IBM

developerWorks®

How XQuery extends XPath

Things you can do in XQuery but not XPath

Donnie Cameron

April 01, 2008

XPath and XQuery are similar in some ways. XPath is even an integral part of XQuery. Both languages allow you to select bits of data from an XML document or an XML document store. In this article, you'll find descriptions of XPath and XQuery, and learn how XQuery extends XPath.

Although both XPath and XQuery perform some of the same functions, XPath provides simplicity and XQuery provides additional power and flexibility. XPath is the perfect tool for many types of queries. For example, XPath is the easiest way for you to create an unordered list of phone numbers from a subset of records in an XML document. However, if you need a query that expresses more complex record-selection criteria, transforms the result set, or requires recursion, then you need XQuery.

XPath

XPath is a domain-specific language (DSL) that is quickly becoming an important part of other more general-purpose languages. Programming languages are incorporating XPath through modules and classes, and in some cases directly into the languages' syntax. This is similar to what happened with regular expressions some time ago.

XPath is popular because of the considerable amount of time and effort that the language can save a developer when extracting specific bits of data from an XML document. Even individuals who have never dealt with XPath before can quickly harness its power. Consider the XML fragment in Listing 1.

Listing 1. XML Document

```
<users>
  <user>
    <name>
      <first>Lola</first>
      <last>Solis</last>
    </name>
    <age>2</age>
  </user>
  <user>
    <name>
      <first>Nina</first>
      <last>Serafina</last>
    </name>
    <age>4</age>
    <visits>
      <first>2008-01-15</first>
      <last>2008-02-15</last>
    </visits>
  </user>
  <user>
    <name>
      <first>Tracy</first>
      <last>Keller</last>
    </name>
    <age>35</age>
  </user>
</users>
```

If you want to obtain a list of last names of the children in this document, you can use the following XPath expression.

Listing 2. Selecting the last names of the users that are under 18 years old

```
/user[age lt 18]/name/last/text()

(: Result
     Solis
     Serafina
:)
```

Imagine the code that you'd have to write to extract that data without XPath. Even with the help of regular expressions, you'd need to think a little about how to exclude the value from the last tag that's in the visits node.

The above XPath expression is not only concise, but also quite clear. A quick glance reveals what the expression does, even to people that don't know XPath. XPath works because it is powerful and because it has a long history behind it. The language understands the nodes in an XML document of arbitrary complexity and, more importantly, the relationships among those nodes. So you can write concise expressions that consider not only elements and element values, but also attributes, processing instructions, and so on.

Many XML queries are hard to express in a clearer and conciser manner than with XPath. But the DSL-nature of XPath and the language's goals impose some fairly serious limitations on the programmer. The sections that follow describe the XQuery language briefly and show problems that XPath alone cannot solve. These problems require the programmer to move beyond XPath to a tool like XQuery.

XQuery

Because XQuery supports XPath natively, as part of XQuery's syntax, XQuery clearly can do everything that XPath can do. But XQuery is Turing-complete and can be considered a general-purpose language; it easily overcomes many of the limitations of XPath, at the expense of introducing a little complexity.

Overview

XQuery uses a simple syntax that is a mix of XML, XPath, comments, functions, and a special expression syntax to tie it all together. XQuery code consists entirely of expressions with no statements. All values are sequences and simplicity is important to the language. So both of the expressions `Hi` and `2 * 2` are valid XQuery code that will execute without any prelude or modification. XQuery is a high-level, strongly-typed, functional language (free of side-effects) that is ideal to express a query to obtain data both from an XML document and a large XML document repository. In this last respect, it is much like SQL. But XQuery additionally provides for expressing an arbitrary transformation of the result set. Much like the use of XPath can be rewarding when you want to retrieve some data from an XML document, the use of XQuery can be quite rewarding when you want to retrieve and transform data from a large repository of XML documents.

Transforming the result set

One obvious limitation of XPath is that it doesn't provide for transforming the result set in any way. Suppose you wanted to return the results from the earlier XPath query (Listing 2) in alphabetical order, as shown in Listing 3.

Listing 3. Results in alphabetical order

```
Serafina
Solis
```

You can't do this with XPath. To achieve this, you'd have to write code in another language (like XQuery, for example) or use some special, proprietary XPath extension to sort the results.

