XQuery: XML from Data to Query

Dr Richi Nayak
School of Information Systems

Admin Details

- Consultation Hours:
  - Monday 1-2 pm
  - Tuesday 4-5 pm
- Email: r.nayak@qut.edu.au
- Assignment 2: Group assignment
  - Releasing in Week 9
  - Based on XQuery and XSLT
  - Using the Saxonb8-9j software

Coverage (Weeks 8 & 9)

- Introduction to XQuery
- XQuery Type systems
- XQuery FLOWR Expressions
  - for, let, order by, where, return
- XQuery Advanced Concepts
- XQuery Aggregates
- SQL and XQuery side-by-side
- XML from/to Relational data
- Data Integration

Introduction to XQuery

- Simple examples
- Basic ideas
- XPath vs. XQuery
- Value expression

XML Example users.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
   <user_tuple>
         <userid>U01</userid>
         <name>Tom Jones</name>
         <rating>B</rating>
   </user_tuple>
   <user_tuple>
         <userid>U02</userid>
         <name>Mary Doe</name>
         <rating>A</rating>
   </user_tuple>
   <user_tuple>
         <userid>U03</userid>
         <name>Dee Linq</name>
         <rating>D</rating>
   </user_tuple>
```

```
<user_tuple>
         <userid>U04</userid>
         <name>Roger Smith</name>
         <rating>C</rating>
</user_tuple>
<user_tuple>
         <userid>U05</userid>
         <name>Jack Sprat</name>
         <rating>B</rating>
</user_tuple>
<user_tuple>
         <userid>U06</userid>
         <name>Rip Van Winkle</name>
         <rating>B</rating>
</user_tuple>
</users>
```


XQuery Example 1

- Query 1: Return users with the rating of 'A' (in a particular format).

```
let $users := doc("users.xml")
<Result>
 {
   for $u in $users/users/user_tuple
   where ($u/rating='A')
   return
     <User>
       {$u/userid,
          $u/name,
          $u/rating}
     </User>
 }
</Result>
```

```
<Result>
    <User>
        <userid>U02</userid>
        <name>Mary Doe</name>
        <rating>A</rating>
    </User>
</Result>
```

XQuery Example 2 – Data Integration

- Query 2: Return the details of the users with the ratings below 'C' but the reserve price more than $1000 (in a particular format).

```
<result>
 {
   for $u in $users//user_tuple,
       $i in $items//item_tuple
   where
     ($u/rating > 'C' and $i/reserve_price > 1000
       and $i/offered_by = $u/userid)
   return
     <warning>
       {$u/name}
       {$u/rating}
       {$i/description}
       {$i/reserve_price}
     </warning>
 }
</result>
```

```
<result>
    <warning>
            <name>Dee Linq</name>
            <rating>D</rating>
            <description> …. </description>
            <reserve-price> 1100 </reserve_price>
    </warning>
</result>
```

7

Basic Ideas Of XQuery

- Value expressions
- Sequences of values
- Use XPath expression to "navigate" on the XML tree, to get values or sequences of values
- Integrate information from multiple sources

8

Why not XPath?

- XPath expressivity insufficient
  - no join queries (as in SQL)
  - no changes to the XML structure possible
  - no quantifiers (as in SQL)
  - no aggregation and functions (as in SQL)
- XQuery extends XPath to a query language that has power similar to SQL.

9

XQuery Overview

- XQuery is an expression language.
  - Every statement evaluates to some result
    - let $x := 5 let $y := 6 return 10*$x+$y
    - Evaluates to 56
- Unlike Relation Algebra, with the relation as the sole datatype, XQuery has a subtle type system.

10

XQuery Type System

- Some node types
- Nodes and expressions
- Operators
  - Comparison, Boolean and Set

11

The XQuery Type System

- *Primitive Types or Atomic values*
  - Number, boolean, strings, dates, times, durations, and XML types
- *Nodes*.
  - Seven kinds.
  - We'll only cover four, on next slide.

12

Some Node Types

1. *Element Nodes*
   - Described by !ELEMENT declarations in DTD's.
2. *Attribute Nodes*
   - described by !ATTLIST declarations in DTD's.
3. *Text Nodes* = #PCDATA.
4. *Document Nodes* represent files.
5. Processing Instruction
6. Comment
7. Function

13

Nodes and Expressions

- Various functions to create or return nodes.
  - Doc function reads an XML file to which a query applies.
  - **Form:** doc("<file name>")
  - **Example:** doc("http://www.lms.qut.edu.au/itb295/bib.xml")
  - We will use doc("bib.xml") throughout, but you must use the complete path to run a XQuery
- Element constructor creates a node in the return.

  <doc><par>Blah Blah</par></doc>

14

Comparison Operators

- Use Fortran comparison operators to compare atomic values only.
  - eq, ne, gt, ge, lt, le.
- Arithmetic operators: +, - , *, div, mod.
  - Apply to any expressions that yield arithmetic or date/time values.

15

Document Order

- Comparison by document order
  - << and >>.
- Example:

$b/book[@year=1994] << $b/book[@year=1992]

is true iff the '1994' attribute appears before the '1992' attribute in the document $b.

16

Boolean Operators

- E_1 and E_2, E_1 or E_2, not(E), if (E_1) then E_2 else E_3 apply to any expressions.
- Example: not(3 eq 5 or 0) has value TRUE.
- Also: true() and false() are functions that return values TRUE and FALSE.

17

Set Operators

- union, intersect, except operate on sequences of nodes.
  - Meanings analogous to SQL.
  - Result eliminates duplicates.
  - Result appears in document order.

18

XQuery FLOWR Expressions

- Example Document “bib.xml”
- Semantics of FLOWR expressions
- FLOWR vs XPath
- For clause
- Let clause
- Attribute retrieval
- Grouping or Result Structuring

19

Example XML Document: bib.xml

