XML, DTD, and XML Schema

Introduction to Databases
CompSci 316 Spring 2017

So far

- Relational Data model
- Database design (E/R diagram, normalization)
- SQL
- Database internals (physical organization, index, query processing, query optimization)
- Transaction

- All mostly focused on “structured data”

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:
  1. Ensure data is “well-formatted”
  2. If needed, ensure data is also “well-structured”
     - But make it easy to define and extend this structure
  3. 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
  - Hard for program to read it (italics and bold does not matter to a program)
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
- **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/></is_textbook>
  - Can be abbreviated: <is_textbook/>
- Elements can also have 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:
    - &gt; becomes 
    - & becomes 

- Contains a single root element
- Has properly matched tags and properly nested elements
  - Right: <section>...<subsection>...</subsection>...</section>
  - Wrong: <section>...</subsection>...</subsection>

A tree representation

- **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**
  - <?xml id="x12"?> = "name: Homer" = "name: ...">
  - <?xml id="x12"?> = "name: Marge" = "name: ...">
  - <?xml id="x12"?> = "father: " = "father: "

- **Namespaces** allow external schemas and qualified names
  - <my:myelement:author:... xmlns:myelement="...

- And more...
Now for some more structure...

**Using DTD**

- DTD can be included in the XML source file
  - `<doctype type="external">`
  - `<bibliography>`
- DTD can be external
  - `<xinclude>`
  - `<bibliography>`

**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
- Example
  ```xml
  <doctype type="external">
  <bibliography>
    <book>
      <title>Foundations of Databases</title>
      <author>Hull</author>
      <author>Vianu Abiteboul</author>
      <publisher>Addison Wesley</publisher>
      <year>1995</year>
      <section>
        <title>Introduction</title>
        <content>In this section we introduce…</content>
      </section>
    </book>
  </bibliography>
  </doctype>
  ```

**DTD explained**

- `<doctype type="external">` is the root element of the document
- `<bibliography>` One or more bibliography consists of a sequence of one or more `<book>` elements
- `<book>` Title, zero or more authors, publisher, zero or more sections, in sequence
- `<attlist>` book ISBN ID #REQUIRED
- `<attlist>` book price CDATA #IMPLIED
- `<element>` book (title, author*, publisher?, year?, section*)
- `<element>` bibliography (book+)

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

**DTD explained (cont’d)**

- `<doctype type="external">`
- `<element data-type="text">` is text that will be parsed
- `&lt;` etc. will be parsed as entities
- Use a DTD section to include text verbatim

