

XML and DTD

Introduction to Databases

CompSci 316 Fall 2019



DUKE
COMPUTER SCIENCE

Announcements (Mon. Oct. 14)

- Gradiance XML exercise assigned; due in a week
- Project milestone 1 feedback to be emailed around this weekend
 - Weekly updates to start next week—details will be announced via email
- Midterm scores will be readjusted after all processing all regrading requests
 - Sorry about all the T/F grading issues—please check sample solution posted on Sakai!

Structured vs. unstructured data

- Relational databases are highly structured
 - All data resides in tables
 - You must define schema before entering any data
 - Every row confirms to the table schema
 - Changing the schema is hard and may break many things
- Texts are highly unstructured
 - Data is free-form
 - There is no pre-defined schema, and it's hard to define any schema
 - Readers need to infer structures and meanings

What's in between these two extremes?

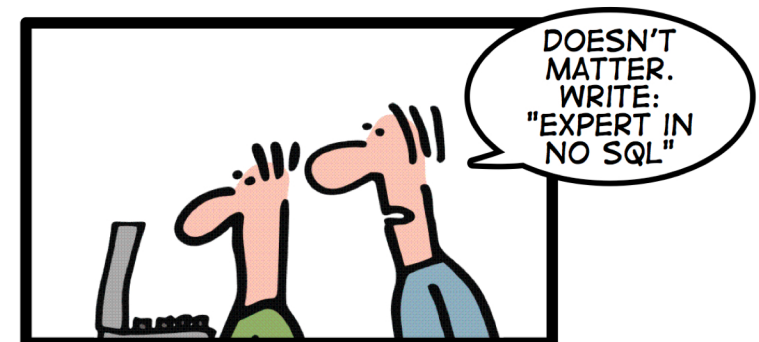
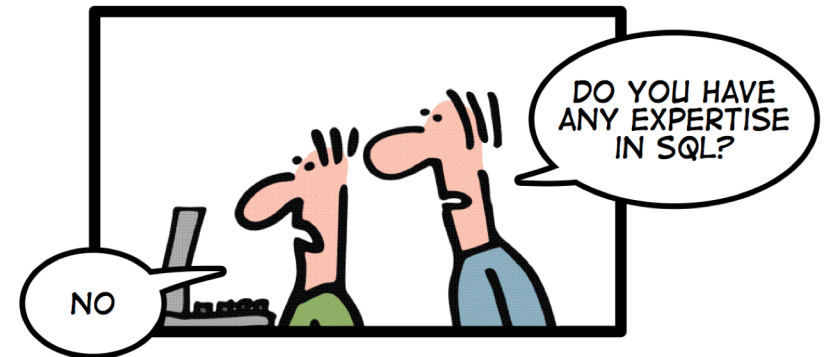
Semi-structured data

- Observation: most data have some structure, e.g.:
 - Book: chapters, sections, titles, paragraphs, references, index, etc.
 - Item for sale: name, picture, price (range), ratings, promotions, etc.
 - Web page: HTML
- Ideas:
 - Ensure data is “well-formatted”
 - If needed, ensure data is also “well-structured”
 - But make it easy to define and extend this structure
 - Make data “self-describing”

SQL vs. NoSQL

- SQL's rigidity in face of semi-structured data is one of the reasons behind the rise of (some) NoSQL systems
 - NoSQL has other motivations, which we hope to get to in a later part of this course

HOW TO WRITE A CV



Leverage the NoSQL boom

Our roadmap thru the NoSQL land

<congress>
 <people>
 <person birthday="1952-11-09" gender="M" id="B000944" name="Sherrod Brown">
 <role district="13" enddate="1995-01-03" party="Democrat" startdate="1993-01-05" state="OH" type="rep"/>
 <role district="13" enddate="1997-01-03" party="Democrat" startdate="1995-01-04" state="OH" type="rep"/>
 <role district="13" enddate="1999-01-03" party="Democrat" startdate="1997-01-07" state="OH" type="rep"/>
 <role district="13" enddate="2001-01-03" party="Democrat" startdate="1999-01-06" state="OH" type="rep"/>
 <role district="13" enddate="2003-01-03" party="Democrat" startdate="2001-01-03" state="OH" type="rep"/>
 <role district="13" enddate="2005-01-03" party="Democrat" startdate="2003-01-07" state="OH" type="rep"/>
 <role district="13" enddate="2007-01-03" party="Democrat" startdate="2005-01-04" state="OH" type="rep"/>
 <role enddate="2013-01-03" party="Democrat" startdate="2007-01-04" state="OH" type="sen"/>
 <role current="1" enddate="2019-01-03" party="Democrat" startdate="2013-01-03" state="OH" type="sen"/>
 </person>
 <person birthday="1958-10-13" gender="F" id="C000127" name="Maria Cantwell">

