Replacing OPENXML with the XML nodes() Function in SQL Server 2005

Written By: Ray Barley -- 10/21/2008

Problem

We have a number of instances where we coded stored procedures to handle transforming an input parameter that is an XML document into a rowset. Prior to SQL Server 2005 we have always used the OPENXML rowset provider to parse the XML and produce a rowset. What new functionality is there in SQL Server 2005 to accomplish this?

Solution

SQL Server 2005 includes native support for XML; there is now an XML column type with a nodes() function that can be used in lieu of OPENXML to transform an XML document into a rowset. Let's take a simple example that uses OPENXML and show how to convert it to use the XML column type and the nodes() function in SQL Server 2005.

To keep our example simple, let's assume we need to perform some sort of processing based on a list of Purchase Orders that is specified in an XML document. The XML document looks like this:

```
<polist>
  <po><ponumber>100</ponumber><podate>2008-09-10</podate></po>
  <po><ponumber>101</ponumber><podate>2008-09-11</podate></po>
</polist>
```

We can transform this XML into a rowset using the following OPENXML:

```
DECLARE @DocHandle int
DECLARE @XmlDocument nvarchar(1000)

SET @XmlDocument = N'<polist>
<po><ponumber>100</ponumber><podate>2008-09-10</podate></po>
<po><ponumber>101</ponumber><podate>2008-09-11</podate></po>
</polist>'

EXEC sp_xml_preparedocument @DocHandle OUTPUT, @XmlDocument

SELECT * FROM OPENXML (@DocHandle, '/polist/po',2)
WITH (ponumber nvarchar(10),
podate datetime)

EXEC sp_xml_removedocument @DocHandle
```

The following is the rowset from executing the above code snippet:

1	100	2008-09-10 00:00:00.000
2	101	2008-09-11 00:00:00.000

Here are the main points from the above code snippet:

- The first step in using OPENXML is to "prepare" the XML document by calling the stored procedure sp_xml_preparedocument which returns an integer that identifies the prepared XML.
- OPENXML is a rowset provider which means you can use it the same as if it were a table. It is essentially a function that takes parameters that specify what to extract from the XML document.
- The final step in using OPENXML is to "free" the prepared XML document by calling the stored procedure sp_xml_removedocument.
- The above xml stored procedures utilize the common MSXML parser component.

The sp_xml_preparedocument stored procedure stores the prepared XML document in SQL Server's internal cache. Calling the sp_xml_removedocument stored procedure is necessary in order to remove the prepared XML document from the cache. According to SQL Server 2005 Books on Line, one-eighth of the total memory available to SQL Server may be used by the MSXML parser. To avoid using up all of this memory, you should call the sp_xml_removedocument stored procedure as soon as you're done with the XML document.

The SQL Server 2005 version of the above code snippet is as follows:

```
DECLARE @xml xml
SET @xml = N'<polist>
              <po><ponumber>100</ponumber><podate>2008-09-10</podate></po>
              <po><ponumber>101</ponumber><podate>2008-09-11</podate></po>
             </polist>'
SELECT
 doc.col.value('ponumber[1]', 'nvarchar(10)') ponumber
,doc.col.value('podate[1]', 'datetime') podate
FROM @xml.nodes('/polist/po') doc(col)
```

Here are the main points from the above code snippet:

- We use the XML column type's nodes() function to transform the XML into a rowset.
- The doc(col) in the nodes() function works like a table and column alias; there is nothing special about the choice of doc and col, i.e. they are not reserved words.
- The rowset is exactly the same as the one produced above using OPENXML.

A minor variation on the SQL Server 2005 example is shown below; in this case the XML document has attributes instead of elements:

```
DECLARE @xml xml
SET @xml = N'<polist>
              <po ponumber="100" podate="2008-09-10" />
              <po ponumber="101" podate="2008-09-11" />
             </polist>'
SELECT
 doc.col.value('@ponumber', 'nvarchar(10)') ponumber
,doc.col.value('@podate', 'datetime') podate
FROM @xml.nodes('/polist/po') doc(col)
```

You reference XML elements using the subscript notation (e.g. ponumber[1]); you reference XML attributes using the '@ponumber' notation.

As far as performance differences between OPENXML and the XML column type with the nodes() function, there doesn't seem to be a crystal clear answer. If you scan through the SQL Server XML Forum you will find some threads where OPENXML is faster than the XML column's nodes() function and vice-versa. As always the best approach may vary depending on your individual circumstances so don't blindly choose the XML column type and the nodes() function over OPENXML.

Next Steps

Next Steps

- Take a look at the OPENXML topic in Books on Line to dig in to the details of this capability.
- We have just scratched the surface on the capabilities of the XML column type in SQL Server 2005; take a look at the XML Data Type Methods topic in Books on Line to familiarize yourself with the other functions such as query(), value(), exist() and modify().