XQuery, on the other hand, allows you to sort the results or transform them into HTML, CSV, SQL, or any other text-based format. Among the most powerful types of transformations that you can do with XQuery are XML to XML transformations. Often, large XML databases can contain a large variety of complex, interrelated XML documents that a client application doesn't need. XQuery allows a client to describe precisely the type of XML document that it would like the server to return. By providing an XQuery interface, a server can often avoid keeping data in multiple schemas. Moreover, using XQuery to transform the data for a client is usually far easier and faster than attempting to transform the data with Perl or Java or some other popular computer language. Certainly, transforming data with XQuery when you retrieve the data is much faster in current implementations than performing a transformation later with XSLT.

For tying together record-selection criteria and result-transformation instructions, XQuery provides a feature called a FLWOR (pronounced "flower") expression. The letters in the acronym stand for `for`, `let`, `where`, `order by`, and `return`. These are the elements that can make up a FLWOR expression. FLWOR expressions include at least some of those elements in roughly the order that the acronym suggests. All FLWOR expressions start with a *for* or a *let* expression and end with a *return* expression. If you're familiar with SQL, you might already see where I am headed with this.

Here's a simple FLWOR expression, which borrows from Edwin Markham's poem, "Outwitted" (see Listing 4).

Listing 4. Simple FLWOR expression

```
let $xml:=
  <a>
    <one>She drew a circle that shut me out</one>
    <two>Heretic rebel, a thing to flout</two>
  </a>

return $xml//one/text()

(: Result
    "She drew a circle that shut me out"
:)
```

Listing 5 shows how you can apply a simple FLWOR expression to the XML in Listing 1. (For brevity, the listing shows the text `_XML from Listing 1_` in place of the actual XML that belongs there.)

Listing 5. Simple FLWOR expression

```
let $xml:= _XML from Listing 1_
for $user in $xml//user[age lt 18]
order by $user/name/last
return $user/name/last/text()

(: Result
    Serafina
    Solis
:)
```

If you wanted the query to return an HTML fragment representing the result as a numbered list, you can apply the XQuery from Listing 6.

Listing 6. Simple FLWOR expression that outputs a numbered list in HTML

```
let $xml:= _XML from Listing 1_
return
  <ol>{
    for $user in $xml//user[age lt 18]
    order by $user/name/last
    return <li>{$user/name/last/text()}</li>
  }</ol>

(: Result
    <ol><li>Serafina</li><li>Solis</li></ol>
:)
```

Notice how XML and XQuery mix so intuitively and effectively in Listing 6.

Expressing more complex record-selection criteria

Aside from transforming the data that it retrieves, XQuery is also a lot better than XPath at finding data in the first place. XQuery and XPath often provide redundancy, which can help programmers be more expressive with queries. For example, Listing 7 shows how you can move the XPath expression fragment `age lt 18` into a `where` clause in the FLWOR expression.

Listing 7. Expressing XPath constraints in XQuery

```
let $xml:= _XML from Listing 1_
return
  <ol>{
    for $user in $xml//user
    where $user/age lt 18
    order by $user/name/last
    return <li>{$user/name/last/text()}</li>
  }</ol>
```

The result that the expression in Listing 7 produces is exactly the same as the result of the expression in Listing 6. But XQuery's `where` clause is significantly more flexible than the XPath syntax for expressing constraints. The XQuery `where` clause can consist of nested expressions of arbitrary complexity that can even include function calls. XQuery doesn't impose limitations on record-selection expressions.

Using functions and recursion

While XPath doesn't support functions, XQuery provides a substantial collection of built-in functions and operators and also allows users to define functions of their own. XQuery functions are strongly typed, support recursion, and can be declared as internal or external. An internal function is a standard function where the function body follows the function declaration. An external function is a type of function declaration that opens the door for implementations to allow the user to define the body of the function in a different programming language.

While recursion might not be the best approach for the tasks that many of developers undertake day to day, it often comes in handy when you work with XML, which can contain arbitrarily nested nodes. Consider the `transform-names` function defined in Listing 8.

