Advanced XML Processing

Introduction

In the previous article, we have seen some examples which demonstrate the XML processing capabilities of SQL Server 2005. We had seen several examples which shows how to read values from an XML variable/field.

In this installment, we will look into the different ways to generate an XML buffer/variable with the results of a query. Most of the times you would need this if you need to call a function/SP which takes an XML variable as a parameter.

The results of a query can be transformed to XML format by using the **FOR XML** TSQL keyword. **FOR XML** should always be used with any of the following keywords: AUTO, RAW, PATH or EXPLICIT. In this article we will see the usages of AUTO and RAW.

Examples

Step 1

```
/*
    Let us create the sample table and populate it.
*/

CREATE TABLE [dbo].[OrderDetails](
    [OrderDetailID] [int] IDENTITY(1,1) NOT NULL,
    [OrderNumber] VARCHAR(10) NOT NULL,
    [ItemNumber] [varchar](20) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [Qty] [int] NULL,
    [Rate] FLOAT NULL,
    [QtyPicked] INT NULL
) ON [PRIMARY]

INSERT INTO OrderDetails (OrderNumber, ItemNumber, Qty, Rate, QtyPicked)
    SELECT '00001', 'A001', 10, 11.25, 0
    UNION SELECT '00001', 'A002', 20, 15, 0
    UNION SELECT '00001', 'A003', 30, 23.75, 0
```

Step 2

```
/*
    The simplest way to return values in XML format is to use the keyword
    FOR XML with AUTO.
*/

SELECT OrderNumber, ItemNumber, Qty FROM OrderDetails FOR XML AUTO

/*
OUTPUT:

```

```
<OrderDetails OrderNumber="00001" ItemNumber="A001" Qty="10" />
<OrderDetails OrderNumber="00001" ItemNumber="A002" Qty="20" />
<OrderDetails OrderNumber="00001" ItemNumber="A003" Qty="30" />
*/
```

Step 3

```
/*
    Though the query returns the results in XML format,
    it is not an XML data type. The result is NVARCHAR.
    To return the results as an XML data type, use the
    keyword TYPE.
*/

SELECT OrderNumber, ItemNumber, Qty FROM OrderDetails FOR XML AUTO, TYPE

/*
OUTPUT:

<OrderDetails OrderNumber="00001" ItemNumber="A001" Qty="10" />
<OrderDetails OrderNumber="00001" ItemNumber="A002" Qty="20" />
<OrderDetails OrderNumber="00001" ItemNumber="A003" Qty="30" />
*/
```

Step 4

```
/*
    By default, SQL SERVER returns the values as attributes.
    Some times you might need the values as nodes. Use the
    ELEMENTS keyword for that.
*/

SELECT OrderNumber, ItemNumber, Qty FROM OrderDetails FOR XML AUTO, TYPE,
ELEMENTS

/*
OUTPUT:

<OrderDetails>
  <OrderNumber>00001</OrderNumber>
  <ItemNumber>A001</ItemNumber>
  <Qty>10</Qty>
</OrderDetails>
<OrderDetails>
  <OrderNumber>00001</OrderNumber>
  <ItemNumber>A002</ItemNumber>
  <Qty>20</Qty>
</OrderDetails>
<OrderDetails>
  <OrderNumber>00001</OrderNumber>
  <ItemNumber>A003</ItemNumber>
  <Qty>30</Qty>
</OrderDetails>
*/
```

