http://www.sqlservercentral.com/articles/61821/
Printed 2008/02/13 05:12AM

XML Workshop XIV - Generating an XML Tree

By Jacob Sebastian, 2007/12/31

Introduction

I guess some of you might have come across the need for a hierarchical data structure in the day-to-day programming life. The most common task is the creation of a menu. Many applications store the menu data in a table with parent-child relationships duly applied. At run time, the menu data is read from the table and depending upon the user rights, filtered and the menu is created. To create a menu, we need a tree-like structure. We could do this easily with recursive calls to the data source (Local Data Set or from the Database Server).

There are many other situations where we need to work with a tree-structured data. But we usually get data in the shape of relational tables (Data Sets and Data Tables) and we might often need to do recursive calls to format the data to the shape that we need.

Since XML is good at managing and storing hierarchical data, at times it might make things easier if we can work with data in XML format. Many of the *User Interface Controls* available today (in different application development platforms) support data binding to an XML data source. In those cases, you might need to create an XML document from the data stored in one or more relational tables. So the question now is, *how to generate an XML tree from relational data*.

Generating an XML tree

If you need a tree structure with unlimited levels, a recursive function might be one choice. There is a an interesting white paper available at SQL Server Developer Center at MSDN web site. I found it to be pretty good. Towards the end of the document, it shows an approach which generates an XML tree by using a recursive function. I am sure that you will appreciate it.

Recently, in one of the MSDN forums, some one posted a question asking for an alternate way. He did not want to use a recursive function. He wanted to see if there is a way to generate an XML tree with a TSQL Query.

Here is the source table.

ID	Parent	Name
1	NULL	car
2	1	engine
3	1	body
4	3	door
5	3	fender
6	4	window
7	2	piston

Here is the required result.

```
<Part id="1" name="car">
  <Part id="2" name="engine">
    <Part id="7" name="piston" />
  </Part>
  <Part id="3" name="body">
    <Part id="4" name="door">
      <Part id="6" name="window" />
    </Part>
    <Part id="5" name="fender" />
  </Part>
</Part>
```

FOR XML EXPLICIT

We have seen how to use *FOR XML EXPLICIT* in XML WORKSHOP IV. I have also written a bit about EXPLICIT in my blog too. EXPLICIT gives good control over the structure of the XML being generated. The tough part is that EXPLICIT expects the data in a specific structure. We need to generate the data in the required structure and pass it to the XML processing engine. In this session, we will generate an XML tree using EXPLICIT.

Writing the code

Let us start writing the code. Let us create a sample table and populate it. Here is the code to create the table and populate it.

```
CREATE TABLE PARTS(
    id int,
     parent int,
    name nvarchar(500))
GO
INSERT INTO PARTS
  SELECT 1, NULL, N'car'
  UNION
  SELECT 2, 1, N'engine'
  UNION
  SELECT 3, 1, N'body'
  UNION
  SELECT 4, 3, N'door'
  UNION
  SELECT 5, 3, N'fender'
  UNION
  SELECT 6, 4, N'window'
  UNION
  SELECT 7, 2, N'piston'
```

Note: The above code is taken from the white paper mentioned earlier.

Let us now look at what kind of an input result set needs to be passed to FOR XML EXPLICIT. To generate the kind of XML structure that we need, we need to have input in the following structure.

Tag	Parent	Part!1!id	Part!1!name	Part!2!id	Part!2!name	Part!3!id	Part!3!name	Part!4!id	Part!4!name
1	NULL	1	car	NULL	NULL	NULL	NULL	NULL	NULL
2	1	NULL	NULL	2	engine	NULL	NULL	NULL	NULL
3	2	NULL	NULL	NULL	NULL	7	piston	NULL	NULL
2	1	NULL	NULL	3	body	NULL	NULL	NULL	NULL
3	2	NULL	NULL	NULL	NULL	4	door	NULL	NULL
4	3	NULL	NULL	NULL	NULL	NULL	NULL	6	window
3	2	NULL	NULL	NULL	NULL	5	fender	NULL	NULL

