SQL Server 2005: Intro to XQuery

SQL 2005: Intro to XQuery

One of the coolest new features of SQL Server 2005 is its native XQuery support. The new `XML` data type we discussed in the last article, with its built-in XQuery support, makes server-side XML querying and manipulation easier than ever.

This article demonstrates XQuery in SQL Server 2005. We'll begin with a definition of XQuery.

What is XQuery?

XQuery (short for "XML Query") is a standard from the same folks who brought you XML[1], XSL[2], and dozens of other related specifications: the World Wide Web Consortium (W3C[3]). The W3C defines XQuery 1.0 as an extension to XPath 2.0, borrowing heavily from several other query language standards including XPath 1.0, XQL, XML-QL, SQL, and OQL.

XQuery is a declarative, functional query language that operates on instances of the XQuery/XPath Data Model (XDM[4]). The XDM defines a rich type system, allowing XQuery to operate on "the abstract, logical structure of an XML document, rather than its surface syntax" (W3C XQuery 1.0 standard). What this means is the XDM allows XQuery to query your XML using a "tree-like" logical representation of your XML. The "XML as a tree" concept should be familiar to Web programmers who dynamically manipulate HTML and XML using the Document Object Model (DOM) [although the similarities between DOM and XDM for the most part end there]. Each branch (or "node") of the XDM tree maintains a set of attributes describing the node. In the tree each node has an XML node type, XDM data type information, node content (string and typed representations), parent/child information, and possibly some other information specific to the node type.

Consider Listing 1 below which defines an XML Schema collection, creates a typed `XML` variable associated with the XML Schema collection, and then populates the `XML` variable with a valid XML document.

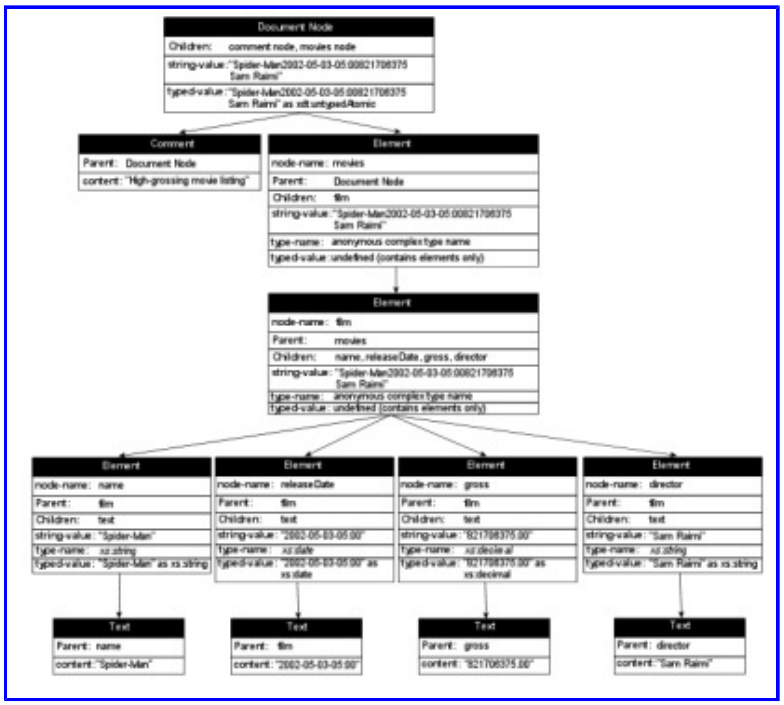
Listing 1. Typed XML example

```
/* Create an XML Schema collection */
CREATE XML SCHEMA COLLECTION MovieSchema AS N'<?xml version="1.0" encoding="UTF-16"?>
<xs:schema xmlns="http://schemas.sqlservercentral.com/MovieSchema"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="movies">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="film" minOccurs="1" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="name" minOccurs="1" maxOccurs="1" type="xs:string"/>
              <xs:element name="releaseDate" minOccurs="1" maxOccurs="1" type="xs:date"/>
              <xs:element name="gross" minOccurs="1" maxOccurs="1" type="xs:decimal"/>
              <xs:element name="director" minOccurs="1" maxOccurs="1" type="xs:string"/>
            </xs:sequence>
          </xs:complexType>
```

```
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>';
GO

/* Typed xml variable */
DECLARE @doc XML (DOCUMENT MovieSchema);

/* Populate with a valid XML document */
SELECT @doc = N'<?xml version="1.0" encoding="UTF-16"?>
<!-- High-grossing movie listing -->
<movies>
  <film>
    <name>Spider-Man</name>
    <releaseDate>2002-05-03-05:00</releaseDate>
    <gross>821706375.00</gross>
    <director>Sam Raimi</director>
  </film>
</movies>';

SELECT @doc;
```

