

# XML, DTD, and XML Schema

CompSci 316  
Introduction to Database Systems

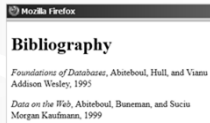
## Announcements (Thu. Oct. 17)

- ❖ Project milestone #1 due today!
- ❖ Midterm being graded; sample solution available this weekend

## From HTML to XML (eXtensible Markup Language)

- ❖ HTML describes presentation of content

```
<h1>Bibliography</h1>
<p><i>Foundations of Databases</i>,
Abiteboul, Hull, and Vianu
<br>Addison Wesley, 1995
</p>...
```



- ❖ XML describes only the content

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

- ☞ 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">

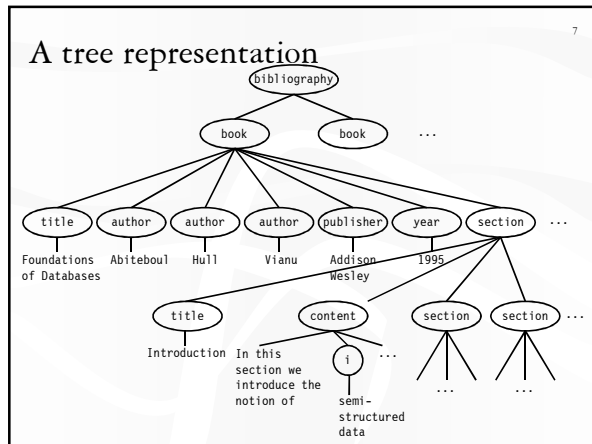
☞ Ordering generally matters, except for attributes

```
<bibliography>
<book ISBN="ISBN-10" price="80.00">
  <title>Foundations of Databases</title>
  <is_textbook/>
  <author>Abiteboul</author>
  <author>Hull</author>
  <author>Vianu</author>
  <publisher>Addison Wesley</publisher>
  <year>1995</year>
</book>
</bibliography>
```

## 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>...</section>...</subsection>



- ### 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
 

```
<person id="012"><name>Homer</name>.</person>
<person id="034"><name>Marge</name>.</person>
<person id="056" father="012" mother="034"><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
 

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

- ### DTD explained
- ```
<!DOCTYPE bibliography [
  bibliography is the root element of the document
  <!ELEMENT bibliography (book+)>
  bibliography consists of a sequence of one or more book elements
  <!ELEMENT book (title, author*, publisher?, year?, section*)>
  book consists of a title, zero or more authors,
  an optional publisher, and zero or more sections, in sequence
  <!ATTLIST book ISBN ID #REQUIRED>
  book has a required ISBN attribute which is a unique identifier
  <!ATTLIST book price CDATA #IMPLIED>
  book has an optional (#IMPLIED) price attribute which contains
  character data
  </!DOCTYPE>
```
- 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*)>
```
- PCDATA is text that will be parsed
    - `&lt;`; etc. will be parsed as entities
    - Use a CDATA section to include text verbatim
  - author, publisher, year, and i contain parsed character data
  - content contains mixed content: text optionally interspersed with i elements
  - Recursive declaration:
 

```
<section><title>Introduction</title>
<content>In this section we introduce
the notion of <i>semi-structured data</i>.
</content>
<section><title>XML</title>
<content>XML stands for...</content>
</section>
<section><title>DTD</title>
<section><title>Definitions</title>
<content>DTD stands for...</content>
</section>
<section><title>Usages</title>
<content>You can use DTD to...</content>
</section>
</section>
```

- ### Using DTD
- ❖ DTD can be included in the XML source file
 

```
<?xml version="1.0" ?>
<!DOCTYPE bibliography [
  ...
]>
<bibliography>
  ...
</bibliography>
```
  - ❖ DTD can be external
 

```
<?xml version="1.0" ?>
<!DOCTYPE bibliography SYSTEM "../dtds/bib.dtd">
<bibliography>
  ...
</bibliography>
```

