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, but typically easier
- XQuery 1.0 standard late in 2002

XQuery Basics

- General structure:

  FOR *variable declarations*
  WHERE *condition*
  RETURN *document*

  FLWR expression

- Example:

```
FOR  $t  IN  document("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" />
```

This document on next slide

transcript.xml

```
<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 (cont'd)

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

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

FLWR inside XML

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

104

Document Restructuring with XQuery

- *Reconstruct lists of students taking each class using the* Transcript *records*:

```
FOR  $c  IN  distinct(document("transcript.xml")/CrsTaken)
RETURN
   <ClassRoster   CrsCode = {$c/@CrsCode}   Semester = {$c/@Semester}>
   {
    FOR  $t  IN  document("transcript.xml")/Transcript
    WHERE   $t/CrsTaken/@CrsCode = $c/CrsCode
              AND  $t/CrsTaken/@Semester = $c/@Semester
    RETURN   $t/Student
                 SORTBY ($t/Student/@StudId)
   }
   </ClassRoster>
   SORTBY ($c/@CrsCode)
```

FLWR inside RETURN – similar to query inside SELECT in OQL

105

Document Restructuring (cont'd)

- *Output elements have the form*:

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

- *Problem*: the above element <u>will be output *twice*</u> – once when $c is bound to

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

and once when it is bound to

Bart Simpson's

John Doe's

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

Note: grades are different – distinct( ) won't eliminate transcript records that refer to same class!

106

Document Restructuring (cont'd)

- *Solution*: instead of

```
FOR  $c  IN  distinct(document("transcript.xml")/CrsTaken)
```

use

Document on next slide

```
FOR  $c  IN document("classes.xml")/CrsTaken
```

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

Then $c is bound to each class exactly once, so each class roster will be output exactly once

107

http://xyz.edu/classes.xml

```
<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:

Test that classes aren't empty

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

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

110

XQuery Semantics (cont'd)

- *Step 1*: Produce a list of bindings for variables
  - The FOR clause binds each variable to an *ordered* 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

111

XQuery Semantics (cont'd)

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

112

XQuery Semantics (cont'd)

- *Step 2*: filter the bindings via the WHERE clause
  - Use each tuple-binding to substitute its conponents for variables; retain those bindings that make WHERE true
  - Example: WHERE $A/CrsTaken/@CrsCode = $B/Class/@CrsCode
    - Binding: $A/w, where w = <CrsTaken CrsCode="CS308" …/>
      $B/x, where x = <Class CrsCode="CS308" … />
    - Then w/CrsTaken/@CrsCode = x/Class/@CrsCode, so the WHERE condition is satisfied & binding retained

113

XQuery Semantics (cont'd)

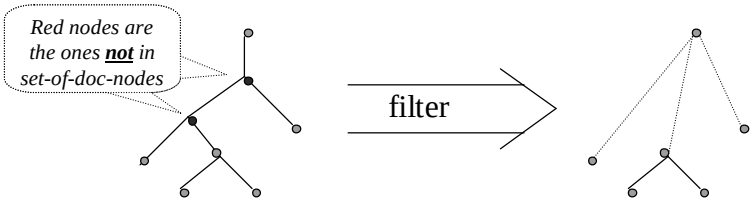
- *Step 3*: Construct result
  - For each retained tuple of bindings, instantiate the RETURN clause
  - This creates a fragment of the output document
  - Do this for each retained tuple of bindings in sequence

114

The filter( ) Operator

- filter(*document, set-of-doc-nodes*)
  - *set-of-doc-nodes* specified using Xpath
  - Delete every node that *does not* occur in *set-of-doc-nodes* from *document*

    set-of-doc-nodes is treated literally as a set of nodes; children not included
  - Connect disconnected children to appropriate ancestor



115

The filter( ) operator (cont'd)

- *Example*: filtering http://xyz.edu/classes.html

  filter(//Class, //Class | //Class/CrsName)

  yields:

  <Class><CrsName>SE</CrsName></Class>
  <Class><CrsName>Circuits</CrsName></Class>
  <Class><CrsName>Databases</CrsName></Class>
  <Class><CrsName>TP</CrsName></Class>
  <Class><CrsName>Algebra</CrsName></Class>

  Deleted nodes:

  /Classes/Class/@CrsCode, /Classes/Class/@Semester, /Classes/Class/Instructor

116

Fixing the Restructuring Query Using filter( )

Declare variable, a syntactic sugar

Get rid of @Grade and thus of duplicate CrsTaken elements

```
LET $ct := document("transcript.xml")/Transcript/CrsTaken
FOR $c IN distinct(filter($ct, $ct | $ct/@CrsCode | $ct/@Semester))
RETURN
   <ClassRoster  CrsCode = {$c/@CrsCode}  Semester = {$c/@Semester} >
   {
     FOR $t IN document("transcript.xml")/Transcript
     WHERE  $t/CrsTaken/@CrsCode = $c/CrsCode
              AND $t/CrsTaken/@Semester = $c/@Semester
     RETURN  $t/Student
                 SORTBY ($t/Student/@StudId)
   }
   </ClassRoster>
   SORTBY ($c/@CrsCode)
```

117

User-defined Functions

- Can define functions, even recursive ones
- Functions can be called from within a FLWR expression
- Body of function is an XQuery expression
- Result of expression is returned
  - Result can be a primitive data type (integer, string), an element, a list of elements, a list of arbitrary document nodes, …