XQuery can query the typed XML document via a tree structure generated by the XDM. Figure 1 below shows the XDM tree generated by this XML Schema for the sample XML document (click the image for a large version in its own pop-up window). The XDM can also generate trees for untyped XML, in which case it assigns types of `xdt:untyped` to element nodes and `xdt:untypedAtomic` to attributes, meaning they are not validated against an XML Schema.

Figure 1. XDM tree for sample document and XML Schema



The SQL Server 2005 XQuery implementation is a subset of the W3C XQuery 1.0 standard, and certain parts of that standard were left unimplemented in the current iteration of SQL Server XQuery support. This is important to know if you are trying to convert existing XQuery scripts from other sources, such as existing applications,

books on XQuery, or even samples from the W3C website.

Expressions and Sequences

Expressions are the building block of XQuery queries. Expressions come in many flavors including literals, variable references, function calls, and others. Examples of expressions include:

```
3.141592
"Hello Mrs. Robinson"
$count
fn:avg( (10, 20, 30, 40) )
```

Sequences are ordered collections of items. An example of a sequence might include:

```
(1, 1, 3, 5, 4, 10, 7, 6, 9, 2, 8)
```

A sequence can contain duplicate values and the order of the sequence is maintained. Note that the items in the example sequence are not sorted numerically or alphabetically, but are stored in the order they are added to the sequence. When a sequence is the result of an XQuery expression, the items are normally added to the sequence in "document order".

One important property of sequences is that a sequence of one item is equivalent to a singleton atomic value. So the sequence (2.718281828459) is equivalent to the singleton atomic value 2.718281828459.

The query() Method

The `xml` data type `query()` method allows you to query XML data using XQuery. A simple XQuery query consists of a path expression. Path expressions determine which nodes should be returned by an XQuery. The location path is read from left to right, corresponding to moving down and to the right in the XML node tree. A path expression that begins with a single forward slash ("/") starts at the root of the XML input. A path expression beginning with a double forward slash ("//") returns nodes that match the path anywhere in the XML input.

Borrowing from our sample XML in the previous article, we'll create a sample XML document we can use the xml data type's `query()` on. I'll assume you're a Family Guy fan like me (*Giggity-Giggity!*) for this example. The sample XML document, and its schema collection, looks like this:

