Introduction to XQuery and XML Databases

TEI@Oxford

July 2009

XQuery

While XSLT is good for transforming XML to other formats (XML, HTML, PDF, Text, etc.) sometimes you may wish to query a large database of XML documents and extract only matching records. In cases such as this, XQuery might be more appropriate.

XML in Relational Databases vs Native XML Databases

XML-enabled Relational Databases: All XML structures are mapped onto to a traditional database form of fields in tables, but it is able to accept XML as input and render XML as output. The database itself does the conversion rather than relying on some form of middleware.

Native XML Databases: XML documents are used as the fundamental unit or storage and the internal model uses this XML. This does not mean that the documents are stored as text files, but may be stored as an indexed relationships of nodes.

Reasons for XML Databases

- Leverage advantages of XML
  - Easy interoperability with other systems
  - Maintain arbitrarily deeply-nested hierarchical documents as data source
  - Extensibility and customisability of XML
- Integrating XML into existing relational database systems is difficult:
  - Tables and fields are basic storage element instead of arbitrary hierarchy
  - XML structures need to be mapped in and out of tables for every query
  - High number of joins to recreate XML structure

Querying XML Documents

- Queries on XML documents are usually about the hierarchical structure (relationships between nodes)
- Types of relationships:
  - parent-child
  - ancestor-descendant
  - sibling (previous and following)
- Processing such queries without native XML databases can be difficult and slow

Storage in XML Databases: Collections of Trees

Trees XML documents are hierarchical trees, so storing them as trees benefits applications which need to navigate around that tree

Collections Collection-based implementations are useful for applications that query a collection of XML documents

Access XPath and XQuery are the main ways to access collections of XML trees

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 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[@xml:lang='lat']
//foreign[@xml: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>
```

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

Result: Element Constructors

A series of elements like:

```
<div>
  <p>This woman was born in
  <placeName> England </placeName>
   and died in
  <placeName> Ethiopia</placeName>.
  </p>
</div>
<!-- and more divs after this -->
```

XQuery Example: Traversing the Tree

You are not limited to one section of the document:

```
 (: Getting the issue title for each person :)
 declare namespace tei="http://www.tei-c.org/ns/1.0";
let $issues := collection('/db/pc')//tei:TEI
 for $person in $issues//tei:person[@sex ='2']
let $birthCountry := $person/tei:birth//tei:country
let $deathCountry := $person/tei:death//tei:country
let $name := $person/tei:persName
let $title := $person/ancestor::tei:TEI/tei:teiHeader//tei:title[1]
return
 <div>
  <head>{$name}</head>

   <p>This woman mentioned in {$title} was born in
    <placeName>{$birthCountry}</placeName> and died in
    <placeName>{$deathCountry}</placeName>.</p>
 </div>
```

Result: Traversing the Tree

A series of elements like:

```
<div>
<head><persName>Sylvia Pankhurst</persName></head>
<p> This woman mentioned in <title>Punch, or the London
Charivari, Vol. 147, July 1, 1914</title> was born in
    <placeName> England</placeName>
       and died in<placeName> Ethiopa </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 $issues := collection('/db/punch')//tei:TEI
 for $person in $issues//tei:person[@sex ='2']
let $birthCountry := $person/tei:birth//tei:country
let $birthDate := $person/tei:birth/@when
let $name := $person/tei:persName
let $title := $person/ancestor::tei:TEI/tei:teiHeader//tei:title[1]
return
 <div>
  <head>{$name}</head>
 <p>This woman mentioned in {$title} was born in
      <placeName>{$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 elements like:

```
<div>
<head><persName>Sylvia Pankhurst</persName></head>
<p> This woman mentioned in <title>Punch, or the London
Charivari, Vol. 147, July 1, 1914</title> was born in
     <placeName> England</placeName>.
      The date of their birth was in the year
     <date when="1882-05-05"> 1882 </date>.</p>
</div>
```

XQuery Example: Nesting For Loops

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

Result: Nesting For Loops

A series of elements like:

```
<div>
 <head>Punch, or the London Charivari, Vol. 147, July 1, 1914</head>
   <p> This issue has 30 people.</p>
   <p> This issue has a woman whose first name is Sylvia. </p>
   <p> This issue has a woman whose first name is Hilda. </p>
```

XQuery Example: Embedding in HTML

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

Result: Embedding in HTML

An HTML document like:

```
<html>
 <head>
  <title> Issues with numbers </title>
 </head>
 <body>
  <div>
   <h1>Punch, or the London Charivari, Vol. 147, July 1, 1914</h1>
   <p> This issue has 20 people. </p>
  </div>
  <div>
   <h1>>Punch, or the London Charivari, Vol. 147, July 15, 1914</h1>
   <p> This issue has 30 people. </p>
  </div>
   <!-- Many more issues-->
 </body>
</html>
```

XQuery in Practice

- You can handle requests passed from HTML forms inside your XQueries
- That XQuery is being used is invisible to the user
- eXist is often run 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