CSE544
XML, XQuery

Monday+Wednesday,
April 11+13, 2004

Announcements

Readings:

- Simeon, Wadler: The essence of XML. POPL 2003. Review due Wednesday
- J. Shanmugasundaram, K. Tufte, C. Zhang, G. He, D. J. DeWitt, J. F. Naughton. Relational Databases for Querying XML Documents: Limitations and Opportunities VLDB 1999. Review due on Monday
- Other suggested readings:
  - http://www.xml.com/pub/a/98/10/guide0.html
  - http://www.w3.org/TR/2003/WD-xquery-use-cases-20030502/
- Guest Lecturer: Alan Fekete, Transactions, Monday, Wednesday

Outline

- XML: syntax, semantics, data, DTDs
- XPath
- Xquery

XML Syntax

- tags: book, title, author, …
- start tag: <book>, end tag: </book>
- elements: <book>…<book>,<author>…</author>
- elements are nested
- empty element: <red></red> abbrv. <red/>
- an XML document: single *root element*

well formed XML document: if it has matching tags

XML Syntax

```
<book price = "55" currency = "USD">
  <title> Foundations of Databases </title>
  <author> Abiteboul </author>
  …
  <year> 1995 </year>
</book>
```

attributes are alternative ways to represent data

XML Syntax

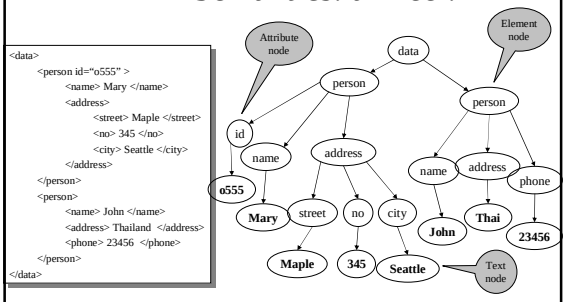
```
<person id="o555">  <name> Jane </name> </person>

<person id="o456">  <name> Mary </name>
                    <children idref="o123 o555"/>
</person>

<person id="o123" mother="o456"><name>John</name>
</person>
```

oids and references in XML are just syntax

XML Semantics: a Tree !




```
<data>
    <person id="o555" >
        <name> Mary </name>
        <address>
            <street> Maple </street>
            <no> 345 </no>
            <city> Seattle </city>
        </address>
    </person>
    <person>
        <name> John </name>
        <address> Thailand </address>
        <phone> 23456 </phone>
    </person>
</data>
```

Order matters !!!

7

XML Data

- XML is self-describing
- Schema elements become part of the data
  - Reational schema: persons(name,phone)
  - In XML <persons>, <name>, <phone> are part of the data, and are repeated many times
- Consequence: XML is much more flexible
- XML = semistructured data

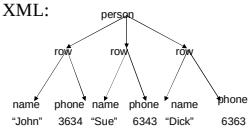
8

Relational Data as XML

person

name	phone
John	3634
Sue	6343
Dick	6363

XML:




```
<person>
    <row> <name>John</name>
          <phone> 3634</phone></row>
    <row> <name>Sue</name>
          <phone> 6343</phone>
    <row> <name>Dick</name>
          <phone> 6363</phone></row>
</person>
```

9

XML is Semi-structured Data

- Missing attributes:

```
<person>   <name> John</name>
           <phone>1234</phone>
 </person>

<person>  <name>Joe</name>
</person>
```

ß no phone !

- Could represent in a table with nulls

name	phone
John	1234
Joe	-

10

XML is Semi-structured Data

- Repeated attributes

```
<person> <name> Mary</name>
         <phone>2345</phone>
         <phone>3456</phone>
</person>
```

ß two phones !

- Impossible in tables:

name	phone		
Mary	2345	3456	???

11

XML is Semi-structured Data

- Attributes with different types in different objects

```
<person> <name>  <first> John </first>
                 <last> Smith </last>
         </name>
         <phone>1234</phone>
</person>
```

ß structured name !

