Introduction to Recursive CTEs

Day 7 of Common Table Expression Month (June) at SteveStedman.com, today we will be taking a look at the introduction to recursive CTEs.

These queries will be using that database that was set up in a previous posting on the CTE_DEMO Sample Database, if you haven't set up the sample database, download it and set it up now.

The recursive feature of CTEs is perhaps one of the most powerful things you can do with a Common Table Expression

What is Recursion

Recursion is a concept in programming where a function calls itself. Many of the typical C programming interview questions around data structures like trees, linked lists, and other structures often times use recursion. For instance in the following T-SQL example, the scalar function Fibonacci calculates the Fibonacci sequence using recursion, by calling itself. This is accomplished by calling the function name inside of the body of the function.

To call the function you simply include in a select statement, with the first parameter being the number of Fibonacci numbers to calculate, and the second and third parameters are always 0 and 1. The second and third parameters are set to 0 and 1 to prime the recursive function, and are used internally to pass the recursive call the current and previous values.

Which produces the following output using SSMS:

SQLQuery5.sql - server2012.Northwind (sa (58))*

```
CREATE FUNCTION dbo.Fibonacci (@Num integer, @prev integer, @next integer)
RETURNS VARCHAR(4000)
AS
BEGIN
    DECLARE @returnValue as VARCHAR (4000) = cast(@prev as varchar(4000));
    IF (@Num > 0)
    BEGIN
        IF (LEN(@returnValue) > 0)
        BEGIN
            SET @returnValue = @returnValue + ',';
        END
        SET @returnValue = @returnValue + dbo.Fibonacci(@Num - 1, @next, @next + @prev) ;
    END

    RETURN @returnValue;
END
GO

select dbo.Fibonacci(10, 0, 1);
```

100 %

Results | Messages

	(No column name)
1	0,1,1,2,3,5,8,13,21,34,55

Query executed successfully. | server2012 (11.0 RTM) | sa (58) | Northwind | 00:00:00 | 1 rows

Recursion With a CTE

Doing recursion as shown above with a function could be considered the brute force way of doing it. The common table expression way of doing recursion is much more elegant, and you can use it even if you don't have the database permissions needed to create stored procs or functions.

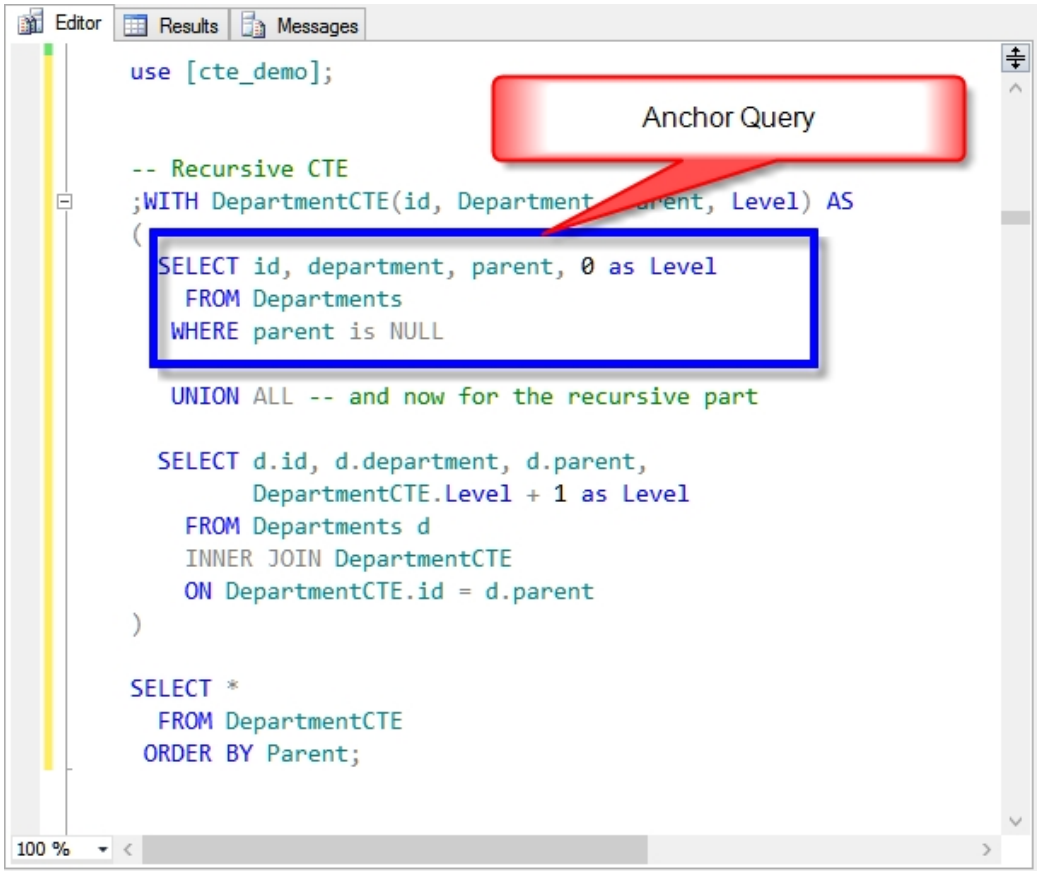
Here is an example of a recursive CTE that queries a heirarchy of store departments:

