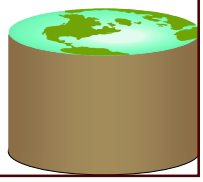
Introduction to XML, XPath, & XQuery

CS186, Fall 2005
R &G - Chapters 7-27

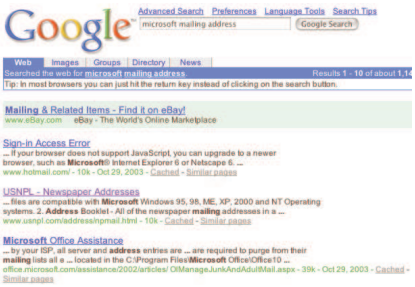
Bill Gates, The Revolution, and a Network of Trees
(based on a true story)



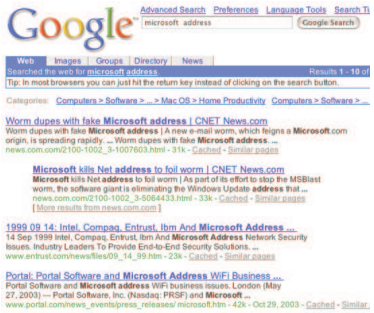



Letter to Bill Gates

"Microsoft mailing address"



"Microsoft address"



Web Search Today

- **Web document: bag of words**
- **HTML: presentation language**

```
<I>
   Microsoft<BR>
   One Microsoft Way<BR>
   Redmond, WA<BR>
</I>
```

```
<I>
   Terriyaki sauce<BR>
   One egg<BR>
   New York steak<BR>
</I>
```

- **Difficult to identify structure/semantics**

A first step - XML

- **Focus on structure/semantics instead of layout**

"Microsoft mailing address"

```
<I>
   Microsoft<BR>
   One Microsoft Way<BR>
   Redmond, WA<BR>
</I>
```

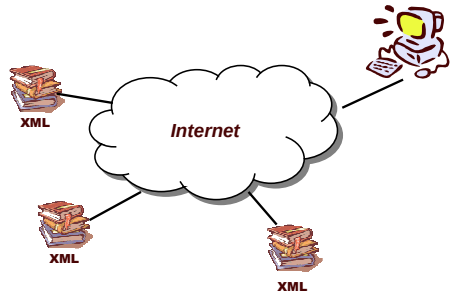
address[.*name="Microsoft"]

```
<address>
  <company name="Microsoft">
  <street>One Microsoft way</street>
  <city>Redmond</city>
  <state>WA</state>
</address>
```

HTML vs. XML

- **HTML**
  - Fixed set of tags for markups
  - Semantically poor : tags only describe presentation of data
- **XML**
  - *Extensible* set of *semantically-rich* tags
  - Describe *meaning/semantics* of the data

The Revolution



XML Data (Text)

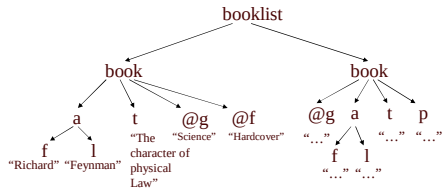
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<booklist>
        <book genre="Science" format="Hardcover">
                <author>
                        <firstname>Richard</firstname>
                        <lastname>Feynman</lastname>
                </author>
                <title>The character of Physical Law</title>
        </book>
        <book genre="Fiction">
                <author>
                        <firstname>R.K.</firstname>
                        <lastname>Narayan</lastname>
                </author>
                <title>Waiting for the Mahatma</title>
                <published>1981</published>
        </book>
</booklist>
```

Content

Element

Nesting

XML Data (Tree)



XML Basics

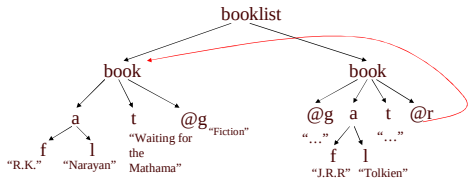
- **Elements**
  - Encode "concepts" in the XML database
  - Nesting denotes association/inclusion
- **Attributes**
  - Record information specific to an element (e.g., the genre of a book)
- **References**
  - Links between elements in different parts of the document

Example of XML References