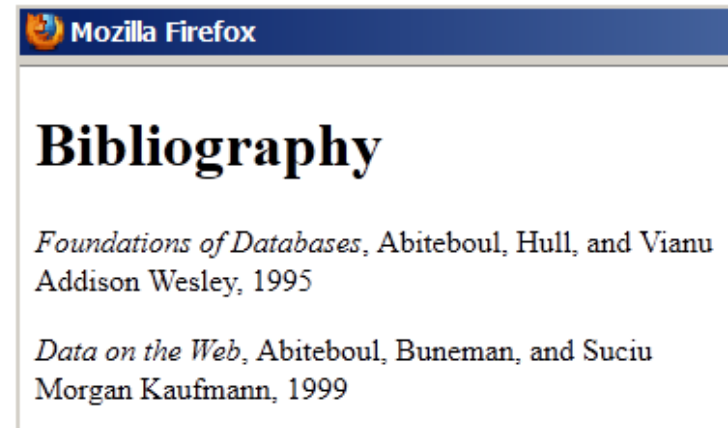
- But can't relational databases do XML?

```
{ "_id" : "B000944", "birthday" : ISODate("1952-11-09T00:00:00Z"), "gender" : "M", "name" : "Sherrod Brown", "roles" : [ {
  "district" : 13, "enddate" : ISODate("1995-01-03T00:00:00Z"), "party" : "Democrat", "startdate" : ISODate("1993-01-
05T00:00:00Z"), "state" : "OH", "type" : "rep" }, { "district" : 13, "enddate" : ISODate("1997-01-03T00:00:00Z"), "party"
: "Democrat", "startdate" : ISODate("1995-01-04T00:00:00Z"), "state" : "OH", "type" : "rep" }, { "district" : 13,
"enddate" : ISODate("1999-01-03T00:00:00Z"), "party" : "Democrat", "startdate" : ISODate("1997-01-07T00:00:00Z"), "state"
: "OH", "type" : "rep" }, { "district" : 13, "enddate" : ISODate("2001-01-03T00:00:00Z"), "party" : "Democrat",
"startdate" : ISODate("1999-01-06T00:00:00Z"), "state" : "OH", "type" : "rep" }, { "district" : 13, "enddate" :
ISODate("2003-01-03T00:00:00Z"), "party" : "Democrat", "startdate" : ISODate("2001-01-03T00:00:00Z"), "state" : "OH",
"type" : "rep" }, { "district" : 13, "enddate" : ISODate("2005-01-03T00:00:00Z"), "party" : "Democrat", "startdate" :
ISODate("2003-01-07T00:00:00Z"), "state" : "OH", "type" : "rep" }, { "district" : 13, "enddate" : ISODate("2007-01-
03T00:00:00Z"), "party" : "Democrat", "startdate" : ISODate("2005-01-04T00:00:00Z"), "state" : "OH", "type" : "rep" }, {
"enddate" : ISODate("2013-01-03T00:00:00Z"), "party" : "Democrat", "startdate" : ISODate("2007-01-04T00:00:00Z"), "state"
: "OH", "type" : "sen" }, { "current" : 1, "enddate" : ISODate("2019-01-03T00:00:00Z"), "party" : "Democrat", "startdate"
: ISODate("2013-01-03T00:00:00Z"), "state" : "OH", "type" : "sen" } ] }
{ "_id" : "C000127", "birthday" : ISODate("1958-10-13T00:00:00Z"), "gender" : "F", "name" : "Maria Cantwell", "roles" : [
{ "district" : 1, "enddate" : ISODate("1995-01-03T00:00:00Z"), "party" : "Democrat", "startdate" : ISODate("1993-01-
05T00:00:00Z"), "state" : "WA", "type" : "rep" }, { "enddate" : ISODate("2007-01-03T00:00:00Z"), "party" : "Democrat",
"startdate" : ISODate("2001-01-03T00:00:00Z"), "state" : "WA", "type" : "sen" }, { "enddate" : ISODate("2013-01-
03T00:00:00Z"), "party" : "Democrat", "startdate" : ISODate("2007-01-04T00:00:00Z"), "state" : "WA", "type" : "sen" }, {
"current" : 1, "enddate" : ISODate("2019-01-03T00:00:00Z"), "party" : "Democrat", "startdate" : ISODate("2013-01-
03T00:00:00Z"), "state" : "WA", "type" : "sen" } ] }
```



