CS/INFO 330
XQuery

Mirek Riedewald
mirek@cs.cornell.edu

(Based on slides by Dan Suciu)

Announcements

- Brief web service HW discussion on Friday
- More detailed requirements for midterm demos and design document will be out soon
  - Just a longer version of what I announced in class last week

Recall: XPath

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…

XQuery

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

FLWR (“Flower”) Expressions

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

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>
```

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 (abstracted version):

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


FOR-WHERE-RETURN

Equivalently (perhaps more geek-ish)

```
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
```


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 !

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>
    …
```


FOR-WHERE-RETURN

- Notice the use of "{" and "}" in previous example
- 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>
```


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

XQuery

Result:

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


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

Aggregates

Same thing:

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

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>
```


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

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 ???)

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>
```


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 !

SQL and XQuery Side-by-side

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

Find all products made in Ithaca

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

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() = "Ithaca"
RETURN { $x/name }
```

XQuery

Cool XQuery

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

Result

```
<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 vs. LET

FOR
- Binds *node variables* → iteration

LET
- Binds *collection variables* → one value

FOR vs. 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>
```


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

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

Returns:

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

XQuery

Summary:

- FOR-LET-WHERE-RETURN = FLWR