From HTML to XML (eXtensible Markup Language)

- HTML describes presentation of content
  
  From HTML to XML (eXtensible Markup Language)
  
  `<h1>Bibliography</h1>`
  
  `<p>`
  
  `<i>Foundations of Databases</i>`, Abiteboul, Hull, and Vianu
  
  `<p>`
  
  `<br>Addison Wesley, 1995`
  
  `<p>`
  
  ...`
  
  `<p>`
  
  XML describes only the content
  
  `<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

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: `&lt;` becomes `<`; and `&` becomes `&amp;`
  
  - Contains a single root element
  
  - Has properly matched tags and properly nested elements
    
    - Right: `<section><subsection>...</subsection></section>`
    
    - Wrong: `<section><subsection>...</subsection></section>`

XML, DTD, and XML Schema

CompSci 316
Introduction to Database Systems

Announcements (Thu. Oct. 18)

- Project milestone #1 due today!
- Midterm being graded; sample solution available next Tuesday
- Homework #3 available next Thursday

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>...</book>`
  
  - Empty elements: `<is_textbook>`
    
    - Can be abbreviated: `<is_textbook/>`
  
  - Elements can also have attributes: `<book ISBN="" price="80.00"/>`
  
  - Ordering generally matters, except for attributes
A tree representation

Bibliography

book

... 

title

author

author

author

year

publisher

Introduction

We introduce the notion of semi-structured data... 

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.

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

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
  ```
  <person id="123"><name>Homer</name><name>Marge</name><name>...</name><name><person>
  </person>
  ...
  ```
- Namespaces allow external schemas and qualified names
  ```
  <book xmlns="http://...">...
  ```
- And more...

DTD explained

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

Using DTD

- DTD can be included in the XML source file
  ```
  <!DOCTYPE bibliography SYSTEM "...">
  ```
- DTD can be external
  ```
  <!DOCTYPE html PUBLIC "..." "...">
  ```
Annoyances: element type declarations

- Consider this element content (children) declaration:
  ```xml
  <!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*)>`

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 definition (XSD)

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

- Defines `xs` to be the namespace described in the URL
- Uses of `xs` within the `xs:schema` element now refer to tags from this namespace

XSD example

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

- Declares a structure with child elements/attributes as opposed to just text
- Declares a sequence of child elements, like “(…, …, …)” in DTD
- A leaf element with string content
- A leaf element with integer content
- Reference to element `section` defined elsewhere
- The attribute has a decimal value, and it is optional
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:element ref="section" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Another title definition; can be different from book/title
Declares mixed content (text interspersed with structure below)
A compositor like `xs:sequence`; this one declares a list of alternatives, like `(…|…|…)`
imin/maxOccurs can be attached to compositors too
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
<xs:element name="bibliography">
  <xs:complexType>
    <xs:key name="bookKey">
      <xs:selector xpath="./book"/>
      <xs:field xpath="@ISBN"/>
    </xs:key>
  </xs:complexType>
</xs:element>
```

Under any bibliography element, elements reachable by `./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:element name="bibliography">
  <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:complexType>
</xs:element>
```

Suppose content can reference books
```xml
<xs:element name="bibliography">
  <xs:complexType>
    <xs:selector xpath="./book-ref"/>
    <xs:field xpath="@isbn"/>
  </xs:complexType>
</xs:element>
```

Under any bibliography element, for elements reachable by selector `./book-ref` (i.e., any book-ref element underneath), values for field "ISBN" (i.e., ISBN attributes) must appear as values of `bookKey`, the key being referred
- Make sure keyref is declared in the same scope as the key it refers to
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