HTML: language of the Web

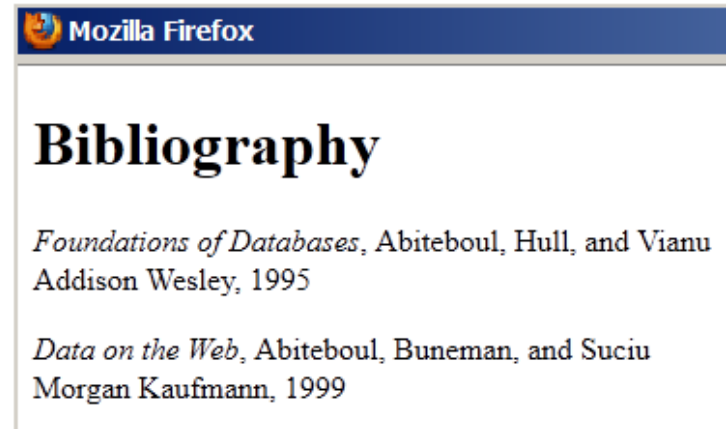
```
<h1>Bibliography</h1>
<p><i>Foundations of Databases</i>,
Abiteboul, Hull, and Vianu
<br>Addison Wesley, 1995
<p>...
```



- It's mostly a “formatting” language
- It mixes presentation and content
 - Hard to change presentation (say, for different displays)
 - Hard to extract content

XML: eXtensible Markup Language

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



- Text-based
- Capture data (content), not presentation
- Data self-describes its structure
 - Names and nesting of tags have meanings!

Other nice features of XML

- **Portability**: Just like HTML, you can ship XML data across platforms
 - Relational data requires heavy-weight API's
- **Flexibility**: You can represent any information (structured, semi-structured, documents, ...)
 - Relational data is best suited for structured data
- **Extensibility**: Since data describes itself, you can change the schema easily
 - Relational schema is rigid and difficult to change

XML terminology

```
<bibliography>
  <book ISBN="ISBN-10" price="80.00">
    <title>Foundations of Databases</title>
    <author>Abiteboul</author>
    <author>Hull</author>
    <author>Vianu</author>
    <publisher>Addison Wesley</publisher>
    <year>1995</year>
  </book>...
</bibliography>
```

- **Tag names:** `book`, `title`, ...
- **Start tags:** `<book>`, `<title>`, ...
- **End tags:** `</book>`, `</title>`, ...
- An **element** is enclosed by a pair of start and end tags: `<book>...</book>`
 - Elements can be nested:
`<book>...<title>...</title>...</book>`
 - Empty elements: `<is_textbook></is_textbook>`
 - Can be abbreviated: `<is_textbook/>`
- Elements can also have **attributes**:
`<book ISBN="..." price="80.00">`

