CSE544: Lecture 6

XML: Schemas, Queries

Wednesday, 4/17/2002

Project Ideas

1. System catalog for Piazza.
2. Query optimization in Piazza.
3. Update propagation in Piazza. (taken)
4. Encryption, security in Piazza and the XML Toolkit
5. Define events in the XMLTK. [talk to me]
6. Improved index computation in the XML toolkit (SIX) [talk to me]
7. Xpath containment/equivalence algorithm.

Outline

- XML DTDs
- [we skip XML Schema: please see slides from last lecture]
- XPath
- XQuery

Document Type Definitions DTD

- part of the original XML specification
- an XML document may have a DTD
- terminology for XML:
  - well-formed: if tags are correctly closed
  - valid: if it has a DTD and conforms to it
- validation is useful in data exchange

Very Simple DTD

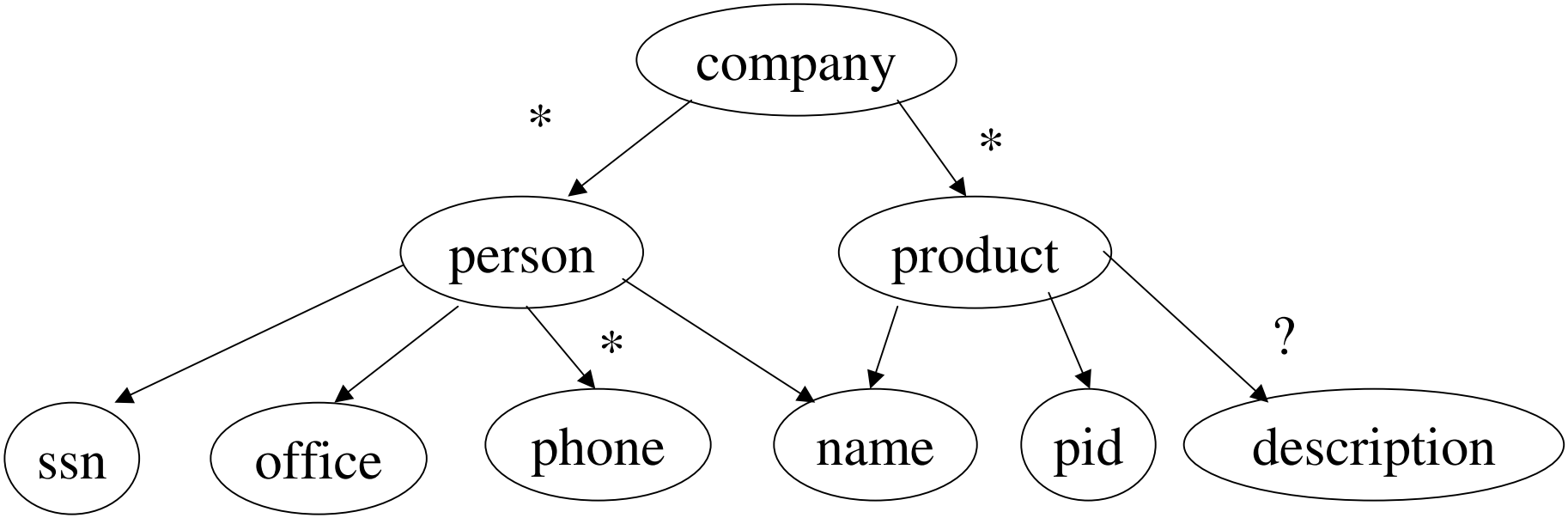
```
<!DOCTYPE company [
  <!ELEMENT company   ((person|product)*)>
  <!ELEMENT person  (ssn, name, office, phone*)>
  <!ELEMENT ssn        (#PCDATA)>
  <!ELEMENT name     (#PCDATA)>
  <!ELEMENT office     (#PCDATA)>
  <!ELEMENT phone    (#PCDATA)>
  <!ELEMENT product  (pid, name, description?)>
  <!ELEMENT pid    (#PCDATA)>
   <!ELEMENT description    (#PCDATA)>
]>
```

Very Simple DTD

Example of valid XML document:

```
<company>
   <person>  <ssn> 123456789 </ssn>
             <name> John </name>
             <office> B432 </office>
             <phone> 1234 </phone>
             <phone> 6789 </phone>
   </person>
   <person>  <ssn> 987654321 </ssn>
             <name> Jim </name>
             <office> B123 </office>
   </person>
   <product> ...  </product>
   ...
</company>
```

Very Simple DTD



Graph representation of a DTD (notice: not necessarily a tree)
Lose order information (office before or after phone ?)