Step 5

```
/*
    You will notice that the result does not have a ROOT element.
    A correct XML document/fragment should always have a ROOT element.
    Let us have this added by using the ROOT keyword.
*/
```

```
 6
 7 SELECT OrderNumber, ItemNumber, Qty FROM OrderDetails FOR XML AUTO, TYPE,
ELEMENTS, ROOT
 8
 9 /*
10 OUTPUT:
11
12 <root>
13   <OrderDetails>
14     <OrderNumber>00001</OrderNumber>
15     <ItemNumber>A001</ItemNumber>
16     <Qty>10</Qty>
17   </OrderDetails>
18   <OrderDetails>
19     <OrderNumber>00001</OrderNumber>
20     <ItemNumber>A002</ItemNumber>
21     <Qty>20</Qty>
22   </OrderDetails>
23   <OrderDetails>
24     <OrderNumber>00001</OrderNumber>
25     <ItemNumber>A003</ItemNumber>
26     <Qty>30</Qty>
27   </OrderDetails>
28 </root>
29 */
```

Step 6

```
 1 /*
 2     Well, we have a <root> element now. However, the name is not good.
 3     Let us change the name of the root element.
 4 */
 5
 6 SELECT OrderNumber, ItemNumber, Qty FROM OrderDetails FOR XML AUTO, TYPE,
ELEMENTS, ROOT('orderInfo')
 7
 8 /*
 9 OUTPUT:
10
11 <orderInfo>
12   <OrderDetails>
13     <OrderNumber>00001</OrderNumber>
14     <ItemNumber>A001</ItemNumber>
15     <Qty>10</Qty>
16   </OrderDetails>
17   <OrderDetails>
18     <OrderNumber>00001</OrderNumber>
19     <ItemNumber>A002</ItemNumber>
20     <Qty>20</Qty>
21   </OrderDetails>
22   <OrderDetails>
23     <OrderNumber>00001</OrderNumber>
24     <ItemNumber>A003</ItemNumber>
25     <Qty>30</Qty>
26   </OrderDetails>
27 </orderInfo>
28 */
```

Step 7

```
 1 /*
```

```
    So far, we have seen how to assign a custom name to the <root> element as
    well as a custom name to each row. it is also possible to give a custom
    name to each element by using a column alias. The following example shows
that.
*/

SELECT
    OrderNumber as 'OrderNum',
    ItemNumber as ItemCode,
    Qty as Quantity
FROM OrderDetails FOR XML AUTO, TYPE, ELEMENTS, ROOT('itemInfo')

/*
OUTPUT:

<itemInfo>
  <OrderDetails>
    <OrderNum>00001</OrderNum>
    <ItemCode>A001</ItemCode>
    <Quantity>10</Quantity>
  </OrderDetails>
  <OrderDetails>
    <OrderNum>00001</OrderNum>
    <ItemCode>A002</ItemCode>
    <Quantity>20</Quantity>
  </OrderDetails>
  <OrderDetails>
    <OrderNum>00001</OrderNum>
    <ItemCode>A003</ItemCode>
    <Quantity>30</Quantity>
  </OrderDetails>
</itemInfo>
*/
```

Step 8

```
/*
    So far, we have seen, how to rename the <root> element
    as well as the columns. Now let us see how to rename the
    names of rows. By default the AUTO keyword generates rows
    with the name of the table/alias.
*/

SELECT
    OrderNumber,
    ItemNumber,
    Qty
FROM OrderDetails itemInfo FOR XML AUTO, TYPE, ELEMENTS, ROOT('orderInfo')

/*
OUTPUT:

<orderInfo>
  <itemInfo>
    <OrderNumber>00001</OrderNumber>
    <ItemNumber>A001</ItemNumber>
    <Qty>10</Qty>
  </itemInfo>
  <itemInfo>
    <OrderNumber>00001</OrderNumber>
```

```
    <ItemNumber>A002</ItemNumber>
    <Qty>20</Qty>
  </itemInfo>
  <itemInfo>
    <OrderNumber>00001</OrderNumber>
    <ItemNumber>A003</ItemNumber>
    <Qty>30</Qty>
  </itemInfo>
</orderInfo>
*/
```