- Nested collections (non 1NF)
- Heterogeneous collections:
  - <db> contains both <book>s and <publisher>s

12

Document Type Definitions DTD

- Part of the original XML specification
- To be replaced by XML Schema
  - Much more complex
- An XML document may have a DTD
- XML document:

  well-formed = if tags are correctly closed

  Valid = if it has a DTD and conforms to it
- Validation is useful in data exchange

13

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)>
]>
```

14

Very Simple DTD

Example of valid XML document:

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

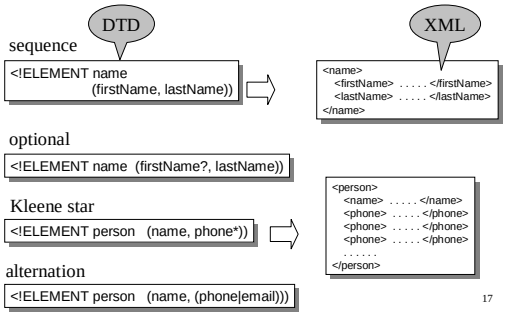
15

DTD: The Content Model

- Content model:
  - Complex = a regular expression over other elements
  - Text-only = #PCDATA
  - Empty = EMPTY
  - Any = ANY
  - Mixed content = (#PCDATA | A | B | C)*

16

DTD: Regular Expressions



17

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)>
]>
```

18

Very Simple DTD

Example of valid XML document:

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

19

Attributes in DTDs

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

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

20

Attributes in DTDs

```
<!ELEMENT person  (ssn, name, office, phone?)>
<!ATTLIST    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

22

Attributes in DTDs

Kind:

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

23

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)

24

Querying XML Data

- XPath = simple navigation through the tree
- XQuery = the SQL of XML
- XSLT = recursive traversal
  – will not discuss in class

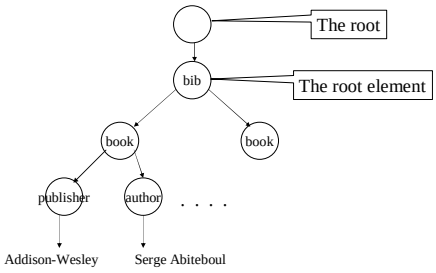
25

Sample Data for 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>
```

26

Data Model for XPath



27

XPath: Simple Expressions

/bib/book/year

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

/bib/paper/year

Result: empty (there were no papers)

28

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 Rick </first-name>

29

Xpath: Text Nodes

/bib/book/author/text()

Result: Serge Abiteboul
Victor Vianu
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

30

Xpath: Wildcard

//author/*

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

* Matches any element

31

Xpath: Attribute Nodes

/bib/book/@price

Result: “55”

@price means that price is has to be an attribute

32

Xpath: Predicates

/bib/book/author[firstname]

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

33

Xpath: More Predicates

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

Result: <lastname> … </lastname>
<lastname> … </lastname>

34

Xpath: More Predicates

/bib/book[@price < 60]

/bib/book[author/@age < 25]

/bib/book[author/text()]

35

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…

36

XQuery

- Based on Quilt, which is based on XML-QL
- Uses XPath to express more complex queries

37

FLWR ("Flower") Expressions

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

38

Sample Data for Queries (more or less)

```
<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>
```

39

FOR-WHERE-RETURN

Find all book titles published after 1995:

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

Result:

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

40

FOR-WHERE-RETURN

Equivalently (perhaps more geekish)

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

And even shorter:

```
document("bib.xml")/bib/book[year/text() > 1995] /title
```

41

FOR-WHERE-RETURN

- Find all book titles and the year when they were published:

```
FOR $x IN document("bib.xml")/bib/book
RETURN <answer>
            <what>{ $x/title/text() } </what>
            <when>{ $x/year/text() } </when>
       </answer>
```

We can construct whatever XML results we want !