Once we have the results in the above structure, the rest of the work will be done by the EXPLICIT operator. Let us now look at the query that will generate the result structure that we need. The *parts* table has a hierarchical relationship set between the records using the *ID* and *Parent* . We need to generate the results in such a way that the *Tag* column should contain the level or depth of the current node. To calculate the depth of each record, we need to use some sort of recursion. Since we do not want to use a recursive function, we will use a CTE. By using a CTE, we can write a recursive query in SQL Server 2005.

```
;WITH Parts1
AS
(
    SELECT
        0 AS [Level],
        [id],
        Parent,
        [name],
```

```
        CAST( [id] AS VARBINARY(MAX)) AS Sort
    FROM parts
    WHERE Parent IS NULL

    UNION ALL
    SELECT
        [Level] + 1,
        p.[id],
        p.Parent,
        p.[name],
        CAST( SORT + CAST(p.[id] AS BINARY(4)) AS VARBINARY(MAX))
    FROM parts p
    INNER JOIN Parts1 c ON p.parent = c.id
)
SELECT * FROM Parts1
```

The above query generates the following result.

Level	ID	Parent	Name	Sort
0	**1**	**NULL**	**car**	**0x00000001**
1	2	1	engine	0x0000000100000002
1	3	1	body	0x0000000100000003
2	4	3	door	0x000000010000000300000004
2	5	3	fender	0x000000010000000300000005
3	6	4	window	0x00000001000000030000000400000006
2	7	2	piston	0x000000010000000200000007

The first column shows the depth level. The last column is to keep the records in the correct order after we process all the records recursively. As you could see in the code, each level will add a binary string to the column used for the sorting. Sorting is very important in FOR XML EXPLICIT. The XML result set will be processed exactly in the same order as the rows appear in it. So we need to make sure that the result set follow the correct order that we need in the resultant XML. Let us move to the next step.

```
;WITH Parts1
AS
(
    SELECT
        0 AS [Level],
        [id],
        Parent,
        [name],
        CAST( [id] AS VARBINARY(MAX)) AS Sort
    FROM parts
    WHERE Parent IS NULL

    UNION ALL
    SELECT
        [Level] + 1,
        p.[id],
        p.Parent,
        p.[name],
        CAST( SORT + CAST(p.[id] AS BINARY(4)) AS VARBINARY(MAX))
    FROM parts p
    INNER JOIN Parts1 c ON p.parent = c.id
),
Parts2 AS (
    SELECT
        [Level] + 1 AS Tag,
        [id],
        Parent,
        [name],
        sort
    FROM Parts1
)
SELECT * FROM Parts2
```

Tag	ID	Parent	Name	Sort

1	1	NULL	car	0x00000001
2	2	1	engine	0x0000000100000002
2	3	1	body	0x0000000100000003
3	4	3	door	0x000000010000000300000004
3	5	3	fender	0x000000010000000300000005
4	6	4	window	0x00000001000000030000000400000006
3	7	2	piston	0x000000010000000200000007

We did not do anything complex processing at this stage. We just renamed the *Level* to *Tag*. Let us move to the next step.

```
;WITH Parts1
AS
(
    SELECT
        0 AS [Level],
        [id],
        Parent,
        [name],
        CAST( [id] AS VARBINARY(MAX)) AS Sort
    FROM parts
    WHERE Parent IS NULL

    UNION ALL
    SELECT
        [Level] + 1,
        p.[id],
        p.Parent,
        p.[name],
        CAST( SORT + CAST(p.[id] AS BINARY(4)) AS VARBINARY(MAX))
    FROM parts p
    INNER JOIN Parts1 c ON p.parent = c.id
),
Parts2 AS (
    SELECT
        [Level] + 1 AS Tag,
        [id],
        Parent,
        [name],
        sort
    FROM Parts1
),
Parts3 AS (
    SELECT
        *,
        (SELECT Tag FROM Parts2 r2 WHERE r2.ID = r1.parent) AS ParentTag
    FROM Parts2 r1
)
SELECT * FROM Parts3
```

Here is the result set that we have at this stage.