Content Model

- Element *content*: what we can put in an element (aka *content model*)
- Content model:
  - Complex = a regular expression over other elements
  - Text-only = #PCDATA
  - Empty = EMPTY
  - Any = ANY
  - Mixed content = (#PCDATA | A | B | C)*
    - (i.e. very restrictied)

Attributes in DTDs

```
<!ELEMENT person  (ssn, name, office, phone?)>
<!ATTLIS      person age CDATA  #REQUIRED>
```

```
<person age="25">
      <name> ....</name>
       ...
</person>
```

Attributes in DTDs

```
<!ELEMENT person  (ssn, name, office, phone?)>
<!ATTLIS      person age          CDATA  #REQUIRED
                     id           ID       #REQUIRED
                     manager    IDREF   #REQUIRED
                     manages    IDREFS   #REQUIRED
>
```

```
<person age="25"
        id="p29432"
        manager="p48293"  manages="p34982 p423234">
    <name> ....</name>
     ...
</person>
```

Attributes in DTDs

Types:

- CDATA = string
- ID = key
- IDREF = foreign key
- IDREFS = foreign keys separated by space
- (Monday | Wednesday | Friday) = enumeration
- NMTOKEN = must be a valid XML name
- NMTOKENS = multiple valid XML names
- ENTITY = you don’t want to know this

Attributes in DTDs

Kind:

- #REQUIRED
- #IMPLIED = optional
- value = default value
- value #FIXED = the only value allowed

Using DTDs

- Must include in the XML document
- Either include the entire DTD:
  - <!DOCTYPE rootElement [.....]>
- Or include a reference to it:
  - <!DOCTYPE rootElement SYSTEM "http://www.mydtd.org">
- Or mix the two... (e.g. to override the external definition)

DTDs as Grammars

```
<!DOCTYPE paper [
  <!ELEMENT paper   (section*)>
  <!ELEMENT section ((title,section*) | text)>
  <!ELEMENT title      (#PCDATA)>
  <!ELEMENT text       (#PCDATA)>
]>
```

```
<paper> <section> <text> </text> </section>
          <section> <title> </title> <section> … </section>
                                          <section> … </section>
          </section>
</paper>
```

DTDs as Grammars

- A DTD = a grammar
- A valid XML document = a parse tree for that grammar

DTDs as Schemas

Not so well suited:

- impose unwanted constraints on order

  <!ELEMENT person (name,phone)>

- references cannot be constrained
- can be too vague:

  <!ELEMENT person ((name|phone|email)*)>

From DTDs to Relational Schemas

Shanmugasundaram et al. *Relational databases for querying XML documents"*

Design a relational schema for:

```
<!DOCTYPE company [
  <!ELEMENT company   ((person|product)*)>
  <!ELEMENT person  (ssn, name, office?, phone*)>
  <!ELEMENT ssn       (#PCDATA)>
  <!ELEMENT name     (#PCDATA)>
  <!ELEMENT office     (#PCDATA)>
  <!ELEMENT phone    (#PCDATA)>
  <!ELEMENT product  (pid, name, description?)>
  <!ELEMENT pid    (#PCDATA)>
   <!ELEMENT description    (#PCDATA)>
]>
```

Where Should We Store XML Data ?

- "Native" XML databases:
  - Total market in 2001: only $77 million [research firm IDC]
  - But growing dramatically (six fold per year)
- Main players:
  - Software AG with *Tamino XML*: 40.5% share of the market.
  - eXcelon (formerly ObjectStore): 23.3%
  - Computer Associates: 19.4%
  - Poet Software: 1.3%
  - Remaining 15%: Linux-based open source tools (many based around the dbXML core), startups (Ipedo, Ixiasoft)

What is the risk for these companies ?