Step 9

```
/*
    In the previous example, we had assigned an alias to the table
    in order to customize the element names of each row. This approach works
well.
    However, if the query is complex, some times, it will be very confusing
to
    use alias names just to format the XML node names.

    The AUTO keyword does not provide a way to customize the name of rows. By
using
    the RAW keyword, instead of AUTO, we can easily customize the rows. The
following
    example demonstrates this by using the RAW keyword.
*/

SELECT
    OrderNumber,
    ItemNumber,
    Qty
FROM OrderDetails FOR XML RAW('itemInfo'), TYPE, ELEMENTS, ROOT('orderInfo')

/*
OUTPUT:

<orderInfo>
  <itemInfo>
    <OrderNumber>00001</OrderNumber>
    <ItemNumber>A001</ItemNumber>
    <Qty>10</Qty>
  </itemInfo>
  <itemInfo>
    <OrderNumber>00001</OrderNumber>
    <ItemNumber>A002</ItemNumber>
    <Qty>20</Qty>
  </itemInfo>
  <itemInfo>
    <OrderNumber>00001</OrderNumber>
    <ItemNumber>A003</ItemNumber>
    <Qty>30</Qty>
  </itemInfo>
</orderInfo>
*/
```

Step 10

```
/*
    So far, we are able to format the XML results in the way we wish.
    We are able to rename the root node, rows and element names.
```

```
 4
 5     Now let us look at a different case. When SQL Server generates the
 6     XML results, it will skip the columns which has NULL values. Let us
 7     look at an example.
 8
 9     The code below, updates a column to NULL. Look at the results. The
10     third row does not have <Qty> element.
11 */
12
13 UPDATE OrderDetails SET Qty = NULL WHERE OrderDetailID = 3
14
15 SELECT
16     OrderNumber,
17     ItemNumber,
18     Qty
19 FROM OrderDetails FOR XML RAW('itemInfo'), TYPE, ELEMENTS, ROOT('orderInfo')
20
21 /*
22 OUTPUT:
23
24 <orderInfo>
25   <itemInfo>
26     <OrderNumber>00001</OrderNumber>
27     <ItemNumber>A001</ItemNumber>
28     <Qty>10</Qty>
29   </itemInfo>
30   <itemInfo>
31     <OrderNumber>00001</OrderNumber>
32     <ItemNumber>A002</ItemNumber>
33     <Qty>20</Qty>
34   </itemInfo>
35   <itemInfo>
36     <OrderNumber>00001</OrderNumber>
37     <ItemNumber>A003</ItemNumber>
38   </itemInfo>
39 </orderInfo>
40 */
```

Step 11

```
 1 /*
 2     In the previous example, we have seen that, if a column is
 3     NULL, then the element will not be present in the generated XML.
 4
 5     This can create problems most of the times. For example, if we
 6     need to pass this XML fragment to another function/SP and if
 7     that function/SP expects the element to be present in all the rows
 8     the function/SP will fail.
 9
10     So, we need a way to generate an empty element when the value is NULL.
11     The keyword XSINIL does the trick.
12 */
13
14 SELECT
15     OrderNumber,
16     ItemNumber,
17     Qty
18 FROM OrderDetails FOR XML RAW('itemInfo'), TYPE, ELEMENTS XSINIL, ROOT
('orderInfo')
19
```

```
/*
OUTPUT:

<orderInfo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <itemInfo>
    <OrderNumber>00001</OrderNumber>
    <ItemNumber>A001</ItemNumber>
    <Qty>10</Qty>
  </itemInfo>
  <itemInfo>
    <OrderNumber>00001</OrderNumber>
    <ItemNumber>A002</ItemNumber>
    <Qty>20</Qty>
  </itemInfo>
  <itemInfo>
    <OrderNumber>00001</OrderNumber>
    <ItemNumber>A003</ItemNumber>
    <Qty xsi:nil="true" />
  </itemInfo>
</orderInfo>
*/
```

Conclusions

In this article, we have seen the usage of keywords AUTO and RAW for formatting the results as XML. In the next article we will see the usage PATH and EXPLICIT.