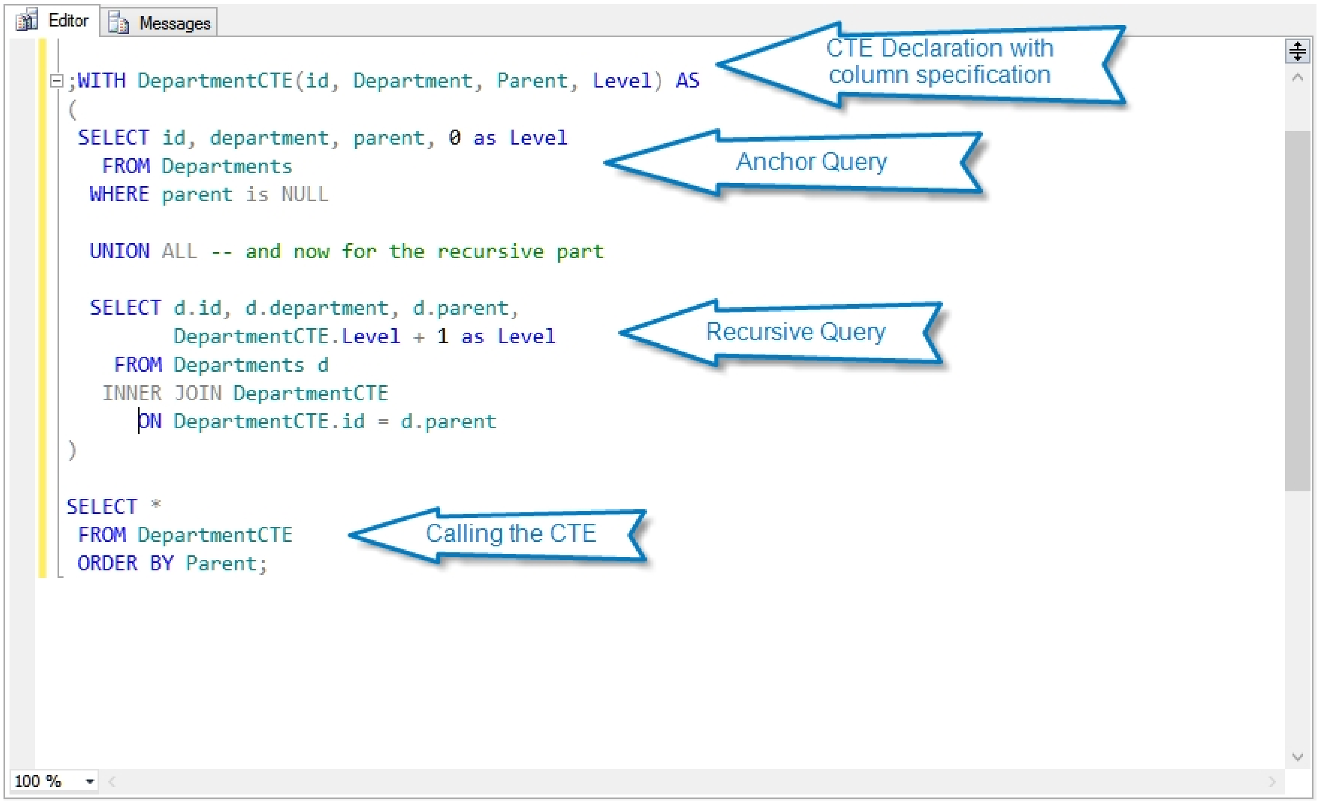
Recursive CTE for Dates In A Year

Day 8 of Common Table Expression Month (June) at SteveStedman.com, today I will be building on the intro to recursive CTEs from yesterday and showing how a recursive CTE can be used to calculate information about dates of the year. This would be useful if you were trying to build a calendar.

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.

Recursive Review

Yesterday's topic was the introduction to recursive CTE's.

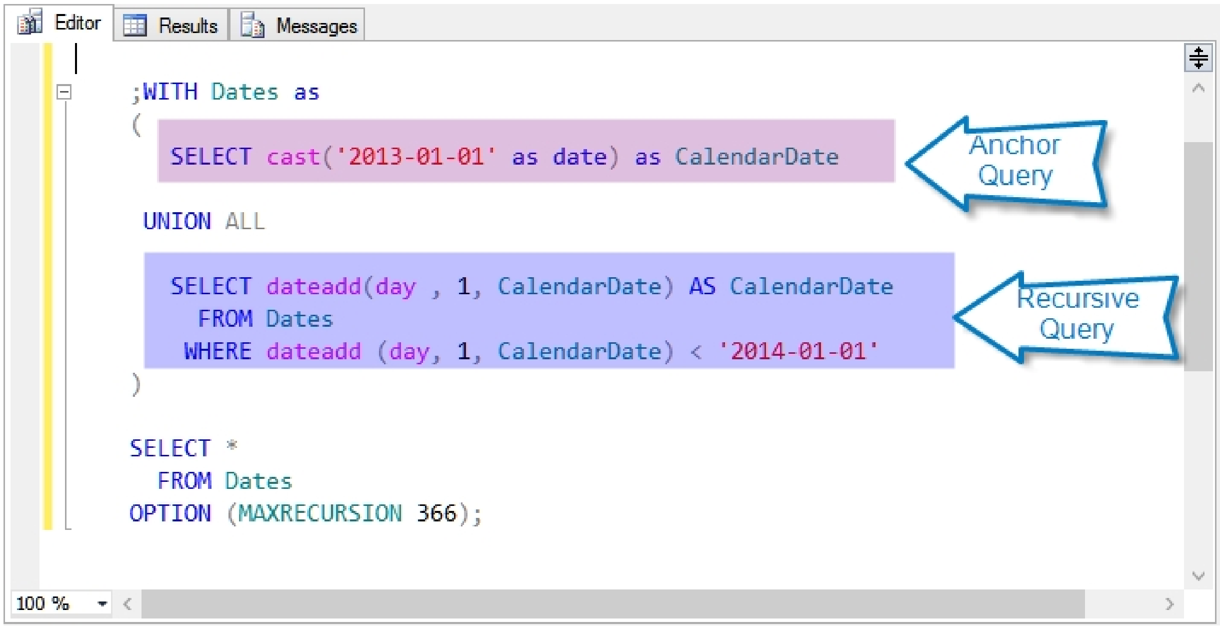


```sql
;WITH DepartmentCTE(id, Department, Parent, Level) AS
(
 SELECT id, department, parent, 0 as Level
   FROM Departments
  WHERE parent is NULL

  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;
```

