XML, DTD, and XML Schema

Announcements (Thu. Oct. 18)

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

From HTML to XML (eXtensible Markup Language)

- HTML describes presentation of content

```
<p>Foundations of Databases</p>
```

- XML describes only the content

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

- 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">...
- 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 &lt; 0...</section>
    - Other special entities: \ becomes &lt; 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>
More XML features

- Processing instructions for apps: `<? ... java applet ... ?>`
  - 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="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:
  - `<book xmlns:myCitationStyle="http://.../mySchema">
    <myCitationStyle:title>...</myCitationStyle:title>
    <myCitationStyle:author>...</myCitationStyle:author>...
  </book>`
- And more...

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

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

Bibliography

Bibliography is the root element of the document. It consists of a sequence of one or more `book` elements.

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

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

DTD explained (cont’d)

```
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT i (#PCDATA)>
<!ELEMENT content (#PCDATA|i)*>
<!ELEMENT section (title, content?, section*)>
```

Content:

```
<section><title>Introduction</title>
<content>In this section we introduce the notion of <i>semi-structured data</i>…</content>
</section>
<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>
```

PCDATA:

- Text that will be parsed.
- Entities like `<` will be parsed as entities.
- Use a CDATA section to include text verbatim.

PCDATA contains mixed content: text optionally interspersed with `i` elements.

Recursive declaration:

```
Each section begins with a title, followed by an optional content, and then zero or more (sub)sections.
```
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
  - 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|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
dering 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:schema>
```

XSD example

```xml
<xs:complexType name="book">
  <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>
```

This attribute has a decimal value, and it is optional
XSD example cont’d

```xml
<xsd:element name="section">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="title" type="xsd:string"/>
      <xsd:element name="content" minOccurs="0" maxOccurs="1">
        <xsd:complexType mixed="true">
          <xsd:choice minOccurs="0" maxOccurs="unbounded">
            <xsd:element name="i" type="xsd:string"/>
            <xsd:element name="b" type="xsd:string"/>
          </xsd:choice>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Another title definition; can be different from book/title
Declares mixed content
A compose like xs:sequence, also use xs:element name="i" type="xsd:string"/>
Like "(…|…|…)" in DTD
min/maxOccurs can be attached to compositors too
Recursive definition

XSD example cont’d

- To complete bib.xsd:
  ```xml
  <xsd:element name="bibliography">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="book" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  ```

- To use bib.xsd in an XML document:
  ```xml
  <?xml version="1.0"?>
    <book>… …</book>
    <book>… …</book>
    … …
  </bibliography>
  ```

Named types

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

- Use elsewhere in XSD:
  ```xml
  <xsd:element name="title" type="formattedTextType"/>
  <xsd: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>

<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 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:element name="book-ref">
  <xs:complexType>
    <xs:attribute name="ISBN" type="xs:string"/>
  </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

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

```xml
<geo_db>
  <state>
    <name xs:string/>
    <capital_city_id xs:string/>
    <area xs:decimal/>
    <id xs:string/>
    <name xs:string/>
    <population xs:integer/>
    ...
  </state>
  <county>
    <name xs:string/>
    <capital_city_id xs:string/>
    <id xs:string/>
    <name xs:string/>
    <area xs:decimal/>
    <population xs:integer/>
    ...
  </county>
  <city>
    <name xs:string/>
    <area xs:decimal/>
    <id xs:string/>
    <name xs:string/>
    <population xs:integer/>
    ...
  </city>
  ...
</geo_db>
```

Declare stateKey in geo_db with
- Selector ./state
- Field name

Declare countyInStateKey in state with
- Selector ./county
- Field name

Declare cityInCountyKey in county with
- Selector ./city
- Field name

Declare capitalCityIdKeyRef in geo_db referencing cityIdKey, with
- Selector ./state
- Field capital_city_id

Declare cityIdKey in geo_db with
- Selector ./state/count/city
- Field id
```