42

Answer

```
<answer>
    <what> How to cook a Turkey </what>
    <when> 2003 </when>
</answer>
<answer>
    <what> Cooking While Watching TV </what>
    <when> 2004 </when>
</answer>
<answer>
    <what> Turkeys on TV</what>
    <when> 2002 </when>
</answer>
. . . . .
```

43

FOR-WHERE-RETURN

- Notice the use of "{" and "}"
- What is the result without them ?

```
FOR $x IN document("bib.xml")/bib/book
RETURN <answer>
            <title>  $x/title/text() </title>
            <year> $x/year/text() </year>
        </answer>
```

44

XQuery: Nesting

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

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

In the RETURN clause comma concatenates XML fragments 45

XQuery

Result:

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

46

Aggregates

Find all books with more than 3 authors:

```
FOR $x IN document("bib.xml")/bib/book
WHERE count($x/author)>3
RETURN $x
```

count = a function that counts
avg = computes the average
sum = computes the sum
distinct-values = eliminates duplicates

47

Aggregates

Same thing:

```
FOR $x IN document("bib.xml")/bib/book[count(author)>3]
RETURN  $x
```

48

Aggregates

Print all authors who published more than 3 books

```
FOR $b IN document("bib.xml")/bib,
        $a IN $b/book/author/text())
WHERE count($b/book[author/text()=$a)>3
RETURN <author> { $a } </author>
```

What's wrong ?

Aggregates

Be aware of duplicates !

```
FOR $b IN document("bib.xml")/bib,
      $a IN distinct-values($b/book/author/text())
WHERE count($b/book[author/text()=$a)>3
RETURN <author> { $a } </author>
```


XQuery

Find books whose price is larger than average:

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

LET binds a variable to one value;
FOR iterates a variable over a list of values
We will come back to that

FOR-WHERE-RETURN

- "Flatten" the authors, i.e. return a list of (author, title) pairs

```
FOR $b IN document("bib.xml")/bib/book,
      $x IN $b/title/text(),
      $y IN $b/author/text()
RETURN <answer>
              <title> { $x } </title>
              <author> { $y } </author>
          </answer>
```

```
Answer:
<answer>
   <title> abc </title>
   <author> efg </author>
</answer>
<answer>
   <title> abc </title>
   <author> hkj </author>
</answer>
```


FOR-WHERE-RETURN

- For each author, return all book titles he/she wrote

```
FOR $b IN document("bib.xml")/bib,
      $x IN $b/book/author/text()
RETURN
  <answer>
    <author> { $x } </author>
    { FOR $y IN $b/book[author/text()=$x]/title
      RETURN $y }
  </answer>
```

```
Answer:
<answer>
   <author> efg </author>
   <title> abc </title>
   <title> klm </title>
   . . . .
</answer>
```

What about duplicate authors ?

FOR-WHERE-RETURN

- Same, but eliminate duplicate authors:

```
FOR $b IN document("bib.xml")/bib
LET $a := distinct-values($b/book/author/text())
FOR $x IN $a
RETURN
  <answer>
    <author> $x </author>
    { FOR $y IN $b/book[author/text()=$x]/title
      RETURN $y }
  </answer>
```

FOR-WHERE-RETURN

- Same thing:

```
FOR $b IN document("bib.xml")/bib,
      $x IN distinct-values($b/book/author/text())
RETURN
  <answer>
    <author> $x </author>
    { FOR $y IN $b/book[author/text()=$x]/title
      RETURN $y }
  </answer>
```

55

SQL and XQuery Side-by-side

Product(pid, name, maker, price)

Find all product names, prices, sort by price

```
SELECT x.name,
       x.price
FROM Product x
ORDER BY x.price
```

SQL

```
FOR $x in document("db.xml")/db/Product/row
ORDER BY $x/price/text()
RETURN <answer>
          { $x/name,  $x/price }
       </answer>
```

XQuery

56

Answers

Name	Price
abc	7
def	23
. . .	. . .

```
<answer>
     <name>  abc  </name>
     <price>  7 </price>
</answer>
<answer>
     <name>  def  </name>
     <price>  23 </price>
</answer>
  . . . .
```

Notice: this is NOT a well-formed document ! (WHY ???)

57

Producing a Well-Formed Answer

```
<myQuery>
  {  FOR $x in document("db.xml")/db/Product/row
     ORDER BY $x/price/text()
     RETURN <answer>
               { $x/name,  $x/price }
            </answer>
  }
</myQuery>
```

58

Xquery's Answer

```
<myQuery>
  <answer>
       <name>  abc  </name>
       <price>  7 </price>
  </answer>
  <answer>
       <name>  def  </name>
       <price>  23 </price>
  </answer>
    . . . .
</myQuery>
```

Now it is well-formed !