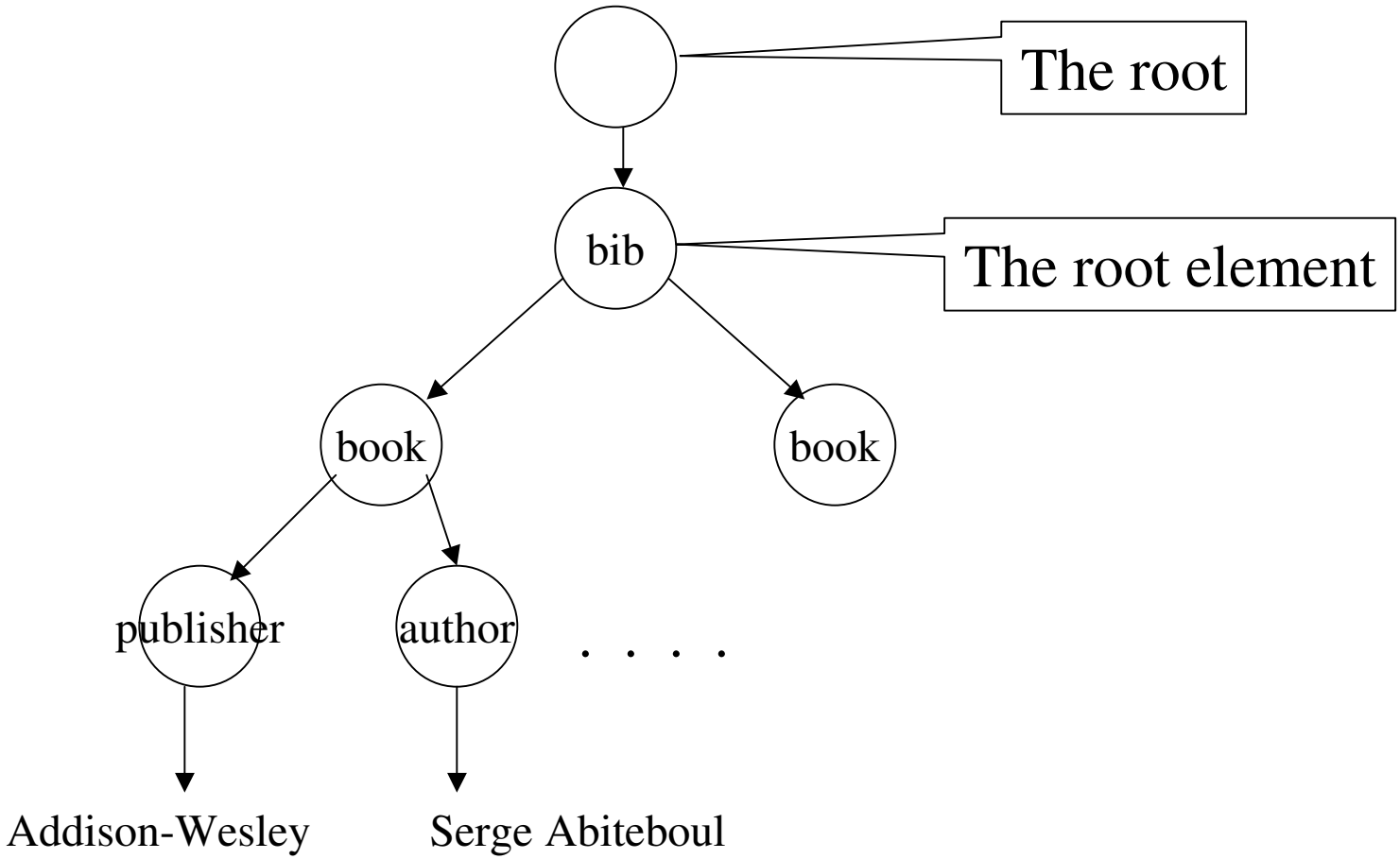
XPath

- http://www.w3.org/TR/xpath (11/99)
- Building block for other W3C standards:
  - XSL Transformations (XSLT)
  - XML Link (XLink)
  - XML Pointer (XPointer)
  - XML Query
- Was originally part of XSL

