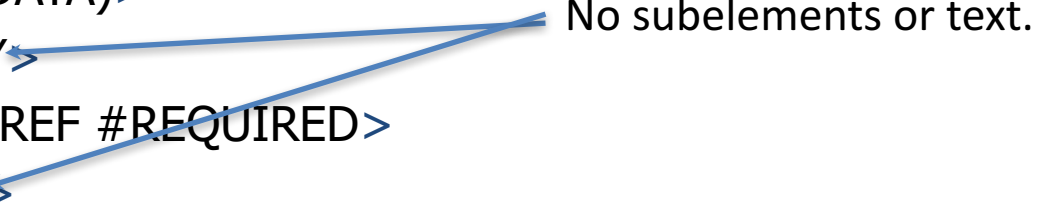
- ID stands for identifier. The values across all IDs must be distinct.

- IDREF stands for identifier reference. If an attribute is declared as IDREF, then …
  - the associated value must exist as the value of some ID attribute (i.e., no dangling "pointers").

- IDREFS specifies "several" (0 or more) identifiers.

An Alternative DTD Specification

```
<!DOCTYPE family [
  <!ELEMENT family   (person)*>
  <!ELEMENT person  (name, mother?, father?, children?)>
  <!ATTLIST person id ID #REQUIRED>
  <!ELEMENT name    (#PCDATA)>
  <!ELEMENT mother EMPTY>
  <!ATTLIST mother idref IDREF #REQUIRED>
  <!ELEMENT father EMPTY>
  <!ATTLIST father idref IDREF #REQUIRED>
  <!ELEMENT children EMPTY>
  <!ATTLIST children idrefs IDREFS #REQUIRED>
]>
```



Revised Data for the Alternative DTD

```
<family>
    <person  id = "jane”>
        <name> Jane Doe </name>
        <mother idref = "mary”></mother>
        <father idref = "john"></father>
    </person>
    <person id = "john”>
        <name> John Doe </name>
        <children idrefs = "jane jack"> </children>
    </person>
    ...
</family>
```

A Useful Abbreviation for Empty

- When an element has empty content we can write:

  <tag blahblahbla/> instead of <tag blahblahbla></tag>

For example:

```
<family>
    <person  id = "jane”>
        <name> Jane Doe </name>
    <mother idref = "mary”/>
    <father idref = "john”/>
    </person>
    ...
</family>
```

movieschema.dtd

```
<!DOCTYPE db [
  <!ELEMENT   db       (movie+, actor+)>
  <!ELEMENT   movie    (title, director, cast, budget)>
    <!ATTLIST          movie   id   ID   #REQUIRED>
  <!ELEMENT   title    (#PCDATA)>
  <!ELEMENT   director (#PCDATA)>
  <!ELEMENT   cast     EMPTY>
    <!ATTLIST cast     idrefs  IDREFS  #REQUIRED>
  <!ELEMENT   budget   (#PCDATA)>
```

movieschema.dtd (cont'd)

```
  <!ELEMENT   actor      (name, acted_In, age?, directed*)>
  <!ATTLIST   actor      id      ID       #REQUIRED>
  <!ELEMENT   name       (#PCDATA)>
  <!ELEMENT   acted_In   EMPTY>
     <!ATTLIST  acted_In        idrefs   IDREFS  #REQUIRED>
  <!ELEMENT   age        (#PCDATA)>
  <!ELEMENT   directed   (#PCDATA)>
]>
```

Connecting the Document with its DTD

- In line:

```
<?xml version="1.0"?>
<!DOCTYPE db [<!ELEMENT ...> … ]>
<db> ... </db>
```

Includes everything from movieschema.dtd

- Another file:

```
<!DOCTYPE db SYSTEM ”movieschema.dtd">
```

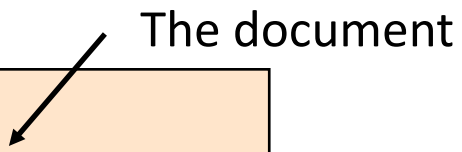
Note word SYSTEM

- A URL:

```
<!DOCTYPE db SYSTEM
        "http://www.schemaauthority.com/movieschema.dtd">
```

First Example

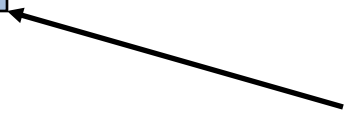
```
<?xml version = "1.0" standalone = "no" ?>
<!DOCTYPE BARS [
    <!ELEMENT BARS (BAR*)>
    <!ELEMENT BAR (NAME, BEER+)>
    <!ELEMENT NAME (#PCDATA)>
    <!ELEMENT BEER (NAME, PRICE)>
    <!ELEMENT PRICE (#PCDATA)>
]>
<BARS>
    <BAR><NAME>Joe's Bar</NAME>
      <BEER><NAME>Bud</NAME> <PRICE>2.50</PRICE></BEER>
      <BEER><NAME>Miller</NAME> <PRICE>3.00</PRICE></BEER>
    </BAR>
    <BAR> …
</BARS>
```



Second Example

- 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

Well-Formed and Valid Documents

- We say that an XML document is *well-formed* if the document (with or without an associated DTD) has proper nesting of tags and the attributes of every element are all unique.

- We say that an XML document x is *valid* with respect to a DTD D if x conforms to D. That is, if the document x conforms to the regular expression grammar and constraints given by D.

DTDs versus Schemas (or Types)

- By database (or programming language) standards DTDs are rather weak specifications.
  - Only one base type -- PCDATA
  - No useful "abstractions" e.g., no sets
  - IDREFs are untyped. They allow you to reference something, but you don't know what!
  - Few constraints. E.g., "Local keys" as opposed to global IDs.
  - Tag definitions are *global.*

- XML Schema:
  - An extension of DTDs that allows one to impose a schema or type on an XML document.

XML Schema

- A more powerful way to describe the structure of XML documents.

- XML-Schema declarations are themselves XML documents.
  - They describe “elements” and the things doing the describing are also “elements”.
  - See textbook, Section 11.4.

Query Languages for XML

- XPath: Language for navigating through an XML document.
  - See textbook, Section 12.1.
- XQuery: Query language for XML, similar in power to SQL.
  - See textbook, Section 12.2.
- XSLT: Language for extracting information from an XML document and transforming it.
  - See textbook, Section 12.3.