59

SQL and XQuery Side-by-side

Product(pid, name, maker, price)
Company(cid, name, city, revenues)

Find all products made in Seattle

```
SELECT x.name
FROM Product x, Company y
WHERE x.maker=y.cid
      and y.city="Seattle"
```

SQL

```
FOR $r in document("db.xml")/db,
       $x in $r/Product/row,
       $y in $r/Company/row
WHERE
   $x/maker/text()=$y/cid/text()
    and  $y/city/text() = "Seattle"
RETURN { $x/name }
```

XQuery

Cool XQuery

```
FOR $y in /db/Company/row[city/text()="Seattle"],
       $x in /db/Product/row[maker/text()=$y/cid/text()]
RETURN { $x/name }
```

```
<product>
      <row> <pid> 123 </pid>
            <name> abc </name>
            <maker> efg </maker>
      </row>
      <row> …. </row>
      …
</product>
<product>
  . . .
</product>
. . . .
```


SQL and XQuery Side-by-side

For each company with revenues < 1M count the products over $100

```
SELECT y.name, count(*)
FROM Product x, Company y
WHERE x.price > 100 and x.maker=y.cid and y.revenue < 1000000
GROUP BY y.cid, y.name
```

```
FOR $r in document("db.xml")/db,
    $y in $r/Company/row[revenue/text()<1000000]
RETURN
  <proudCompany>
    <companyName> { $y/name/text() } </companyName>
    <numberOfExpensiveProducts>
        { count($r/Product/row[maker/text()=$y/cid/text()][price/text()>100]) }
    </numberOfExpensiveProducts>
  </proudCompany>
```

SQL and XQuery Side-by-side

Find companies with at least 30 products, and their average price

```
SELECT y.name, avg(x.price)
FROM Product x, Company y
WHERE x.maker=y.cid
GROUP BY y.cid, y.name
HAVING count(*) > 30
```

An element

A collection

```
FOR $r in document("db.xml")/db,
    $y in $r/Company/row
LET $p := $r/Product/row[maker/text()=$y/cid/text()]
WHERE count($p) > 30
RETURN
  <theCompany>
    <companyName> { $y/name/text() }
    </companyName>
    <avgPrice> avg($p/price/text()) </avgPrice>
  </theCompany>
```


FOR v.s. LET

FOR

- Binds *node variables* à iteration

LET

- Binds *collection variables* à one value

FOR v.s. LET

```
FOR $x IN /bib/book
RETURN <result> { $x } </result>
```

Returns:

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

```
LET $x := /bib/book
RETURN <result> { $x } </result>
```

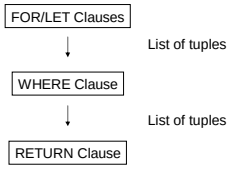
Returns:

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


XQuery

Summary:

- FOR-LET-WHERE-RETURN = FLWR



Collections in XQuery

- Ordered and unordered collections
  - /bib/book/author/text() = an *ordered* collection: result is in *document order*
  - distinct-values(/bib/book/author/text()) = an unordered collection: the output order is implementation dependent
- LET $a := /bib/book à $a is a collection
- $b/author à a collection (several authors...)

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

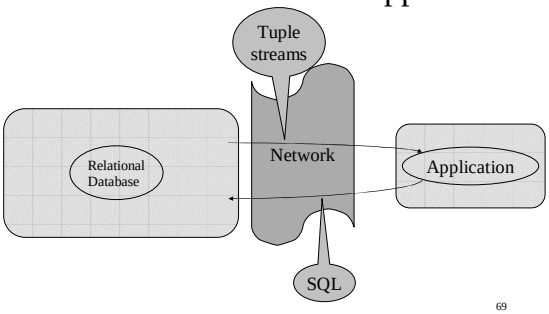
Returns:

