XML Query Languages

References

- http://www.w3.org/TR/xpath
- http://www.w3.org/Style/XSL/
- http://www.w3.org/XML/Query/
- Semantic Web Primer: Chapter 2 (2.5 and 2.6)

XML Query Languages

- Market, convenience, …
- XPath, XSLT, XQuery: three query languages for XML
- XPath – simple & efficient
- XSLT – full feature programming language, powerful query capabilities
- XQuery – SQL style query language – most powerful query capabilities

XPath

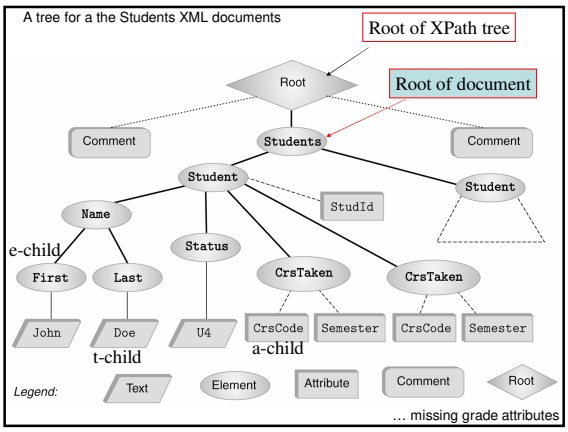
- Idea comes from *path expression* of OQL in object databases
- Extends the path expressions with query facilities by allowing search condition to occur in path expressions
- XPath data model:
  - view documents as trees (see picture), providing operators for tree traversing, use *absolute* and *relative path expression*
  - provide a way for computing a string-value for each type of node
- A XPath expression takes a document tree, returns a set of nodes in the tree, a value (boolean, number, string)

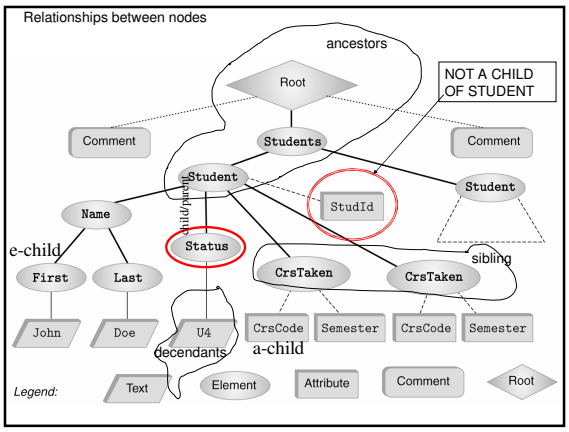
XPath – Basic concepts

- Node and the relationships between node in a tree
- String-value of a node:
  - element (root or any element): concatenation of the string-values of ALL TEXT NODE descendants of the element node in document order
  - attribute: the string which is the value of the attribute within the element
- Expanded-name of a node: a pair consisting of a localpart and a namespace part (specified the location of the namespace in which the element is defined)
- Expressions
- Context node
- Node tests

XPath – Example Document: Transcripts of Students

```
<?xml version="1.0" ?>
<!– List of the students with their courses -->
<Students Date="09/11/06">
    <Student StudID="111111111">
            <Name> <First> John </First> <Last> Doe </Last> </Name>
            <Status>U4</Status>
            <CrsTaken Semester="F2006" CrsCode="CS579" Grade="A"/>
            <CrsTaken Semester="F2006" CrsCode="CS598" Grade="B" />
            <CrsTaken Semester="F2006" CrsCode="CS510" Grade="B"/>
    </Student>
    <Student StudID="222222222">
            <Name> <First> Abraham </First> <Last> Fury </Last> </Name>
            <Status>U4</Status>
            <CrsTaken Semester="F2006" CrsCode="CS579" Grade="C"/>
            <CrsTaken Semester="F2006" CrsCode="CS570" Grade="A"/>
            <CrsTaken Semester="F2006" CrsCode="CS510" Grade="B"/>
    </Student>
    … …
</Students>
```





