XML, DTD, and XML Schema

Introduction to Databases

CompSci 316 Spring 2017

DUKE
COMPUTER SCIENCE

So far

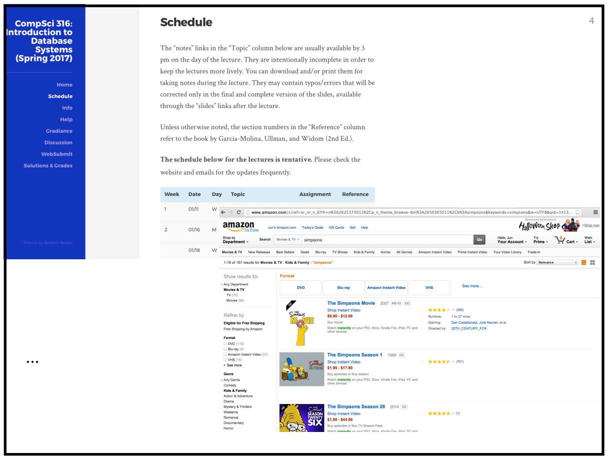
- Relational Data model
- Database design (E/R diagram, normalization)
- SQL
- Database internals (physical organization, index, query processing, query optimization)
- Transaction

- 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?



CompSci 316: Introduction to Database Systems (Spring 2017)

Schedule

The "notes" links in the "Topic" column below are usually available by 3 pm on the day of the lecture. They are intentionally incomplete in order to keep the lectures more lively. You can download and/or print them for taking notes during the lecture. They may contain typos/errors that will be corrected only in the final and complete version of the slides, available through the "slides" links after the lecture.

Unless otherwise noted, the section numbers in the "Reference" column refer to the book by Garcia-Molina, Ullman, and Widom (2nd Ed.).

The schedule below for the lectures is tentative. Please check the website and emails for the updates frequently.

Week	Date	Day	Topic	Assignment	Reference
1	01/11	W			
2	01/16	M			
	01/18	W			

www.amazon.com/s/ref=sr_nr_n_6?rh=n%3A2625373011%2Cp_n_theme_browse-bin%3A

1-16 of 167 results for Movies & TV : Kids & Family : "simpsons"

Show results for
Any Department
Movies & TV
TV (70)
Movies (94)

Refine by
Eligible for Free Shipping
Free Shipping by Amazon
Format
DVD (110)
Blu-ray (4)
Amazon Instant Video (37)
VHS (15)
See more

The Simpsons Movie 2007 PG-13 CC
Shop Instant Video
$9.99 - $12.99
Buy movie
Watch instantly on your PS3, Xbox, Kindle Fire, iPad, PC and other devices.

The Simpsons Season 1 1989 CC
Shop Instant Video

The Simpsons Season 26 2014
Shop Instant Video

…

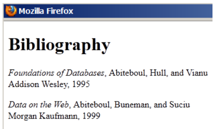
Semi-structured data

- 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

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



Bibliography

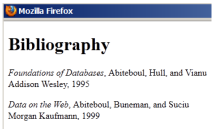
Foundations of Databases, Abiteboul, Hull, and Vianu
Addison Wesley, 1995

Data on the Web, Abiteboul, Buneman, and Suciu
Morgan Kaufmann, 1999

- 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

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



Bibliography

Foundations of Databases, Abiteboul, Hull, and Vianu
Addison Wesley, 1995

Data on the Web, Abiteboul, Buneman, and Suciu
Morgan Kaufmann, 1999

- 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

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

- 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">

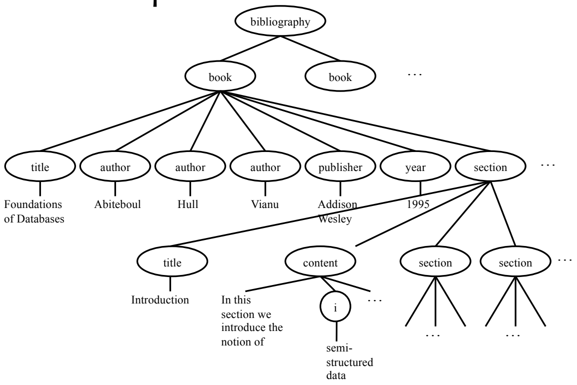
☞Ordering generally matters (can specify), except for attributes

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: > 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>

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

```
<person id="o12"><name>Homer</name>…</person>
<person id="o34"><name>Marge</name>…</person>
<person id="o56" father="o12" mother="o34">
  <name>Bart</name>…
</person>…
```

