

XML, DTD, and XPath

CPS 116
Introduction to Database Systems

Announcements (October 18) 2

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

From HTML to XML (eXtensible Markup Language) 3

- ❖ 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>
      </book>_</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

4

- ❖ 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

5

- ❖ 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">

```
<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>
```

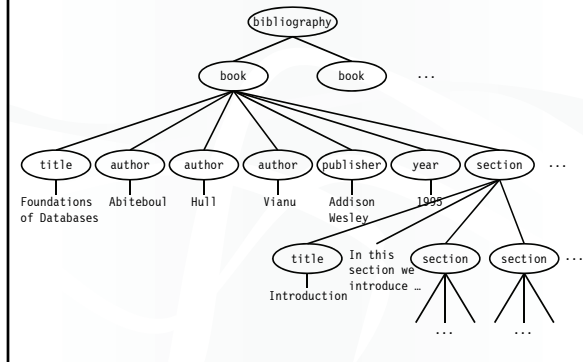
Well-formed XML documents

6

- A well-formed XML document
- ❖ Follows XML lexical conventions
 - Wrong: <section>We show that $x < 0$...</section>
 - Right: <section>We show that $x \leq 0$...</section>
 - Other special entities: > becomes > and & becomes &
 - ❖ 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>...</section>...</subsection>

A tree representation

7



More XML features

8

- ❖ Comments: `<!-- Comments here -->`
- ❖ CDATA: `<![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

```
<book xmlns:myCitationStyle="http://mySchema">
  <myCitationStyle:title>...</myCitationStyle:title>
  <myCitationStyle:author>...</myCitationStyle:author>
</book>
```
- ❖ Processing instructions for apps: `<? ...java applet... ?>`
- ❖ And more...

Valid XML documents

9

- ❖ 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 section (title, (#PCDATA)?, section*)>
]>
```

DTD explained

10

```
<!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 more  
      └ 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> ── price attribute which contains  
  └ book has an optional (#IMPLIED) character data  
    └ price attribute which contains character data  
Other attribute types include IDREF (reference to an ID),  
IDREFS (space-separated list of references), enumerated list, etc.
```

DTD explained (cont'd)

11

```
<!ELEMENT title (#PCDATA)> ── PCDATA is text that will be parsed  
<!ELEMENT author (#PCDATA)> ── (<...> will be treated as a markup tag  
<!ELEMENT publisher (#PCDATA)> ── and &lt; etc. will be treated as entities);  
<!ELEMENT year (#PCDATA)> ── CDATA is unparsed character data  
  └ 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  
] >  
  <section><title>Introduction</title>  
  In this section we introduce XML and DTD.  
  <section><title>XML</title>  
  XML stands for...  
  </section>  
  <section><title>DTD</title>  
  <section><title>Definitions</title>  
  DTD stands for...  
  </section>  
  <section><title>Usages</title>  
  You can use DTD to...  
  </section>  
</section>  
</section>
```

“Deterministic” content declaration

12

- ❖ Catch: the following declaration may not work:
 - `<!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

13

❖ 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>
```

Why use DTD's?

14

❖ Benefits of not using DTD

❖ Benefits of using DTD

XML versus relational data

15

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) 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

Query languages for XML

16

❖ 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

17

```
<?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, (#PCDATA)?, 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>
```

XPath

18

- ❖ 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

19

- / 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

20

- ❖ 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

21

- [*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

22

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

23

`/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

24

Frequently used in conditions:

$x + y$, $x - y$, $x * y$, $x \text{ div } y$, $x \text{ mod } y$

`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

25

- ❖ 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`
 - A shorthand: `/bibliography/book/section[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

26

- ❖ 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

27

`id(identifier)` returns the element with *identifier*

- ❖ Suppose that books can reference 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”
`/bibliography/book[@ISBN='ISBN-10']`
`//bookref[id(@ISBN)/author='Abiteboul']`
Or simply:
`id("ISBN-10")//bookref[id(@ISBN)/author="Abiteboul"]`

General XPath location steps

28

- ❖ 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

†These reverse axes produce result node-sets in reverse document order; others (forward axes) produce node-sets in document order

Example of verbose syntax

29

Verbose (axis, node test, predicate):

```
/child::bibliography  
  /child::book[attribute::ISBN='ISBN-10']  
  /descendant-or-self::node()  
  /child::title
```

Abbreviated:

```
/bibliography/book[@ISBN='ISBN-10']//title
```

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

One more example

30

- ❖ 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]

Some technical details on evaluation 31

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