```
<bib>
   <book year="1994">
         <title> TCP/IP Illustrated</title>
         <author>
                  <last> Stevens</last>
                  <first>W.</first>
         </author>
         <publisher> Addison-Wesley</publisher>
         <price> 65.95</price>
   </book>
   <book year="1992">
         <title>Advanced Programming in the Unix environment</title>
         <author><last>Stevens</last><first>W.</first></author>
         <publisher>Addison-Wesley</publisher>
         <price>60.95</price>
   </book>
```

20

Example XML Document: bib.xml

```
   <book year="2000">
         <title>Data on the Web</title>
         <author><last>Abiteboul</last><first>Serge</first></author>
         <author><last>Buneman</last><first>Peter</first></author>
         <author><last>Suciu</last><first>Dan</first></author>
         <publisher>Morgan Kaufmann Publishers</publisher>
         <price> 39.95</price>
   </book>
   <book year="1999">
         <title>The Economics of Technology and Content for Digital TV</title>
         <editor> <last>Gerbarg</last><first>Darcy</first>
                  <affiliation>CITI</affiliation>
         </editor>
         <publisher>Kluwer Academic Publishers</publisher>
         <price>129.95</price>
   </book>
</bib>
```

21

A Simple XQuery Example

- Query 3: Return the author names of all books.

```
let $b := doc(“bib.xml")
return <result>{$b/bib/book/author}</result>
```

=

```
<result>
<author><last>Stevens</last><first>W.</first></author>
<author><last>Stevens</last><first>W.</first></author>
<author><last>Abiteboul</last><first>Serge</first></author>
<author><last>Buneman</last><first>Peter</first></author>
<author><last>Suciu</last><first>Dan</first></author>
</result>
```

22

FLWOR Expressions

1. One or more for and/or let clauses.
2. Then optional where and/or order by clauses.
3. A return clause.

23

A FLWOR Query Example

- Query 4: Return the title, price and publisher for each book that is published after 1998. The output should be sorted according to the publishers name.

```
let $d := doc(“bib.xml")/bib
for $b in $d/book
where $b/@year > ‘1998’
order by $b/publisher
return <book>
            {$b/title, $b/price, $b/publisher}
       </book>
```

24

Query 5 Output

```
<?xml version="1.0" encoding="UTF-8"?>
<book>
  <title>The Economics of Technology and Content for Digital TV</title>
  <price>129.95</price>
  <publisher>Kluwer Academic Publishers</publisher>
</book>
<book>
  <title>Data on the Web</title>
  <price> 39.95</price>
  <publisher>Morgan Kaufmann Publishers</publisher>
</book>
```

25

Semantics of FLWOR Expressions

- Each for creates a loop.
  - let produces only a local definition.
- At each iteration of the nested loops, if any, evaluate the where clause.
- If the where clause returns TRUE, invoke the return clause, and append its value to the output.
- The order by clause defines an ordering of the result sequence.
  - With optional ascending or descending.

26

FLWOR vs. XPath expressions

Query 5: Find the book titles published after 1995.

