http://www.sqlservercentral.com/articles/SS2K5+-+XML/3022/
Printed 2007/11/07 02:20PM

XML Workshop - FOR XML PATH

By Jacob Sebastian, 2007/05/22

Introduction

In the last 2 sessions on XML processing, we have seen various examples of reading and generating XML data. In case you have not seen the last two sessions, you could find them here.

1. Advanced XML Processing
   This article shows several examples that generates XML data using the *FOR XML* keyword. This article demonstrated examples using *AUTO* and *RAW* modes.
2. More Advanced XML Processing Examples
   This article shows several examples that queries/retrieves values from an XML variable/column

In this session, we will see XML processing examples using *PATH* mode. I would suggest you read the previous article which presented several examples using *RAW* and *AUTO* modes.

FOR XML PATH

RAW and *AUTO* modes are good enough for most of the XML formatting requirements. *PATH* mode provides a few additional formatting capabilities that we will examine through the examples given below.

Example 1: Generating a hierarchy of XML nodes

```
    1 /*
    2     Using the PATH mode, you can generate hierarchies of XML nodes. The XML
generation is
    3     controlled by the ALIAS of the column. In the example given below, a
new node <item> is
    4     created which holds "ItemNumber" and "Quantity". Note the slash ("/")
used within the
    5     column name, which really controls the XML generation.
    6 */
    7
    8 SELECT
    9     OrderNumber AS 'orderNumber',
   10     ItemNumber AS 'item/itemNumber',
   11     Qty AS 'item/Quantity'
   12 FROM OrderDetails FOR XML PATH('orderInfo'), TYPE, ELEMENTS, ROOT('order')
   13
   14 /*
   15 OUTPUT:
   16
   17 <order>
   18   <orderInfo>
   19     <orderNumber>00001</orderNumber>
   20     <item>
   21       <itemNumber>A001</itemNumber>
   22       <Quantity>10</Quantity>
   23     </item>
```

```
  </orderInfo>
  <orderInfo>
    <orderNumber>00001</orderNumber>
    <item>
      <itemNumber>A002</itemNumber>
      <Quantity>20</Quantity>
    </item>
  </orderInfo>
</order>
*/
```

Example 2: Generating a List of Values

```
/*
    We just saw how the PATH keyword can be used to re-structure the XML
    generated by FOR XML. Let us now look at a few other scenarios where
    the PATH keyword is useful.

    One of the situations, where the PATH keyword can come handy is to
create a list
    of values. The following query returns a list of Item Numbers from the
Order table.
*/

SELECT
    ItemNumber AS 'data()'
FROM
    OrderDetails
FOR XML PATH('')

/*
Output:

A001 A002 A003
*/

/*
    You will note that the above query returns the results as a SPACE
separated
    value. Most of the times we work with COMMA separated values, not SPACE
separated
    values. So let us see how to generate a COMMA separated value list.

    There is no direct way to do this. But we can do it with a small trick.
Look at
    the following example
*/

SELECT
    ',' + ItemNumber AS 'data()'
FROM
    OrderDetails
FOR XML PATH('')

/*
Output:

,A001 ,A002 ,A003
*/

/*
    Well, we are almost done. But it is not really good still. You will
note that the
    string starts with a COMMA. Lets use the STUFF function to remove this.
```

```
Here is the
   46     final query.
   47 */
   48
   49 SELECT STUFF(
   50     (SELECT
   51         ',' + ItemNumber AS 'data()'
   52     FROM
   53         OrderDetails
   54     FOR XML PATH('')),1,1,'')
   55
   56 /*
   57     Output:
   58
   59     A001 ,A002 ,A003
   60 */
   61
   62 /*
   63     I just found Jamie's blog which suggests that using the above approach
is
   64     better than the other options to create a list of values. You can find
the blog
   65     here:
http://blogs.conchango.com/jamiethomson/archive/2007/04/05/T_2D00_SQL_3A00_-A-
T_2D00_SQL-Poser--_2D00_--Part-3.aspx
   66 */
```

PATH v/s AUTO and RAW

PATH provides more control over the hierarchy by manipulating the column alias. This would be very handy most of the times. *EXPLICIT* provides more control over the XML generation, but the usage is pretty complex. I will present a step by step approach to master the *EXPLICIT* usage in the next article.

