Lecture 13: XQuery
XML Publishing, XML Storage
Monday, October 28, 2002

XQuery

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>
```

Question:

Why do we get duplicates?

```
XQuery

Same as before, but eliminate duplicates:

```
FOR $x IN bib/book/title/text() = "Database Theory"/author/text()
  $y IN distinct(bib/book/author/text() = $x)/title/text()
RETURN <answer> { $y } </answer>
```

Result:

```
<answer> abc </answer>
<answer> def </answer>
<answer> ghi </answer>
```

distinct = a function that eliminates duplicates
Need to apply to a collection of text values, not of elements – note how query has changed

FLWR (“Flower”) Expressions

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

XQuery

Find book titles by the coauthors of “Database Theory”:

```
FOR $x IN bib/book/title/text() = "Database Theory"
  $y IN bib/book/author/text() = $x/author/text()
RETURN <answer> { $y/title/text() } </answer>
```

Result:

```
<answer> abc </answer>
<answer> def </answer>
<answer> ghi </answer>
```

Question:

Why do we get duplicates?

```
XQuery

SQL and XQuery Side-by-side

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

Find all products made in Seattle

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

```
FOR $x IN /db/Company/row
  $y IN /db/Company/row
  WHERE $x/maker/text() = $y/cid/text() 
    and $y/city/text() = "Seattle"
RETURN { $x/name }
```

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

Cool XQuery
XQuery: Nesting

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

```xml
FOR $a IN /bib/book[publisher/text()="Morgan Kaufmann"/author]
RETURN $a
```

In the RETURN clause comma concatenates XML fragments.

---

XQuery

Result:

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

---

XQuery

Find books whose price is larger than average:

```
LET $a := avg(/bib/book/price/text())
FOR $b in /bib/book
WHERE $b/price/text() > $a
RETURN { $b }
```

---

XQuery

Find all publishers that published more than 100 books:

```
<big_publishers>
  ( FOR $p IN distinct(/publisher/text())
    WHERE count($b) > 100
    RETURN <publisher> [ $p ] </publisher>
  )
</big_publishers>
```

$\$b$ is a collection of elements, not a single element.

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

---

XQuery

Summary:

- FOR-LET-WHERE-RETURN = FLWR

- FOR/LET Clause
  - List of tuples
- WHERE Clause
  - List of tuples
- RETURN Clause
  - Instance of XQuery data model
FOR v.s. LET

FOR
• Binds node variables $\rightarrow$ iteration

LET
• Binds collection variables $\rightarrow$ one value

Collections in XQuery

• Ordered and unordered collections
  – /bib/book/author/text() = an ordered collection: result is in document order
  – distinct/bib/book/author/text()) = an unordered collection: the output order is implementation dependent

• LET $a := /bib/book$ $\rightarrow$ $a$ is a collection
• Sh/author $\rightarrow$ a collection (several authors...)

XML from/to Relational Data

• XML publishing:
  – relational data $\rightarrow$ XML
• XML storage:
  – XML $\rightarrow$ relational data

The Role of XML Data

• XML is designed for data exchange, not to replace relational or E/R data
• Sources of XML data:
  – Created manually with text editors: not really data
  – Generated automatically from relational data (will discuss next)
  – Text files, replacing older data formats: Web server logs, scientific data (biological, astronomical)
  – Stored/processed in native XML engines: very few applications need that today
XML Publishing

- Exporting the data is easy: we do this already for HTML
- Translating XQuery → SQL is hard

XML publishing systems:
- Research: Experanto (IBM/DB2), SilkRoute (AT&T Labs and UW)
  - XQuery → SQL
- Commercial: SQL Server, Oracle
  - only Xpath → SQL and with restrictions

XML Publishing

First thing to do: design the DTD:

```xml
<xmi-view>
  <course>
    <title> Operating Systems </title>
    <room> MGH04 </room>
    <student> names=John, Jane, Mary </address>
  </student>
  <student> names=John, Jane </address>
  <student> names=John </address>
</course>
</xmi-view>
```

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

```xml
<xmi-view>
  FOR $x IN /db/Course/row
  RETURN
  <course>
    <title> $x/title/text() </title>
    <room> $x/room/text() </room>
    FOR $y IN /db/Enroll/row[ $x/cid/text() = $y/cid/text() ]
    RETURN <student>
      <name> $y/name/text() </name>
      <address> $y/address/text() </address>
      <grade> $y/grade/text() </grade>
    </student>
  </course>
</xmi-view>
```

XML Publishing

Will follow SilkRoute, more or less

```
student  enroll  course
```

- Relational schema:
  Student(sid, name, address)
  Course(cid, title, room)
  Enroll(sid, cid, grade)

XML Publishing

Group by courses: redundant representation of students

Other representations possible too

```
<FOR $x IN /xmi-view/course*>
  <FOR $y IN $x/student*>(name/text() = "Mary")
  RETURN <answer> $y/grade/text() </answer>
</FOR>
```

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
```

SilkRoute does this automatically
XML Publishing

How do we choose the output structure?
• Determined by agreement, with our partners, or dictated by committees
  – XML dialects (called applications) = DTDs
• XML Data is often nested, irregular, etc
• No normal forms for XML

XML Storage

• Often the XML data is small and is parsed directly into the application (DOM API)
• Sometimes it is big, and we need to store it in a database
• The XML storage problem:
  – How do we choose the schema of the database?
• Much harder than XML publishing (why?)

XML Storage

Two solutions:
• Schema derived from DTD
• Storing XML as a graph: “Edge relation”

Designing a Schema from DTD

Design a relational schema for:

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

First, construct the DTD graph:

```
  company
    / | \
   /  |  \     /
person  product
    / | \
   /  |  \     /
    ssn  name  office
          / | \
          /  |  \  *
         phone  pid  price  avail  descr
```

Next, design the relational schema, using common sense.

```
  company
    / | \
   /  |  \     /
person  product
    / | \
   /  |  \     /
     ssn  name  office
          / | \
          /  |  \  *
         phone  pid  price  avail  descr
```

Which attributes may be null?
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

Storing XML as a Graph

- Every XML instance is a tree
- Hence we can store it as any graph, using an Edge table
- In addition we need a Value table to store the data values (#PCDATA)

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

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