```
for $x in doc("bib.xml")/bib/book
where $x/@year > '1995'
return $x/title
```

Result:

```
<title> Data on the Web </title>
<title> The Economics of Technology and Content for Digital TV </title>
```

27

FLWOR vs. XPath expressions

Equivalently

```
for $x in doc("bib.xml")/bib/book[@year > 1995] /title
return $x
```

And even shorter:

```
doc("bib.xml")/bib/book[@year > 1995] /title
```

28

FOR Clauses

for <variable> in <expression>, . . .

- Variables begin with $.
- A for-variable takes on each item in the sequence denoted by the expression, in turn.
- Whatever follows this for is executed once for each value of the variable.

29

Example: FOR

Query 6: Return the author names of all books.

"Expand the enclosed string by replacing variables and path exps. by their values."

```
for $b in doc("bib.xml") /bib/book/author
return <result>{$b}</result>
```

- $b ranges over the 'author' elements of all books in our example document.

30

Result of Query 7

- Result is a list of tagged authors, like

```
<result><author><last>Stevens</last><first>W.</first></author>
    </result>
<result> <author><last>Stevens</last><first>W.</first></author>
    </result>
<result>
    <author><last>Abiteboul</last><first>Serge</first></author>
    </result>
<result>
    <author><last>Buneman</last><first>Peter</first></author>
    </result>
<result> <author><last>Suciu</last><first>Dan</first></author>
    </result>
```

31

LET Clauses

let <variable> := <expression>, . . .

- Value of the variable becomes the *sequence* of items defined by the expression.
- Note let does not cause iteration; for does.

32

Example: LET

Revisit Query 3

- Is there any difference in result?

33

FOR vs. LET

- FOR $x in expr -- binds $x to each value in the list expr
  - Binds *node variables* → iteration
- LET $x = expr -- binds $x to the entire list expr
  - Binds *collection variables* → one value
  - Useful for common subexpressions and for aggregations

34

FOR vs. LET

```
for $x in doc("bib.xml")/bib/book
return <result> {$x} </result>
```

Returns:

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

```
let $x := doc("bib.xml")/bib/book
return <result> {$x} </result>
```

Returns:

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

35

Attribute Retrieval

Query 7: Return the list of years of book publications.

```
for $x in doc ("bib.xml")/bib/book/@year
return <BookYears> { $x  } </BookYears>
```

- $x ranges over the 'year' attributes of all books in our example document.
- Result is a list of tagged years, like

```
<BookYears year = "1994" />
<BookYears year = "1992" />
<BookYears year = "2000" />
<BookYears year = "1999" />
```

36

XQuery: Advanced Concepts

- A simple example
- Grouping (Result Structuring)
- Nesting (join)
- Sorting (order by)
- Aggregates – count, distinct, avg
- If-then-else
- Functions
- Universal and existential quantifiers

37

Result Structuring

- Query 8: Find the book titles and their corresponding publication year:

```
for $x in doc("bib.xml")/ bib/book
return <answer>
          <title> { $x/title/text() } </title>
          <year> { $x/@year } </year>
       </answer>
```

Braces { } denote evaluation of enclosed expression

38

Query 8 Output

```
<?xml version="1.0" encoding="UTF-8"?>
<answer>
  <title>TCP/IP Illustrated</title>
  <year year="1994"/>
</answer>
<answer>
  <title>Advanced Programming in the Unix environment</title>
  <year year="1992"/>
</answer>
<answer>
  <title>Data on the Web</title>
  <year year="2000"/>
</answer>
<answer>
  <title>The Economics of Technology and Content for Digital TV</title>
  <year year="1999"/>
</answer>
```

39

Result Structuring – text()

- Notice the use of text() in getting the value of title
- What is the result without it ?