OS Directories vs Path expressions

Operating system:	XPath:
/ = the root directory	/students = the root element (if named *students*)
/usr/tson/tmp = the file named tmp in tson in usr	/students/student/crstaken = *every* crstaken element in every student in the documents
tmp = the file named tmp in the current directory	crstaken = *every* crstaken element that is a child of the current element
. = the current directory	. = the current element
.. = the parent directory	.. = parent of the current element
/usr/tson/* = all the files in /usr/tson	/students/student/* = all the elements in /students/student

Absolute vs relative path

- A path that begins with a / represents an absolute path, starting from the top of the document
  - Example: /students/student
  - Note that even an absolute path can select *more than one* element
  - A slash by itself means "the whole document"
- A path that does *not* begin with a / represents a path starting from the current element
  - Example: student/name
- A path that begins with // can start from *anywhere* in the document
  - Example: //student/crscode selects every element crscode that is a child of an element student
  - This can be expensive, since it involves searching the entire document

Condition

- A number in brackets selects a particular matching child (counting starts from 1, except in Internet Explorer)
  - Example: /students/student[1] selects the first student of the list
  - Example: /student/crstaken[2] selects the second crstaken of every student in the XML document
  - Example: /students/student[1]/crstaken[2]
  - Only *matching* elements are counted; for example, the student has name and crstaken elements, the former are ignored when counting crstaken.
- The function last() in brackets selects the last matching child
  - Example: : /students/student[last()]
  - You can even do simple arithmetic
  - Example: /students/student[last()-1]

XML Document

```
<?xml version="1.0" ?>
<!– List of the students with their courses -->
<Students  Date="09/11/06">
    <Student StudID="111111111">
            <Name> <First> John </First> <Last> Doe </Last> </Name>
            <Status>U4</Status>
            <CrsTaken Semester="F2006"  CrsCode="CS579"  Grade="A"/>
            <CrsTaken Semester="F2006"  CrsCode="CS598"  Grade="B" />
            <CrsTaken Semester="F2006"  CrsCode="CS510"  Grade="B"/>
    </Student>
    <Student StudID="222222222">
            <Name> <First> Abraham </First> <Last>  Fury </Last> </Name>
            <Status>U4</Status>
            <CrsTaken Semester="F2006"  CrsCode="CS579"  Grade="C"/>
            <CrsTaken Semester="F2006"  CrsCode="CS570"  Grade="A"/>
            <CrsTaken Semester="F2006"  CrsCode="CS510"  Grade="B"/>
    </Student>
    … …
</Students>
```

Result of XPath expressions

- /Students/Student[1]/CrsTaken[2]:
  - <CrsTaken Semester="F2006" CrsCode="CS598" Grade="B"/>
- /Students/Student/CrsTaken/@CrsCode
  CS579
  CS598
  CS510
  …
- Students/Student/Name/First/text()
  John
  Abraham

Xpath: Summary from example

students	matches a students element
*	matches any element
/	matches the root element
/students	matches a students element under root
students/student	matches a student in students
students//student	matches a student in students, at any depth
//student	matches a students at any depth
student|crstaken	matches a student or a crstaken
@semester	matches a semester attribute
students/student/@studid	matches studid attribute in student, in students
student/name/[@first="John"]/..	matches all Johns in the doc

Axes – Specified a set of nodes

ancestor – contains the ancestors of the context node
ancestor-or-self – contains the ancestors of the context node including the current node
attribute – contains the attributes of the context node (empty if context node is not an element)
child – contains the children of the context node
descendant – contains the descendants of the context node
descendant-or-self
following – contains all nodes in the same document as the context node that are after the context node in document order, excluding any descendants and excluding attribute nodes and namespace nodes
following-sibling – …
namespace – contains the namespace nodes of the context node
parent – contains the parent of the context node, if there is one
preceding – like following but take those come before the context node
preceding-sibling – …
self – context node