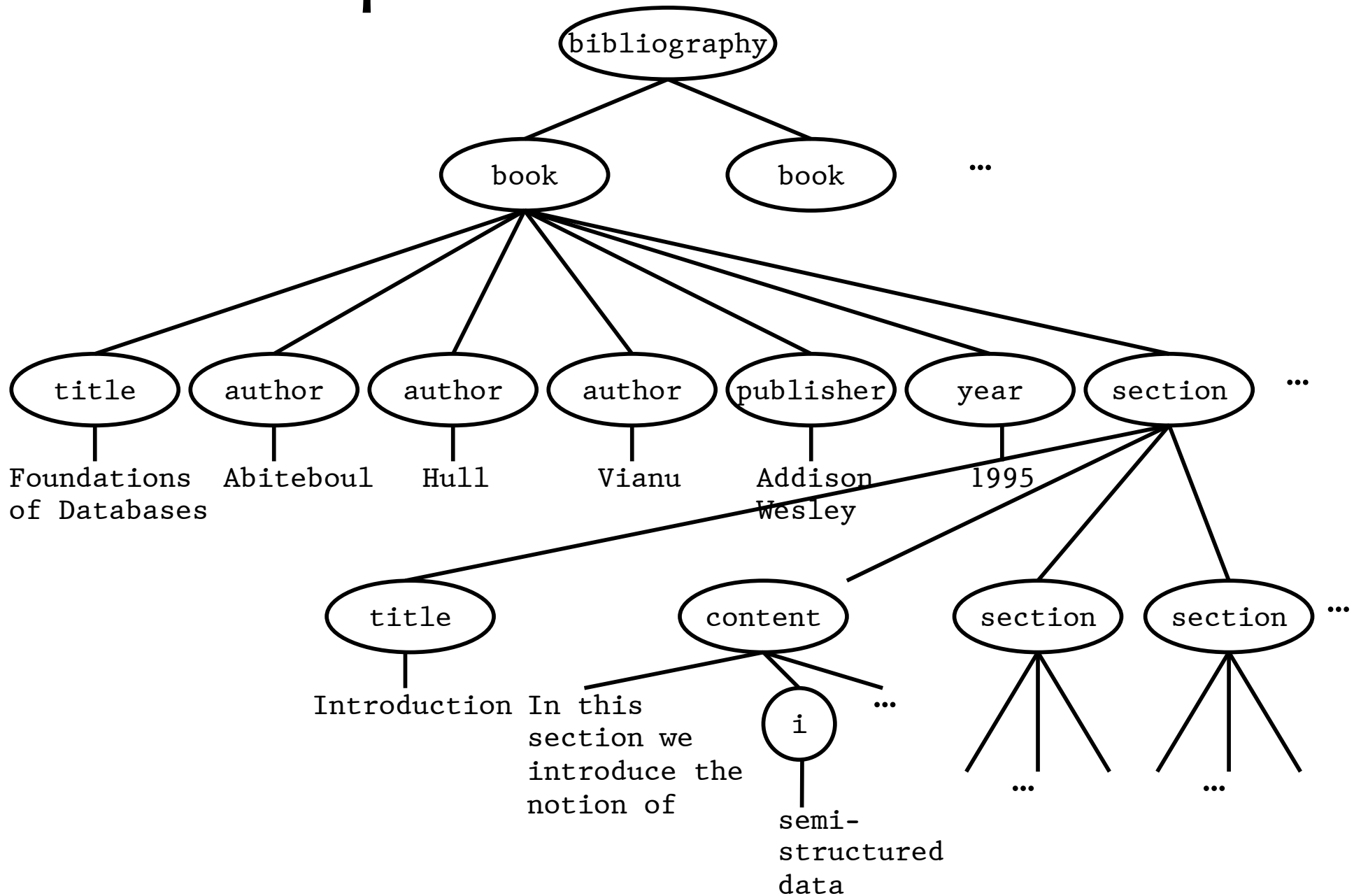
☞ Ordering generally matters, except for attributes

Well-formed XML documents

A **well-formed** XML document

- Follows XML lexical conventions
 - Wrong: `<section>We show that x < 0...</section>`
 - Right: `<section>We show that x < 0...</section>`
 - Other special entities: `>` becomes `>`; and `&` becomes `&`;
- Contains a single root element
- Has properly matched tags and properly nested elements
 - Right:
`<section>...<subsection>...</subsection>...</section>`
 - Wrong:
`<section>...<subsection>...</section>...</subsection>`

A tree representation



More XML features

- **Processing instructions** for apps: `<? ... ?>`
 - An XML file typically starts with a version declaration using this syntax: `<?xml version="1.0"?>`
- **Comments**: `<!-- Comments here -->`
- **CDATA section**: `<![CDATA[Tags: <book>, ...]]>`
- **ID's and references**
 - ID value must start with a non-digit


```
<person id="o12"><name>Homer</name>...</person>
<person id="o34"><name>Marge</name>...</person>
<person id="o56" father="o12" mother="o34">
  <name>Bart</name>...
</person>...
```
- **Namespaces** allow external schemas and qualified names


```
<myCitationStyle:book xmlns:myCitationStyle="http://.../mySchema">
  <myCitationStyle:title>...</myCitationStyle:title>
  <myCitationStyle:author>...</myCitationStyle:author>...
</book>
```
- And more...



Now for some
more structure...

Valid XML documents

- A **valid** XML document conforms to a **Document Type Definition (DTD)**
 - A DTD is optional
 - A DTD specifies a grammar for the document
 - Constraints on structures and values of elements, attributes, etc.