```
for $x in doc("bib.xml")/ bib/book
return <answer>
          <title>{ $x/title/text() } </title>
          <year>{ $x/@year } </year>
       </answer>
```

40

Result Structuring - “{“ and “}”

- 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>
```

41

Result Structure - Grouping

Query 9: Return a list of (author, title) pairs

```
for $b in doc("bib.xml")/bib/book,
    $x in $b/title, $y in $b/author
return
  <answer>
       {$y}
       <title> { $x/text()} </title>
  </answer>
```

```
Result:
<answer>
   <author> ..stevens.. </author>
   <title> TCP/IP… </title>
</answer>
<answer>
   <author> ..stevens </author>
   <title> Advanced… </title>
</answer>
…
```

42

Result Structure: Grouping

Query 10: For each author, return all titles of his books.

```
for $b in doc("bib.xml")/bib,
      $x in $b/book/author
return
  <answer>
    { $x }
    {for $y in $b/book[author=$x]/title
      return $y}
  </answer>
```

Required Result:

```
<answer>
   <author> ..stevens.. </author>
    <title> TCP/IP… </title>
    <title> Advanced… </title>
</answer>
<answer>
    <author> ..Abiteboul </author>
    <title> Data… </title>
</answer>
..
```

43

Query 10 Result

```
<answer>
      <author> <last>Stevens</last>     <first>W.</first> </author>
       <title> TCP/IP... </title>
       <title> Advanced... </title>
</answer>
<answer>
      <author> <last>Stevens</last>     <first>W.</first> </author>
       <title> TCP/IP... </title>
       <title> Advanced... </title>
</answer>
<answer>
      <author> ..Abiteboul </author>
   <title> Data... </title>
</answer>
```

? What do you observe in this result?

44

Grouping: Result Structure

Query 11: Eliminating duplicates

```
for $b in doc("bib.xml")/bib,
      $x in distinct-values ($b/book/author)
return
  <answer>
    <author> { $x } </author>
    { for $y in $b/book[author=$x]/title
      return $y }
  </answer>
```

Result:

```
<answer>
   <author> StevensW </author>
    <title> TCP/IP… </title>
    <title> Advanced… </title>
</answer>
<answer>
    <author> AbiteboulSerge </author>
    <title> Data… </title>
</answer>
```

45

XQuery Nesting

Query 12: For each author of a book by Morgan Kaufmann, list the titles of the books the author has published:

```
for $b in doc("bib.xml")/bib,
      $a in $b/book[publisher/text()="Morgan Kaufmann Publishers"]/author
return <result>
            { $a,
              for $t in $b/book[author=$a]/title
              return $t
            }
          </result>
```

In the return clause comma concatenates XML fragments.

Is there an alternate way?

46

XQuery Nesting: Output

Query 12 Result:

```
<result>
   <author>
      <last>Abiteboul</last>
      <first>Serge</first>
   </author>
   <title>Data on the Web</title>
</result>
<result>
   <author>
      <last>Buneman</last>
      <first>Peter</first>
   </author>
   <title>Data on the Web</title>
</result>
<result>
   <author>
      <last>Suciu</last>
      <first>Dan</first>
   </author>
   <title>Data on the Web</title>
</result>
```

47

Sorting in XQuery

- Query 13: List publishers in alphabetical order. No duplicates are allowed.

```
let $x := doc("bib.xml")//publisher
return
<publisher_list>
{
   for $p in distinct-values($x)
   order by  $p
   return <publisher> {$p} </publisher>
}
</publisher_list>
```

48

Query 13 output:

```
<?xml version="1.0" encoding="UTF-8"?>
<publisher_list>
   <publisher> Addison-Wesley   </publisher>
   <publisher> Kluwer Academic Publishers
        </publisher>
   <publisher> Morgan Kaufmann Publishers
        </publisher>
</publisher_list>
```

49

Multiple Sorting in XQuery

- Query 14: List publishers names in alphabetical order. For each publisher, list the book titles in descending order with the price.

```
let $x := doc("bib.xml")//publisher
return
<publisher_list>
{
   for $p in distinct-values ($x)
   order by  $p
   return
      <publisher>
      { <name> {$p} </name>,
            for $b in doc("bib.xml")//book[publisher = $p]
            order by $b/title descending
            return <book> {$b/title} {$b/price} </book>
       }
       </publisher>
}
</publisher_list>
```

