Announcements (October 18)

- Feedback on project milestone 1 by this weekend
- Homework #3 will be assigned next Tuesday

From HTML to XML (eXtensible Markup Language)

- HTML describes presentation of content

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

- XML describes only the content

```xml
<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/> and can be abbreviated: <is_textbook/>
- Elements can also have attributes: <book ISBN="80.00" 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 &lt; 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>
A valid XML document

- 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, (#PCDATA)?, section*)>
  ]>
  <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>
  
    <title>Advanced Databases</title>
    <author>Smith</author>
    <author>Doe</author>
    <publisher>Cambridge</publisher>
    <year>2000</year>
  </book>
  ```

DTD explained (cont’d)

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

PCDATA is text that will be parsed (<...> will be treated as markup tags and &lt;etc. will be treated as entities)

CO DATA is unparsed character data

- Each section starts with a title, followed by some optional text and then zero or more subsections

“Deterministic” content declaration

- Catch: the following declaration may 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> ...
  </bibliography>
  ```

- DTD can be external
  ```xml
  <?xml version="1.0"?>
  <!DOCTYPE bibliography SYSTEM "../dtds/bib.dtd">  
  <bibliography> ...
  </bibliography>
  ```

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
  - DTD can serve as a schema for the XML data
    - Guards against errors
    - Helps with processing
  - DTD facilitates information exchange
    - People can agree to use a common DTD 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
- Data exchange is problematic
- “Native” support in all serious commercial DBMS

XML data

- Well-formed XML does not require predefined, fixed schema
- Nested structure; ID/IDREF(S)
- Ordering forced by document format; may or may not be important
- Data exchange is easy
- Designed for easy exchange
- Often implemented as an “add-on” on top of relations

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*)*)>  
]>  
</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>
```

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

XPath

- XPath specifies path expressions that match XML data by navigating down (and occasionally up and across) the tree
- Example
  ```xml
  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
//@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
  /bibliography/book/@ISBN
- 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
  /bibliography/book[40<=@price and @price<=50]
- Books authored by “Abiteboul” or those with price lower than $50
  /bibliography/book[@author='Abiteboul' or @price<50]

Predicates involving node-sets

/bibliography/book[@author='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
  /bibliography/book[@author='Abiteboul' and author!='Abiteboul']
  - Will it return any books?

XPath operators and functions

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

- All elements whose tag names contain "section" (e.g., "subsection")
  ```xml
 //*[contains(name(), 'section')]
  ```
- Title of the first section in each book
  ```xml
  /bibliography/book/section[position()=1]/title
  ```
- Title of the last section in each book
  ```xml
  /bibliography/book/section[position()=last()]/title
  ```
- Books with fewer than 10 sections
  ```xml
  /bibliography/book[count(section)<10]
  ```
- All elements whose parent’s tag name is not “book”
  ```xml
  /[name()!='book']/*
  ```

A tricky example

- Suppose that price is a child element of book, and there may be multiple prices per book
  ```xml
  How about:
  /bibliography/book
  [price >= 20 and price <= 50]
  ```
- Correct answer:
  ```xml
  /bibliography/book
  [price[. >= 20 and . <= 50]]
  ```

De-referencing IDREF’s

- `id(identifier)` returns the element with identifier
- Suppose that books can reference other books
  ```xml
  <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”
  ```xml
  //bookref[@ISBN/author='Abiteboul']
  ```
  Or simply:
  ```xml
  ```

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`, `descendent`, `descendent-or-self`, `following`, `following-sibling`, `preceding`, `preceding-sibling`, and `namespace`
  - A node-test: either a name test (e.g., `book`), a type test (e.g., `text()`), separated from the axis by `::`
  - Zero or more predicates (or conditions) enclosed in square brackets

Example of verbose syntax

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

Abbreviated:
  ```xml
  ```

- child is the default axis
- `//` stands for `/descendent-or-self::node()`

One more example

- Which of the following queries correctly find the third author in the entire input document?
  ```xml
  //author[position()=3]
  ```
  - Finds all third authors (for each publication)
  ```xml
  /descendant-or-self::node()
  [name()='author' and position()=3]
  ```
  - Returns the third element in the document if it is an author
  ```xml
  /descendant-or-self::node()
  [name()='author']
  [position()=3]
  ```
  - Correct
  - After the first condition is passed, the evaluation context changes:
    - Context size: # of nodes that passed the first condition
    - Context position: position of the context node within the list of nodes
Some technical details on evaluation

Given a context node, evaluate a location path as follows:

1. Start with node-set $N$ containing the context node
2. For each location step, from left to right:
   - $U \leftarrow \emptyset$
   - For each node $n$ in $N$:
     - Using $n$ as the context node, compute a node-set $N'$ from the axis and the node-test
     - Each predicate in turn filters $N'$
       - For each node $n'$ in $N'$, evaluate predicate with the following context:
         - Context node is $n'$
         - Context size is the number of nodes in $N'$
         - Context position is the position of $n'$ within $N'$
       - $U \leftarrow U \cup N'$
     - $N \leftarrow U$
3. Return $N$