Tag	ID	Parent	Name	Sort	Parent Tag
1	1	NULL	car	0x00000001	NULL
2	2	1	engine	0x0000000100000002	1
2	3	1	body	0x0000000100000003	1
3	4	3	door	0x000000010000000300000004	2
3	5	3	fender	0x000000010000000300000005	2
4	6	4	window	0x00000001000000030000000400000006	3
3	7	2	piston	0x000000010000000200000007	2

Note that we have generated the *Parent Tag* at this stage. We can now ignore *ID* and *ParentID* and work with *Tag* and *ParentTag*.

At this stage we have everything that we need to write the query for FOR XML EXPLICIT. Let us now write the query for

the final step.

```
;WITH Parts1
AS
(
    SELECT
        0 AS [Level],
        [id],
        Parent,
        [name],
        CAST( [id] AS VARBINARY(MAX)) AS Sort
    FROM parts
    WHERE Parent IS NULL

    UNION ALL
    SELECT
        [Level] + 1,
        p.[id],
        p.Parent,
        p.[name],
        CAST( SORT + CAST(p.[id] AS BINARY(4)) AS VARBINARY(MAX))
    FROM parts p
    INNER JOIN Parts1 c ON p.parent = c.id
),
Parts2 AS (
    SELECT
        [Level] + 1 AS Tag,
        [id],
        Parent,
        [name],
        sort
    FROM Parts1
),
Parts3 AS (
    SELECT
        *,
        (SELECT Tag FROM Parts2 r2 WHERE r2.ID = r1.parent) AS ParentTag
    FROM Parts2 r1
)

SELECT
Tag,
ParentTag as Parent,
CASE WHEN tag = 1 THEN [id] ELSE NULL END AS 'Part!1!id',
CASE WHEN tag = 1 THEN [name] ELSE NULL END AS 'Part!1!name',
CASE WHEN tag = 2 THEN [id] ELSE NULL END AS 'Part!2!id',
CASE WHEN tag = 2 THEN [name] ELSE NULL END AS 'Part!2!name',
CASE WHEN tag = 3 THEN [id] ELSE NULL END AS 'Part!3!id',
CASE WHEN tag = 3 THEN [name] ELSE NULL END AS 'Part!3!name',
CASE WHEN tag = 4 THEN [id] ELSE NULL END AS 'Part!4!id',
CASE WHEN tag = 4 THEN [name] ELSE NULL END AS 'Part!4!name'
FROM Parts3
ORDER BY sort
```

Here is the result

Tag	Parent	Part!1!id	Part!1!name	Part!2!id	Part!2!name	Part!3!id	Part!3!name	Part!4!id	Part!4!name
1	**NULL**	**1**	**car**	**NULL**	**NULL**	**NULL**	**NULL**	**NULL**	**NULL**
2	1	NULL	NULL	2	engine	NULL	NULL	NULL	NULL
3	2	NULL	NULL	NULL	NULL	7	piston	NULL	NULL
2	1	NULL	NULL	3	body	NULL	NULL	NULL	NULL
3	2	NULL	NULL	NULL	NULL	4	door	NULL	NULL
4	3	NULL	NULL	NULL	NULL	NULL	NULL	6	window
3	2	NULL	NULL	NULL	NULL	5	fender	NULL	NULL

This is what we needed. Let us apply FOR XML EXPLICIT to this query and see the results.

```
;WITH Parts1
```

```
AS
(
    SELECT
        0 AS [Level],
        [id],
        Parent,
        [name],
        CAST( [id] AS VARBINARY(MAX)) AS Sort
    FROM parts
    WHERE Parent IS NULL

    UNION ALL
    SELECT
        [Level] + 1,
        p.[id],
        p.Parent,
        p.[name],
        CAST( SORT + CAST(p.[id] AS BINARY(4)) AS VARBINARY(MAX))
    FROM parts p
    INNER JOIN Parts1 c ON p.parent = c.id
),
Parts2 AS (
    SELECT
        [Level] + 1 AS Tag,
        [id],
        Parent,
        [name],
        sort
    FROM Parts1
),
Parts3 AS (
    SELECT
        *,
        (SELECT Tag FROM Parts2 r2 WHERE r2.ID = r1.parent) AS ParentTag
    FROM Parts2 r1
)

SELECT
Tag,
ParentTag as Parent,
CASE WHEN tag = 1 THEN [id] ELSE NULL END AS 'Part!1!id',
    CASE WHEN tag = 1 THEN [name] ELSE NULL END AS 'Part!1!name',
CASE WHEN tag = 2 THEN [id] ELSE NULL END AS 'Part!2!id',
    CASE WHEN tag = 2 THEN [name] ELSE NULL END AS 'Part!2!name',
CASE WHEN tag = 3 THEN [id] ELSE NULL END AS 'Part!3!id',
    CASE WHEN tag = 3 THEN [name] ELSE NULL END AS 'Part!3!name',
CASE WHEN tag = 4 THEN [id] ELSE NULL END AS 'Part!4!id',
    CASE WHEN tag = 4 THEN [name] ELSE NULL END AS 'Part!4!name'
FROM Parts3
ORDER BY sort
FOR XML EXPLICIT
```