50

Query 14 output:

```
<?xml version="1.0" encoding="UTF-8"?>
<publisher_list>
  <publisher>
    <name>Addison-Wesley</name>
    <book>
      <title>TCP/IP Illustrated</title>
      <price> 65.95</price>
    </book>
    <book>
      <title>Advanced Programming in the Unix environment</title>
      <price>60.95</price>
    </book>
  </publisher>
  <publisher>
    <name>Kluwer Academic Publishers</name>
    <book>
      <title>The Economics of Technology and Content for Digital TV</title>
      <price>129.95</price>
    </book>
  </publisher>
  <publisher>
    <name>Morgan Kaufmann Publishers</name>
    <book>
      <title>Data on the Web</title>
      <price> 39.95</price>
    </book>
  </publisher>
</publisher_list>
```

51

Aggregates

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

52

Aggregates: Count

Query 15: Find books with multiple authors (more than 2).

```
for $x in doc("bib.xml")/bib/book
where count($x/author)>2
return $x
```

```
<book year="2000">
        <title>Data on the Web</title>
        <author><last>Abiteboul</last><first>Serge</first></author>
        <author><last>Buneman</last><first>Peter</first></author>
        <author><last>Suciu</last><first>Dan</first></author>
        <publisher>Morgan Kaufmann Publishers</publisher>
        <price> 39.95</price>
</book>
```

53

Aggregates: Count

Query 16: Write the query 15 without the 'where' clause

```
for $x in doc("bib.xml")/bib/book[count(author)>2]
return  $x
```

```
<book year="2000">
        <title>Data on the Web</title>
        <author><last>Abiteboul</last><first>Serge</first></author>
        <author><last>Buneman</last><first>Peter</first></author>
        <author><last>Suciu</last><first>Dan</first></author>
        <publisher>Morgan Kaufmann Publishers</publisher>
        <price> 39.95</price>
</book>
```

54

Aggregates: distinct-values

Query 17: Print authors who have published more than 1 book.

```
let $b := doc("bib.xml")/bib/book
for $a in distinct-values($b/author)
let  $c := $b[author= $a]
where count($c)>1
return <author> { $a } </author>
```

$c is a *collection* of elements, not a single element

55

Aggregates: distinct-values

Query 18: Find publishers that have published more than 1 book.

```
let $x := doc("bib.xml")/bib/book
return
<big_publishers>
{    for $p in distinct-values($x/publisher)
     let  $b := $x[publisher = $p]
     where count($b) > 1
     return <publisher> { $p } </publisher>
}
</big_publishers>
```

$b is a *collection* of elements, not a single element

56

Aggregates: avg

Query 19: Find books with price larger than average:

```
let $b in doc("bib.xml")/bib
let $a:=avg($b/book/price/text())
for $x in $b/book
where $x/price/text() > $a
return $x
```

Use of text() – makes any difference?

57

If-Then-Else

Query 20: Return the titles in a collection along with the details of the editors if the title is a journal otherwise the authors details.

```
for $h in doc("collection.xml")/bib/collection
order by $h/title
return <holding>
                { $h/title,
                if $h/@type = "Journal"
                        then return $h/editor
                else return $h/author
                }
            </holding>
```

58

Functions

```
declare function reverse ($items)
{
        let $cptr := count($items)
                for $i in 0 to $cptr
                return $items[$cptr - $i]
};
```

Reverse(1 to 5)

- Note: (1 to 5) = (1, 2, 3, 4, 5)

59

Existential Quantifiers

Query 21: Return the titles in a book if the book has some paragraphs that contains the words "sailing" and "windsurfing".

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

60

Universal Quantifiers

Query 22: Return the titles in a book if every paragraph of the book contains the words “sailing”.

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

61

SQL and XQuery side-by-side

62

Sample relational schema and its equivalent XML representation

• Relational schema: Product(pid, name, maker, price)
Company(cid, name, city, revenue)

• An equivalent XML representation: db.xml

```
<db>
   <product>
      <row> <pid> ??? </pid>
            <name> ??? </name>
            <maker> ??? </maker>
            <price> ??? </price>
      </row>
      <row> …. </row>
      …
   </product>
   . . . .
</db>
```

63

SQL and XQuery Side-by-side

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

Query 23: 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 doc(“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
```

SQL and XQuery Side-by-side

Product(pid, name, maker, price)

Query 24: Find all product names and prices, and sort them by price.

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

SQL

```
for $x in doc(“db.xml”)/db/Product/row
order by $x/price
return <answer>
          { $x/name}  {$x/price}
       </answer>
```

XQuery

65

XQuery’s Answer

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

Is this a well-formed document? (WHY ???)

66

Producing a Well-Formed Answer

