XML, DTD, and XML Schema

CPS 116
Introduction to Database Systems

Announcements (Thu. Oct. 13)
- Project milestone #1 due today!
- Midterm graded; sample solution available
  - Highest: 49.5/50
  - Average: 40.0/50
  - Median: 43.0/50
- Check your grades on Blackboard
- Graded Homework #2 available soon

Announcements (Tue. Oct. 18)
- Project milestone #1 feedback available by this weekend via email
- Graded Homework #2 available this Thursday
- Homework #3 available this Thursday

From HTML to XML (eXtensible Markup Language)

- HTML describes presentation of content
- 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

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>…</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 &lt; 0. </section>`
  - Right: `<section> We show that x &lt; 0. </section>`
- 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

- Comments: `<!-- Comments here -->`
- CDATA section: `<![CDATA[Tags: <book>,...]]>`
- ID's and references
  - `<book xmlns:myCitationStyle="http://…/mySchema”>`
- Namespaces allow external schemas and qualified names
  - `<myCitationStyle:author>`...
- Processing instructions for apps: `<java applet>`...
- And more...

DTD explained

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

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

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
- Contains constraints on structures and values of elements, attributes, etc.
- Example
  ```
  <!DOCTYPE bibliography [...
  <!ELEMENT bibliography (book)>
  <!ELEMENT book (title, author*, publisher?, year?, section*)>
  ...
]>
```
Using DTD

- DTD can be included in the XML source file
  ```xml
  <?xml version="1.0"?><bibliography>
  … …
  </bibliography>
  ```

- DTD can be external
  ```xml
  <?xml version="1.0"?><bibliography SYSTEM "../dtds/bib.dtd">
  … …
  </bibliography>
  ```

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|[b|i]|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 '([b|i]|math)*'>`
    - `<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: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>
```

19 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

Like `author*` in DTD

Like `publisher?` in DTD

A leaf element with integer content

Reference to element `section` defined elsewhere

Declares an attribute under `book`… and this attribute is required

This 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>
```

20 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 "(…|…|…)" in DTD

min/maxOccurs can be attached to compositors too

Like `(i|b)*` in DTD

Recursive definition

Named types

 Define once:

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

22 Restrictions

```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>
```

23 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)

- In general, a key can consist of multiple fields (multiple `<xs:field>` elements under `<xs:key>`)
- More on XPath in next lecture

24 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>
```

21 XSD example cont’d

- To complete bib.xsd:

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

- To use bib.xsd in an XML document:

```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"/>
```

25 © 2001. W3C (MIT, INRIA, Keio), All Rights Reserved.

Restrictions

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

26 Keys

```xml
<xs:complexType 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:complexType>
```

© 2001. W3C (MIT, INRIA, Keio), All Rights Reserved.
Foreign keys

- Suppose content can reference books

```xml
<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>
```

- 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

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
- 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 (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