- Example

```
<!DOCTYPE bibliography [
    <!ELEMENT bibliography (book+)>
    <!ELEMENT book (title, author*, publisher?, year?, section*)>
    <!ATTLIST book ISBN ID #REQUIRED>
    <!ATTLIST book price CDATA #IMPLIED>
    <!ELEMENT title (#PCDATA)>
    <!ELEMENT author (#PCDATA)>
    <!ELEMENT publisher (#PCDATA)>
    <!ELEMENT year (#PCDATA)>
    <!ELEMENT i (#PCDATA)>
    <!ELEMENT content (#PCDATA|i)*>
    <!ELEMENT section (title, content?, section*)>
]>
```

DTD explained

```
<!DOCTYPE bibliography [
  ↳ bibliography is the root element of the document
  <!ELEMENT bibliography (book+)> ↳ One or more
  ↳ bibliography consists of a sequence of one or more book elements
  <!ELEMENT book (title, author*, publisher?, year?, section*)>
    ↳ Zero or more
    ↳ Zero or one
    ↳ book consists of a title, zero or more authors,
      an optional publisher, and zero or more section's, in sequence
  <!ATTLIST book ISBN ID #REQUIRED>
    ↳ book has a required ISBN attribute which is a unique identifier
  <!ATTLIST book price CDATA #IMPLIED>
    ↳ book has an optional (#IMPLIED)
      price attribute which contains
      character data
```

Other attribute types include
 IDREF (reference to an ID),
 IDREFS (space-separated list of references),
 enumerated list, etc.

```
<bibliography>
  <book ISBN="ISBN-10" price="80.00">
    <title>Foundations of Databases</title>
    <author>Abiteboul</author>
    <author>Hull</author>
    <author>Vianu</author>
    <publisher>Addison Wesley</publisher>
    <year>1995</year>
  </book>...
</bibliography>
```

DTD explained (cont'd)

```
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT i (#PCDATA)>
```

PCDATA is text that will be parsed

- < etc. will be parsed as entities
- Use a CDATA section to include text verbatim

↳ author, publisher, year, and i contain **parsed character data**

```
<!ELEMENT content (#PCDATA|i)*>
```

↳ content contains **mixed content**: text optionally interspersed with i elements

```
<!ELEMENT section (title, content?, section*)>
```

↳ **Recursive declaration**:

Each section begins with a title,
followed by an optional content,
and then zero or more
(sub) section's

```
<section><title>Introduction</title>
  <content>In this section we introduce
    the notion of <i>semi-structured data</i>...
  </content>
  <section><title>XML</title>
    <content>XML stands for...</content>
  </section>
  <section><title>DTD</title>
    <section><title>Definition</title>
      <content>DTD stands for...</content>
    </section>
    <section><title>Usage</title>
      <content>You can use DTD to...</content>
    </section>
  </section>
</section>
```

]>

Using DTD

- DTD can be included in the XML source file

- ```
<?xml version="1.0"?>
 <!DOCTYPE bibliography [
 ...
]>
 <bibliography>
 ...
 </bibliography>
```

- DTD can be external

- ```
<?xml version="1.0"?>
  <!DOCTYPE bibliography SYSTEM "../dtds/bib.dtd">
  <bibliography>
    ...
  </bibliography>
```
- ```
<?xml version="1.0"?>
 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
 <html>
 ...
 </html>
```

# Annoyance: content grammar

- Consider this declaration:

```
<!ELEMENT pub-venue
 ((name, address, month, year) |
 (name, volume, number, year))>
```

- “|” means “or”
- Syntactically legal, but won’t work
  - Because of SGML compatibility issues
  - When looking at name, a parser would not know which way to go without looking further ahead
  - Requirement: content declaration must be “deterministic” (i.e., no look-ahead required)
  - Can we rewrite it into an equivalent, deterministic one?
- Also, you cannot nest mixed content declarations
  - **Illegal:** `<!ELEMENT Section (title, (#PCDATA|i)*, section*)>`



# Annoyance: element name clash

- Suppose we want to represent book titles and section titles differently
  - Book titles are pure text: (`#PCDATA`)
  - Section titles can have formatting tags: (`#PCDATA | i | b | math`)\*
- But DTD only allows one `title` declaration!
- Workaround: rename as `book-title` and `section-title`?
  - Not nice—why can't we just infer a `title`'s context?