Xpath: Abbreviated syntax

- Examples:
  - child::name/child::lastname = name/lastname
  - child::student/descendant::grade = student//grade
  - child::student/parent::* = student/..
  - child::student/attribute::studid = student/@age
- child can be omitted
- decendant-or-self can be replaced by //
- attribute can be replaced by @
- …

Basic functions in XPath

- last()
- position()
- count()
- id()
- name()
- string()
- …
- See specification http://www.w3.org/TR/xpath

XPointer

- Use the features of XPath to navigate within an XML document
- Syntax:
  *someURL#*xpointer*(XPathExpr1)*xpointer*(XPathExpr2)…*
- Example:
  http://www.foo.edu/Report.xml#xpointer(//Student[…])

XSLT

- Part of XSL – an *extensible stylesheet langage of XML,* a transformation language for XML: converting XML documents into any type of documents (HTML, XML, etc)
- A functional programming language
- XML syntax
- Provide instructions for converting/extracting information
- Output XML
- A comprehensive tutorial at http://www.zvon.org/xxl/XSLTutorial/Books/Book1/index.html

XSLT Idea

- To convert a XML document into another type of document is similar to process a tree or a set of nodes from the tree
  - A XML document – tree
  - XPath expression – set of nodes of the tree
- To do so, one only needs to add instructions for processing the nodes (different types of nodes, different ways of processing, etc.)

XSLT Stylesheet

- Stylesheet: specifies a transformation of one type of document into another type
- Specifies by a command in the XML document

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl"
        href="http://xyz.edu/Report/report.xsl"?>
<Report Date="2002-03-01"
....
</Report>
```

What parser should be used! (can be omitted)

Location of the stylesheet

XSLT - Example

```
<?xml version="1.0"?>
<StudentList xmlns:xsl=
   "http://www.w3.org/1999/XSL/Transform" xsl:version="1.0">
   <xsl:copy-of select= "//Student/Name"/>
</StudentList >
```

Result:

```
<StudentList>
   <Name><First>John</First><Last>Doe</Last></Name>
   <Name>.....</Name>
   ......
</ StudentList>
```

See http://www.cs.nmsu.edu/~tson/classes/fall06-479/XSL/students-xsl.xml

XSLT – Instructions

- copy-of
- if-then
- for-each
- value-of
- …..
- See specification in the document (http://www.w3.org/TR/xslt)

XSLT – Instructions

```
<?xml version="1.0"?>
<StudentList xmlns:xsl=
   "http://www.w3.org/1999/XSL/Transform"
   xsl:version="1.0">
<xsl:for-each select= "//Student">
  <xsl:if test="count (CrsTaken) &gt; 1">
     <FullName> <xsl:value-of select="*/Last"/>,
                <xsl:value-of select="*/First"/>
     </FullName>  </xsl:if>
</xsl:for-each>
</StudentList >
```

XSLT – Instructions

```
<?xml version="1.0"?>
<StudentList xmlns:xsl=
   "http://www.w3.org/1999/XSL/Transform"
   xsl:version="1.0">
<xsl:for-each select= "//Student">
  <xsl:if test="count (CrsTaken) &gt; 1">
     <FullName> <xsl:value-of select="*/Last"/>,
                    <xsl:value-of select="*/First"/>
     </FullName>  </xsl:if>
</xsl:for-each>
</StudentList >
```

Result:

```
<StudentList>
   <FullName>
   John, Doe
   …..
   …..
  </FullName>
</StudentList>
```

XSLT – Template

- Template: contains instructions for selecting and processing a set of nodes
  - Recursive traversal of the structures of the document
  - Often defined recursively
- Need: algorithm for processing a XSLT template

Example of template for "for-each"