The first time I looked at a recursive CTE, I was just confused by the syntax. Lets walk through the different parts and it may make more sense, to start with I will add some white space to the query to split it up a bit:

```sql
use [cte_demo];


-- Recursive CTE
;WITH DepartmentCTE(id, Department, Parent, Level) AS
(
  SELECT id, department, parent, 0 as Level
    FROM Departments
   WHERE parent is NULL

   UNION ALL -- and now for the recursive part

  SELECT d.id, d.department, d.parent,
         DepartmentCTE.Level + 1 as Level
    FROM Departments d
    INNER JOIN DepartmentCTE
    ON DepartmentCTE.id = d.parent
)

SELECT *
  FROM DepartmentCTE
 ORDER BY Parent;
```

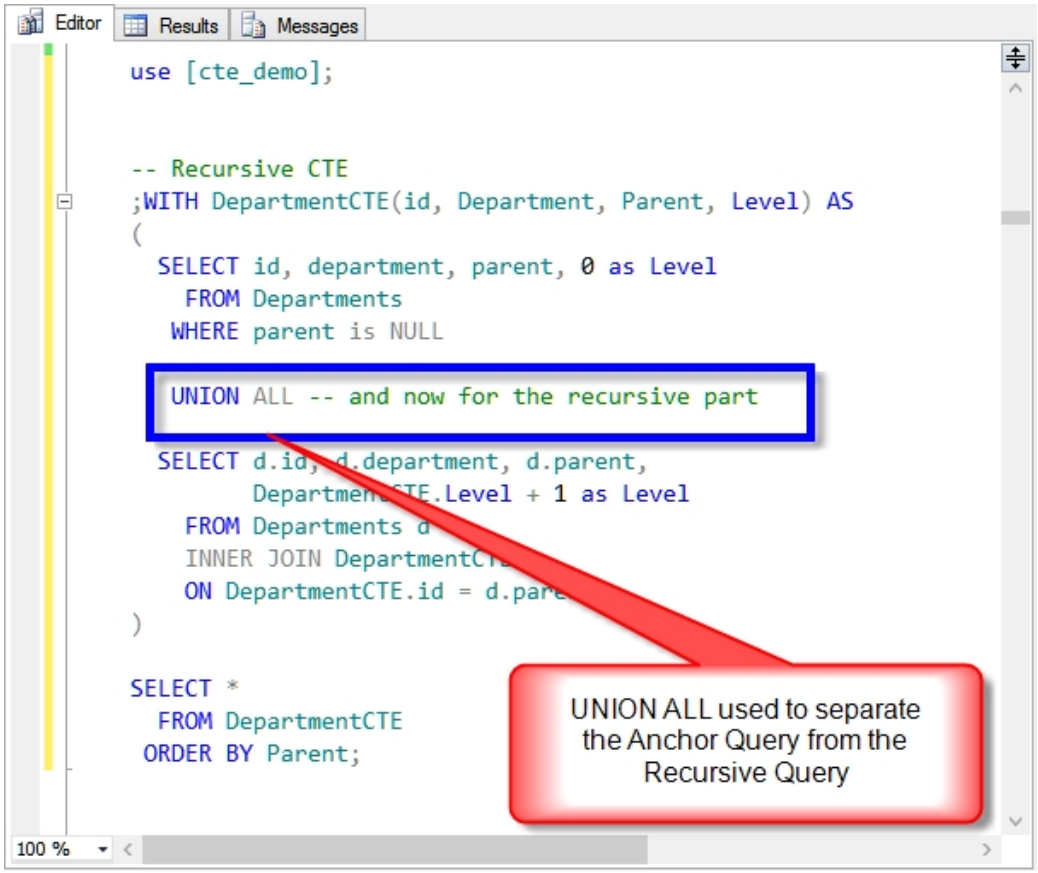
Anchor Query

The anchor query is the part of the CTE that starts the recursion. This is the part of the query that is run first that the recursive part will build from.