**Annoyance: content grammar**

- Consider this declaration:
  ```xml
  <element pub-venue (name, address, month, year)>
  <venue>
    <name>Acme Street</name>
    <address>123 Main St</address>
  </venue>
  <year>2023</year>
  </element>
  ```
- `<attlist>` pub-venue ID #REQUIRED
- `<attlist>` pub-venue address CDATA #IMPLIED
- `<element>` pub-venue (name, address, month, year)
- `<element>` pub-venue (IDREF?, address)
- `<element>` pub-venue (IDREF?, address, month, year)
- `<element>` pub-venue (IDREF?, address, month, year, ID)
- Consider this declaration:
  ```xml
  <element pub-venue (name, address, month, year)>
  <venue>
    <name>Acme Street</name>
    <address>123 Main St</address>
  </venue>
  <year>2023</year>
  </element>
  ```
- `<attlist>` pub-venue ID #REQUIRED
- `<attlist>` pub-venue address CDATA #IMPLIED
- `<element>` pub-venue (name, address, month, year)
- `<element>` pub-venue (IDREF?, address)
- `<element>` pub-venue (IDREF?, address, month, year)
- `<element>` pub-venue (IDREF?, address, month, year, ID)
- Consider this declaration:
  ```xml
  <element pub-venue (name, address, month, year)>
  <venue>
    <name>Acme Street</name>
    <address>123 Main St</address>
  </venue>
  <year>2023</year>
  </element>
  ```
- `<attlist>` pub-venue ID #REQUIRED
- `<attlist>` pub-venue address CDATA #IMPLIED
- `<element>` pub-venue (name, address, month, year)
- `<element>` pub-venue (IDREF?, address)
- `<element>` pub-venue (IDREF?, address, month, year)
- `<element>` pub-venue (IDREF?, address, month, year, ID)
- Consider this declaration:
  ```xml
  <element pub-venue (name, address, month, year)>
  <venue>
    <name>Acme Street</name>
    <address>123 Main St</address>
  </venue>
  <year>2023</year>
  </element>
  ```
- `<attlist>` pub-venue ID #REQUIRED
- `<attlist>` pub-venue address CDATA #IMPLIED
- `<element>` pub-venue (name, address, month, year)
- `<element>` pub-venue (IDREF?, address)
- `<element>` pub-venue (IDREF?, address, month, year)
- `<element>` pub-venue (IDREF?, address, month, year, ID)
- Consider this declaration:
  ```xml
  <element pub-venue (name, address, month, year)>
  <venue>
    <name>Acme Street</name>
    <address>123 Main St</address>
  </venue>
  <year>2023</year>
  </element>
  ```
- `<attlist>` pub-venue ID #REQUIRED
- `<attlist>` pub-venue address CDATA #IMPLIED
- `<element>` pub-venue (name, address, month, year)
- `<element>` pub-venue (IDREF?, address)
- `<element>` pub-venue (IDREF?, address, month, year)
- `<element>` pub-venue (IDREF?, address, month, year, ID)
- Consider this declaration:
  ```xml
  <element pub-venue (name, address, month, year)>
  <venue>
    <name>Acme Street</name>
    <address>123 Main St</address>
  </venue>
  <year>2023</year>
  </element>
  ```
- `<attlist>` pub-venue ID #REQUIRED
- `<attlist>` pub-venue address CDATA #IMPLIED
- `<element>` pub-venue (name, address, month, year)
- `<element>` pub-venue (IDREF?, address)
- `<element>` pub-venue (IDREF?, address, month, year)
- `<element>` pub-venue (IDREF?, address, month, year, ID)
- Consider this declaration:
  ```xml
  <element pub-venue (name, address, month, year)>
  <venue>
    <name>Acme Street</name>
    <address>123 Main St</address>
  </venue>
  <year>2023</year>
  </element>
  ```
- `<attlist>` pub-venue ID #REQUIRED
- `<attlist>` pub-venue address CDATA #IMPLIED
- `<element>` pub-venue (name, address, month, year)
- `<element>` pub-venue (IDREF?, address)
- `<element>` pub-venue (IDREF?, address, month, year)
- `<element>` pub-venue (IDREF?, address, month, year, ID)
- Consider this declaration:
  ```xml
  <element pub-venue (name, address, month, year)>
  <venue>
    <name>Acme Street</name>
    <address>123 Main St</address>
  </venue>
  <year>2023</year>
  </element>
  ```
- `<attlist>` pub-venue ID #REQUIRED
- `<attlist>` pub-venue address CDATA #IMPLIED
- `<element>` pub-venue (name, address, month, year)
- `<element>` pub-venue (IDREF?, address)
- `<element>` pub-venue (IDREF?, address, month, year)
- `<element>` pub-venue (IDREF?, address, month, year, ID)
- Consider this declaration:
  ```xml
  <element pub-venue (name, address, month, year)>
  <venue>
    <name>Acme Street</name>
    <address>123 Main St</address>
  </venue>
  <year>2023</year>
  </element>
  ```
- `<attlist>` pub-venue ID #REQUIRED
- `<attlist>` pub-venue address CDATA #IMPLIED
- `<element>` pub-venue (name, address, month, year)
- `<element>` pub-venue (IDREF?, address)
- `<element>` pub-venue (IDREF?, address, month, year)
- `<element>` pub-venue (IDREF?, address, month, year, ID)
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|ib|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, blah, foo?, bar*, ...)
    • want to define E2 to have the same structure
  • Parameter entities in DTD provide a workaround
    • &ENTITY % E.struct [blah, blah, foo?, bar*, ...];
  • E struct (i.e., already defined element)
  • 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:element name="book">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="title" type="xs:string" maxOccurs="1" minOccurs="0"/>
        <xs:element name="year" type="xs:integer" maxOccurs="1" minOccurs="0"/>
        <xs:element name="author" type="xs:string" maxOccurs="unbounded" minOccurs="0"/>
        <xs:element name="price" type="xs:decimal" minOccurs="0" maxOccurs="1"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

XSD example

We are now defining an element named book.

- `book`: Defines `xs` to be the namespace described in the URL.
- `book-title`: A leaf element with string content.
- `year`: A leaf element with integer content.
- `author`: Like section in DTD, section is defined elsewhere.
- `price`: This attribute has a decimal value, and it is optional.
- `title`, `year`, `author`: Required.

```xml
<book>
  <title>...</title>
  <year>0</year>
  <author>...</author>
  <price>10.5</price>
</book>
```
Keys

```xml
<x:element name="bibliography">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="book" type="xs:complexType">
        <xs:attribute name="@ISBN" type="xs:string"/>
        <xs:complexType>
          <xs:sequence>
            <xs:element name="title" type="xs:string"/>
            <xs:element name="author" type="xs:string"/>
            <xs:element name="year" type="xs:integer"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</x: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

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
- Exchange is problematic
- “Native” support in all "Native" support in all 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

```xml
<geo_db>
  <state>
    <state_key/>
    <name>/naming
definitions/</state>
    <area>
      <area_key/>
      <naming/defininitions/
definitions/>
    </area>
    <population>...
  </state>
  <county>
    <county_key/>
    <state_key/>
    <name>/naming/defininitions/definitions/>
    <area>
      <area_key/>
      <naming/defininitions/definitions/>
    </area>
    <population>...
  </county>
  <city>
    <city_key/>
    <county_key/>
    <name>/naming/defininitions/definitions/>
    <population>...
  </city>
</geo_db>
```

- Declare `stateKey` in `geo_db` with
- Declare `cityIdKey` in `geo_db` referencing `cityIdKey` with
- Declare `capitalCityIdKeyRef` in `geo_db` with

Foreign keys

- Suppose content can reference books
    - 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
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:element/>
      </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"
    xsi:noNamespaceSchemaLocation="file:bib.xsd">
    <book>
      ...
    </book>
    ...
  </bibliography>

Restrictions

  <xs:simpleType name="priceType">
    <xs:restriction base="xs:decimal">
      <xs:totalDigits value="0"/>
    </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>

Named types

• Define once:
  <xs:complexType name="formattedTextType" mixed="true">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element name="b" type="xs:string"/>
      <xs:element name="i" type="xs:string"/>
      <xs:element name="b\i" type="xs:string"/>
    </xs:choice>
  </xs:complexType>

• Use elsewhere in XSD:
  ...
  ...