Here is the XML result

```
<Part id="1" name="car">
  <Part id="2" name="engine">
    <Part id="7" name="piston" />
  </Part>
  <Part id="3" name="body">
    <Part id="4" name="door">
      <Part id="6" name="window" />
    </Part>
    <Part id="5" name="fender" />
  </Part>
</Part>
```

Adding More Levels

What do we do if we have more levels? Well, this solution is not good if you cannot guess the maximum levels that you

are expecting. Most of the times we might need 5 or 6 levels maximum. In such cases this approach might work. But if you really need to work with more levels or unlimited levels, you might go with some other options, probably with a recursive function.

At present, this query is written for maximum 4 levels. Let us see how to add support for one more level. Here is how we could extend this query to support one more level. I have marked the changes in yellow which will show you how to add more levels. Using that approach, you can add any number of levels to the current query.

```
;WITH Parts1
AS
(
    SELECT
        0 AS [Level],
        [id],
        Parent,
        [name],
        CAST( [id] AS VARBINARY(MAX)) AS Sort
    FROM parts
    WHERE Parent IS NULL

    UNION ALL
    SELECT
        [Level] + 1,
        p.[id],
        p.Parent,
        p.[name],
        CAST( SORT + CAST(p.[id] AS BINARY(4)) AS VARBINARY(MAX))
    FROM parts p
    INNER JOIN Parts1 c ON p.parent = c.id
),
Parts2 AS (
    SELECT
        [Level] + 1 AS Tag,
        [id],
        Parent,
        [name],
        sort
    FROM Parts1
),
Parts3 AS (
    SELECT
        *,
        (SELECT Tag FROM Parts2 r2 WHERE r2.ID = r1.parent) AS ParentTag
    FROM Parts2 r1
)

SELECT
Tag,
ParentTag as Parent,
CASE WHEN tag = 1 THEN [id] ELSE NULL END AS 'Part!1!id',
    CASE WHEN tag = 1 THEN [name] ELSE NULL END AS 'Part!1!name',
CASE WHEN tag = 2 THEN [id] ELSE NULL END AS 'Part!2!id',
    CASE WHEN tag = 2 THEN [name] ELSE NULL END AS 'Part!2!name',
CASE WHEN tag = 3 THEN [id] ELSE NULL END AS 'Part!3!id',
    CASE WHEN tag = 3 THEN [name] ELSE NULL END AS 'Part!3!name',
CASE WHEN tag = 4 THEN [id] ELSE NULL END AS 'Part!4!id',
    CASE WHEN tag = 4 THEN [name] ELSE NULL END AS 'Part!4!name',
CASE WHEN tag = 5 THEN [id] ELSE NULL END AS 'Part!5!id',
    CASE WHEN tag = 5 THEN [name] ELSE NULL END AS 'Part!5!name'
FROM Parts3
ORDER BY sort
FOR XML EXPLICIT
```

Conclusions

The approach presented in this session may not be the best possible solution. There must be other ways to do this too. I have not tested the performance factors. Based up on your specific requirement, you should make a decision about the

approach to be taken. The intention of this article is to show a method, that some might find dirty and others might feel handy. The purpose of XML Workshop is to show what we could do with SQL Server XML. It does not recommend any specific method, instead it tries to present the 'XML way of doing' and you should make your own decision whether to take an XML approach or NON XML approach.