```
<xsl:template match="//Student">
   <xsl:if test="count (CrsTaken) &gt; 1">
      <FullName>
         <xsl:value-of select="*/Last"/>,
         <xsl:value-of select="*/First"/>
      </FullName>
   </xsl:if>
</xsl:template>
```

Same result as using "for-each"

See: http://www.cs.nmsu.edu/~tson/classes/fall06-479/XSL/students_for_each_template.xml

Template rules

- One stylesheet might contain several templates
- Each template specified instructions for selecting and processing a set of nodes

➔ what happens if several templates specify the same set of nodes

- explicit specification of priority
- more specific rules get applied
- if cannot be decided using specificity: either an error or the one that appears last in the stylesheet

http://www.zvon.org/xxl/XSLTutorial/Books/Output/example69_ch2.html

Examples

- A stylesheet with more than one template (http://www.cs.nmsu.edu/~tson/classes/fall06-479/XSL/students_many_tmp.xml)
- A stylesheet with conflicting templates (http://www.cs.nmsu.edu/~tson/classes/fall06-479/XSL/students_many_temp_conf.xml)
- Can put into HTML format etc. (http://www.cs.nmsu.edu/~tson/classes/fall06-479/XSL/students_many_tmp_html.xml)

XQuery
FLWR ("Flower") Expressions

- Syntax similar to SQL

```
FOR  .. LET …
   FOR .. LET …      variable declaration
WHERE …              condition
RETURN …             result
```

- FOR → FROM
- WHERE → WHERE
- RETURN → SELECT

SQL correspondence

Transcripts.xml

```
<?xml version="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>
    <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 - Example

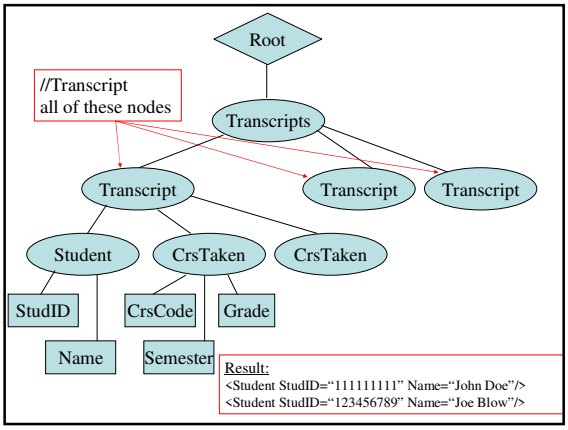
```
FOR $t IN
   document("http://xyz.edu/transcripts.xml")
   //Transcript
WHERE $t/CrsTaken/@CrsCode = "MA123"
RETURN $t/Student
```

Declare $t and its range

→ Return the Student element of $t

Find all transcripts containing "MA123"

Return the set of Student's elements of those transcripts

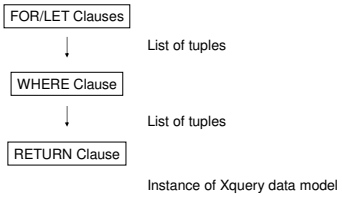


Putting it in well-formed XML

```
<StudentList>
(FOR $t IN
   document("http://xyz.edu/transcripts.xml")
   //Transcript
WHERE $t/CrsTaken/@CrsCode = "MA123"
RETURN $t/Student
)
</StudentList>
```

XQuery – How does it work?

- FOR-LET-WHERE-RETURN = FLWR



Instance of Xquery data model

Construction of class rosters from transcripts: first try

```
FOR $c IN distinct(document("http://xyz.edu/transcripts.xml")
                   //CrsTaken)
RETURN <ClassRoster CrsCode=$c/@CrsCode Semester=$c/@Semester>
    (
        FOR $t IN document("http://xyz.edu/transcripts.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)
```

For each class $c, find the students attending the class and output his information

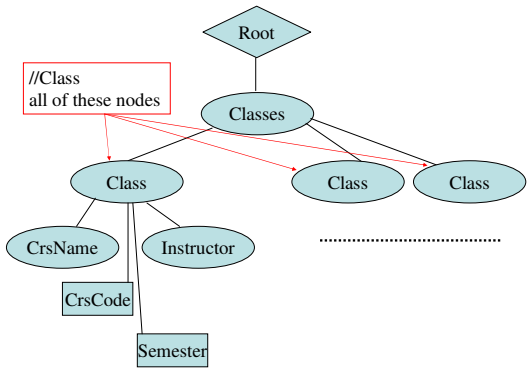
=➔ output one class roster for each *CrsTaken* node ➔ possibly more than one if different students get different grade

Fix ?