Example for XPath Queries

```
<bib>
    <book>  <publisher> Addison-Wesley </publisher>
            <author> Serge Abiteboul </author>
            <author> <first-name> Rick </first-name>
                     <last-name> Hull </last-name>
            </author>
            <author> Victor Vianu </author>
            <title> Foundations of Databases </title>
            <year> 1995 </year>
    </book>
    <book price="55">
            <publisher> Freeman </publisher>
            <author> Jeffrey D. Ullman </author>
            <title> Principles of Database and Knowledge Base Systems </title>
            <year> 1998 </year>
    </book>
</bib>
```

Data Model for XPath



XPath: Simple Expressions

```
/bib/book/year
```

Result: <year> 1995 </year>
<year> 1998 </year>

```
/bib/paper/year
```

Result: empty (there were no papers)

XPath: Restricted Kleene Closure

```
//author
```

Result: <author> Serge Abiteboul </author>
<author> <first-name> Rick </first-name>
<last-name> Hull </last-name>
</author>
<author> Victor Vianu </author>
<author> Jeffrey D. Ullman </author>

```
/bib//first-name
```

Result: <first-name> Rick </first-name>

Xpath: Text Nodes

```
/bib/book/author/text()
```

Result: Serge Abiteboul
Jeffrey D. Ullman

Rick Hull doesn't appear because he has firstname, lastname

Functions in XPath:

- text() = matches the text value
- node() = matches any node (= * or @* or text())
- name() = returns the name of the current tag

Xpath: Wildcard

```
//author/*
```

Result: <first-name> Rick </first-name>
<last-name> Hull </last-name>

* Matches any element

Xpath: Attribute Nodes

```
/bib/book/@price
```

Result: “55”

@price means that price is has to be an attribute

Xpath: Qualifiers

```
/bib/book/author[firstname]
```

Result: <author> <first-name> Rick </first-name>
<last-name> Hull </last-name>
</author>

Xpath: More Qualifiers

```
/bib/book/author[firstname][address[//zip][city]]/lastname
```

Result: <lastname> … </lastname>

<lastname> … </lastname>

Xpath: More Qualifiers

```
/bib/book[@price < “60”]
```

```
/bib/book[author/@age < “25”]
```

```
/bib/book[author/text()]
```

Xpath: Summary

bib	matches a bib element
*	matches any element
/	matches the root element
/bib	matches a bib element under root
bib/paper	matches a paper in bib
bib//paper	matches a paper in bib, at any depth
//paper	matches a paper at any depth
paper|book	matches a paper or a book
@price	matches a price attribute
bib/book/@price	matches price attribute in book, in bib

bib/book/[@price<“55”]/author/lastname matches…

Xpath: More Details

- An Xpath expression, p, establishes a relation between:
  - A *context node*, and
  - A node in the *answer set*
- In other words, p denotes a function:
  - S[p] : Nodes -> {Nodes}
- Examples:
  - author/firstname
  - . = self
  - .. = parent
  - part/*/*/subpart/../name = part/*/*[subpart]/name

The Root and the Root

- <bib> <paper> 1 </paper> <paper> 2 </paper> </bib>
- bib is the "***document element***"
- The "***root***" is above bib

- /bib = returns the document element
- / = returns the root

- Why ? Because we may have comments before and after <bib>; they become siblings of <bib>

- This is advanced xmlogy

Xpath: More Details

- We can navigate along 13 axes:
  - ancestor
  - ancestor-or-self
  - attribute
  - child
  - descendant
  - descendant-or-self
  - following
  - following-sibling
  - namespace
  - parent
  - preceding
  - preceding-sibling
  - self

Xpath: More Details

- Examples:
  - child::author/child:lastname = author/lastname
  - child::author/descendant::zip = author//zip
  - child::author/parent::* = author/..
  - child::author/attribute::age = author/@age
- What does this mean ?
  - paper/publisher/parent::*/author
  - /bib//address[ancestor::book]
  - /bib//author/ancestor::*//zip

Xpath: Even More Details

- name() = the name of the current node
  - /bib//*[name()=book] same as /bib//book
- What does this mean ?
  /bib//*[ancestor::*[name()!=book]]
  - In a different notation bib.[^book]*._
- Navigation axis give us strictly more power !