```
<booklist>
        <book id="narayan_w4m" genre="Fiction">
                <author>
                        <firstname>R.K.</firstname>
                        <lastname>Narayan</lastname>
                </author>
                <title>Waiting for the Mahatma</title>
        </book>
        …
        <book id="tolkien_lotr" genre="Fiction">
                <author>
                        <firstname>J.R.R.</firstname>
                        <lastname>Tolkien</lastname>
                </author>
                <title>The Lord of the Rings</title>
                <related ref="narayan_w4m"/>
        </book>
</booklist>
```

Data becomes a graph

XML Data with References



What about a schema?

- **XML does not require a schema**
  - After all, data is *self-describing*
  - More flexibility, less usability!
- **There are two means for defining a "schema":**
  - A Document Type Definition (DTD)
  - An XML Schema
  - Fix vocabulary of tags (and semantics)
    - Match information across different XML documents
  - Describe nesting structure
    - Know where to look for what information

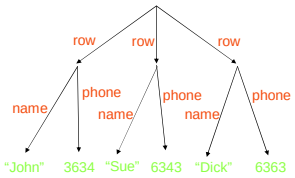
Document Type Definition

```
<!DOCTYPE BOOKLIST [
<!ELEMENT BOOKLIST (BOOK)*>
        <!ELEMENT BOOK (AUTHOR,TITLE,PUBLISHED?) >
                <!ELEMENT FIRSTNAME (#PCDATA)>
                <!ELEMENT LASTNAME (#PCDATA)>
        <!ELEMENT TITLE (#PCDATA)>
        <!ELEMENT PUBLISHED (#PCDATA)>
        <!ATTLIST BOOK GENRE (Science|Fiction) #REQUIRED>
        <!ATTLIST BOOK FORMAT (Paperback|Hardcover) "Paperback">
]>
```

- **DTD specifies a *regular expression* for every element**
- **Does not specify the type of content**
- **"Loosely" structured data compared to relational tables**
  - ***Semistructured data***

XML vs. Relational Data

name	phone
John	3634
Sue	6343
Dick	6363



Relation

XML

XML vs. Relational Data

- **A relation instance is basically a tree with:**
  - Unbounded fanout at level 1 (i.e., any # of rows)
  - Fixed fanout at level 2 (i.e., fixed # fields)
- **XML data is essentially an arbitrary tree**
  - Unbounded fanout at all nodes/levels
  - Any number of levels
  - Variable # of children at different nodes, variable path lengths

Query Language for XML

- **Must be high-level; "SQL for XML"**
- **Must conform to DTD/XML Schema**
  - But also work in absence of schema info
- **Support simple and complex/nested datatypes**
- **Support universal and existential quantifiers, aggregation**
- **Operations on sequences and hierarchies of document structures**
- **Capability to transform and create XML structures**

Overview of XQuery

- **Path expressions (XPath)**
- **Element constructors**
- **FLWOR ("flower") expressions**
  - Several other kinds of expressions as well, including conditional expressions, list expressions, quantified expressions, etc.
- **Expressions evaluated w.r.t. a context:**
  - Context item (current node)
  - Context position (in sequence being processed)
  - Context size (of the sequence being processed)
  - Context also includes namespaces, variables, functions, date, etc.

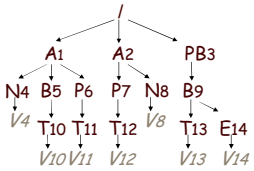
XPath Expressions

Examples:

- **/booklist/book**
- **/booklist/book/author**
- **/booklist/book/author/lastname**

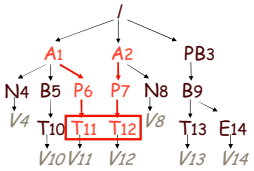
Given an XML document, the *value* of a path expression *p* is a set of elements (= XML subtrees)

Path Expressions



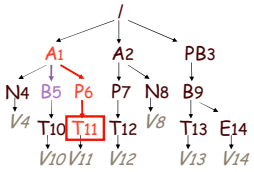
- **XPath expressions**
  - Simple: /A/P/T
  - Branching: /A[B]/P/T
  - Values: /A/P/T[=v11]
- **Result is a <u>set</u>**

Path Expressions



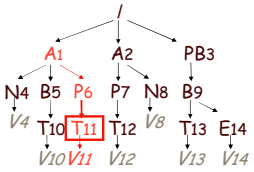
- **XPath expressions**
  - Simple: /A/P/T
  - Branching: /A[B]/P/T
  - Values: /A/P/T[=v11]
- **Result is a <u>set</u>**

Path Expressions



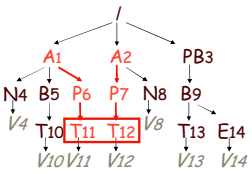
- **XPath expressions**
  - Simple: /A/P/T
  - Branching: /A[B]/P/T
  - Values: /A/P/T[=v11]
- **Result is a <u>set</u>**

Path Expressions



- **XPath expressions**
  - Simple: /A/P/T
  - Branching: /A[B]/P/T
  - Values: /A/P/T[=v11]
- **Result is a <u>set</u>**

Path Expressions



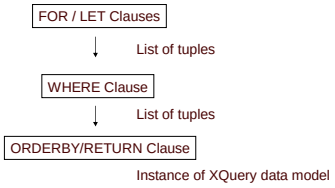
- **XPath expressions**
  - Simple: /A/P/T
  - Branching: /A[B]/P/T
  - Values: /A/P/T[=v11]
- **Result is a set**

XPath Syntax

- **Path wildcards**
  - // = descendant at any level (or self)
  - * = any (single) tag
  - Example: **/booklist//lastname**
- **Query attributes and attribute content**
  - Use "@"
  - Examples: **/booklist//book[@format="Paperback"], /booklist//book/@genre**
- **Branching predicates: A[*pred*]**
  - Predicate on A's subtree using *logical connectives* (and, or, etc.), *path expressions*, *built-in functions* (e.g., contains()), etc.
  - Example: **//author[contains(./lastname, "Fey")]**

XQuery FLWOR Expressions

- **FOR-LET-WHERE-ORDERBY-RETURN = FLWOR**



FOR vs. LET

- **FOR $x IN path-expression**
  - Binds $x in turn to each element in the expression
- **LET $x := path-expression**
  - Binds $x to the *entire list of elements* in the expression
  - Useful for common sub-expressions and for aggregations

FOR vs. LET: Example