- Assume that the list of classes is available – write a different query
- Use the filter operation

Classes.xml

```
<?xml version="1.0" ?>
<Classes>
    <Class CrsCode="CS308" Semester="F1997">
        <CrsName>Market Analysis</CrsName>
        <Instructor>Adrian Jones</Instructor>
    </Class>
    <Class CrsCode="EE101" Semester="F1995">
        <CrsName>Electronic Circuits</CrsName>
        <Instructor>David Jones</Instructor>
    </Class>
    <Class CrsCode="CS305" Semester="F1995">
        <CrsName>Database Systems</CrsName>
        <Instructor>Mary Doe</Instructor>
    </Class>
    <Class CrsCode="CS315" Semester="S1997">
        <CrsName>Transaction Processing</CrsName>
        <Instructor>John Smyth</Instructor>
    </Class>
    <Class CrsCode="MAT123" Semester="F1997">
        <CrsName>Algebra</CrsName>
        <Instructor>Ann White</Instructor>
    </Class>
</Classes>
```



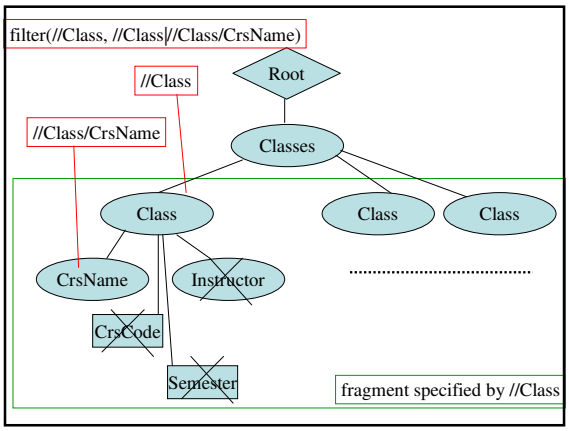
See Pg. 604 for XQuery (next slide)

```
FOR $c IN document("http://xyz.edu/classes.xml")//Class
RETURN
  <ClassRoster CrsCode=$c/@CrsCode Semester=$c/@Semester>
  $c/CrsName   $c/Instructor
  (FOR $t IN document("http://xyz.edu/transcripts.xml")//Transcript
     WHERE $t/CrsTaken/@CrsCode =  $c/@CrsCode
     RETURN $t/Student
     SORTBY($t/Student/@StudID)
  )
  </ClassRoster>
  SORTBY($c/@CrsCode)
```

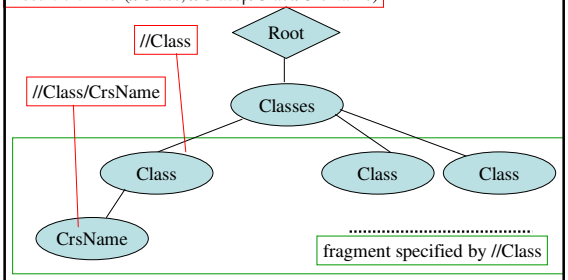
Give the "correct" result:
All ClassRoster, each only once

Filtering

- Syntax: filter(*argument1, argument2*)
- Meaning: return a document fragment obtained by
  - deleting from the set of nodes specified by *argument1* the nodes that do not occur in *argument2*
  - reconnecting the remaining nodes according to the child-parent relationship of the document specified by *argument1*



Result of: filter(//Class, //Class|//Class/CrsName)