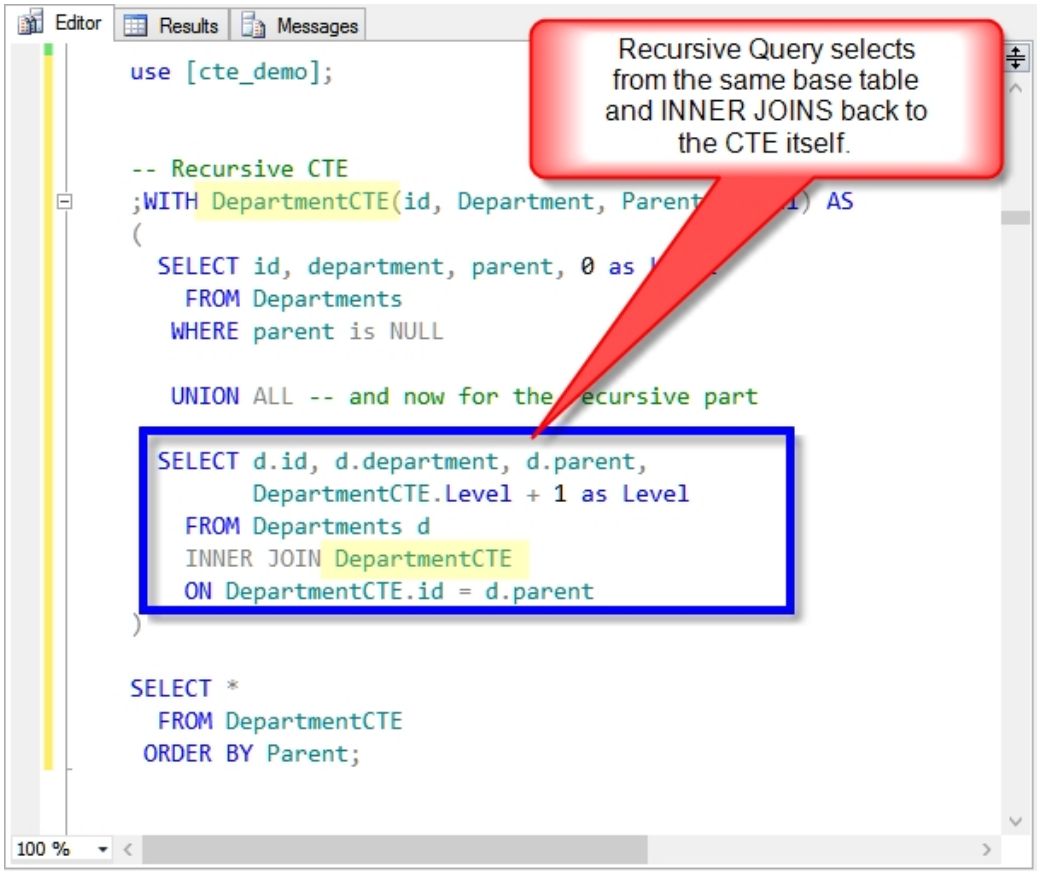


Recursive Query

The UNION ALL keyword is used to separate the anchor query from the recursive part.



After the UNION ALL comes the Recursive Query



What makes the recursive part of the CTE query recursive is the way that it references itself

Executing the Recursive CTE

Once the recursive CTE has been written, it can be executed just like any other CTE, SELECT everything FROM the CTE name, in this case we are also using ORDER BY to sort the results.

