XQuery

CS 445

Fall 2008

Some slide content courtesy of Ramakrishnan & Gehrke, Dan Suciu, Zack Ives.

Querying XML Data

- XPath = simple navigation through the tree
- XQuery = the SQL of XML

Query Language and Data Model

- A query language is "closed" w.r.t. its data model if input and output of a query conform to the model
- SQL
  - Set of tuples in, set of tuples out
- XPath 1.0
  - A tree of nodes (well-formed XML) in, a **node set** out.
- XQuery 1.0
  - Sequence of items in, sequence of items out
- Compositionality of a language
  - Output of Query 1 can be used as input to Query 2

XQuery Values

- Item = node or atomic value.
- Value = ordered sequence of zero or more items.
- Examples:
  - () = empty sequence.
  - (“Hello”, “World”)
  - (“Hello”, <PRICE>2.50</PRICE>, 10)

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>
```

Document Nodes

- Form:
  - doc("<file name>").
- Establishes a document to which a query applies.
- Example:
  - doc("/courses/445/bib.xml")

FLWOR expressions

- FLOWR is a high-level construct that
  - supports iteration and binding of variables to intermediate results
  - is useful for joins and restructuring data
- Syntax: For-Let-Where-Order by-Return

```
for $x in expression1                          /* similar to FROM in SQL */
[let $y := expression2 ]                       /* no analogy in SQL */
[where expression3 ]                           /* similar to WHERE in SQL */
[order by  expression4 (ascending|descending)? ]
                                               /* similar to ORDER-BY in SQL */
return expression4                             /* similar to SELECT in SQL */
```

Example FLOWR Expression

```
for $x in doc("bib.xml")/bib/book        // iterate, bind each item to $x
let $y := $x/author                      // no iteration, bind a sequence to $y
where $x/title="XML"                     // filter each tuple ($x, $y)
order by $x/@year descending             // order tuples
return count($y)                         // one result per surviving tuple
```

- The `for` clause iterates over all books in an input document, binding $x to ***each*** book in turn.
- For each binding of $x, the `let` clause binds $y to ***all*** authors of this book.
- The result of `for` and `let` clauses is a tuple stream in which each tuple contains a pair of bindings for $x and $y, i.e. ($x, $y).
- The `where` clause filters each tuple ($x, $y) by checking predicates.
- The `order by` clause orders surviving tuples.
- The `return` clause returns the count of $y for each surviving tuple.

FOR-WHERE-RETURN

Find all book titles published after 1995:

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

Result:

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

FOR-WHERE-RETURN

Equivalently (perhaps more geekish)

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

And even shorter:

```
doc("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 doc("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 "}"
- What is the result without them ?

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

More Examples of WHERE

- Selections

```
for $b in doc("bib.xml")/bib/book
where $b/publisher = “Addison Wesley" and
      $b/@year = "1998"
return $b/title

for $b in doc("bib.xml")/bib/book
where empty($b/author)
return $b/title

for $b in doc("bib.xml")/bib/book
where count($b/author) = 1
return $b/title
```

Aggregates over a sequence: count, avg, sum, min, max

Aggregates

Find all books with more than 3 authors:

```
for $x in doc("bib.xml")/bib/book
where count($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 doc("bib.xml")/bib/book[count(author)>3]
RETURN  $x
```

FOR v.s. LET

FOR

- Binds *node variables* → iteration

LET

- Binds *collection variables* → 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>
```

FOR-WHERE-RETURN

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

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

Answer:

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

XQuery: Nesting

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

```
for $b in doc("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>
```

Getting Distinct Values from FOR

- Distinct values: the *fn:distinct-values* function eliminates duplicates in a sequence *by value*
  - The for expression evaluates to a sequence of nodes
    - *fn:distinct-values* converts it to a sequence of atomic values and removes duplicates

```
for $a in distinct-values(doc(“bib.xml”)/book/author)
return   <author-name> {$a} </author-name>
```

versus

```
for $a in doc(“bib.xml”)/book/author
return $a
```

Value Comparison

```
<author>
  <first>Jeffery</first>
  <last>Ullman</last>
</author>
```

- Value comparison *"eq"*: compares single values
- "eq" applies *atomization* (*fn:data( )*) to each operand
  - Given a sequence of nodes, *fn:data( )* returns an *atomic value* for each node which consists of:
    - a <u>string value</u>, i.e., the concatenation of the string values of ***all*** its *Text Node descendants* in *document order*
    - a <u>type</u>, e.g., xdt:untypedAtomic
  - For each operand, "eq" uses the fn:data() result if it evaluates to a singleton sequence, o.w. runtime error.

✓
```
for $a in doc("bib.xml")/bib/book/author
where $a eq "JefferyUllman"
return $a/..
```

✗
```
for $b in doc("bib.xml")/bib/book
where $b/author eq "JefferyUllman"
return $b/author
```

General Comparison

- General comparison operators (=, !=, <, >, <=, >=): *existentially* quantified comparisons, applied to *operand sequences of any length*
- Atomization (fn:data()) is applied to each operand to get a sequence of atomic values
- Comparison is true if one value from a sequence satisfies the comparison

```
for $b in doc(“bib.xml”)/bib/book
where $b/author = “JefferyUllman”
return $b/author
```

String Operations

- Functions for string matching

  ```
  fn:contains(xs:string, xs:string)
  fn:starts(ends)-with(xs:string, xs:string
  fn:substring-before(after)(xs:string, xs:string)
  ```

  …

- Again, atomization (fn:data()) is applied to each function parameter to get an atomic value.

```
for $a in doc("bib.xml")//author
where contains($a, "Ullman")
return $a
```

```
<author>
  <name>Jeffery Ullman</name>
</author>
```

```
<author>
  <first>Jeffery</first>
  <last>Ullman</last>
</author>
```

Element Construction

```
<bib>
{ for $b in doc("bib.xml")/bib/book
  where $b/publisher = "Addison-Wesley"
        and $b/@year > 1991
  return <book year="{ $b/@year }">
             { $b/title } </book>
}
</bib>
```