118

XQuery Functions: Example

- Count the number of *e*-children recursively:

```
DEFINE FUNCTION countNodes(element $e) RETURNS integer {
   RETURN
     IF empty($e/*) THEN 0
     ELSE
        sum(FOR $n IN $e/* RETURN countNodes($n))
         + count($e/*)
}
```

Function signature

XQuery expression

Built-in functions sum, count, empty

119

Class Rosters (again) Using Functions

```
DEFINE FUNCTION extractClasses(element $e) RETURNS element* {
   FOR $c IN $e//CrsTaken
   RETURN <Class CrsCode={$c/@CrsCode} Semester={$c/@Semester} />
}
<Rosters>
    FOR $c IN
          distinct( FOR $d IN document("transcript.xml") RETURN extractClasses($d) )
    RETURN
       <ClassRoster CrsCode = {$c/@CrsCode} Semester = {$c/@Semester} >
       {
         LET $trs := document("transcript.xml")
         FOR $t IN $trs//Transcript[CrsTaken/@CrsCode=$c/@CrsCode and
                                    CrsTaken/@Semester=$c/@Semester]
         RETURN $t/Student
         SORTBY ($t/Student/@StudId)
       }
       </ClassRoster>
</Rosters>
```


Converting Attributes to Elements with XQuery

- An XQuery reformulation of a previous XSLT query – much more straightforward (but ignores text nodes)

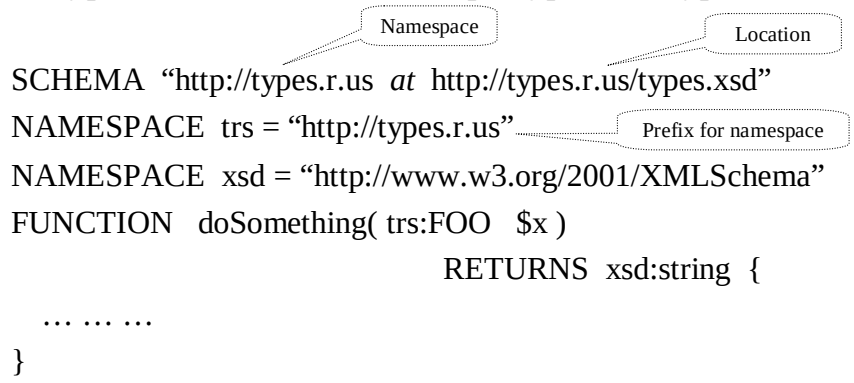
```
DEFINE FUNCTION convertAttributes(attribute $a) RETURNS element {
   RETURN element {name($a)} {value($a)}
}
FUNCTION convertElement($e) RETURNS AnyElement {
   RETURN element {name($a)}
      {
         { FOR $a IN $e/@* RETURN convertAttribute ($a) }
         IF empty($e/*) THEN $e/text( )
         ELSE { FOR $n IN $e/* RETURN convertElement($n) }
      }
}

RETURN convertElement(document("my-document")/*)
```

Computed element

The actual query: Just a RETURN statement!!

Integration with XML Schema and Namespaces

- Let type FOO be defined in http://types.r.us/types.xsd:



122

Grouping and Aggregation

- Does not use separate grouping operator
  - Recall that OQL does not need one either
  - 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)

123

Aggregation Example

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

```
FOR  $t  IN  document("transcripts.xml")//Transcript,
       $s  IN  $t/Student
LET   $c :=  $t/CrsTaken
RETURN
     <StudentSummary  StudId = {$s/@StudId}  Name = {$s/@Name}
                              TotalCourses = {count(distinct($c))} />
SORTBY (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*:

```
FOR  $t  IN  document("transcript.xml")//Transcript
WHERE  SOME  $ct  IN  $t/CrsTaken
                  SATISFIES  $ct/@CrsCode = "MAT123"
RETURN  $t/Student
```

Almost equivalent to:

```
FOR  $t  IN  document("transcript.xml")//Transcript,
       $ct  IN  $t/CrsTaken
WHERE  $ct/@CrsCode = "MAT123"
RETURN  $t/Student
```

– *Not equivalent, if students can take same course twice*!

Implicit Quantification

- Note: in SQL, variables that occur in FROM, but not SELECT are implicitly quantified with ∃
- In XQuery, variables that occur in FOR, but not RETURN are similar to those in SQL. However:
  - In XQuery variables are bound to document nodes
    - Two nodes may look textually the same (e.g., two different instances of the same course element), but they are still different nodes and thus different variable bindings
    - Instantiations of the RETURN expression produced by binding variables to different nodes are output even if these instantiations are textually identical
  - In SQL a variable can be bound to the same value only once; identical tuples are not output twice (in theory)
  - *This is why the two queries in the previous slide are not equivalent*

126

Quantification (cont'd)

- Retrieve all classes (from classes.xml) where each student took MAT123
  - Hard to do in SQL (before SQL-99) because of the lack of explicit quantification

```
FOR  $c  IN  document(classes.xml)//Class
LET  $g := {  -- Transctipt records that correspond to class $c
   FOR  $t  IN  document("transcript.xml")//Transcript
   WHERE  $t/CrsTaken/@Semester = $c/@Semester
             AND  $t/CrsTaken/@CrsCode = $c/CrsCode
   RETURN  $t
          }
WHERE  EVERY  $tr  IN  $g  SATISFIES
                    NOT empty($tr[CrsTaken/@CrsCode="MAT123])
RETURN  $c  SORTBY($c/@CrsCode)
```

127