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

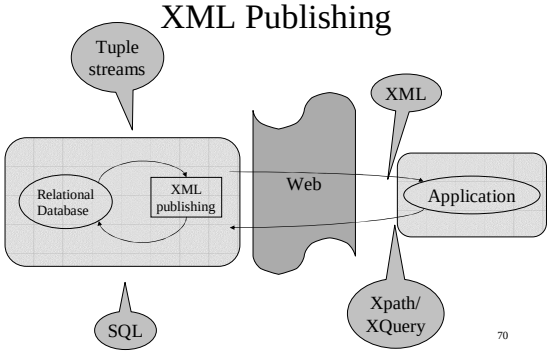

XML from/to Relational Data

- XML publishing:
  - relational data à XML
- XML storage:
  - XML à relational data

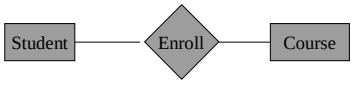
Client/server DB Apps



XML Publishing



XML Publishing



- Relational schema:

  Student(sid, name, address)

  Course(cid, title, room)

  Enroll(sid, cid, grade)

XML Publishing

```
<xmlview>
    <course>   <title> Operating Systems </title>
               <room> MGH084 </room>
               <student> <name> John </name>
                         <address> Seattle </address >
                         <grade> 3.8 </grade>
               </student>
               <student> …</student>
        …
    </course>
    <course>   <title> Database </title>
               <room> EE045 </room>
               <student> <name> Mary </name>
                         <address> Shoreline </address >
                         <grade> 3.9 </grade>
               </student>
               <student> …</student>
        …
    </course>
    …
</xmlview>
```

Group by courses: redundant representation of students

Other representations possible too

XML Publishing

First thing to do: design the DTD:

```
<!ELEMENT xmlview (course*)>
<!ELEMENT course (title, room, student*)>
<!ELEMENT student (name,address,grade)>
<!ELEMENT name    (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT grade   (#PCDATA)>
<!ELEMENT title   (#PCDATA)>
```

73

Now we write an XQuery to export relational data à XML
Note: result is is the right DTD

```
<xmlview>
{ FOR $x IN /db/Course/row
  RETURN
    <course>
      <title>   { $x/title/text() } </title>
      <room> { $x/room/text() } </room>
      { FOR  $y IN /db/Enroll/row[cid/text() = $x/cid/text()]
             $z IN /db/Student/row[sid/text() = $y/sid/text()]
        RETURN <student> <name>    { $z/name/text() } </name>
                         <address> { $z/address/text() } </address>
                         <grade>   { $y/grade/text() } </grade>
               </student>
      }
    </course>
}
</xmlview>
```

74

XML Publishing

Query: find Mary's grade in Operating Systems

XQuery

```
FOR $x IN /xmlview/course[title/text()="Operating Systems"],
      $y IN $x/student/[name/text()="Mary"]
RETURN  <answer> $y/grade/text() </answer>
```

⇩ Can be done automatically

SQL

```
SELECT Enroll.grade
FROM Student, Enroll, Course
WHERE Student.name="Mary" and Course.title="OS"
      and Student.sid = Enroll.sid and Enroll.cid = Course.cid
```

75

XML Publishing

How do we choose the output structure ?

- Determined by agreement with partners/users
- Or dictated by committees
  - XML dialects (called *applications*) = DTDs
- XML Data is often nested, irregular, etc
- No normal forms for XML

76

XML Storage

- Most often the XML data is small
  - E.g. a SOAP message
  - Parsed directly into the application (DOM API)
- Sometimes XML data is large
  - need to store/process it in a database
- The XML storage problem:
  - How do we choose the schema of the database ?