Let us take a trial assignment and see how each XML mode (AUTO, RAW and PATH) can be used to accomplish the given task. Let us assume that we have 3 tables: Customer, City and Country. We need to retrieve data from the above tables, in the XML structure given below.

```
 1 <customersByRegion>
 2   <country name="USA" currency="US Dollars">
 3     <city name="NY">
 4       <customer id="MK" name="John Mark" phone="111-111-1111"/>
 5       <customer id="WS" name="Will Smith" phone="222-222-2222"/>
 6     </city>
 7     <city name="NJ">
 8       <customer id="EN" name="Elizabeth Lincoln" phone="333-333-3333"/>
 9     </city>
10   </country>
11   <country name="England" currency="Pound Sterling">
12     <city name="London">
13       <customer id="TH" name="Thomas Hardy" phone="444-444-4444"/>
14     </city>
15   </country>
16   <country name="India" currency="Rupees">
17     <city name="New Delhi">
18       <customer id="JS" name="Jacob Sebastian" phone="555-555-5555"/>
19     </city>
20   </country>
21 </customersByRegion>
```

Let us run the script to generate the tables and insert the data that we need.

```
CREATE TABLE Countries (CountryID INT, CountryName VARCHAR(20), Currency
VARCHAR(20))
CREATE TABLE Cities (CityID INT, CityName VARCHAR(20), CountryID INT)
CREATE TABLE Customers (CustomerNumber VARCHAR(2), CustomerName VARCHAR
(40), Phone VARCHAR(20), CityID INT)

INSERT INTO Countries(CountryID, CountryName, Currency)
    SELECT 1 AS CountryID, 'USA' AS CountryName, 'US Dollars' as Currency
UNION
    SELECT 2, 'England', 'Pound Sterling' UNION
    SELECT 3, 'India', 'Rupee'

INSERT INTO Cities(CityID, CityName, CountryID)
    SELECT 1 AS CityID, 'NY' AS CityName, 1 AS CountryID UNION
    SELECT 2, 'NJ', 1 UNION
    SELECT 3, 'London', 2 UNION
    SELECT 4, 'New Delhi', 3

INSERT INTO Customers(CustomerNumber, CustomerName, Phone, CityID)
    SELECT 'MK' AS CustomerNumber, 'John Mark' AS CustomerName, '111-111-
1111' AS Phone, 1 AS CityID UNION
    SELECT 'WS', 'Will Smith', '222-222-2222', 1 UNION
    SELECT 'EN', 'Elizabeth Lincoln', '333-333-3333', 2 UNION
    SELECT 'TH', 'Thomas Hardy', '444-444-4444', 3 UNION
    SELECT 'JS', 'Jacob Sebastian', '555-555-5555', 4
```

Let us start with *PATH* and see if we can generate the above XML structure with it. Let us write the first version of our TSQL as follows.

```
SELECT
        Country.CountryName AS 'country/name',
        Country.Currency AS 'country/currency',
        City.CityName AS 'country/city/name',
        Customer.CustomerNumber AS 'country/city/customer/id',
        Customer.CustomerName AS 'country/city/customer/name',
        Customer.Phone AS 'country/city/customer/phone'
    FROM
        Customers Customer
        INNER JOIN Cities City ON (City.CityID = Customer.CityID)
        INNER JOIN Countries Country ON (Country.CountryID =
City.CountryID)
    ORDER BY CountryName, CityName
    FOR XML PATH

/*
OUTPUT:

<row>
  <country>
    <name>England</name>
    <currency>Pound Sterling</currency>
    <city>
      <name>London</name>
      <customer>
        <id>TH</id>
        <name>Thomas Hardy</name>
        <phone>444-444-4444</phone>
      </customer>
    </city>
  </country>
</row>
<row>
  <country>
```

```
    <name>India</name>
    <currency>Rupee</currency>
    <city>
      <name>New Delhi</name>
      <customer>
        <id>JS</id>
        <name>Jacob Sebastian</name>
        <phone>555-555-5555</phone>
      </customer>
    </city>
  </country>
</row>
<row>
...
</row>
...
*/
```