XQuery

- Based on Quilt
  (which is based on XML-QL)
- http://www.w3.org/TR/xquery/
  2/2001
- XML Query data model (of course ☺)
  - Ordered !

FLWR (“Flower”) Expressions

FOR ... LET... FOR... LET...
WHERE...
RETURN...

XQuery

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

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

XQuery

Result:

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

XQuery

- FOR $x in expr -- binds $x to each value in the list expr

- LET $x = expr -- binds $x to the entire list expr
  - Useful for common subexpressions and for aggregations

XQuery

```
<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>
```

count = a (aggregate) function that returns the number of elms

XQuery

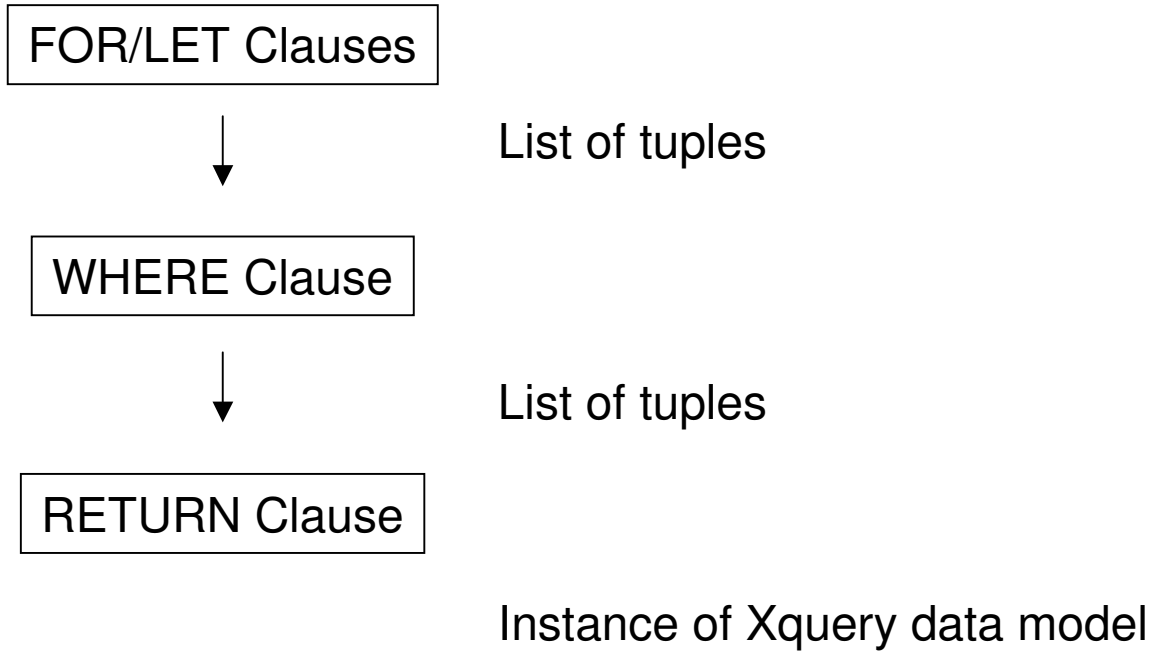
Find books whose price is larger than average:

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

XQuery

Summary:

- FOR-LET-WHERE-RETURN = FLWR



FOR v.s. LET

FOR

- Binds *node variables* → iteration

LET

- Binds *collection variables* → one value

FOR v.s. LET

```
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>
...
```

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

Returns:

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

Collections in XQuery

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

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

Returns:

```
<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 ??
- $b/price * ($b/quant1 + $b/quant2) ≠
  $b/price * $b/quant1 + $b/price * $b/quant2 !!

Sorting in XQuery

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

Sorting in XQuery

- Sorting arugments: refer to the name space of the RETURN clause, not the FOR clause

If-Then-Else

```
FOR $h IN //holding
RETURN <holding>
                {  $h/title,
                   IF $h/@type = "Journal"
                            THEN $h/editor
                   ELSE $h/author
                 }
              </holding> SORTBY (title)
```

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
  - Currently: arbitrary recursion
  - Perhaps more restrictions in the future ?