Listing 8. Simple function to change node names in any XML document

```
(: Part 1 :)
define function transform-names($node as node()) as node() {
  element{replace(name($node), "_", "-")} {
    $node/text(), for $subnode in $node/* return transform-names($subnode)
  }
}

(: Part 2 :)
let $xml:=
<item>
  <item_type>book</item_type>
  <contributors>
    <author>
      <first_name>Charles</first_name>
      <last_name>Edward</last_name>
      <home_address>
        <home_street>206 S. Solomon St.</home_street>
        <home_city>New Orleans</home_city>
        <home_state>LA</home_state>
        <home_zip>70119</home_zip>
      </home_address>
    </author>
    <artist>
      <last_name>Salinas</last_name>
    </artist>
  </contributors>
</item>
```

```
return transform-names($xml)

(: Result
    <item>
      <item-type>book</item-type>
      <contributors>
        <author>
          <first-name>Charles</first-name>
          <last-name>Edward</last-name>
          <home-address>
            <home-street>206 S. Solomon St.</home-street>
            <home-city>New Orleans</home-city>
            <home-state>LA</home-state>
            <home-zip>70119</home-zip>
          </home-address>
        </author>
        <artist>
          <last-name>Salinas</last-name>
        </artist>
      </contributors>
    </item>
:)
```

The `transform-names` function, which amounts merely to the code that appears in Part 1 of Listing 8, accepts an XML document or node of arbitrary complexity. In every XML tag name, the function replaces any underscore character (_) with a dash character (-).

Recursion in this case makes it trivial for the function to traverse the structure of the document. As a result, the function is succinct (3 lines!), easy to maintain, and works with any valid XML document or node that doesn't use attributes. Even if the function seems a little difficult to grasp completely at first—especially for programmers that don't often resort to recursion—one might quickly guess how to modify the function to delete underscores instead of replacing them with dashes.

Expressing joins

XPath doesn't provide a means to join XML nodes in a query. However, just like SQL provides a natural syntax to express table-joins in queries, XQuery provides an intuitive (at least to SQL users) way to join sets of XML nodes. The code in Listing 9 describes how joins work in XQuery.

Listing 9. XQuery join expression

```
(: Part 1 :)
let $authors:=
  <authors>
    <author>
      <name>Harold Abelson</name>
      <books>
        <isbn>978-0-07-000422-1</isbn>
        <isbn>978-0-262-01063-4</isbn>
      </books>
    </author>
    <author>
      <name>Paul Graham</name>
      <books>
        <isbn>978-0-13-370875-2</isbn>
        <isbn>978-0-13-030552-7</isbn>
        <isbn>978-0-596-00662-4</isbn>
      </books>
    </author>
    <author>
      <name>Apostolos-Paul Refenes</name>
      <books>
        <isbn>978-0-471-94364-8</isbn>
        <isbn>978-981-02-2819-4</isbn>
      </books>
    </author>
  </authors>

(: Part 2 :)
let $books:=
  <books>
    <book>
      <title>Structure and Interpretation of Computer Programs</title>
      <isbn>978-0-07-000422-1</isbn>
    </book>
    <book>
      <title>Turtle Geometry</title>
      <isbn>978-0-262-01063-4</isbn>
    </book>
    <book>
      <title>ANSI Common LISP</title>
      <isbn>978-0-13-370875-2</isbn>
    </book>
    <book>
      <title>On LISP</title>
      <isbn>978-0-13-030552-7</isbn>
    </book>
    <book>
      <title>Hackers and Painters</title>
      <isbn>978-0-596-00662-4</isbn>
    </book>
    <book>
      <title>Neural Networks in the Capital Markets</title>
      <isbn>978-0-471-94364-8</isbn>
    </book>
    <book>
      <title>Neural Networks in Financial Engineering</title>
      <isbn>978-981-02-2819-4</isbn>
    </book>
    <book>
      <title>Handbook of Artificial Intelligence</title>
      <isbn>978-0-201-16889-1</isbn>
    </book>
    <book>
      <title>Artificial Intelligence Programming</title>
      <isbn>978-0-89859-609-0</isbn>
    </book>
    <book>
      <title>A New Guide to Artificial Intelligence</title>
      <isbn>978-0-89391-607-7</isbn>
    </book>
    <book>
      <title>Artificial Intelligence</title>
      <isbn>978-0-08-034112-5</isbn>
    </book>
    <book>
      <title>Artificial Intelligence</title>
      <isbn>978-0-631-18385-3</isbn>
    </book>
  </books>

(: Part 3 :)
return
  <books-complete-info>{
    for $book in $books/*
      for $author in $authors/*
```

```
        where $book/isbn = $author//isbn
          and (
            contains($book/title, "LISP")
            or contains($book/title, "Neural"))
    order by $book/title
    return <book>{$book/*, $author/name}</book>
  }</books-complete-info>
```