Well, not very good. You must have noticed that the format is not what we needed. We need the values as *Attributes*, but this query returns values as XML *nodes*. So, let us look at the following query which generates results where values are presented as *Attributes*.

```
SELECT
        Country.CountryName AS 'country/@name',
        Country.Currency AS 'country/@currency',
        City.CityName AS 'country/city/@name',
        Customer.CustomerNumber AS 'country/city/customer/@id',
        Customer.CustomerName AS 'country/city/customer/@name',
        Customer.Phone AS 'country/city/customer/@phone'
    FROM
        Customers Customer
        INNER JOIN Cities City ON (City.CityID = Customer.CityID)
        INNER JOIN Countries Country ON (Country.CountryID =
City.CountryID)
    ORDER BY CountryName, CityName
    FOR XML PATH(''), ROOT('CustomersByRegion')

/*
<CustomersByRegion>
  <country name="England" currency="Pound Sterling">
    <city name="London">
      <customer id="TH" name="Thomas Hardy" phone="444-444-4444" />
    </city>
  </country>
  <country name="India" currency="Rupee">
    <city name="New Delhi">
      <customer id="JS" name="Jacob Sebastian" phone="555-555-5555" />
    </city>
  </country>
  <country name="USA" currency="US Dollars">
    <city name="NJ">
      <customer id="EN" name="Elizabeth Lincoln" phone="333-333-3333" />
    </city>
  </country>
  <country name="USA" currency="US Dollars">
    <city name="NY">
      <customer id="MK" name="John Mark" phone="111-111-1111" />
    </city>
  </country>
  <country name="USA" currency="US Dollars">
    <city name="NY">
      <customer id="WS" name="Will Smith" phone="222-222-2222" />
    </city>
```

```
41   </country>
42 </CustomersByRegion>
43 */
```

We have results very close to what we need. But we are missing something. You will notice that the results are not grouped. “USA” has 3 nodes in the XML data generated by the query. This should be grouped into a single node. However, PATH does not provide a way to do that. Let us look into other modes to see if they can help in this scenario.

Let us move on to *AUTO* mode. The following code tries to generate the required XML structure using the *AUTO* mode.

```
 1 SELECT
 2     Country.CountryName AS [name],
 3     Country.Currency,
 4     City.CityName AS [name],
 5     Customer.CustomerNumber AS [id],
 6     Customer.CustomerName AS [name],
 7     Customer.Phone
 8 FROM
 9     Customers Customer
10     INNER JOIN Cities City ON (City.CityID = Customer.CityID)
11     INNER JOIN Countries Country ON (Country.CountryID = City.CountryID)
12 ORDER BY CountryName, CityName
13 FOR XML AUTO
14
15 /*
16 OUTPUT:
17
18 <Country name="England" Currency="Pound Sterling">
19   <City name="London">
20     <Customer id="TH" name="Thomas Hardy" Phone="444-444-4444" />
21   </City>
22 </Country>
23 <Country name="India" Currency="Rupee">
24   <City name="New Delhi">
25     <Customer id="JS" name="Jacob Sebastian" Phone="555-555-5555" />
26   </City>
27 </Country>
28 <Country name="USA" Currency="US Dollars">
29   <City name="NJ">
30     <Customer id="EN" name="Elizabeth Lincoln" Phone="333-333-3333" />
31   </City>
32   <City name="NY">
33     <Customer id="MK" name="John Mark" Phone="111-111-1111" />
34     <Customer id="WS" name="Will Smith" Phone="222-222-2222" />
35   </City>
36 </Country>
37 */
```

