XML Technologies and Applications

Rajshekhar Sunderraman

Department of Computer Science
Georgia State University
Atlanta, GA 30302
raj@cs.gsu.edu

V (c). XML Querying: XQuery

December 2005
Outline

- Introduction
- XML Basics
- XML Structural Constraint Specification
  - Document Type Definitions (DTDs)
  - XML Schema
- XML/Database Mappings
- XML Parsing APIs
  - Simple API for XML (SAX)
  - Document Object Model (DOM)
- XML Querying and Transformation
  - XPath
  - XSLT
  - XQuery
- XML Applications
XQuery – XML Query Language

- Integrates XPath with earlier proposed query languages: XQL, XML-QL
- SQL-style, not functional-style
- Much easier to use as a query language than XSLT
- Can do pretty much the same things as XSLT and more, but typically easier
- 2004: XQuery 1.0
<Transcripts>

<Transcript>
  <Student StudId="111111111" Name="John Doe"/>
  <CrsTaken CrsCode="CS308" Semester="F1997" Grade="B"/>
  <CrsTaken CrsCode="MAT123" Semester="F1997" Grade="B"/>
  <CrsTaken CrsCode="EE101" Semester="F1997" Grade="A"/>
  <CrsTaken CrsCode="CS305" Semester="F1995" Grade="A"/>
</Transcript>

<Transcript>
  <Student StudId="987654321" Name="Bart Simpson"/>
  <CrsTaken CrsCode="CS305" Semester="F1995" Grade="C"/>
  <CrsTaken CrsCode="CS308" Semester="F1994" Grade="B"/>
</Transcript>

... cont'd ...
transcript.xml (cont’d)

<Transcript>
  <Student StudId="123454321" Name="Joe Blow" />
  <CrsTaken CrsCode="CS315" Semester="S1997" Grade="A" />
  <CrsTaken CrsCode="CS305" Semester="S1996" Grade="A" />
  <CrsTaken CrsCode="MAT123" Semester="S1996" Grade="C" />
</Transcript>

<Transcript>
  <Student StudId="023456789" Name="Homer Simpson" />
  <CrsTaken CrsCode="EE101" Semester="F1995" Grade="B" />
  <CrsTaken CrsCode="CS305" Semester="S1996" Grade="A" />
</Transcript>

</Transcripts>
XQuery Basics

• General structure (FLWR expressions):

```
FOR variable declarations
LET variable := expression,
    variable := expression, ...
WHERE condition
RETURN document
```

• Example:

```
(: students who took MAT123 :) 
FOR $t$ IN doc("http://xyz.edu/transcript.xml")//Transcript
WHERE $t$/CrsTaken/@CrsCode = “MAT123”
RETURN $t$/Student
```

• Result:

```
<Student StudId=”111111111” Name=”John Doe” />
<Student StudId=”123454321” Name=”Joe Blow” />
```
XQuery Basics (cont’d)

• Previous query doesn’t produce a well-formed XML document; the following does:

```xml
<StudentList>
  { 
    FOR $t IN doc("transcript.xml")//Transcript
    WHERE $t/CrsTaken/@CrsCode = "MAT123"
    RETURN $t/Student
  }
</StudentList>
```

• FOR binds $t to Transcript elements one by one, filters using WHERE, then places Student-children as e-children of StudentList using RETURN
FOR vs LET

**FOR** $x$ IN doc("transcript.xml")
**RETURN** <result> { $x } </result>

For: iteration

Returns:

<result> <transcript>...</transcript></result>
<result> <transcript>...</transcript></result>
<result> <transcript>...</transcript></result> ...

**LET** $x$ := doc("transcript.xml")
**LET** $x$ := doc("transcript.xml")
**RETURN** <result> { $x } </result>

Let: set value is assigned to variable.

Returns:

<result>
  <transcript>...</transcript>
  <transcript>...</transcript>
  <transcript>...</transcript>
  ...
</result>
Reconstruct lists of students taking each class using the Transcript records:

```xml
FOR $c$ IN distinct values(doc("transcript.xml")//CrsTaken)
RETURN
<ClassRoster CrsCode={$c/@CrsCode} Semester={$c/@Semester}>
{
    FOR $t$ IN doc("transcript.xml")//Transcript
    WHERE $t/CrsTaken/[@CrsCode = $c/@CrsCode and @Semester = $c/@Semester]
    RETURN $t/Student
ORDER BY $t/Student/@StudId
}
</ClassRoster>
ORDER BY $c/@CrsCode
```

Query inside
RETURN – similar to query inside
SELECT in OQL
Document Restructuring (cont’d)

• Output elements have the form:

```xml
<ClassRoster CrsCode="CS305" Semester="F1995">
    <Student StudId="111111111" Name="John Doe"/>
    <Student StudId="987654321" Name="Bart Simpson"/>
</ClassRoster>
```

• **Problem:** the above element will be output *twice* – for each of the following two bindings of $c$:

```xml
<CrsTaken CrsCode="CS305" Semester="F1995" Grade="C"/>
<CrsTaken CrsCode="CS305" Semester="F1995" Grade="A"/>
```

*Note:* grades are different – distinct-values( ) won’t eliminate transcript records that refer to same class!
Document Restructuring (cont’d)

• *Solution*: instead of

\[
\text{FOR } \$c \text{ IN distinct-values(doc(“transcript.xml”)//CrsTaken)}
\]

\[
\text{use}
\]

\[
\text{FOR } \$c \text{ IN doc(“classes.xml”)//Class}
\]

where *classes.xml* lists course offerings (course code/semester) *explicitly* (no need to extract them from transcript records) – shown on next slide