Parts 1 and 2 of Listing 9 assign XML documents to the variables authors and books. Some of the nodes in books relate to nodes in authors, such that a book node has an ISBN that is among the ones listed for an author node.

Part 3 of the listing contains an XQuery join expression that assembles a new XML document, books-complete-info (look ahead in Listing 10), that contains book nodes that include the author's name.

Note a few remarkable things about the code in Part 3 of Listing 9. The two `for` expressions near the beginning of that code hint to XQuery that this will be a join expression. The where clause is similar conceptually to what one might write in SQL to achieve the join. But notice that an author node can have multiple ISBNs, which requires that the `where` clause effectively mean: "where the book's ISBN is among the author's ISBNs". This compares more to a sub-select within an SQL where clause, but the XQuery syntax seems more intuitive and natural. And certainly the XQuery expression is more concise.

Listing 10. Results from an XQuery join expression

```
<books-complete-info>
  <book>
    <title>ANSI Common LISP</title>
    <isbn>978-0-13-370875-2</isbn>
    <name>Paul Graham</name>
  </book>
  <book>
    <title>On LISP</title>
    <isbn>978-0-13-030552-7</isbn>
    <name>Paul Graham</name>
  </book>
  <book>
    <title>Neural Networks in the Capital Markets</title>
    <isbn>978-0-471-94364-8</isbn>
    <name>Apostolos-Paul Refenes</name>
  </book>
  <book>
    <title>Neural Networks in Financial Engineering</title>
    <isbn>978-981-02-2819-4</isbn>
    <name>Apostolos-Paul Refenes</name>
  </book>
</books-complete-info>
```

Summary

XPath is a mature DSL that should be your first choice to get to a piece of data that is buried deep in an XML document or repository. But, XPath was not designed to handle many kinds of problems. As you saw in this article, XQuery extends XPath vastly, emerging as the tool of choice when you have complex data selection requirements or you need to return results that are sorted, specially formatted, or otherwise transformed.

Related topics

- W3C Recommendation for XQuery 1.0 Specification: Read the details on this query language designed to be broadly applicable across many types of XML data sources.
- W3C Recommendation for XQuery 1.0 and XPath 2.0 Functions and Operators: Explore this catalog of the functions and operators required for XPath 2.0, XML Query 1.0 and XSLT 2.0.
- W3C XML Query Use Cases: Look at usage scenarios for XQuery.
- What is XQuery (Per Bothner. O'Reilly xml.com, October 2002): Read this high level view of XQuery that introduces main ideas that you should understand before you go deeper or actually try to use it.
- Process XML using XQuery (Nicholas Chase, developerWorks, March 2007): In the tutorial , see how to use XQuery to retrieve information from an XML document stored in an XQuery-enabled database.
- An Introduction to XQuery (Howard Katz, developerWorks, January 2006): Get some background history, a road map into the documentation, and a snapshot of the current state of the XQuery specification.
- XML Schema Part 2: Datatypes Second Edition: Peruse the specification for the XML Schema language as it defines facilities for defining datatypes to be used in XML Schemas as well as other XML specifications.
- XPath Recommendation: Read more on XPath, a language for addressing parts of an XML document that is designed for use by both XSLT and XPointer.
- XML Path Language (XPath) 2.0: See where XPath is going.
- Turing test: Read about a proposal for a test of a machine's capability to demonstrate intelligence, on Wikipedia.
- : Find out how you can become an IBM-Certified Developer.
- XML technical library: See the developerWorks XML Zone for a wide range of technical articles and tips, tutorials, standards, and IBM Redbooks.
- DB2 Express-C 9.5: Download and try the XML database used for this tutorial.
- IBM trial software: Build your next development project with trial software available for download directly from developerWorks.