CTE Hierarchy compared to the alternative

After my **CTE** presentation at a while back I was asked many questions, and received several great suggestions from people.

One question was how does the performance compare between a recursive **CTE** to generate a hierarchical tree path listing and a query using self JOINs and UNION ALL to generate similar results. To test this I created a simple table for departments in an online store that contained the following rows.

The I wrote two queries to generate a tree path for all departments in the list.

First the CTE Query:

```
-- Performance Differences
-- Remember Ctrl+M to turn on Execution Plan
-- Recursive CTE compared to Multiple Self Joins
;WITH departmentcte(deptid, department, parent, LEVEL, treepath) AS
( SELECT id AS deptid, department, parent, 0 AS LEVEL,
CAST(department AS VARCHAR(1024)) AS treepath
FROM departments
WHERE parent IS NULL
UNION ALL -- and now for the recursive part
SELECT d.id AS deptid, d.department, d.parent,
departmentcte.LEVEL + 1 AS LEVEL,
CAST(departmentcte.treepath + ' -> ' +
CAST(d.department AS VARCHAR(1024))
AS VARCHAR(1024)) AS treepath
FROM departments d
INNER JOIN departmentcte
ON departmentcte.deptid = d.parent
)
SELECT deptid, treepath
FROM departmentcte
ORDER BY treepath;
```

Then the Non CTE Query (ugly):

```
-- Multiple Self Joins Unioned
-- Difficult to display Parent categories
SELECT d.id AS deptid, d.department AS treepath
FROM departments d
WHERE d.parent IS NULL
UNION ALL
SELECT da2.id AS deptid,
da1.department + ' -> ' +
da2.department AS treepath
FROM departments da1
INNER JOIN departments da2 ON da1.id = da2.parent
WHERE da1.parent IS NULL
UNION ALL
SELECT db3.id AS deptid,
db1.department + ' -> ' +
db2.department + ' -> ' +
db3.department AS treepath
FROM departments db1
INNER JOIN departments db2 ON db1.id = db2.parent
INNER JOIN departments db3 ON db2.id = db3.parent
WHERE db1.parent IS NULL
UNION ALL
SELECT dc3.id AS deptid,
dc1.department + ' -> ' +
dc2.department + ' -> ' +
dc3.department + ' -> ' +
dc4.department AS treepath
FROM departments dc1
INNER JOIN departments dc2 ON dc1.id = dc2.parent
INNER JOIN departments dc3 ON dc2.id = dc3.parent
INNER JOIN departments dc4 ON dc3.id = dc4.parent
WHERE dc1.parent IS NULL
UNION ALL
SELECT dd3.id AS deptid,
dd1.department + ' -> ' +
dd2.department + ' -> ' +
dd3.department + ' -> ' +
dd4.department + ' -> ' + dd5.department AS treepath
```

```
FROM departments dd1
INNER JOIN departments dd2 ON dd1.id = dd2.parent
INNER JOIN departments dd3 ON dd2.id = dd3.parent
INNER JOIN departments dd4 ON dd3.id = dd4.parent
INNER JOIN departments dd5 ON dd4.id = dd5.parent
WHERE dd1.parent IS NULL

ORDER BY treepath
;
```

The two queries were confirmed to produce the same results:

But when looking at the actual execution plan for the queries, there is a very big difference. The self JOIN, and UNION ALL query took 13 times as long to run as the CTE query did.

Results:

The CTE compared to the self JOIN with UNION ALL is a double win, first the CTE is much easier to read, second the CTE is much faster with 1/13th of the time it took to execute the self JOIN, UNION ALL solution. Also in the examples shown we showed the queries were used to show up to 5 levels of categories. If we wanted to go further than 5 levels, the difference gets even worse between the CTE and the self JOIN, UNION ALL solution.

Related Links:

- My Book on Common Table Expressions
- cte_demo sample database

Common Table Expressions Book

If you enjoyed this posting, and want to learn more about common table expressions, please take a look at my book on CTE's at Amazon.com. The book is titled Common Table Expressions - Joes 2 Pros® - A CTE Tutorial on Performance, Stored Procedures, Recursion, Nesting and the use of Multiple CTEs.