Introduction to XQuery

James Cummings

February 2006

What is XQuery?

- It is a domain-specific method for accessing and manipulating XML
- It is meant for querying XML
- It is built upon XPath
- It is like SQL but for XML
- A W3C proposed recommendation

XQuery: Expressions

path expressions return a nodeset

element constructors return a new element

FLWOR expressions analogous to SQL Select statement

list expressions operations on lists or sets of values

conditional expressions traditional if then else construction

qualified expressions boolean operations over lists or sets of values

datatype expressions test datatypes of values

XQuery: Path Expression

The simplest kind of XQuery that you've already seen:

```
document("test.xml")//p
//p/foreign[@lang='lat']
//foreign[@lang='lat']/text()
//tei:TEI//tei:person[@age >= 25]
```

XQuery: Element constructor

You may construct elements and embed XQueries or create well-formed results in the output you return. These may contain literal text or variables:

```
<latin>o tempora o mores</latin>
<latin>{$s}</latin>
<li>Name: {$surname}, {$forename}</li>
<li>Birth Country:
{data($person/tei:birth/tei:placeName/tei:country)}
</li>
```

XQuery: FLWOR expressions

For - Let - Where - Order - Return

```
declare namespace tei="http://www.tei-c.org/ns/1.0";
for $t in document("book.xml")//tei:text
let $latinPhrases := $t//tei:foreign[@xml:lang='lat']
where count($latinPhrases) > 1
order by count($latinPhrases)
return
  <list><item>ID: {data($t/@xml:id)}</item>
  <item>Phrases: {$latinPhrases} </item></list>
```

- `for` defines a cursor over an xpath
- `let` defines a name for the contents of an xpath
- `where` selects from the nodes
- `order` sorts the results
- `return` specifies the XML fragments to construct
- Curly braces are used for grouping, and defining scope

XQuery: List Expressions

XQuery expressions manipulate lists of values, for which many operators are supported:

- constant lists: (7, 9, <thirteen/>)
- integer ranges: i to j
- XPath expressions
- concatenation
- set operators: | (or union), intersect, except
- functions: remove, index-of, count, avg, max, min, sum, distinct-values ...

XQuery: List Expressions (cont.)

When lists are viewed as sets:

- XML nodes are compared on their node identity
- Any duplicates which exist are removed
- Unless re-ordered the database order is preserved

XQuery: Conditional Expressions


```
<div> {
  IF document("xqt")//tei:title/text()
        ="Introduction to XQuery"
  THEN <p>This is true.</p>
  ELSE <p>This is false.</p> }
</div>
```

More often used in user-defined functions

XQuery: Qualified Expressions

- some-in-satisfies

```
declare namespace tei="http://www.tei-c.org/ns/1.0";
for $b in document("book.xml")//tei:text
where some $p in $b//tei:p satisfies
 (contains($p,"sailing")
 AND contains($p,"windsurfing"))
return
 $b/ancestor::tei:TEI/tei:teiHeader//tei:title[1]
```

- every-in-satisfies

```
declare namespace tei="http://www.tei-c.org/ns/1.0";
for $b in document("book.xml")//tei:text
where every $p in $b//tei:p satisfies
 contains($p,"sailing")
return
 $b/ancestor::tei:TEI/tei:teiHeader//tei:title[1]
```

XQuery: Datatype Expressions

- XQuery supports all datatypes from XML Schema, both primitive and complex types
- Constant values can be written:
  - as literals (like string, integer, float)
  - as constructor functions (true(), date("2001-06-07"))
  - as explicit casts (cast as xsd:positiveInteger(47))
- Arbitrary XML Schema documents can be imported into an XQuery
- An `instance of` operator allows runtime validation of any value relative to a datatype or a schema.
- A `typeswitch` operator allows branching based on types.

XQuery and Namespaces

- As with XPath queries, if your documents are in a particular namespace then this namespace must be declared in your query
- Namespace prefixes must be used to access any element in a namespace
- Multiple namespaces can be declared and used
- Output can be in a different namespace to the input

XQuery Example: Multiple Variables

One of the real benefits that XQuery gives you over XPath queries is that you can define multiple variables:

```
 (: All Person Elements  with their Stone's Description :)
 declare namespace tei="http://www.tei-c.org/ns/1.0";
 for $stones in collection('/db/pc')//tei:TEI
 let $stoneDesc := $stones//tei:stoneDescription
 let $people := $stones//tei:person
return
 <div>{$people} {$stoneDesc}</div>
```

XQuery Example: Element Constructors

You can construct the results into whatever elements you want:

```
( : Women's Birth and Death Countries in placeName elements :)
 declare namespace tei="http://www.tei-c.org/ns/1.0";
 let $stones := collection('/db/pc')//tei:TEI
 for $person in $stones//tei:person[@sex = '2']
 let $birthCountry := $person/tei:birth//tei:country/text()
 let $deathCountry := $person/tei:death//tei:country/text()
return
 <div><p>This woman was born in
      <placeName>{$birthCountry}</placeName>
      and died in
      <placeName>{$deathCountry}</placeName>.</p>
  </div>
```

