From HTML to XML (eXtensible Markup Language)

- HTML describes presentation of content
  ```html
  <h1>Bibliography</h1>
  <p><i>Foundations of Databases</i>
  Abiteboul, Hull, and Vianu
  <br>Addison Wesley, 1995
  </p>
  ```
- XML describes only the content
  ```xml
  <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>
  ```
- Separation of content from presentation simplifies content extraction and allows the same content to be presented easily in different looks

Other nice features of XML

- Portability: Just like HTML, you can ship XML data across platforms
  - Relational data requires heavy-weight protocols, e.g., JDBC
- 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

- 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"/>

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 &lt; 0.</section>
  - Other special entities: \ becomes &lt; and \& becomes &amp;
- Contains a single root element
- Has tags that are properly matched and elements that are properly nested
  - Wrong: <section>, subsection>, subsection>...</subsection></subsection>...</section>
  - Right: <section>, subsection>, subsection>...</subsection></subsection>...</section>
DTD explained (cont’d)

<ELEMENT title (#PCDATA)>
<ELEMENT author (#PCDATA)>
<ELEMENT year (#PCDATA)>
<ELEMENT content (#PCDATA)>
<ELEMENT publisher (#PCDATA)>
<ELEMENT section (#PCDATA)>

Recursive declaration:
Each section begins with a title,
followed by an optional content,
and then zero or more (sub) sections.

DTD explained

<DOCTYPE bibliography [
  <!ELEMENT bibliography (book)>]
  <!ELEMENT book (title, author, publisher, year, section)*>
  <!ELEMENT title (#PCDATA)>
  <!ELEMENT publisher (#PCDATA)>
  <!ELEMENT year (#PCDATA)>
  <!ELEMENT content (#PCDATA)>
Annoyances: element type declarations

- Consider this element content (children) declaration:
  ```xml
  <!ELEMENT pub-venue ( (name, address, month, year) | (name, volume, number, year) )>
  ```
- You can nest element content (children) declarations
- But the above nesting may be illegal
  - 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*)>`

Annoyances: 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 one infer title’s contexts from data?

Annoyances: 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 value, even if they are different types of elements (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"?

XML Schema

- A more powerful way of defining the structure and constraining the contents of XML documents
- An XML Schema definition is itself an XML document
  - Typically stored as a standalone .xsd file
  - XML (data) documents refer to external .xsd files
- W3C recommendation
  - Unlike DTD, XML Schema is separate from the XML specification

XML Schema definition (XSD)

```xml
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  ...<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:attribute name="ISBN" type="xs:string" use="required"/>
      <xs:attribute name="price" type="xs:decimal" use="optional"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```
XSD example cont’d

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

Another title definition; can be different from book/title
Declares mixed content
A compositor like xs:sequence; this one declares a list of alternatives,
like "(…|…|…)" in DTD
Recursive definition

To complete bib.xsd:

```xml
<xs:element name="bibliography">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="book" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

To use bib.xsd in an XML document:

```xml
<?xml version="1.0"?>
<bibliography xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="file:bib.xsd">
  <book>… …</book>
  <book>… …</book>
  … …
</bibliography>
```

Named types

Define once:

```xml
<xs:complexType name="formattedTextType" mixed="true">
  <xs:choice minOccurs="0" maxOccurs="unbounded">
    <xs:element name="i" type="xs:string"/>
    <xs:element name="b" type="xs:string"/>
  </xs:choice>
</xs:complexType>
```

Use elsewhere in XSD:

```xml
<xs:element name="title" type="formattedTextType"/>
<xs:element name="content" type="formattedTextType" minOccurs="0" maxOccurs="1"/>
```

Restrictions

```xml
<xs:simpleType name="priceType">
  <xs:restriction base="xs:decimal">
    <xs:minInclusive value="0.00"/>
  </xs:restriction>
</xs:simpleType>
```

```xml
<xs:simpleType name="statusType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="in stock"/>
    <xs:enumeration value="out of stock"/>
    <xs:enumeration value="out of print"/>
  </xs:restriction>
</xs:simpleType>
```

Keys

```xml
<xsl:element name="bibliography">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="bookKey">
        <xs:complexType>
          <xs:attribute name="ISBN" type="xs:string"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Under any bibliography element, elements reachable by selector "/book" (i.e., book child elements) must have unique values for field "ISBN" (i.e., ISBN attributes)
- In general, a key can consist of multiple fields (multiple <xs:field> elements under <xs:key>)
- More on XPath in next lecture

Foreign keys

```xml
<xs:simpleType name="bookKey">
  <xs:complexType>
    <xs:attribute name="ISBN" type="xs:string"/>
  </xs:complexType>
</xs:element>
```

Suppose content can reference books

```xml
<xs:element name="content">
  <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element name="i" type="xs:string"/>
      <xs:element name="b" type="xs:string"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

Under any content element, for elements reachable by selector "/book-ref" (i.e., book-ref child elements), values for field "ISBN" (i.e., ISBN attributes) must appear as values of bookKey, the key being referred

```xml
<xsl:element name="bookref">
  <xs:complexType>
    <xs:attribute name="ISBN" type="xs:string"/>
  </xs:complexType>
</xs:element>
```
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

<table>
<thead>
<tr>
<th>Relational data</th>
<th>XML data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema is always fixed in advance and difficult to change</td>
<td>Well-formed XML does not require predefined, fixed schema</td>
</tr>
<tr>
<td>Simple, flat table structures</td>
<td>Nested structure; ID/IDREF(S) permit arbitrary graphs</td>
</tr>
<tr>
<td>Ordering of rows and columns is unimportant</td>
<td>Ordering forced by document format; may or may not be important</td>
</tr>
<tr>
<td>Data exchange is problematic</td>
<td>Designed for easy exchange</td>
</tr>
<tr>
<td>“Native” support in all serious commercial DBMS</td>
<td>Often implemented as an “add-on” on top of relations</td>
</tr>
</tbody>
</table>

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

```
declare stateKey in geo_db with
  selector ./state
  field name xs:string

declare countyInStateKey in state with
  selector ./county
  field name xs:string

declare cityInCountyKey in county with
  selector ./city
  field name xs:string

declare capitalCityIdKeyRef in geo_db referencing cityIdKey, with
  selector ./state
  field @capital_city_id

declare cityIdKey in geo_db with
  selector ./state/county/city
  field @id
```