

Xquery

Prof. Alex Brodsky

INFS 740

GMU

Element constructors

- To create or add to XML documents
- Can use straight XML
 - Create a student with name and ID

```
<student>  
  <name>Laurie Reeves</name>  
  <ID>123456</ID>  
</student>
```

Element constructors

- Also, can use embedded XQuery statements

```
<student>  
  {document("students.xml")//courses[title="SENG513"]/student[ID="123456"]/name}  
  <ID>123456</ID>  
</student>
```

Another FLWR Example

Return a list of students who have taken at least
100 courses

```
FOR $p IN document("students.xml")//student
LET $b := document("courses.xml")//course[student =
    $p]
WHERE count($b) > 100
RETURN $p
```

Sequence Operators

- Comma separated values
 - 1, 2, 3
- Optional brackets
 - (1, 2, 3)
 - (1,2,(3,4)) = (1,2,3,4)
- TO
 - binary operator
 - Changes operands to integers
 - e.g. 2 to 5 = (2,3,4,5)

Sequence Operators

- UNION, INTERSECT and EXCEPT

– e.g.

1, 2, 3 UNION 4, 5, 6

Conditional expressions

- IF, THEN, and ELSE

```
FOR $h IN //item
RETURN
  <item>
    {$h/name,
      IF ($h/@type = "Student")
      THEN $h/ID
      ELSE $h/description
    }
  </item>
```

Order in XML

- **BEFORE and AFTER**

```
LET $p := //course[1]
FOR $e IN //* AFTER ($p//midterm_1)[1] BEFORE
    ($p//midterm_2)[2]
RETURN $e
```

Order in XML

- Order By

```
LET $p := //course[1]
FOR $e IN //* AFTER ($p//midterm_1)[1] BEFORE
    ($p//midterm_2)[2]
ORDER BY $e/title
RETURN $e
```

- Unordered

```
Unordered (
LET $p := //course[1]
FOR $e IN //* AFTER ($p//midterm_1)[1] BEFORE
    ($p//midterm_2)[2]
RETURN $e
)
```

Quantifiers

- Tests for existence of some elements that satisfy a condition
- Also used to test whether all elements in a collection satisfy a condition
- XQuery provides for existential (“some”) and universal (“every”) quantifiers
- Key words **satisfies** and **contains**

Quantified expressions

SOME variable **IN** expr **SATISFIES** expr

e.g. Find the names of faculty who teach a course in which the description contains Java and XML

```
FOR $b IN //faculty
WHERE SOME $p IN $b//course/description
  SATISFIES (contains($p, "java") AND
  contains($p, "XML"))
RETURN $b/name
```

Functions

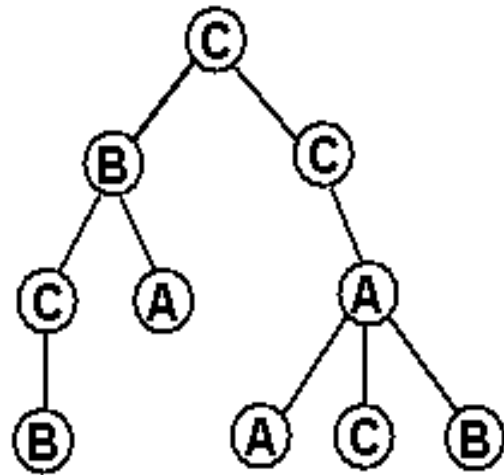
- XQuery provides a core library of built in functions
- document, distinct, empty, avg, sum, count, max, and min
- XQuery allows user to define their own functions
- Each function must declare the datatypes of its parameters and result, and body of function

Filtering

- “filter” is a function in the XQuery core function library
- Takes two operands, each of which is an expression
- Returns copies of the nodes in the first operand that are top level nodes in the second operand
- If the two operands don’t have a common root, then filter returns an empty list

Filtering – Diagram

`filter(/C, //A | //B)`



Before Filtering: /C



After Filtering: /C FILTER //A | //B

Data Types in XQuery

- XQuery is *strongly typed*
 - Every expression and sub-expression has a well-defined data type: primitive values *and* elements, attributes, etc.
- Based on XML Schema 1.0

Test or modify datatypes

- **INSTANCEOF** – Boolean binary operator, returns true if left operand is the same datatype as the right
 - \$x INSTANCEOF zoonames:animal
 - <a>{5} instance of element(*, xs:integer)
- **CAST** – changes the datatype
 - (x DIV y) CAST AS integer
- **TREAT** – treats one datatype as another for a given expression
 - \$mypet TREAT AS Cat

Explicit Data Types

```
import schema "ipo.xsd"
declare namespace ipo="http://www.example.com/IPO"

define function check-address( $a as ipo:address? )
  as xsd:boolean
{ ... }

for $x in document("foo")//ipo:invoice
where $x/ipo:shipTo instance of ipo:USAddress
...
```

Typeswitch

- Allows for polymorphism

```
TYPESWITCH ($a)
```

```
  CASE duck RETURN quack($a)
```

```
  CASE dog  RETURN woof($a)
```

```
  DEFAULT RETURN "No sound"
```

Conformance Features

- Basic XQuery
 - Cannot import XML Schemas; use “built-in” type names only
 - But data do have types
 - Types are checked dynamically
- Schema Import Feature
 - Import schemas: give names to element and attribute types

Conformance Features

- Static Typing Feature
 - Use the type rules specified in the Formal Semantics Document to check for type errors before query execution.

Conformance Features

- Four possibilities:
 - Basic
 - Basic+Schema
 - Basic+Static
 - Basic+Schema+Static

Basic+Schema+Static

- Documents have schemas
- Validate constructed XML
- When appropriate, XQuery will automatically
 - Extract data from simple elements/attributes
 - Convert numeric to double
 - Handle missing data
- But in this case, XQuery does **not**
 - Convert strings to numbers or vice versa

`$year + 3 {-- error if year is a string value --}`

- Allow path expressions that cannot exist
- Allow creation of XML that doesn't conform to schema

Basic

- Basic
Documents *don't* have schemas;
all elements are marked `xs:anyType` and all simple values are marked `xdt:untypedAtomic`
- `dt:untypedAtomic` will automatically be converted to the needed type in most expressions

`$year + 3`

this is fine if `$year` is `xdt:untypedAtomic`

Relational Databases

- Basic (possibly+Static)
- Just because there is no schema does not mean there are no data types
 - XQuery is defined to work on a data model form
 - The data model includes type annotations
- So for relational data, the types on columns and elements can be “built-in” to the model.

Validation

- Specify conformance level: When validation is on, elements are validated according to XML Schema

```
validate strict {  
  <bib:shortbook>  
    <bib:author>{$b/firstname, “ “, $b/lastname}  
      </bib:author>  
    {$b/bib:title}  
    <bib:publisher>  
      {$b/bib:publisher/@*}  
      {$b/bib:publisher/bib:name}  
    </bib:publisher>  
  </bib:shortbook>  
}
```