## 13 Annoyances: element type declarations

- ❖ Consider this element content (children) declaration:  

```
<!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*)>`

## 14 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|i|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?

## 15 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”?

## 16 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

## 17 XML Schema definition (XSD)

```
<?xml version="1.0"?>  
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">  
  ...  
</xs:schema>
```

↳ Defines `xs` to be the namespace described in the URL

Uses of `xs:` within the `xs:schema` element now refer to tags from this namespace

## 18 XSD example

```
<xs:element name="book">  
  <xs:complexType> Declares a structure with child elements/attributes as opposed to just text  
  <xs:sequence> Declares a sequence of child elements, like "(, -, ..)" in DTD  
  <xs:element name="title" type="xs:string"/> A leaf element with string content  
  <xs:element name="author" type="xs:string" minOccurs="0" maxOccurs="unbounded"/> Like author* in DTD  
  <xs:element name="publisher" type="xs:string" minOccurs="0" maxOccurs="1"/> Like publisher? in DTD  
  <xs:element name="year" type="xs:integer" minOccurs="0" maxOccurs="1"/> A leaf element with integer content  
  <xs:element ref="section" minOccurs="0" maxOccurs="1"/> Reference to element section defined elsewhere  
  </xs:sequence>  
  <xs:attribute name="ISBN" type="xs:string" use="required"/>  
    Declares an attribute under book... and this attribute is required  
  <xs:attribute name="price" type="xs:decimal" use="optional"/>  
    This attribute has a decimal value, and it is optional  
</xs:complexType>  
</xs:element>
```

## XSD example cont'd

19

```

<xs:element name="section">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="title" type="xs:string"/> from book/title
      <xs:element name="content" minOccurs="0" maxOccurs="1">
        <xs:complexType mixed="true">
          <xs:choice minOccurs="0" maxOccurs="unbounded"> min/maxOccurs can be
            (text interspersed with structure below)
            <xs:element name="i" type="xs:string"/> attached to compositors too
            <xs:element name="b" type="xs:string"/>
          </xs:choice>
        </xs:complexType> Like (#PCDATA|i|b)* in DTD
      </xs:sequence> Recursive definition
    </xs:complexType>
  </xs:element>
</xs:element>

```

A compositor like `<xs:sequence>` this one declares a list of alternatives, like `"(..|...|..)"` in DTD

## XSD example cont'd

20

❖ To complete `bib.xsd`:

```

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

```

## Named types

21

### ❖ 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

22

```

<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

23

```

<xs:element name="bibliography">
  <xs:complexType>... </xs:complexType>
  <xs:key name="bookKey">
    <xs:selector xpath="./book"/>
    <xs:field xpath="@ISBN"/>
  </xs:key>
</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

24

### ❖ Suppose content can reference books

```

<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:choice>
    </xs:complexType>
  </xs:element>
</xs:element>

```

```

<xs:element name="bibliography">
  <xs:complexType>... </xs:complexType>
  <xs:key name="bookKey">
    <xs:selector xpath="./book"/>
    <xs:field xpath="@ISBN"/>
  </xs:key>
  <xs:keyref name="bookForeignKey"
    refer="bookKey">
    <xs:selector xpath="./book-ref"/>
    <xs:field xpath="@ISBN"/>
  </xs:keyref>
</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?

25

- ❖ 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

26

- | Relational data   | XML data  |
|---|---|
| ❖ Schema is always fixed in advance and difficult to change | ❖ Well-formed XML does not require predefined, fixed schema       |
| ❖ Simple, flat table structures                             | ❖ Nested structure; ID/IDREF(S) permit arbitrary graphs           |
| ❖ Ordering of rows and columns is unimportant               | ❖ Ordering forced by document format; may or may not be important |
| ❖ Data exchange is problematic                              | ❖ Designed for easy exchange                                      |
| ❖ "Native" support in all serious commercial DBMS           | ❖ Often implemented as an "add-on" on top of relations            |

## Case study

27

- ❖ 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

28