Here again, we are able to get perfect XML except the ROOT element. XML is not always useful without the REQUIRED root element. AUTO mode does not provide a way to specify a ROOT element for the resultant XML. Let us see if we can find a workaround to get this done. The following code shows a workaround to fix the above limitation.

```
1 SELECT CAST ('<CustomersByRegion>' + (SELECT
2     Country.CountryName AS [name],
3     Country.Currency,
4     City.CityName AS [name1],
5     Customer.CustomerNumber AS [id],
6     Customer.CustomerName AS [name2],
```

```
    Customer.Phone
FROM
    Customers Customer
    INNER JOIN Cities City ON (City.CityID = Customer.CityID)
    INNER JOIN Countries Country ON (Country.CountryID = City.CountryID)
ORDER BY CountryName, CityName
FOR XML AUTO) + '</CustomersByRegion>' AS XML)

/*
<CustomersByRegion>
  <Country name="England" Currency="Pound Sterling">
    <City name1="London">
      <Customer id="TH" name2="Thomas Hardy" Phone="444-444-4444" />
    </City>
  </Country>
  <Country name="India" Currency="Rupee">
    <City name1="New Delhi">
      <Customer id="JS" name2="Jacob Sebastian" Phone="555-555-5555" />
    </City>
  </Country>
  <Country name="USA" Currency="US Dollars">
    <City name1="NJ">
      <Customer id="EN" name2="Elizabeth Lincoln" Phone="333-333-3333" />
    </City>
    <City name1="NY">
      <Customer id="MK" name2="John Mark" Phone="111-111-1111" />
      <Customer id="WS" name2="Will Smith" Phone="222-222-2222" />
    </City>
  </Country>
</CustomersByRegion>
*/
```

Now let us see if we can use *RAW* mode for generating the result that we need. The *RAW* mode does not provide a way to generate a hierarchy of XML nodes. However, a combination of RAW and AUTO can be used to generate the results that we need. The following code demonstrates it.

```
SELECT CAST(
    (SELECT
        Country.CountryName AS [name],
        Country.Currency,
        City.CityName AS [name],
        Customer.CustomerNumber AS [id],
        Customer.CustomerName AS [name],
        Customer.Phone
    FROM
        Customers Customer
        INNER JOIN Cities City ON (City.CityID = Customer.CityID)
        INNER JOIN Countries Country ON (Country.CountryID =
City.CountryID)
    ORDER BY CountryName, CityName
    FOR XML AUTO) AS XML)
FOR XML RAW('CustomersByRegion')

/*
OUTPUT:

<CustomersByRegion>
  <Country name="England" Currency="Pound Sterling">
    <City name="London">
      <Customer id="TH" name="Thomas Hardy" Phone="444-444-4444" />
    </City>
  </Country>
  <Country name="India" Currency="Rupee">
```

```
27      <City name="New Delhi">
28        <Customer id="JS" name="Jacob Sebastian" Phone="555-555-5555" />
29      </City>
30    </Country>
31    <Country name="USA" Currency="US Dollars">
32      <City name="NJ">
33        <Customer id="EN" name="Elizabeth Lincoln" Phone="333-333-3333" />
34      </City>
35      <City name="NY">
36        <Customer id="MK" name="John Mark" Phone="111-111-1111" />
37        <Customer id="WS" name="Will Smith" Phone="222-222-2222" />
38      </City>
39    </Country>
40  </CustomersByRegion>
41  */
```

Conclusions

We have seen several examples of *AUTO*, *RAW* and *PATH* modes and I guess the usage of the above modes might be clearly demonstrated in the examples we have seen so far. There is one more mode, *EXPLICIT*, which I have not presented yet. The *EXPLICIT* mode provides much more control over the structure of the generated XML. However, it is much more complex to use. In the next article, I will present a step-by-step tutorial to understand the usage of the *EXPLICIT* mode.