Listing 2. The sample XML document

```
/* The Family Guy schema collection */
CREATE XML SCHEMA COLLECTION FamilyGuySchema AS N'<?xml version="1.0" encoding="UTF-16"?>
<xs:schema xmlns="http://schemas.sqlservercentral.com/FamilyGuySchema"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="show">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="episode" minOccurs="1" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="season" minOccurs="1" maxOccurs="1" type="xs:integer"/>
              <xs:element name="number" minOccurs="1" maxOccurs="1" type="xs:integer"/>
              <xs:element name="title" minOccurs="1" maxOccurs="1" type="xs:string"/>
              <xs:element name="airdate" minOccurs="1" maxOccurs="1" type="xs:date"/>
              <xs:element name="synopsis" minOccurs="1" maxOccurs="1" type="xs:string"/>
              <xs:element name="quote" minOccurs="1" maxOccurs="1" type="xs:string"/>
            </xs:sequence>
```

```
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="name" type="xs:string" use="required"/>
    </xs:complexType>
  </xs:element>
</xs:schema>';
GO

/* Will hold a well-formed XML document */
DECLARE @doc XML (DOCUMENT FamilyGuySchema);

/* Populate the XML document */
SELECT @doc = N'<?xml version = "1.0" encoding = "UTF-16"?>
<show name = "Family Guy">
  <episode>
    <season>1</season>
    <number>1</number>
    <title>Death Has A Shadow</title>
    <airdate>1999-01-31-05:00</airdate>
    <synopsis>
      After getting drunk at a bachelor party and getting fired from his
      job at the Happy-go-Lucky toy factory for being hung over, Peter
      applies for welfare... and receives a check for $150,000!
    </synopsis>
    <quote>
      Peter: Guys, our money problems are over; we are officially on
      welfare! Come on kids, help me scatter car parts on the front lawn.
    </quote>
  </episode>
  <episode>
    <season>1</season>
    <number>2</number>
    <title>I Never Met The Dead Man</title>
    <airdate>1999-04-11-05:00</airdate>
    <synopsis>
      Peter crashes into the town satellite dish, knocking out the TV
      across the city.  He promises to buy his daughter Meg a car if
      she takes the blame.  Peter goes crazy without TV, as Stewie
      builds a weather control device to destroy the world&amp;apos;s
      broccoli.
    </synopsis>
    <quote>
      Brian:  Hey barkeep, whose leg do you gotta hump to get a dry
      martini around here?
    </quote>
  </episode>
  <episode>
    <season>1</season>
    <number>3</number>
    <title>Chitty Chitty Death Bang</title>
    <airdate>1999-04-11-05:00</airdate>
    <synopsis>
      Peter lets Meg to go to a party with her new suicidal cult-member
      friend on Stewie&amp;apos;s birthday.  Stewie takes on the cult leader,
      who he believes is the &amp;quot;Man in White&amp;quot;, trying to return
      him to the womb.
    </synopsis>
    <quote>
      Peter:  Hey, Lois, look. The two symbols of the Republican party: an
      elephant and a big fat white guy who&amp;apos;s threatened by change.
    </quote>
  </episode>
  <episode>
```

```
    <season>1</season>
    <number>4</number>
    <title>Mind Over Murder</title>
    <airdate>1999-04-18-05:00</airdate>
    <synopsis>
      Peter gets into a fight with a manly-looking woman at a soccer game,
      and gets put on house arrest.  On the advice of a ghost he turns
      his basement into a bar for his friends.  Lois becomes the main
      attraction when she starts singing at Peter&amp;apos;s bar.  Meanwhile,
      Stewie builds a time machine in order to avoid teething pain.
    </synopsis>
    <quote>
      Stewie:  For the love of God, shake me! Shake me like a British
      nanny!
    </quote>
  </episode>
</show>';

SELECT @doc;
```

The first sample XQuery query will use an absolute path expression to retrieve all of the episode titles from the XML document in Listing 2:

```
/* Retrieve all show titles - absolute location path */
SELECT @doc.query('/show/episode/title');
```

The second sample retrieves all the quotes from the XML document using a relative path expression:

```
/* Retrieve all quote nodes anywhere in the XML */
SELECT @doc.query('//quote');
```

The wild-card character "*" can be used to match the name of any node within the path. For instance the sample query below retrieves all nodes beneath the /show/episode nodes, no matter what their names:

```
/* Retrieve all nodes beneath the /show/episode nodes */
SELECT @doc.query('/show/episode/*');
```

Because of its heritage, XQuery can dynamically build XML output from XML input. We can use the sample that retrieves show titles to dynamically build a well-formed XML document:

```
/* Retrieve all show titles - absolute location path */
SELECT @doc.query('<titles>
  { /show/episode/title };
</titles>
');
```

Simple path expressions like the above are fine for solving simple problems, but XQuery's real power lies in its FLWOR expressions.