In the introduction to recursive CTE's we covered the declaration of the CTE, the Anchor Query which starts the recursive process, the Recursive Query which continues the recursion, and the Query that calls the CTE.

Recursive CTE for dates in a Year

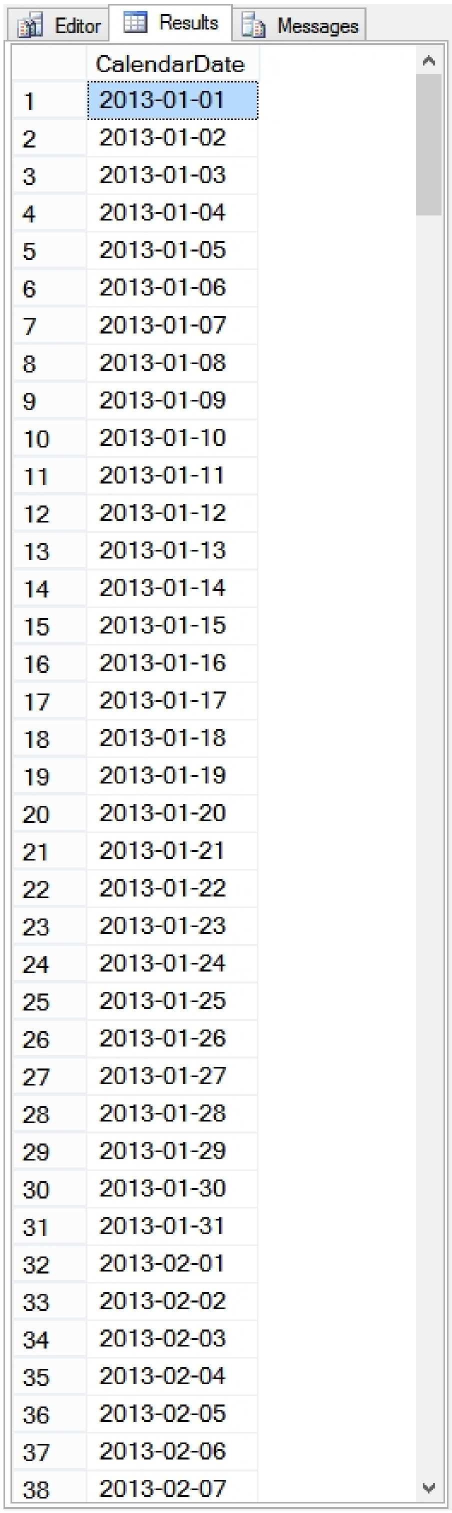
In the following picture, the CTE is named Dates, the anchor query start out by just selecting January 1st of 2013. Next the recursive part selects CalendarDate from the Dates CTE and it adds a single day to it. This all continues recursively as long as the date is less than January 1st 2014.



```
;WITH Dates as
(
   SELECT cast('2013-01-01' as date) as CalendarDate

 UNION ALL

   SELECT dateadd(day , 1, CalendarDate) AS CalendarDate
     FROM Dates
    WHERE dateadd (day, 1, CalendarDate) < '2014-01-01'
)

SELECT *
  FROM Dates
OPTION (MAXRECURSION 366);
```

There is an additional setting. The OPTION (MAXRECURSION 366) has been added to go past the default 100 levels of recursion.

When we run the query we get the following results:



	CalendarDate
1	2013-01-01
2	2013-01-02
3	2013-01-03
4	2013-01-04
5	2013-01-05
6	2013-01-06
7	2013-01-07
8	2013-01-08
9	2013-01-09
10	2013-01-10
11	2013-01-11
12	2013-01-12
13	2013-01-13
14	2013-01-14
15	2013-01-15
16	2013-01-16
17	2013-01-17
18	2013-01-18
19	2013-01-19
20	2013-01-20
21	2013-01-21
22	2013-01-22
23	2013-01-23
24	2013-01-24
25	2013-01-25
26	2013-01-26
27	2013-01-27
28	2013-01-28
29	2013-01-29
30	2013-01-30
31	2013-01-31
32	2013-02-01
33	2013-02-02
34	2013-02-03
35	2013-02-04
36	2013-02-05
37	2013-02-06
38	2013-02-07

Which continues all the way to December 31st 2013.

How Would This Be Useful:

Chapter 10 of the CTE book has a section on finding holes in patterns. Basically you want to query for things that you have it is generally straightforward, but if you want to query for things that you don't have it is not as easy. Lets say you are working on a scheduling application that needs to look at a list of dates and find the dates that a venue may be available. You can probably easily query the dates the venue is in use, and with the Dates CTE you could then do a left join with exclusions where you left join the Dates CTE to the dates that a venue is in use, then only select the results where the venue date is null. This would return the dates that the venue is available.

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.