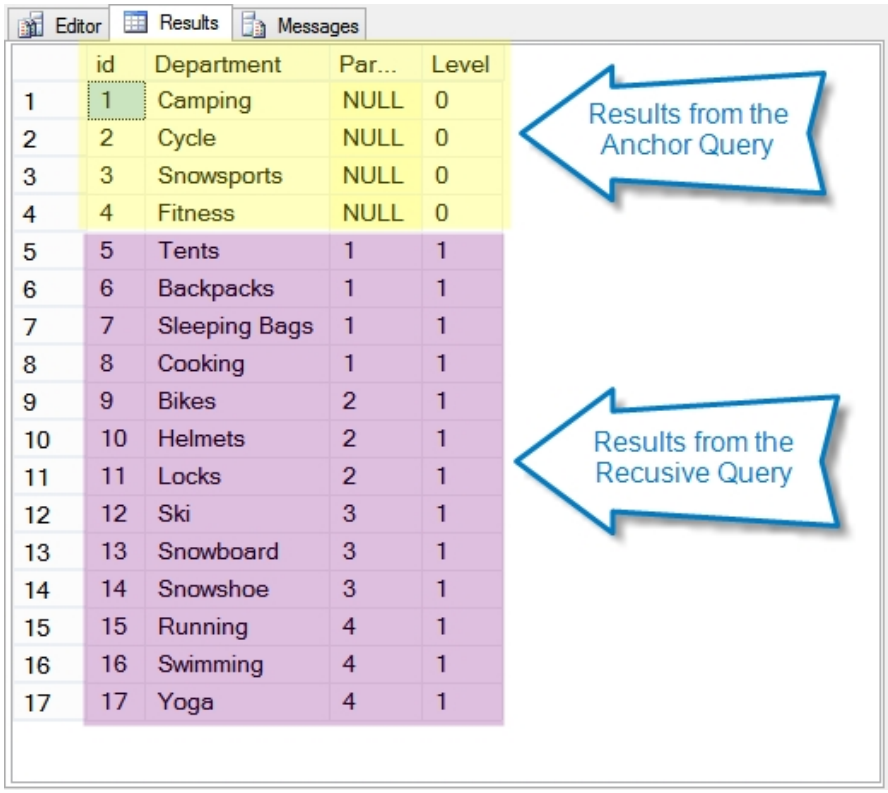
```sql
use [cte_demo];


-- Recursive CTE
;WITH DepartmentCTE(id, Department, Parent, Level) AS
(
  SELECT id, department, parent, 0 as Level
    FROM Departments
   WHERE parent is NULL

   UNION ALL -- and now for the recursive part

  SELECT d.id, d.department, d.parent,
         DepartmentCTE.Level + 1 as Level
    FROM Departments d
    INNER JOIN DepartmentCTE
    ON DepartmentCTE.id = d.parent
)

SELECT *
  FROM DepartmentCTE
 ORDER BY Parent;
```

With the CTE complete, lets take a look at the results.

Editor | Results | Messages

	id	Department	Par...	Level
1	1	Camping	NULL	0
2	2	Cycle	NULL	0
3	3	Snowsports	NULL	0
4	4	Fitness	NULL	0
5	5	Tents	1	1
6	6	Backpacks	1	1
7	7	Sleeping Bags	1	1
8	8	Cooking	1	1
9	9	Bikes	2	1
10	10	Helmets	2	1
11	11	Locks	2	1
12	12	Ski	3	1
13	13	Snowboard	3	1
14	14	Snowshoe	3	1
15	15	Running	4	1
16	16	Swimming	4	1
17	17	Yoga	4	1

Now lets break the results up into what was generated by the anchor and what was generated by the recursive query.



	id	Department	Par...	Level
1	1	Camping	NULL	0
2	2	Cycle	NULL	0
3	3	Snowsports	NULL	0
4	4	Fitness	NULL	0
5	5	Tents	1	1
6	6	Backpacks	1	1
7	7	Sleeping Bags	1	1
8	8	Cooking	1	1
9	9	Bikes	2	1
10	10	Helmets	2	1
11	11	Locks	2	1
12	12	Ski	3	1
13	13	Snowboard	3	1
14	14	Snowshoe	3	1
15	15	Running	4	1
16	16	Swimming	4	1
17	17	Yoga	4	1

Whats Next

Now that you understand recursive CTE's we can get into some really interesting recursive options over the rest of CTE month at SteveStedman.com. Stick around something new most every day.

Related Links:

- Introduction to CTEs Video
- Recursive Scalar Function in T-SQL

- My Book on Common Table Expressions
- cte_demo sample database