XML, DTD, and XPath

CPS 116
Introduction to Database Systems

Announcements (October 17)

- Project milestone #1 feedback will be ready by Thursday
- Homework #3 will be assigned Thursday

From HTML to XML (eXtensible Markup Language)

- HTML describes presentation of content
  
  <html>
  <body>
  <h1>Bibliography</h1>
  <p><i>Foundations of Databases</i>
  Abiteboul, Hull, and Vianu
  <br>Addison Wesley, 1995
  </p>
  </body>
  </html>

- 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:
  - Empty elements: <is_textbook/>
    - Can be abbreviated: <is_textbook/>
  - Elements can also have attributes: <book ISBN="" price="80.00"/>

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 tags that are properly matched and elements that are properly nested
  - Right: <section>...</subsection>...</subsection>...</section>
  - Wrong:
    - <section>...</subsection>...</subsection>...</section>
More XML features

- Comments: <!-- Comments here -->
- CDATA: <![CDATA[Tags: <book>,...]]>
- ID's and references
  
```xml
<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
  
```xml
<book xmlns:myCitationStyle="http://my/schema">
  <myCitationStyle:title>…</myCitationStyle:title>
  <myCitationStyle:author>…</myCitationStyle:author>…
</book>
```

- Processing instructions for apps: <? ... java applet ... ?>
- 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

```xml
<!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 section (title, (#PCDATA)?, section*)>
]>
```

DTD explained

```xml
<!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 section (title, (#PCDATA)?, section*)>
]>
```

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

```xml
<ELEMENT title (#PCDATA)>
<ELEMENT author (#PCDATA)>
<ELEMENT publisher (#PCDATA)>
<ELEMENT year (#PCDATA)>
  title, author, publisher, and year all contain parsed character data (#PCDATA)
<ELEMENT section (title, (#PCDATA)?, section*)>
  Each section starts with a title, followed by some optional text and then zero or more subsections
</ELEMENT>
```

“Deterministic” content declaration

- Catch: the following declaration does not work:
  ```xml
  <!ELEMENT pub-venue 
    ( (name, address, month, year) | 
    (name, volume, number, year) )>
  ```
  Because when looking at `name`, the XML processor 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 the above declaration into an equivalent, but deterministic one?

Using DTD

- DTD can be included in the XML source file
  ```xml
  <?xml version="1.0"?>
  <!DOCTYPE bibliography [
    "bibliography"
  ]>
  ```
- DTD can be external
  ```xml
  <?xml version="1.0"?>
  <!DOCTYPE bibliography SYSTEM "../dtds/bib.dtd">
  ```
  ```xml
  <?xml version="1.0"?>
  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
  ```
Why use DTD’s?

- Benefits of not using DTD
  - Unstructured data is easy to represent
  - Overhead of DTD validation is avoided
- Benefits of using DTD

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>Data 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>

Query languages for XML

- XPath
  - Path expressions with conditions
  - Building block of other standards (XQuery, XSLT, XLink, XPointer, etc.)
- XQuery
  - XPath + full-fledged SQL-like query language
- XSLT
  - XPath + transformation templates
Example DTD and XML

```xml
<?xml version='1.0'?>
<!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, (section)?, section*)>
]>
<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>
    <section>…</section>…
  </book>
  …
</bibliography>
```

A tree representation

