# Lecture 13: XQuery
# XML Publishing, XML Storage

Monday, October 28, 2002



## FLWR ("Flower") Expressions

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



## 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>
```



## 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>
```

Question:
Why do we get duplicates ?

Result:

```
<answer> abc </ answer >
< answer > def </ answer >
< answer > abc </ answer >
< answer > ghk </ answer >
```



## 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>
```

distinct = a function
that eliminates duplicates

Need to apply to a collection
of *text* values, not of *elements – note how query has changed*

Result:

```
<answer> abc </ answer >
< answer > def </ answer >
< answer > ghk </ answer >
```



## 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"
```

SQL

```
FOR $x in /db/Product/row
    $y in /db/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 }
```

## XQuery: Nesting

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

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

In the RETURN clause comma concatenates XML fragments



## 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

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

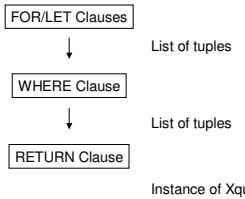
$b is a *collection* of elements, not a single element
count = a (aggregate) function that returns the number of elms



## XQuery

Summary:

- FOR-LET-WHERE-RETURN = FLWR



## FOR v.s. LET

FOR
- Binds *node variables* → iteration

LET
- Binds *collection variables* → one value

13

## 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>
```

14

## 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 → $a is a collection
- $b/author → a collection (several authors...)

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

Returns:

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

15

## 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

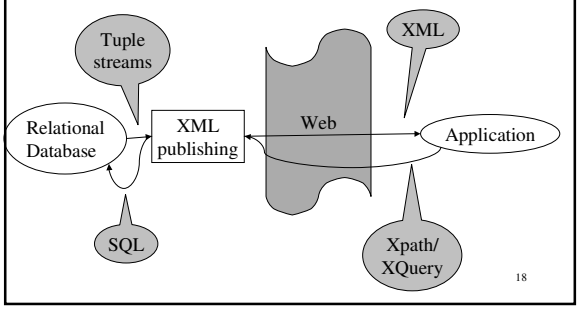
16

## XML from/to Relational Data

- XML publishing:
  - relational data → XML
- XML storage:
  - XML → relational data

17

## XML Publishing




18

# XML Publishing

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

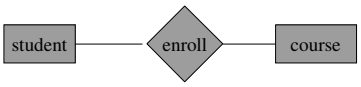
XML publishing systems:

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

19

# XML Publishing

Will follow SilkRoute, more or less




- Relational schema:

  Student(sid, name, address)

  Course(cid, title, room)

  Enroll(sid, cid, grade)

20

# 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

21

# 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)>
```

22

Now we write an XQuery to export relational data → 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()]/row
             $z IN /db/Student/row[sid/text() = $y/sid/text()]/row
        RETURN <student> <name>    { $z/name/text() } </name>
                         <address> { $z/address/text() } </address>
                         <grade>   { $y/grade/text() } </grade>
               </student>
      }
    </course>
}
</xmlview>
```

23

# 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>
```

SilkRoute does this 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
```

24

## 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

25

## 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 ?)

26

## XML Storage

Two solutions:

- Schema derived from DTD
- Storing XML as a graph: "Edge relation"

27

## Designing a Schema from DTD

Design a relational schema for:

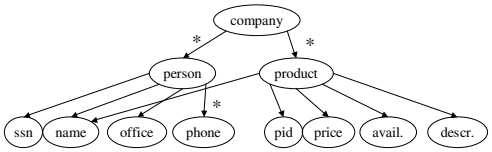
```
<!DOCTYPE company [
  <!ELEMENT company   ((person|product)*)>
  <!ELEMENT person  (ssn, name, office?, phone*)>
  <!ELEMENT ssn       (#PCDATA)>
  <!ELEMENT name    (#PCDATA)>
  <!ELEMENT office    (#PCDATA)>
  <!ELEMENT phone   (#PCDATA)>
  <!ELEMENT product  (pid, name, ((price,availability)|description))>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

28

## Designing a Schema from DTD

First, construct the DTD graph:

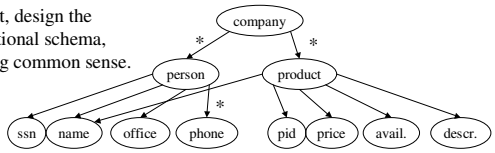
```
<!DOCTYPE company [
  <!ELEMENT company   ((person|product)*)>
  <!ELEMENT person  (ssn, name, office?, phone*)>
  <!ELEMENT ssn       (#PCDATA)>
  <!ELEMENT name    (#PCDATA)>
  <!ELEMENT office    (#PCDATA)>
  <!ELEMENT phone   (#PCDATA)>
  <!ELEMENT product  (pid, name, ((price,availability)|description))>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




29

## Designing a Schema from DTD

Next, design the relational schema, using common sense.




Person(<u>ssn</u>, name, office)
Phone(<u>ssn, phone</u>)
Product(<u>pid</u>, name, price, avail., descr.)

Which attributes may be null ?

30

## 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
```

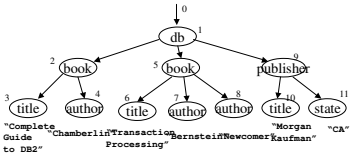
31

## 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)

32

## 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

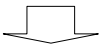
| Source | Val |
|---|---|
| 3 | Complete guide . . . |
| 4 | Chamberlin |
| 6 | . . . |
| . . . | . . . |

33

## Storing XML as a Graph

What happens to queries:

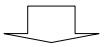
```
FOR $x IN /db/book[author/text()="Chamberlin"]
RETURN  $x/title
```



34

## Storing XML as a Graph

What happens to queries:



```
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 = 'Chamberin 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

36