```
<myQuery>
 {      for $x in doc("db.xml")/db/Product/row
        order by $x/price
        return <answer>
                 { $x/name}  {$x/price}
               </answer>
 }
</myQuery>
```


XQuery's Answer

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

Is it well-formed ?

SQL and XQuery Side-by-side

Query 25: 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 doc("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

Query 26: 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 doc("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>
```


Summary of comparison

- If-then-else
- Universal and existential quantifiers
- Sorting
- Before and After
  - for dealing with order in the input
- Filter
  - deletes some edges in the result tree
- Recursive functions

XML from/to Relational data

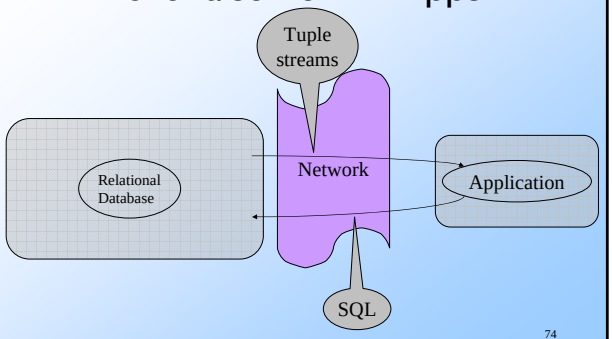
- XML publishing from relational data to its proprietary XML format

XML from/to Relational Data

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

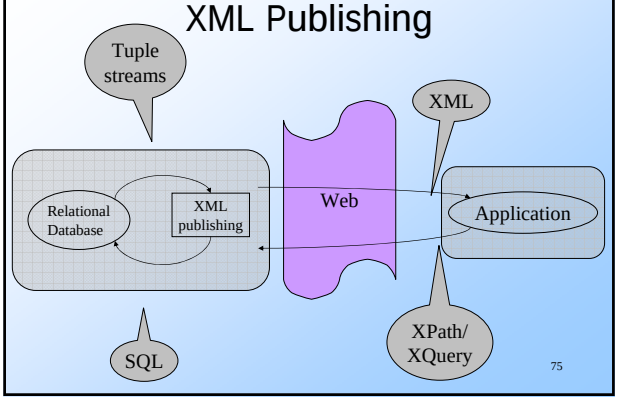
73

Client/server DB Apps



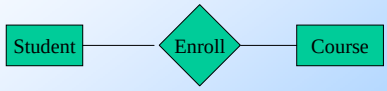
74

XML Publishing



75

XML Publishing



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

76

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

77

XML Publishing

First thing to do: design the DTD or XSD

```
<!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)>
```

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

```
<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>
```

79

XML Publishing

Query 27: 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
```

80

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

81

Data Integration

- Integration of two data sources
  - Union and Grouping
  - Join and Nesting

82

Integration of Two Document/Database Sources

- You are given two databases containing information about movies.
- The database A consists of Movie elements, with subelements Title (unique), Director (unique), and Actor (one or more).
- The database B consists of Actor elements, with subelements Movie that have attributes Title and Director.
- You define the schemas of A and B using XML Schema.
  - There is a requirement that different movies have different titles.
- Define in XQuery the mapping from B to A. Make sure of the following:
  - You don't end up with duplicate movies.
  - The set of actors for a movie is the union of the corresponding sets from A and B (duplicates eliminated).

83

Reference: Dr. Michalis Petropoulos- Data Integration Course notes

sourceA.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="sourceA">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="movie" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="title" type="xs:string"/>
              <xs:element name="director" type="xs:string"/>
              <xs:element name="actors">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="actor" maxOccurs="unbounded"
type="xs:string"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

84

sourceB.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="sourceB">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="actor" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="name" type="xs:string"/>
              <xs:element name="movies">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="movie" maxOccurs="unbounded">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name="title" type="xs:string"/>
                          <xs:element name="director" type="xs:string"/>
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

85

integrated.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="sourceA">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="movie" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="title" type="xs:string"/>
              <xs:element name="director" type="xs:string"/>
              <xs:element name="actors">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="actor" maxOccurs="unbounded"
                                type="xs:string"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

86

sourceA.xml