Result: Element Constructors

A series of `<div>` elements like:

```
<div>
  <p>This woman was born in
  <placeName> France </placeName>
   and died in
  <placeName> Italy </placeName>.
  </p>
</div>
```

XQuery Example: Traversing the Tree

You are not limited to one section of the document:

```
 (: Getting the Stone's Title :)
 declare namespace tei="http://www.tei-c.org/ns/1.0";
 let $stones := collection('/db/pc')//tei:TEI
 for $person in $stones//tei:person[@sex ='2']
 let $birthCountry := $person/tei:birth//tei:country/text()
 let $deathCountry := $person/tei:death//tei:country/text()
 let $title :=
  $person/ancestor::tei:TEI//tei:teiHeader//tei:title[1]/text()
return
 <div>
   <head>{$title}</head>
   <p>This woman was born in
    <placeName>{$birthCountry}</placeName> and died in
    <placeName>{$deathCountry}</placeName>.</p>
 </div>
```

Result: Traversing the Tree

A series of `<div>` elements like:

```
<div>
  <head> Stone 2 </head>
  <p> This woman was born in
    <placeName> France </placeName>
       and died in
    <placeName> Italy </placeName>. </p>
</div>
```

XQuery Example: Using Functions


```
 (: Getting birth date attribute and parts thereof :)
 declare namespace tei="http://www.tei-c.org/ns/1.0";
 let $stones := collection('/db/pc')//tei:TEI
 for $person in $stones//tei:person[@sex ='2']
 let $birthCountry := $person/tei:birth//tei:country/text()
 let $birthDate := $person/tei:birth/@date
 let $title := $person/ancestor::tei:TEI//tei:title[1]/text )
return
 <div>
  <head>{$title}</head>
 <p>This woman was born in
      <placeName>{$birthCountry}</placeName>.
 The date of their birth was in the year
 <date>{$birthDate}
   {data(substring-before($birthDate, '-'))}</date>.
      </p>
 </div>
```

Result: Using Functions

A series of `<div>` elements like:

```
<div>
  <head> Stone 2 </head>
   <p> This woman was born in
     <placeName> France </placeName>.
      The date of their birth was in the year
     <date date="1882-10-02"> 1882 </date>.</p>
</div>
```

XQuery Example: Nesting For Loops


```
 (: Number of people on a stone, women's forenames :)
 declare namespace tei="http://www.tei-c.org/ns/1.0";
 for $stones in collection('/db/pc')//tei:TEI
 let $title := $stones//tei:teiHeader//tei:title[1]/text()
 let $number := count($stones//tei:person)
return
 <div>
  <head>{$title}</head>
  <p>This stone has {$number} people.</p>
  { for $person in $stones//tei:person[@sex ='2']
 let $forename := $person//tei:forename/text()
return
 <p>This stone has woman whose
  first name is {$forename}.</p>
 }</div>
```

Result: Nesting For Loops

A series of `<div>` elements like:

```
<div>
 <head> Stone 2 </head>
   <p> This stone has 3 people.</p>
   <p> This stone has woman whose first name is Hilda. </p>
 </div>
```

XQuery Example: Embedding in HTML


```
 (: XQuery nested inside HTML :)
 declare namespace tei="http://www.tei-c.org/ns/1.0";
<html>
 <head><title>Stones with numbers</title></head>
 <body>
 {
 for $stones in collection('/db/pc')//tei:TEI
 let $title := $stones//tei:teiHeader//tei:title[1]/text()
 let $number := count($stones//tei:person)
return
 <div>
  <h1>{$title}</h1>
  <p>This stone has {$number} people.</p>
 </div>
}
</body>
</html>
```

Result: Embedding in HTML

An HTML document like:

```
<html>
 <head>
  <title> Stones with numbers </title>
 </head>
 <body>
  <div>
   <h1> Stone 1 </h1>
   <p> This stone has 1 people. </p>
  </div>
  <div>
   <h1> Stone 2 </h1>
   <p> This stone has 3 people. </p>
  </div>
   <!-- Many more stones -->
 </body>
</html>
```

XQuery in Practice

- We are using eXist web-form XQuery Interface so you can see the resulting XML
- You can handle requests passed from HTML forms inside your XQueries
- That XQuery is being used is invisible to the user
- eXist may be embedded into Apache's Cocoon web publishing framework
- You can send queries to eXist in all sorts of ways including from within Java programs and via HTTP/REST, XML-RPC, SOAP, WebDAV.
- You can use XUpdate to add/change/delete nodes in the XML database