XML, DTD, and XML Schema

Introduction to Databases
CompSci 316 Fall 2016

Announcements (Thu. Oct. 20)

• Homework #3 assigned today; due in ~2 weeks
• Midterm graded
  • Mean: $77.4 + 1.4$ extra credit = 78.8
  • I will make a 4-point adjustment when calculating final grade
  • Median: 80; highest: 106
  • Sample solution has been posted
  • Two problems where folks had most trouble:
    • FD implication
    • Exactly 2 in RA
• Project milestone #1 feedback to be emailed this weekend

Structured vs. unstructured data

• Relational databases are highly structured
  • All data resides in tables
  • You must define schema before entering any data
  • Every row confirms to the table schema
  • Changing the schema is hard and may break many things
• Texts are highly unstructured
  • Data is free-form
  • There is no pre-defined schema, and it's hard to define any schema
  • Readers need to infer structures and meanings

What’s in between these two extremes?
Semi-structured data

• Observation: most data have some structure, e.g.:
  • Book: chapters, sections, titles, paragraphs, references, index, etc.
  • Item for sale: name, picture, price (range), ratings, promotions, etc.
  • Web page: HTML
• Ideas:
  • Ensure data is “well-formatted”
  • If needed, ensure data is also “well-structured”
    • But make it easy to define and extend this structure
  • Make data “self-describing”

HTML: language of the Web

• It’s mostly a “formatting” language
• It mixes presentation and content
  • Hard to change presentation (say, for different displays)
  • Hard to extract content
XML: eXtensible Markup Language

- Text-based
- Capture data (content), not presentation
- Data self-describes its structure
  - Names and nesting of tags have meanings!

Other nice features of XML

- **Portability**: Just like HTML, you can ship XML data across platforms
  - Relational data requires heavy-weight API's
- **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/>
    - 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
  • Right: `<section>We show that x < 0…</section>`
  • Wrong: `<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>…</section>…</subsection>`

A tree representation

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

• Namespaces allow external schemas and qualified names

• And more...
Now for some more structure...

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 section (title, content?, section*)>
]>```

DTD explained

- `<bible>`, `<book>`, `<title>`, `<author>`, `<publisher>`, `<year>`, `<section>` elements
- `<ISBN>` and `<price>` attributes
- Other attribute types include `IDREF` (reference to an ID), `IDREFS` (space-separated list of references), enumerated list, etc.

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

https://commons.wikimedia.org/wiki/File:Hundertwasser_04.jpg
DTD explained (cont’d)

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

- `PCDATA` is text that will be parsed
- `&lt;` etc. will be parsed as entities
- Use a CDATA section to include text verbatim
- Each section begins with a title, followed by an optional content, and then zero or more (sub)section's
- `content` 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)section's

```xml
<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>
```

Using DTD

- DTD can be included in the XML source file
- DTD can be external
- DTD can be included in the XML source file
- DTD can be external

```xml
<?xml version="1.0"?>
<!DOCTYPE bibliography [ … ]><bibliography>…</bibliography>
```

Annoyance: content grammar

- Consider this declaration:
- “|” 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)*)*, section*)>`
Annoyance: 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 we just infer a title’s context?

Annoyance: 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, even if they have different types (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 '(...)'>
    - <!ELEMENT E1 % E.struct;>
    - <!ELEMENT E2 % E.struct;>
  - Something less “hacky”?

Now for even more structure support...
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">
  <!-- Uses of xs: within the xs:schema element now refer to tags from this namespace -->
  <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>
</xs:schema>
```

XSD example

```xml
<element name="book">... We are now defining an element named book ...
<complexType>... Declares a structure with child elements/attributes as opposed to just text ...
<element name="title" type="xs:string">... A leaf element with string content ...
<element name="author" type="xs:string">... Like author* in DTD ...
<element name="publisher" type="xs:string">... Like publisher? in DTD ...
<element name="year" type="xs:integer">... A leaf element with integer content ...
<element ref="section" minOccurs="0" maxOccurs="unbounded">... Like section* in DTD; section is defined elsewhere ...
</sequence>
<attribute name="ISBN" type="xs:string">... Declares an attribute under book... and this attribute is required ...
<attribute name="price" type="xs:decimal">... This attribute has a decimal value; and it is optional ...
</complexType>
</element>
```
XSD example cont’d

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

XSD example cont’d

• To complete bib.xsd:
  <xs:element name="bibliography">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="book" minOccurs="1" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

• To use bib.xsd in an XML document:
  <?xml version="1.0"?>
  <bibliography xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <book>…</book>
    <book>…</book>
    …
  </bibliography>

Named types

• Define once:
  <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:
  …
  <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>

<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:selector xpath="/book"/>
    <xs:field xpath="@ISBN"/>
  </xs:complexType>
</xs:element>
```

- Under any bibliography, 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="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" type="xs:string">
        <xs:attribute name="ISBN" type="xs:string"/>
      </xs:element>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

- Suppose content can reference books
- Under bibliography, for elements reachable by selector "./book-ref" (i.e., any book-ref underneath):
  - values of field "@ISBN" (i.e., ISBN attributes) must appear as values of bookKey, the key referenced
    - Make sure keyref is declared in the same scope
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>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
    Field @name
Declare countyInStateKey in state with
    Field @name
Declare cityInCountyKey in county with
    Field @name
Declare capitalCityIdKeyRef in geo_db referencing cityIdKey, with
    Field @capital_city_id
```

```
geo_db
    name xs:string capital_city_id xs:string
    state...
    name xs:string area xs:decimal
    id xs:string name xs:string
    population xs:integer
city...
```

```
state
    county...
```

```
county
    city...
```

```
city
    name xs:string
    area xs:decimal
    id xs:string name xs:string
    population xs:integer
city...
```

```
geo_db
    name xs:string capital_city_id xs:string
    state...
    name xs:string area xs:decimal
    id xs:string name xs:string
    population xs:integer
city...
```

```
state
    county...
```

```
county
    city...
```

```
city
    name xs:string
    area xs:decimal
    id xs:string name xs:string
    population xs:integer
city...
```

```
geo_db
    name xs:string capital_city_id xs:string
    state...
    name xs:string area xs:decimal
    id xs:string name xs:string
    population xs:integer
city...
```

```
state
    county...
```

```
county
    city...
```

```
city
    name xs:string
    area xs:decimal
    id xs:string name xs:string
    population xs:integer
city...
```