```
FOR $x IN document("bib.xml")/bib/book
RETURN <result> $x </result>
```

Returns:

```
<result> <book>...</book></result>
<result> <book>...</book></result>
<result> <book>...</book></result>
...
```

Notice that result has several elements

```
LET $x := document("bib.xml")/bib/book
RETURN <result> $x </result>
```

Returns:

```
<result> <book>...</book>
         <book>...</book>
         <book>...</book>
           ...
</result>
```

Notice that result has exactly one element

XQuery Example 1

Find all book titles published after 1995:

```
FOR $x IN document("bib.xml")/bib/book
WHERE $x/year > 1995
RETURN $x/title
```

Result:

```
<title> abc </title>
<title> def </title>
<title> ghi </title>
```

XQuery Example 2

For each author of a book by Morgan Kaufmann, list all books she published:

```
FOR $a IN distinct(
 document("bib.xml"/bib/book[publisher="Morgan Kaufmann"]/author))
RETURN <result>
        $a,
        FOR $t IN /bib/book[author=$a]/title
        RETURN $t
      </result>
```

distinct = a function that eliminates duplicates (after converting inputs to atomic values)

Results for Example 2

```
<result>
    <author>Jones</author>
    <title> abc </title>
    <title> def </title>
</result>
<result>
    <author> Smith </author>
    <title> ghi </title>
</result>
```

Observe how nested structure of result elements is determined by the nested structure of the query.

XQuery Example 3

```
<big_publishers>
    FOR $p IN distinct(document("bib.xml")//publisher)
    LET $b := document("bib.xml")/book[publisher = $p]
    WHERE count($b) > 100
    RETURN $p
</big_publishers>
```

For each publisher p

- Let the list of books published by p be b

Count the # books in b, and return p if b > 100

count = (aggregate) function that returns the number of elements

XQuery Example 4

Find books whose price is larger than average:

```
LET $a := avg(document("bib.xml")/bib/book/price)
FOR $b in document("bib.xml")/bib/book
WHERE $b/price > $a
RETURN $b
```

Collections in XQuery

- **Ordered and unordered collections**
  - /bib/book/author = an ordered collection
  - Distinct(/bib/book/author) = an unordered collection
- **Examples:**
  - LET $a = /bib/book → $a is a collection
  - $b/author → also a collection (several authors...)

However:

```
RETURN <result> $b/author </result>
```

Returns a single collection!

