The Fundamentals of the SQL Server 2005 XML Datatype

Source	http://www.developer.com/db/article.php/10920_3531196_2

Introduction to XQuery

SQL is the data-manipulation language for relational databases. XQuery is the SQL of the XML world. Like all XML specifications, the XQuery specification is maintained and enhanced by the World Wide Web Consortium (W3C). (All of the XQuery samples in this section utilize the Instructions field in the Production.ProductModel table of AdventureWorks.)

Mastering XQuery requires you to understand two things: XPath and FLOWR (For, Let, Order by, Where, Return) statements. XPath provides a way of expressing the location of data within a XML document and a way of performing operations on the data in an XML document. It includes functions for performing string, Boolean, and arithmetic operations on data within a XML document. XPath locations express a set of nodes within the XML document. An XPath expression looks like a file path in Windows Explorer. Keeping with the previous sample data, a typical XPath expression appears as follows:

```
/Inst:root/Inst:Location[1]/Inst:step
```

This XPath expression instructs the XML processor to give the calling program all of the "step" nodes attached to the first Location node, which is attached to the node called "root". (There are other ways to express the same set of Nodes above using XPath. All of the samples in this article follow the examples provided in the SQL Server 2005 Books Online.)

The expression above uses the XQuery qualified name (QName) with a namespace prefix and the more common short-cut syntax. The "[]" symbolizes a predicate. Predicates act like a filter, and they can contain a variety of other expressions. You'll see more predicate examples in a bit. (A complete introduction to all XPath expression syntax is beyond the scope of this article. See the Further Reading section at the end for more information.)

As stated previously, the second key to understanding XQuery is proficiency with the FLOWR statement. If you noticed earlier that the directives Where and Order By are also SQL directives and therefore assumed the statements worked like the SQL directives, you are correct. SQL was one of the inspirations for the FLOWR statement. In fact, the FLOWR statement is mechanically similar to a SQL select statement. Instead of returning rows of a

table, the statement returns a set of XML nodes. The following is a typical FLOWR statement:

```
for $RetVal in /root/Location[1]/step[1]/tool
order by $RetVal descending
return $RetVal
```

This statement returns the tool nodes on the first "step" node of the first "Location" node in descending order. As you can see, the FLOWR statement utilizes XPath to identify the group of nodes on which to operate. The "For" part of the FLOWR statement works something like the "From" clause in SQL. Anything proceeded by a "$" denotes a variable in XPath.

XQuery includes many other operators and functions, including:

- **Value comparison operations**
- **Aggregate functions such as Avg(), Sum(), and Count()**
- **Conversion functions**
- **If.. then.. else constructs**

Now that you understand the basics of XQuery, look at how functions on the XML datatype use XQuery.

Functions on the XML Datatype

The following functions allow you to extract and modify XML data using stored procedures, triggers, and user-defined functions:

- **Query() allows you to get sets of nodes from a XML document.**
- **Value() allows you to return a single value from an element or attribute in a document.**
- **Exist() returns a boolean value true if the XQuery expression returns values and false if the XQuery expression returns nothing.**
- **Modify() allows the developer to change values in a XML document.**

First, look at the Query function. The data returned from a Query function are a set of nodes. The following is an example of the Query function:

```
SELECT Instructions.query('
  declare namespace Inst="http://schemas.microsoft.com/sqlserver/
    2004/07/adventure-works/ProductModelManuInstructions";
```

```
        /Inst:root/Inst:Location[1]/Inst:step
') as ResVal FROM Production.ProductModel
```

One common trait among all of the XML datatype functions is they are used within the context of a regular SQL statement. Another common trait is that the functions are part of the XML datatype rather than a separate SQL function.