77

XML Storage

Three solutions:

- Schema derived from DTD
- Storing XML as a graph: "Edge relation"
- Store it as a BLOB
  - Simple, boring, inefficient
  - Won't discuss in class

78

Designing a Schema from DTD

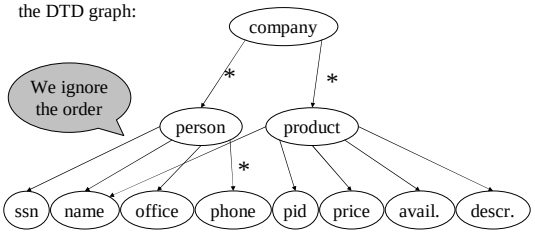
Design a relational schema for:

```
<!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, ((price,availability)|description))>
  <!ELEMENT pid    (#PCDATA)>
  <!ELEMENT description    (#PCDATA)>
]>
```

79

Designing a Schema from DTD

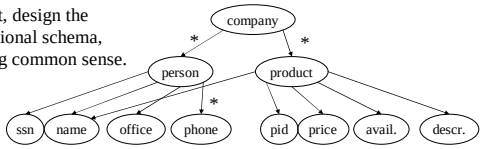
First, construct the DTD graph:



80

Designing a Schema from DTD

Next, design the relational schema, using common sense.



Person(ssn, name, office)
Phone(ssn, phone)
Product(pid, name, price, avail., descr.)

Which attributes may be NULL ? (Look at the DTD)

81

Designing a Schema from DTD

What happens to queries:

```
FOR $x IN /company/product[description]
RETURN  <answer> { $x/name, $x/description } </answer>
```

```
SELECT Product.name, Product.description
FROM    Product
WHERE Product.description IS NOT NULL
```

82

Storing XML as a Graph

Sometimes we don't have a DTD:

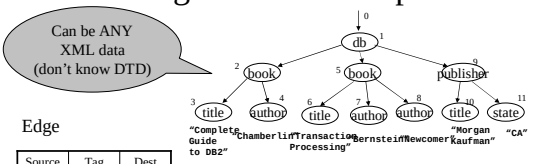
- How can we store the XML data ?

Every XML instance is a *tree*

- Store the edges in an Edge table
- Store the #PCDATA in a Value table

83

Storing XML as a Graph



Edge

Source	Tag	Dest
0	db	1
1	book	2
2	title	3
2	author	4
1	book	5
5	title	6
5	author	7
. . .	. . .	. . .

Value

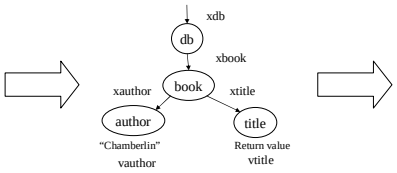
Source	Val
3	Complete guide . . .
4	Chamberlin
6	. . .
. . .	. . .

84

Storing XML as a Graph

What happens to queries:

```
FOR $x IN /db/book[author/text()="Chamberlin"]
RETURN  $x/title
```



Storing XML as a Graph

What happens to queries:

A 6-way join !!!

```
SELECT vtitle.value
FROM    Edge xdb, Edge xbook, Edge xauthor, Edge xtitle,
        Value vauthor, Value vtitle
WHERE xdb.source = 0                    and xdb.tag = 'db'
      and xdb.dest = xbook.source       and xbook.tag = 'book'
      and xbook.dest = xauthor.source   and xauthor.tag = 'author'
      and xbook.dest = xtitle.source    and xtitle.tag = 'title'
      and xauthor.dest = vauthor.source and vauthor.value = 'Chamberlin"
      and xtitle.dest = vtitle.source
```


Storing XML as a Graph

Edge relation summary:

- Same relational schema for every XML document:
  Edge(Source, Tag, Dest)
  Value(Source, Val)
- Generic: works for *every* XML instance
- But inefficient:
  - Repeat tags multiple times
  - Need many joins to reconstruct data

Other XML Topics

- Name spaces
- XML API:
  - DOM = "Document Object Model"
- XML languages:
  - XSLT
- XML Schema
- Xlink, XPointer
- SOAP

Available from www.w3.org
(but don't spend rest of your life reading those standards !)

Old&New XML Research at UW

- Processing:
  - Query languages (XML-QL, a precursor of XQuery)
  - Tukwila
  - XML updates
- XML publishing/storage
  - SilkRoute: silkroute.sourceforge.net
  - STORED
- XML tools
  - XML Compressor: Xmill - a **very** popular tool
  - XML Toolkit (xsort, xagg, xgrep, xtransf, etc): xmltk.sourceforge.net
- Theory:
  - Typechecking
  - Xpath, Xquery containment