# Annoyance: lack of type support

- Too few attribute types: string (`CDATA`), token (e.g., `ID`, `IDREF`), enumeration (e.g., `(red|green|blue)`)
  - What about integer, float, date, etc.?
- ID not typed
  - No two elements can have the same id, even if they have different types (e.g., `book` vs. `section`)
- Difficult to reuse complex structure definitions
  - E.g.: already defined element `E1` as `(blah, bleh, foo?, bar*, ...)`; want to define `E2` to have the same structure
  - **Parameter entities** in DTD provide a workaround
    - `<!ENTITY % E.struct '(blah, bleh, foo?, bar*, ...)'`
    - `<!ELEMENT E1 %E.struct;>`
    - `<!ELEMENT E2 %E.struct;>`
  - Something less “hacky”?



Want even more  
structure support?

# XML Schema

- A more powerful way of defining the structure and constraining the contents of XML documents
  - Supports a rich set of types and user-defined types/structures
  - Supports notions of keys and foreign keys
- An XML Schema definition is itself an XML document
  - Typically stored as a standalone .xsd file
  - XML (data) documents refer to external .xsd files

```

<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

 <xs:element name="bibliography">
 <xs:complexType>
 <xs:sequence>
 <xs:element ref="book" minOccurs="0" maxOccurs="unbounded"/>
 </xs:sequence>
 </xs:complexType>
 <xs:key name="bookKey">
 <xs:selector xpath="./book"/>
 <xs:field xpath="@ISBN"/>
 </xs:key>
 <xs:keyref name="bookForeignKey" refer="bookKey">
 <xs:selector xpath="./book-ref"/>
 <xs:field xpath="@ISBN"/>
 </xs:keyref>
 </xs:element>

 <xs:element name="book">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="title" type="xs:string"/>
 <xs:element name="author" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
 <xs:element name="publisher" type="xs:string" minOccurs="0" maxOccurs="1"/>
 <xs:element name="year" type="xs:integer" minOccurs="0" maxOccurs="1"/>
 <xs:element ref="section" minOccurs="0" maxOccurs="unbounded"/>
 </xs:sequence>
 <xs:attribute name="ISBN" type="xs:string" use="required"/>
 <xs:attribute name="price" type="xs:decimal" use="optional"/>
 </xs:complexType>
 </xs:element>

 <xs:element name="section">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="title" type="xs:string"/>
 <xs:element ref="content" minOccurs="0" maxOccurs="1"/>
 <xs:element ref="section" minOccurs="0" maxOccurs="unbounded"/>
 </xs:sequence>
 </xs:complexType>
 </xs:element>

 <xs:element name="content">
 <xs:complexType mixed="true">
 <xs:choice minOccurs="0" maxOccurs="unbounded">
 <xs:element name="i" type="xs:string"/>
 <xs:element name="b" type="xs:string"/>
 <xs:element name="book-ref">
 <xs:complexType>
 <xs:attribute name="ISBN" type="xs:string"/>
 </xs:complexType>
 </xs:element>
 </xs:choice>
 </xs:complexType>
 </xs:element>

</xs:schema>

```

# Why use DTD or XML Schema?

- Benefits of not using them
  - Unstructured data is easy to represent
  - Overhead of validation is avoided
- Benefits of using them
  - Serve as schema for the XML data
    - Guards against errors
    - Helps with processing
  - Facilitate information exchange
    - People can agree to use a common DTD or XML Schema to exchange data (e.g., XHTML)

# XML versus relational data

## Relational data

- Schema is always fixed in advance and difficult to change
- Simple, flat table structures
- Ordering of rows and columns is unimportant
- Exchange is problematic
- “Native” support in all serious commercial DBMS

## XML data

- Well-formed XML does not require predefined, fixed schema
- Nested structure; ID/IDREF(S) permit arbitrary graphs
- Ordering forced by document format; may or may not be important
- Designed for easy exchange
- Often implemented as an “add-on” on top of relations

# Case study

- Design an XML document representing cities, counties, and states
  - For states, record name and capital (city)
  - For counties, record name, area, and location (state)
  - For cities, record name, population, and location (county and state)
- Assume the following:
  - Names of states are unique
  - Names of counties are only unique within a state
  - Names of cities are only unique within a county
  - A city is always located in a single county
  - A county is always located in a single state



# A possible design