```
```

XPath

- XPath specifies path expressions that match XML data by navigating down (and occasionally up and across) the tree
- Example
  - Query: /bibliography/book/author
  - Like a UNIX path
  - Result: all author elements reachable from root via the path /bibliography/book/author
**Basic XPath constructs**

/ separator between steps in a path

name matches any child element with this tag name

* matches any child element

@name matches the attribute with this name

@* matches any attribute

// matches any descendent element or the current element itself

. matches the current element

.. matches the parent element

---

**Simple XPath examples**

- All book titles
  
  /bibliography/book/title

- All book ISBN numbers
  

- All title elements, anywhere in the document
  
  //title

- All section titles, anywhere in the document
  
  //section/title

- Authors of bibliographical entries (suppose there are articles, reports, etc. in addition to books)
  
  /bibliography/*/author

---

**Predicates in path expressions**

[condition] matches the current element if condition evaluates to true on the current element

- Books with price lower than $50
  
  /bibliography/book[@price<50]
  
  XPath will automatically convert the price string to a numeric value for comparison

- Books with author "Abiteboul"
  
  /bibliography/book[author='Abiteboul']

- Books with a publisher child element
  
  /bibliography/book/publisher

- Prices of books authored by "Abiteboul"
  
  /bibliography/book[author='Abiteboul']/@price
More complex predicates

Predicates can have and's and or's

- Books with price between $40 and $50
  \(/\text{bibliography/book}[40<@\text{price} \land @\text{price}<50]\)
- Books authored by “Abiteboul” or those with price lower than $50
  \(/\text{bibliography/book}[\text{author}='\text{Abiteboul}' \lor @\text{price}<50]\)

Predicates involving node-sets

\(/\text{bibliography/book}[\text{author}=\text{'Abiteboul'}]\)

- There may be multiple authors, so author in general returns a node-set (in XPath terminology)
- The predicate evaluates to true as long as it evaluates true for at least one node in the node-set, i.e., at least one author is “Abiteboul”
- Tricky query
  \(/\text{bibliography/book}[\text{author}=\text{'Abiteboul'} \land \text{author}!=\text{'Abiteboul'}]\)
  • Will it return any books?

XPath operators and functions

Frequently used in conditions:
- \(x + y, x - y, x \times y, x \div y, x \mod y\)
- \(\text{contains}(x, y)\) true if string \(x\) contains string \(y\)
- \(\text{count}(\text{node-set})\) counts the number nodes in \(\text{node-set}\)
- \(\text{position}()\) returns the “context position” (roughly, the position of the current node in the node-set containing it)
- \(\text{last}()\) returns the “context size” (roughly, the size of the node-set containing the current node)
- \(\text{name}()\) returns the tag name of the current element
More XPath examples

- All elements whose tag names contain "section" (e.g., "subsection")
  
  `/*[contains(name(), 'section')]`

- Title of the first section in each book
  
  `/bibliography/book/section[position()=1]/title`

- Title of the last section in each book
  
  `/bibliography/book/section[position()=last()]/title`

- Books with fewer than 10 sections
  
  `/bibliography/book[count(section)<10]`

- All elements whose parent's tag name is not "book"
  
  `/*[name()!='book']/`

A tricky example

- Suppose that `price` is a child element of `book`, and there may be multiple prices per book

- Books with some price in range [20, 50]
  
  - How about:
    
    `/bibliography/book
    [price >= 20 and price <= 50]`

De-referencing IDREF’s

- `id(identifier)` returns the element with the unique `identifier`

- Suppose that books can make references to other books

  `<section><title>Introduction</title>
   XML is a hot topic these days; see <bookref ISBN="ISBN-10"/> for more details...<section>`

- Find all references to books written by "Abiteboul" in the book with "ISBN-10"

General XPath location steps

- Technically, each XPath query consists of a series of location steps separated by /.

- Each location step consists of:
  - An axis: one of self, attribute, parent, child, ancestor, ancestor-or-self, descendant, descendant-or-self, following, following-sibling, preceding, preceding-sibling, and namespace.
  - A node test: either a name test (e.g., book, section, *) or a type test (e.g., text(), node(), comment()), separated from the axis by ::
  - Zero or more predicates (or conditions) enclosed in square brackets.

Example of verbose syntax

Verbose (axis, node test, predicate):
/child::bibliography
/descendant-or-self::node()
/child::title

Abbreviated:
- child is the default axis
- // stands for /descendant-or-self::node()/

One more example

- Which of the following queries correctly find the third author in the entire input document?
  - //author[position()=3]
  - /descendant-or-self::node()
    [name()=author and position()=3]
  - /descendant-or-self::node()
    [name()=author]
    [position()=3]