Then $c$ is bound to each class exactly once, so each class roster will be output exactly once
<Classes>
  <Class  CrsCode="CS308"  Semester="F1997" >
    <CrsName>SE</CrsName>  <Instructor>Adrian Jones</Instructor>
  </Class>
  <Class  CrsCode="EE101"  Semester="F1995" >
    <CrsName>Circuits</CrsName>  <Instructor>David Jones</Instructor>
  </Class>
  <Class  CrsCode="CS305"  Semester="F1995" >
    <CrsName>Databases</CrsName>  <Instructor>Mary Doe</Instructor>
  </Class>
  <Class  CrsCode="CS315"  Semester="S1997" >
    <CrsName>TP</CrsName>  <Instructor>John Smyth</Instructor>
  </Class>
  <Class  CrsCode="MAR123"  Semester="F1997" >
    <CrsName>Algebra</CrsName>  <Instructor>Ann White</Instructor>
  </Class>
</Classes>
Document Restructuring (cont’d)

• More problems: the above query will list classes with no students. Reformulation that avoids this:

```xml
FOR $c IN doc("classes.xml")//Class
WHERE
    doc("transcripts.xml")//CrsTaken[@CrsCode = $c/@CrsCode and @Semester = $c/@Semester]
RETURN
<ClassRoster CrsCode={$c/@CrsCode} Semester={$c/@Semester}>
    {
        FOR $t IN doc("transcript.xml")//Transcript
            WHERE $t/CrsTaken[@CrsCode = $c/@CrsCode and @Semester = $c/@Semester]
                RETURN $t/Student
            ORDER BY $t/Student/@StudId
    }
</ClassRoster>
ORDER BY $c/@CrsCode
```

Test that classes aren’t empty
XQuery Semantics

- So far the discussion was informal
- XQuery *semantics* defines what the expected result of a query is
- Defined analogously to the semantics of SQL
XQuery Semantics (cont’d)

• **Step 1:** Produce a list of bindings for variables

  - The FOR clause binds each variable to a *list* of nodes specified by an XQuery expression.
    The expression can be:
    • An XPath expression
    • An XQuery query
    • A function that returns a list of nodes

  - End result of a FOR clause:
    • Ordered list of tuples of document nodes
    • Each tuple is a binding for the variables in the FOR clause
Example (bindings):
- Let FOR declare $A$ and $B$
- Bind $A$ to document nodes \{v,w\}; $B$ to \{x,y,z\}
- Then FOR clause produces the following list of bindings for $A$ and $B$:
  - $A/v$, $B/x$
  - $A/v$, $B/y$
  - $A/v$, $B/z$
  - $A/w$, $B/x$
  - $A/w$, $B/y$
  - $A/w$, $B/z$
XQuery Semantics (cont’d)

• *Step 2:* filter the bindings via the WHERE clause
  
  - Use each tuple binding to substitute its components for variables; retain those bindings that make WHERE true

  - Example: \( \text{WHERE } $A/CrsTaken/@CrsCode = $B/Class/@CrsCode \)

  • Binding: \( $A/w, \text{ where } w = \langle \text{CrsTaken CrsCode="CS308" …} / \rangle \)
    \( $B/x, \text{ where } x = \langle \text{Class CrsCode="CS308" …} / \rangle \)

  • Then \( w/CrsTaken/@CrsCode = x/Class/@CrsCode, \) so the WHERE condition is satisfied & binding retained
XQuery Semantics (cont’d)

• *Step 3.* Construct result
  
  – For each retained tuple of bindings, instantiate the RETURN clause

  – This creates a fragment of the output document

  – Do this for each retained tuple of bindings in sequence
Grouping and Aggregation

• Does not use separate grouping operator
  
  – OQL does not need one either (XML data model is object-oriented and hence similarities with OQL)
  
  – Subqueries inside the RETURN clause obviate this need (like subqueries inside SELECT did so in OQL)

• Uses built-in aggregate functions count, avg, sum, etc. (some borrowed from XPath)
Aggregation Example

- Produce a list of students along with the number of courses each student took:

  ```xml
  FOR $t IN fn:doc("transcripts.xml")//Transcript,
  $s IN $t/Student
  LET $c := $t/CrsTaken
  RETURN
  <StudentSummary
    StudId = {$s/@StudId}
    Name = {$s/@Name}
    TotalCourses = {fn:count(fn:distinct-values($c))} />
  ORDER BY StudentSummary/@TotalCourses
  ```

- The *grouping effect* is achieved because $c is bound to a new set of nodes for *each* binding of $t
Quantification in XQuery

• XQuery supports explicit quantification:
  - SOME (∃) and EVERY (∀)

• Example: Find students who have taken MAT123.

  FOR $t$ IN fn:doc(“transcript.xml”)//Transcript
  WHERE SOME $ct$ IN $t$/CrsTaken SATISFIES $ct$/@CrsCode = “MAT123”
  RETURN $t$/Student
Quantification (cont’d)

- Retrieve all classes (from classes.xml) where each student took the class.

```xml
FOR $c IN fn:doc(classes.xml)//Class
LET $g := {
  (: Transcript records that correspond to class $c :) 
  FOR $t IN fn:doc("transcript.xml")//Transcript
    WHERE $t/CrsTaken/@Semester = $c/@Semester AND 
      $t/CrsTaken/@CrsCode = $c/@CrsCode
    RETURN $t
}
$h := { FOR $s in fn:doc("transcript.xml")//Transcript
    RETURN $s } (: all transcript records :) 
WHERE EVERY $tr IN $h SATISFIES
  $tr IN $g
RETURN $c ORDER BY $c/@CrsCode
```
XQuery: Summary

FOR-LET-WHERE-RETURN  = FLWR

FOR/LET Clauses

List of tuples

WHERE Clause

List of tuples

RETURN Clause

Instance of Xquery data model