XML, DTD, and XPath

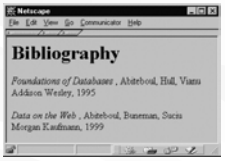
CPS 216

Advanced Database Systems

From HTML to XML (eXtensible Markup Language)

- HTML describes the presentation of the 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

- 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


```
<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>
```

- 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">`

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>…</section>…</subsection>`

More XML features

- 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

- 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

```
<!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 one

→ Zero or more

→ `book` consists of a `title`, zero or more `author`s, 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)>
```
→ `title`, `author`, `publisher`, and `year` all contain parsed character data (`#PCDATA`)

`PCDATA` is text that will be parsed (<…> will be treated as a markup tag and `<` etc. will be treated as entities); `CDATA` is unparsed character data

```
<!ELEMENT section (title, (#PCDATA)?, section*)>
```
→ Each `section` starts with a `title`, followed by some optional text and then zero of 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>Definition</title>
      DTD stands for…
    </section>
    <section><title>Usage</title>
      You can use DTD to…
    </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>
    ```

Why use DTD's?

- Benefits of using DTD
- Benefits of not using DTD

XML versus relational data

Relational data	XML 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	
    ```

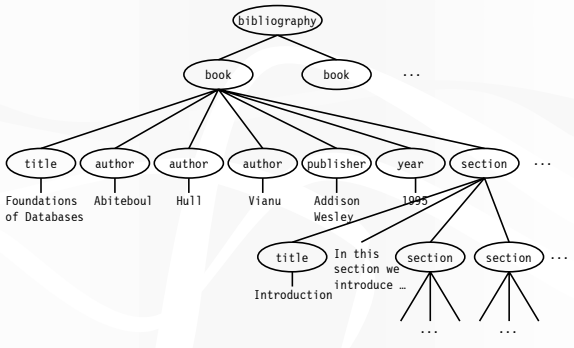
Query languages for XML

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

Example DTD and 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, (#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>
```


A tree representation



XPath

- XPath specifies path expression that match XML data by navigating down (and occasionally up and across) the tree
- Example
  - Query: `/bibliography/book/author`
    - Like a UNIX directory
  - 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

- 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"
- 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 position of the current node in the currently selected node-set

`last()` returns the size of the currently selected node-set

`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
  ```
  - 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

- 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]
    ```
  - Correct answer:

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"

```
/bibliography/book[@ISBN='ISBN-10']
  //bookref[id(@ISBN)/author='Abiteboul']
```

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`, `section`, `*`) or a type test (e.g., `text()`, `node()`, `comment()`), separated from the axis by `::`
  - Zero of more predicates (or conditions) enclosed in square brackets

Example of verbose syntax

Verbose (axis, node test, predicate):

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

Abbreviated:

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

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