```
<result> <author>...</author>
         <author>...</author>
         <author>...</author>
         ...
</result>
```

Collections in XQuery

What about collections in expressions ?

- **$b/price → list of n prices**
- **$b/price * 0.7 → list of n numbers??**
- **$b/price * $b/quantity → list of n x m numbers ??**
  - Valid only if the two sequences have at most one element
  - Atomization
- **$book1/author eq "Kennedy" - Value Comparison**
- **$book1/author = "Kennedy" - General Comparison**

Sorting in XQuery

```
<publisher_list>
  FOR $p IN distinct(document("bib.xml")//publisher)
  ORDERBY  $p
  RETURN <publisher> <name> $p/text() </name> ,
            FOR $b IN document("bib.xml")//book[publisher = $p]
            ORDERBY $b/price DESCENDING
            RETURN <book>
                      $b/title ,
                      $b/price
                   </book>
         </publisher>
</publisher_list>
```

Conditional Expressions: If-Then-Else

```
FOR $h IN //holding
ORDERBY $h/title
RETURN <holding>

          $h/title,

          IF $h/@type = "Journal"

               THEN $h/editor

          ELSE $h/author

       </holding>
```

Existential Quantifiers

```
FOR $b IN //book
WHERE SOME $p IN $b//para SATISFIES
  contains($p, "sailing")
  AND contains($p, "windsurfing")
RETURN $b/title
```

Universal Quantifiers

```
FOR $b IN //book
WHERE EVERY $p IN $b//para SATISFIES
  contains($p, "sailing")
RETURN $b/title
```

Other Stuff in XQuery

- **Before and After**
  - for dealing with order in the input
- **Filter**
  - deletes some edges in the result tree
- **Recursive functions**
- **Namespaces**
- **References, links …**
- **Lots more stuff …**

XML & PostgreSQL

- **Store XML documents as text BLOBs (Binary Large Objects) inside text-valued columns**
- **Load XML *in-memory* and use external User- Defined Functions (UDFs) to process XPath expressions**
  - xpath_bool(xml_text_col, "xpath_query_string")
    - False/true if element set discovered is empty/nonempty
  - xpath_nodeset(xml_text_col, "xpath_query_string")
    - Text result = concatenation of element subtrees
- **No support for full-fledged XQuery**
  - Some support for XSLT transformations -- won't discuss here…
- **Pros/Cons??**

Summary

- **XML has gained momentum as a "universal data format"**
  - Standard for publishing/exchange in business world
- **Jury is still out for the "data model" part**
  - Still need a lot of work on efficient storage/ indexing, query optimization, …
- **Increasing support in commercial systems**
  - BLOB approach is common, others (e.g., DB2) map XML to/from relational
  - A few "native" systems
- **XML is the foundation for the next "Web Revolution"**
  - *Semantic web*, *web services*, *ontologies*, …
  - XML trees will grow everywhere!
    - Click on XML/RSS tabs on web pages, or search for "XML" on your PC

But, don't just take it from me…

"Microsoft has been working with the industry to advance a new generation of software that is interoperable by design, reducing the need for custom development and cumbersome testing and certification. These efforts are centered on using XML, which makes information self-describing – and thus more easily understood by different systems. … This approach is also the foundation for XML-based Web services, which provide an Internet-based set of protocols for distributed computing. This new model for how software talks to other software has been embraced across the industry. It is the cornerstone of Microsoft .NET and the latest generation of our Visual Studio tools for software developers. This approach is also evident in the use of XML as the data interoperability framework for Office 2003 and the Office System set of products."

Bill Gates, MS Executive Email, Feb'05

- **Microsoft's address:**
  - One Microsoft Way
    Redmond, WA

Some Online Resources

- **XPath tutorials**
  - http://www.w3schools.com/xpath/
  - http://www.zvon.org/xxl/XPathTutorial/General/examples.html
- **XQuery tutorials**
  - http://www.w3schools.com/xquery/default.asp
  - http://www.db.ucsd.edu/people/yannis/XQueryTutorial.htm
- **XML reading**
  - http://www.rpbourret.com/xml/XMLAndDatabases.htm