Result:
<Class><CrsName>Market Analysis</CrsName></Class>
<Class><CrsName>Electronic Circuits </CrsName></Class>
…….

```
LET $trs:=document("http://xyz.edu/transcripts.xml")//Transcript
LET $ct:=$trs/CrsTaken
FOR $c IN  distinct(filter($ct, $ct|$ct/@CrsCode|$ct/@Semester))
RETURN
  <ClassRoster CrsCode=$c/@CrsCode Semester=$c/@Semester>
  (FOR $t IN $trs
     WHERE $t/CrsTaken/@CrsCode =  $c/@CrsCode AND
           $t/CrsTaken/@Semester = $c/@Semester
     RETURN $t/Student
     SORTBY($t/Student/@StudID))
  </ClassRoster>
  SORTBY($c/@CrsCode)
```

Give the "correct" result:
All ClassRoster, each only once

FOR v.s. LET

FOR
- Binds *node variables* → iteration

LET
- Binds *collection variables* → one value

FOR v.s. LET

```
FOR $x IN document("transcripts.xml")//
        transcript
RETURN <result> $x </result>
```

Returns:
<result> <transcript>…</transcript></result>
<result> <transcript>…</transcript></result>
<result> <transcript>…</transcript></result>
…

```
LET $x := document("transcripts.xml")//
        transcript
RETURN <result> $x </result>
```

Returns:
<result>
<transcript>…</transcript>
<transcript>…</transcript>
<transcript>…</transcript>
</result>

Collections in XQuery

- Ordered and unordered collections
  - /transcripts/transcript/CrsTaken = an ordered collection
  - Distinct(/ transcripts/transcript/CrsTaken) = an unordered collection
- LET $a := transcripts/transcript/CrsTaken → $a is a collection

Other features

- SORTBY (see example of classroster)
- IF-THEN-ELSE
- SOME (e.g. students who get some A)
- EVERY (e.g. students who get only A)

XQuery – Students with some A

```
FOR $t IN
  document("http://xyz.edu/transcripts.xml")
  //Transcript
WHERE SOME $c IN $t/CrsTaken SATISFIES
                    $c/@Grade = "A"
RETURN $t/Student
```

XQuery – Students with all A

```
FOR $t IN
  document("http://xyz.edu/transcripts.xml")
  //Transcript
WHERE EVERY $c IN $t/CrsTaken SATISFIES
                    $/@Grade = "A"
RETURN $t/Student
```

Advances Features

- User-defined functions
- XQuery and Data types
- Grouping and aggregation

Class rosters constructed with user-defined functions.

```
FUNCTION extractClasses(AnyElement $e)
                    RETURNS LIST(AnyElement){
  FOR $c IN $e//CrsTaken
RETURN <Class CrsCode=$c/@CrsCode Semester=$c/@Semester/>
}

<Rosters>
(
  LET $trs := document("http://xyz.edu/transcripts.xml")
  FOR $c IN distinct(extractClasses($trs))
  RETURN
    <ClassRoster CrsCode=$c/@CrsCode Semester=$c/@Semester>
      (
        FOR $t1 IN  $trs//Transcript[CrsTaken/@CrsCode=$c/@CrsCode and
                                     CrsTaken/@Semester=$c/@Semester]
        RETURN
          $t1/Student
          SORTBY($t1/Student/@StudId)
      )
    </ClassRoster>
)
</Rosters>
```

Transcripts.xml

```
<?xml version="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>
    <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 - Aggregation
Students with less than 4 classes

```
FOR $t IN
  document("http://xyz.edu/transcripts.xml")
  //Transcript
LET $nc := $t/CrsTaken
WHERE count($nc) < 4
RETURN $t/Student
```

Result:

```
<Student StudID="987654321" Name="Bart Simpson"/>
<Student StudID="123454321" Name="Joe Blow"/>
<Student StudID="023456789" Name="Homer Simpson"/>
```