```
<?xml version="1.0"?>
<sourceA xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="sourceA.xsd">
  <movie>
    <title>MovieA</title>
    <director>DirA</director>
    <actors>
      <actor>ActA</actor>
      <actor>ActB</actor>
    </actors>
  </movie>
  <movie>
    <title>MovieB</title>
    <director>DirB</director>
    <actors>
      <actor>ActB</actor>
      <actor>ActC</actor>
    </actors>
  </movie>
</sourceA>
```

87

sourceB.xml

```
<?xml version="1.0"?>
<sourceB xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="sourceB.xsd">
  <actor>
    <name>ActA</name>
    <movies>
      <movie> <title>MovieA</title> <director>DirA</director> </movie>
      <movie> <title>MovieC</title> <director>DirA</director> </movie>
    </movies>
  </actor>
  <actor>
    <name>ActC</name>
    <movies>
      <movie><title>MovieA</title><director>DirA</director></movie>
    </movies>
  </actor>
  <actor>
    <name>ActD</name>
    <movies>
      <movie> <title>MovieC</title> <director>DirA</director></movie>
    </movies>
  </actor>
</sourceB>
```

88

Attempt 1: Union, Restructuring

```
<integrated xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="integrated.xsd">
{
    let $A := doc("sourceA.xml")/sourceA
    return $A/*
union
    (
    let $B := doc("sourceB.xml")/sourceB
    for $titleB in distinct-values($B//title)
    let $dirB := $B//movie[title=$titleB]/director
    return
        <movie>
            <title>{$titleB}</title>
            {$dirB[1]}
            <actors> {
                for $actB in $B/actor
                where $actB//movie/title = $titleB
                return <actor>{data($actB/name)}</actor>
            } </actors>
        </movie>
    )
} </integrated>
```

89

```
<?xml version="1.0" encoding="UTF-8"?>
<integrated xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="integrated.xsd">
  <movie>
    <title>MovieA</title>
    <director>DirA</director>
    <actors>
      <actor>ActA</actor>
      <actor>ActB</actor>
    </actors>
  </movie>
  <movie>
    <title>MovieB</title>
    <director>DirB</director>
    <actors>
      <actor>ActB</actor>
      <actor>ActC</actor>
    </actors>
  </movie>
  <movie>
    <title>MovieA</title>
    <director>DirA</director>
    <actors>
      <actor>ActA</actor>
      <actor>ActC</actor>
    </actors>
  </movie>
  <movie>
    <title>MovieC</title>
    <director>DirA</director>
    <actors>
      <actor>ActA</actor>
      <actor>ActD</actor>
    </actors>
  </movie>
</integrated>
```

90

Attempt 2: Duplicate Elimination, Join

```
<integrated xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="integrated.xsd"> {
      let $A := doc("sourceA.xml")/sourceA
      return $A/*
union
      (
      let $B := doc("sourceB.xml")/sourceB
      for $titleB in distinct-values($B//title)
      let $dirB := $B//movie[title=$titleB]/director
      where not(exists(
                  for $A in doc("sourceA.xml")/sourceA
                  for $titleA in $A//title
                  where $titleA = $titleB
                  return $titleA
      ))
      return
                  <movie>
                              <title>{$titleB}</title>
                              {$dirB[1]}
                              <actors> {
                                          for $actB in $B/actor
                                          where $actB//movie/title = $titleB
                                          return <actor>{data($actB/name)}</actor>
                              } </actors>
                  </movie>
      )
} </integrated>
```



```
<?xml version="1.0" encoding="UTF-8"?>
<integrated xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="integrated.xsd">
  <movie>
    <title>MovieA</title>
    <director>DirA</director>
    <actors>
      <actor>ActA</actor>
      <actor>ActB</actor>
    </actors>
  </movie>
  <movie>
    <title>MovieB</title>
    <director>DirB</director>
    <actors>
      <actor>ActB</actor>
      <actor>ActC</actor>
    </actors>
  </movie>
  <movie>
    <title>MovieC</title>
    <director>DirA</director>
    <actors>
      <actor>ActA</actor>
      <actor>ActD</actor>
    </actors>
  </movie>
</integrated>
```


Attempt 3: Nested and Join