FLWOR Power

FLWOR expressions are named for the keywords they are made of: `for`, `let`, `where`, `order by`, and `return`. SQL Server's XQuery supports all but the `let` clause. A FLWOR expression is a powerful construct that iterates your XML nodes with the `for` clause, limits the results using the `where` clause, sorts the results using the `order by` clause, and returns the results via the `return` clause. Consider the following sample FLWOR expression:

```
/* Retrieve all show titles - absolute location path */
SELECT @doc.query('
  for $ep in (/show/episode)
  where $ep/number le 3
  order by $ep/title
  return <episode>
    { ($ep/title, $ep/number, $ep/synopsis) }
  </episode>
');
```

This sample begins with a `for` clause, which iterates the `/show/episode` nodes, assigning each matching node to the variable `$ep` in turn:

```
for $ep in (/show/episode)
```

The `where` clause limits the results to episode nodes containing a number node with a value less than or equal to 3, discarding nodes that don't match.

```
where $ep/number le 3
```

The `order by` clause is an optional clause that determines the sort order of the results. In this example the results will be ordered by title.

```
order by $ep/title
```

Finally each matching episode's title, number, and synopsis are returned by the `return` clause.

```
return <episode>
  { ($ep/title, $ep/number, $ep/synopsis) }
</episode>
```

Comparison Operators

In the sample FLWOR expression above, I used the `le` comparison operator in the `where` clause to narrow result set to those nodes with a number node value "less than or equal to" 3. XQuery provides several comparison operators divided into three categories: value comparison operators, general comparison operators, and node comparison operators. Figure 2 below is a summary of the comparison operators available to XQuery.

Figure 2. XQuery comparison operators

Value Comparison Operators

`eq`	Equal to
`ne`	Not equal to
`gt`	Greater than
`ge`	Greater than or equal to
`lt`	Less than
`le`	Less than or equal to

General Comparison Operators

`=`	Equal to
`!=`	Not equal to
`>`	Greater than
`>=`	Greater than or equal to
`<`	Less than
`<=`	Less than or equal to

Node Comparison Operators

`is`	Node identity equality
`<<`	Left node precedes right node
`>>`	Left node follows right node

The value comparison operators compare one singleton atomic value to another, as in the following examples:

```
3 ne 4
"harry" lt "sally"
```

The general comparison operators are "existential" operators that can be used to compare sequences of items. Existential simply means the operator returns true if any of the singleton atomic values from the left-hand sequence fulfills the operator comparison with any of the singleton atomic values from the right-hand sequence. As an example, the following expressions return true:

```
(1, 2, 3) = (2, 4, 6)                 (: true because 2 eq 2 evaluates to
true :)
("C", "E", "F") < ("A", "B", "D")     (: true because "C" lt "D" evaluates
to true :)
```

But this expression returns false:

```
("Terry", "Terry", "Michael") = ("Eric", "John", "Graham")   (: false
because none of the items on the left are equal to any items on the
right :)
```

The final type of comparison operators are node comparison operators. The `is` operator returns true if the node on the left-hand side is the node on the right-hand side. Note that this doesn't mean it will return true if the two nodes have the same name, value, and contents: the `is` operator returns true only if the two nodes are actually the exact same node.

The << operator returns true if the left-hand node appears before the right-hand node, in document order. The >> operator returns true if the right-hand node appears after the left-hand node, in document order.

You can easily test XQuery expressions and get immediate feedback from SQL Server with a little snippet of code:

```
DECLARE @x XML;
SELECT @x = N'';
SELECT @x.query('3.141592 eq 3.0 (: replace the expression on the left with
your own expression :)');
```

When you run the snippet of code above (with my sample expression or your own expression plugged in), you will get immediate feedback from SQL Server: either true, false, or an empty sequence.

Conclusion

This article is an introduction to SQL Server 2005 XQuery. SQL Server 2005 also supports a subset of the XQuery 1.0/XPath 2.0 Data Model (XDM), a subset of the XQuery Functions and Operators (F&O) specifications, most of the XQuery-specified math operators, axes, node tests, and other XQuery functionality.

I hope this article provides a starting point for those beginning to explore the full power of SQL Server 2005's XQuery functionality.

Footnotes

[1] The XML 1.0 standard is available at http://www.w3.org/TR/2006/REC-xml-20060816/ and the XML 1.1 standard is available at http://www.w3.org/TR/2006/REC-xml11-20060816/

[2] The XSL family of recommendations is available at http://www.w3.org/Style/XSL/

[3] The W3C website is http://www.w3.org

[4] The XQuery 1.0/XPath 2.0 Data Model (XDM) specification is at http://www.w3.org/TR/xpath-datamodel/