As stated previously, the `Value()` function returns a single value from the XML document. The following are examples of the `Value()` function:

```
SELECT Instructions.value('
  declare namespace Inst="http://schemas.microsoft.com/sqlserver/
    2004/07/adventure-works/ProductModelManuInstructions";
        (/Inst:root/Inst:Location[1]/Inst:step[1]/Inst:tool)[1]
','nvarchar(50)') as ResVal FROM Production.ProductModel
SELECT Instructions.value('
  declare namespace Inst="http://schemas.microsoft.com/sqlserver/
    2004/07/adventure-works/ProductModelManuInstructions";
        (/Inst:root/Inst:Location[1]/@LocationID)[1]
','int') as ResVal FROM Production.ProductModel
```

The first statement returns the value of an element in the document. The second statement returns the value of an attribute in the document (the @ symbol in front of the LocationID denotes an attribute). The `Value()` function requires two parameters to which the XQuery expression and the SQL datatype cast the resulting value.

The `Modify()` function allows a developer to change data in the XML document. XQuery provides no mechanism for changing XML data, so Microsoft implemented the XML Data Modification Language (DML), which has three types of statements:

- **Insert adds new nodes in the XML document.**
- **Delete removes nodes from the XML document.**
- **Replace value of updates the value of a node.**

Below are examples of each DML statement:

Insert

```
UPDATE Production.ProductModel
SET Instructions.modify('
  declare namespace Inst="http://schemas.microsoft.com/sqlserver/
    2004/07/adventure-works/ProductModelManuInstructions";
    insert element Inst:tool { "NewOne" } as last into
        (/Inst:root/Inst:Location[1]/Inst:step)[1]
')
```

Delete

```
UPDATE Production.ProductModel
SET Instructions.modify('
  declare namespace Inst="http://schemas.microsoft.com/sqlserver/
    2004/07/adventure-works/ProductModelManuInstructions";
    delete

(/Inst:root/Inst:Location[1]/Inst:step[1]/Inst:tool)[.="NewOne"]
')
```

Update

```
UPDATE Production.ProductModel
SET Instructions.modify('
  declare namespace Inst="http://schemas.microsoft.com/sqlserver/
    2004/07/adventure-works/ProductModelManuInstructions";
    replace value of

(/Inst:root/Inst:Location[1]/Inst:step[1]/Inst:tool[.="NewOne"])[1]
with "Old One"
')
```

If you are familiar with SQL, DML statements are intuitive. To use the statements properly, you must utilize some more advanced features of XQuery predicates. In the examples above, the [.="NewOne"] predicate works a lot like a where clause, narrowing the data returned by the XPath expression to nodes with specific information.

Like many other SQL datatypes, an XML datatype can be used with variables and as parameters to stored procedures. The following is an example of a stored procedure with an XML parameter:

```
create procedure Production.uspAddProductModel
```

```
@ModelName nvarchar(50),
@Inst xml
as
INSERT INTO [AdventureWorks].[Production].[ProductModel]
           ([Name]
           --,[CatalogDescription]
           ,[Instructions]
           ,[rowguid]
           ,[ModifiedDate])
     VALUES
           (@ModelName
           --,<CatalogDescription,
ProductDescriptionSchemaCollection,>
           ,@Inst
           ,NEWID()
           ,GETDATE())
```

Previously, this article discussed Schema collections. If a XML datatype in your table is mapped to a Schema collection, any XML document you save to the datatype must match an XML Schema in the collection. The following error will appear for any XML that violates the Schema collection:

```
Msg 6905, Level 16, State 3, Line 1
XML Validation: Attribute 'LocationI' is not permitted in this
context.
```

Finally, you can also include SQL datatypes from the SQL statement by executing the XML function inside the XQuery expression using a function called **sql::column()**.

Other Considerations and the Future

You must address other considerations when using XML datatypes with SQL Server 2005:

- **XML views can be created from relational tables by using SQLXML mapping features. You can use XQuery against the XML views. Other XML functions facilitate adding the XML to**

underlying relational tables, so you need not worry about creating all of the necessary relational tables.

- **You can store XML in Text, Image, or nvarchar fields. If you store malformed XML or want to store unmodified XML, the other field may be your only option.**
- **Versioning XML data is different from versioning relational data. You must consider the XSD implications, as well as accommodating existing data in the database, use of user-defined functions, XQuery changes, and so forth. You now can store in a single field what you would normally store in a set of tables.**

Future native file formats of Microsoft Office applications will be XML. It will be interesting to see how SQL Server 2005 integrates with future Office versions.

***~~~ End of Article ~~~***