- Namespaces allow external schemas and qualified names

```
<myCitationStyle:book xmlns:myCitationStyle="http://…/mySchema">
    <myCitationStyle:title>…</myCitationStyle:title>
    <myCitationStyle:author>…</myCitationStyle:author>…
</book>
```

- And more…

Now for some more structure…

https://commons.wikimedia.org/wiki/File:Hundertwasser_04.jpg

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 ID #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+)>
```

One or more

→ bibliography consists of a sequence of one or more book elements

```
 <!ELEMENT book (title, author*, publisher?, year?, section*)>
```

Zero or more

Zero or one

→ book consists of a title, zero or more authors, an optional publisher, and zero or more section's, 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

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

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

DTD explained (cont'd)

```
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT i (#PCDATA)>
```

→ author, publisher, year, and i contain parsed character data

```
<!ELEMENT content (#PCDATA|i)*>
```

→ content contains mixed content: text optionally interspersed with i elements

```
<!ELEMENT section (title, content?, section*)>
```

→ Recursive declaration:
Each section begins with a title, followed by an optional content, and then zero or more (sub) section's

```
]>
```

PCDATA is text that will be parsed

- &lt; etc. will be parsed as entities
- Use a CDATA section to include text verbatim

```
<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>Definition</title>
      <content>DTD stands for…</content>
    </section>
    <section><title>Usage</title>
      <content>You can use DTD to…</content>
    </section>
  </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>
    ```

  -
    ```
    <?xml version="1.0"?>
    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
              "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
    <html>
    … …
    </html>
    ```

Annoyance: content grammar

- Consider this declaration:

```
<!ELEMENT pub-venue
  ( (name, address, month, year) |
    (name, volume, number, year) )>
```

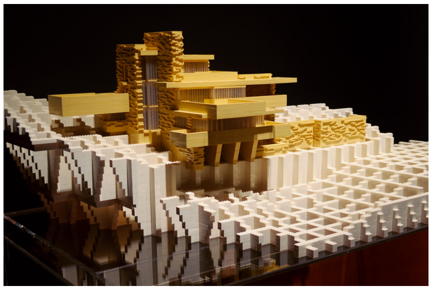
  - "|" means "or"
- Syntactically legal, but won't work
  - Because of SGML (standard generalized markup language) compatibility issues
  - When looking at name, a parser would not know which way to go without looking further ahead
  - Requirement: content declaration must be "deterministic" (i.e., no 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*)>

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 '(blah, bleh, foo?, bar*, …)'>
    - <!ELEMENT E1 %E.struct;>
    - <!ELEMENT E2 %E.struct;>
  - Something less "hacky"?



Now for even
more structure support…

http://thenewsherald.com/content/articles/2012/11/10/entertainment/doc509deb9f9207c2452179065.jpg

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 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

XSD example

`<xs:element name="book">`	We are now defining an element named 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="unbounded"/>`	Like section* in DTD; section is 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>`	

Keys

```
<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, 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

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

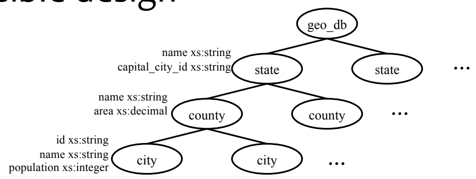
XML versus relational data

Relational data	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
• 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

- 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
  Selector ./state
  Field @name

Declare countyInStateKey in state with
  Selector ./county
  Field @name

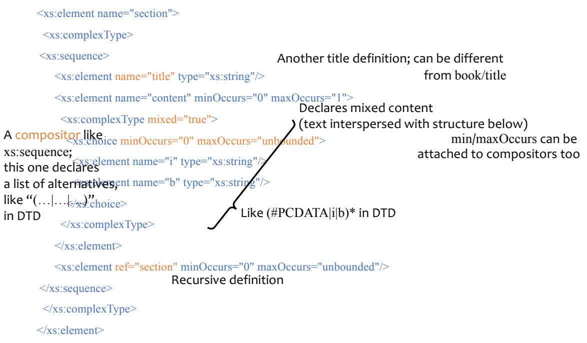
Declare cityInCountyKey in county with
  Selector ./city
  Field @name

Declare cityIdKey in geo_db with
  Selector ./state/county/city
  Field @id

Declare capitalCityIdKeyRef in geo_db referencing cityIdKey, with
  Selector ./state
  Field @capital_city_id

Additional slides

XSD example cont’d



```
<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>
```

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 (#PCDATA|i|b)* 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"
              xsi:noNamespaceSchemaLocation="file:bib.xsd">
  <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

```
<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>
```