```
<integrated xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="integrated.xsd"> {
      ( for $movie in doc("sourceA.xml")/sourceA/movie
        return
                  <movie>
                              {$movie/title}
                              {$movie/director}
                              <actors> {   (   for $actA in $movie/actors/actor return $actA )
                                          union
                                          (  let $B := doc("sourceB.xml")/sourceB
                                             for $actB in $B/actor
                                             where $actB//title = $movie/title and not(exists(
                                                                    for $actA in $movie/actors/actor
                                                                    where $actA  = $actB/name
                                                                    return $actA
                                                         ))
                                             return <actor>{data($actB/name)}</actor>)
                              } </actors>
                  </movie>
      )
union
      ( let $B := doc("sourceB.xml")/sourceB
        for $titleB in distinct-values($B//title)
        let $dirB := $B//movie[title=$titleB]/director
        where not(exists(
                  for $A in doc("sourceA.xml")/sourceA
                  for $titleA in $A//title
                  where $titleA = $titleB
                  return $titleA
      ))
      return
                  <movie>
                              <title>{$titleB}</title>
                              {$dirB[1]}
                              <actors> {
                                          for $actB in $B/actor
                                          where $actB//movie/title = $titleB
                                          return <actor>{data($actB/name)}</actor>
                              } </actors>
                  </movie>
      )
} </integrated>
```


Attempt 3: Nested and Join (1)

```
<integrated xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="integrated.xsd"> {
    ( for $movie in doc("sourceA.xml")/sourceA/movie
      return
          <movie>
                {$movie/title}
                {$movie/director}
                <actors> {   (for $actA in $movie/actors/actor return $actA )
                          union
                          (    let $B := doc("sourceB.xml")/sourceB
                          for $actB in $B/actor
                          where $actB//title = $movie/title and not(exists(
                                    for $actA in $movie/actors/actor
                                    where $actA = $actB/name
                                    return $actA
                                ))
                          return <actor>{data($actB/name)}</actor>)
                } </actors>
          </movie>
    )
union
```


Attempt 3: Nested and Join (2)

```
    ( let $B := doc("sourceB.xml")/sourceB
      for $titleB in distinct-values($B//title)
      let $dirB := $B//movie[title=$titleB]/director
      where not(exists(
            for $A in doc("sourceA.xml")/sourceA
            for $titleA in $A//title
            where $titleA = $titleB
            return $titleA
    ))
    return
          <movie>
                <title>{$titleB}</title>
                {$dirB[1]}
                <actors> {
                      for $actB in $B/actor
                      where $actB//movie/title = $titleB
                      return <actor>{data($actB/name)}</actor>
                } </actors>
          </movie>
    )
} </integrated>
```

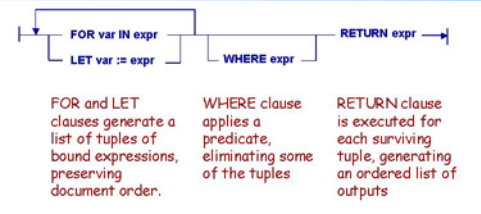


```
<?xml version="1.0" encoding="UTF-8"?>
<integrated xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="integrated.xsd">
  <movie>
    <title>MovieA</title>
    <director>DirA</director>
    <actors>
      <actor>ActA</actor>
      <actor>ActB</actor>
      <actor>ActC</actor>
    </actors>
  </movie>
  <movie>
    <title>MovieB</title>
    <director>DirB</director>
    <actors>
      <actor>ActB</actor>
      <actor>ActC</actor>
    </actors>
  </movie>
  <movie>
    <title>MovieC</title>
    <director>DirA</director>
    <actors>
      <actor>ActA</actor>
      <actor>ActD</actor>
    </actors>
  </movie>
</integrated>
```

Summary: XQuery

Summary:

- FOR-LET-WHERE-RETURN = FLWR



97

Tutorial and Practicals

- Week 8 – Practical
  - Making you familiarise with Saxon that will be used in completing assignment 2 tasks
- Week 9 – Tutorial on XQuery
  - Question 1 to 4.
- Week 10 – Tutorial on XQuery continues..
  - Questions 5 to 8.

98

Reference or Reading Material

- There are numerous online resources of XQuery.
- Official XQuery Web site:
  http://www.w3.org/TR/xquery/

  A complete reference of all the operators, built-in functions, and data types in XQuery 1.0
- http://www.w3schools.com/xquery/default.asp

  An easy and concise tutorial of various features of XQuery.

99