XML, DTD, and XML Schema

CompSci 316
Introduction to Database Systems

Announcements (Thu. Oct. 17)
- Project milestone #1 due today!
- Midterm being graded; sample solution available this weekend

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

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 &gt; 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>

From HTML to XML (eXtensible Markup Language)
- HTML describes presentation of content
  - Extensible Markup Language (eXtensible Markup Language)
  - XML describes only the content
  - Separation of content from presentation simplifies content extraction and allows the same content to be presented easily in different looks

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>...
  - Elements can be nested: <book><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

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

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

More XML features

- Processing instructions for apps: `<? ... ?>`
- An XML file typically starts with a version declaration using this syntax: `<?xml version="1.0"?>`
- Comments: `<!-- ... -->`
- CDATA section: `<![CDATA[<Tags: <book>...]>]]`
- ID's and references
  ```xml
  <person id="632">name="name" /> <person id="925" father="632" mother="426" name="name" /> 
  ```
- Namespaces allow external schemas and qualified names
  ```xml
  </book>
  ```
- And more...

DTD explained

```xml
<!DOCTYPE bibliography [       
  <!ELEMENT bibliography (book+)>     
  <!ELEMENT book (title, author*, publisher?, year?, section*)>     
  <!ELEMENT year (#PCDATA)>       
  <!ELEMENT publisher (#PCDATA)>  
  <!ELEMENT title (#PCDATA)>      
  <!ELEMENT section (title, content, section*)>        
  <!ELEMENT content (#PCDATA|i)*>          
  <!ELEMENT i (#PCDATA)>]>
```

Using DTD

- DTD can be included in the XML source file
  ```xml
  <?xml version="1.0"?>
  <!DOCTYPE bibl [ ... ]>
  </bibl>
  ```
- DTD can be external
  ```xml
  <?xml version="1.0"?>
  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 1.0 Strict//EN"
  "http://www.w3.org/TR/html401/strict.dtd">
  </html>
  ```
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: 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:sequence/>
  ...  
</xs:schema>
```

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

XSD example

```xml
<xs:element name="book">
  <xs:complexType>
    <xs:sequence/>
  </xs:complexType>
</xs:element>
```
XSD example cont’d

```xml
<xs:element name="section">
  <xs:complexType>
    <xs:complexContent>
      <xs:restriction base="xs:string"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
```

Another title definition, can be different:

```xml
<xs:element name="title" type="xs:string"/>
```

Options for element content:

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

Recursive definition:

```xml
(…|…|…)
```

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="content" type="formattedTextType"/>
  ```

Restrictions

- To complete bib.xsd:

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

- To use bib.xsd in an XML document:

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

Foreign keys

- Suppose content can reference books:

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

  ```xml
  <xs:element name="book-ref">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="book" type="bookType"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  ```

  Make sure keyref is declared in the same scope as the key it refers to.

Keys

- 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):

  ```xml
  <xs:element name="bibliography">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="book" type="bookType"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  ```

  In general, a key can consist of multiple fields (multiple `<xs:field>` elements under `<xs:keys>`):

  ```xml
  <xs:element name="bibliography">
    <xs:complexType>
      <xs:keys>
        <xs:field name="ISBN"/>
      </xs:keys>
    </xs:complexType>
  </xs:element>
  ```

  More on XPath in next lecture.
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 important
  - Data 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 (country 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 country is always located in a single state

A possible design

```xml
<geo_db>
  <state>
    <county>
      <city>
        <id xs:string/>
        <name xs:string/>
        <population xs:integer/>
      </city>
    </county>
    <countyInStateKeyRef/>
    <geo_db>
      declare capitalCityIdKey in geo_db referencing cityIdKey;
      declare countyInStateKey in state referencing countyInStateKeyRef;
    </geo_db>
  </county>
  <county>
    <city>
      <id xs:string/>
      <name xs:string/>
      <area xs:decimal/>
      <population xs:integer/>
      <geo_db>
        declare stateIdKey in state referencing stateIdKey;
        declare countyInStateKey in county referencing countyInStateKeyRef;
        declare capitalCityIdKey in city referencing capitalCityIdKeyRef;
      </geo_db>
    </city>
  </county>
</geo_db>
```