What is Common Table Expression (CTE)

A CTE can be thought of as a temporary result set and are similar to a derived table in that it is not stored as an object and lasts only for the duration of the query. A CTE is generally considered to be more readable than a derived table and does not require the extra effort of declaring a Temp Table while providing the same benefits to the user. However; a CTE is more powerful than a derived table as it can also be self-referencing, or even referenced multiple times in the same query.

The basic syntax structure for a CTE is shown below:  
WITH MyCTE  
AS ( SELECT EmpID, FirstName, LastName, ManagerID  
FROM Employee  
WHERE ManagerID IS NULL )  
SELECT \*  
FROM MyCTE

Building a Recursive CTE

In the following examples, you will learn how to harness the power of a recursive CTE query by fulfilling a common business requirement, retrieving hierarchical data. By the time the final query is complete you will be able to easily determine how many levels from the top executive each employee is.

A recursive CTE requires four elements in order to work properly.

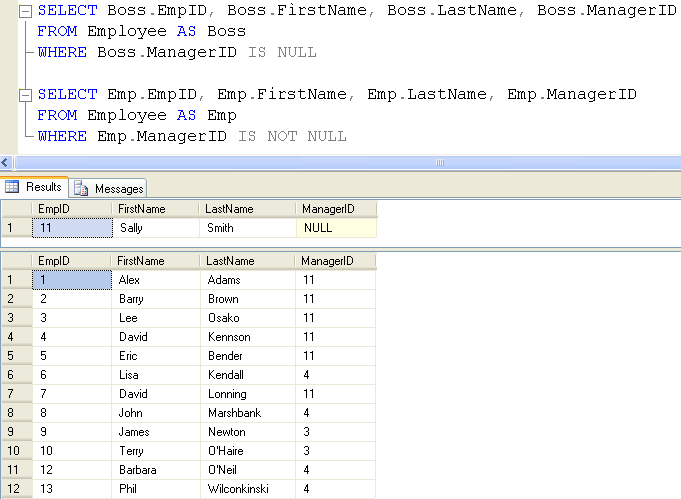
1. Anchor query (runs once and the results ‘seed’ the Recursive query)
2. Recursive query (runs multiple times and is the criteria for the remaining results)
3. UNION ALL statement to bind the Anchor and Recursive queries together.
4. INNER JOIN statement to bind the Recursive query to the results of the CTE.

WITH MyCTE  
AS ( SELECT EmpID, FirstName, LastName, ManagerID  
FROM Employee  
WHERE ManagerID IS NULL  
UNION ALL  
SELECT EmpID, FirstName, LastName, ManagerID  
FROM Employee  
INNER JOIN MyCTE ON Employee.ManagerID = MyCTE.EmpID  
WHERE Employee.ManagerID IS NOT NULL )  
SELECT \*  
FROM MyCTE

Identify the Anchor and Recursive Query

Anyone who does not have a boss is considered to be at the top level of the company and everyone who does have a boss either works for the person(s) at the top level (upper management), or the people that work for them (mid-management thru base employees).

For example, a CEO is at the top level and thus has a ManagerID of null. Likewise, everyone below the CEO will have a ManagerID. This is demonstrated in the two queries below:



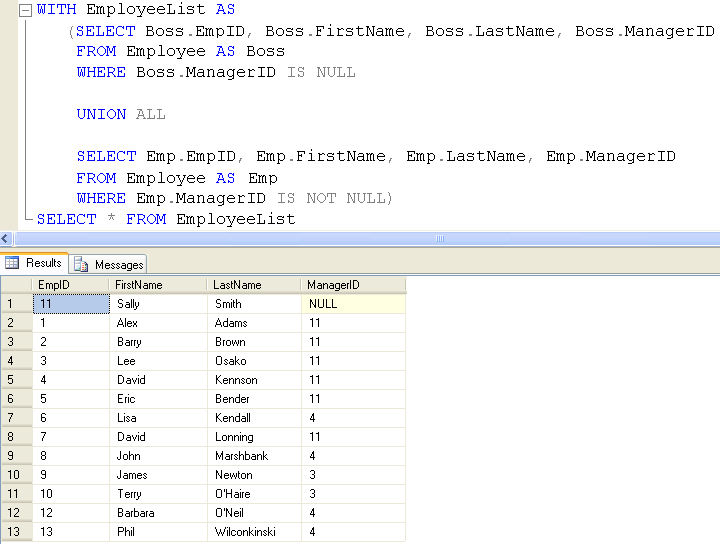
The first SELECT statement will become your Anchor query as it will find the employee that has a ManagerID of null (representing Level 1 of the organization). The second SELECT statement will become your Recursive query and it will find all employees that do have a ManagerID (representing Level 2-3 of this organization).

As you can see from the results so far, these queries are unable to give hierarchical data on which level each employee is at within the organization.

Add the Anchor and Recursive query to a CTE

Begin transforming this entire query into a CTE by placing a UNION ALL statement between the Anchor and Recursive queries. Now add parentheses around the entire query, indenting it, moving it down, and adding the declaration WITH EmployeeList AS before the open parenthesis, and then add SELECT \* FROM EmployeeList on the next line after the close parenthesis.

Your query should now look like the screenshot below:



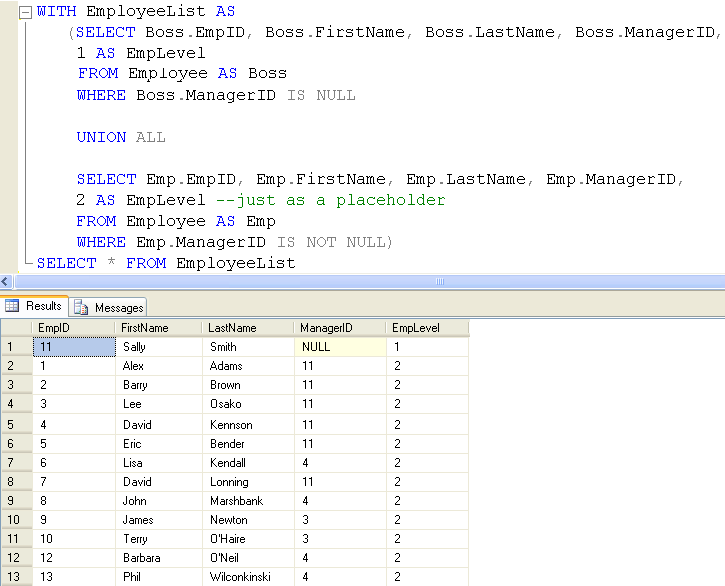
As you can see, the results from your CTE are exactly the same as the results returned from running the anchor and Recursive queries simultaneously in the previous example.

Add an expression to track hierarchical level

The Anchor query (aliased as ‘Boss’) inside the CTE represents everyone at Level 1 (i.e. Sally Smith). The Recursive query (aliased as ‘Emp’) represents everyone at Levels 2 and 3. In order to visualize each level in a result set, you will need to add an expression field to each query.

Add the expression “1 AS EmpLevel” to the Anchor query and the expression “2 AS EmpLevel” to the Recursive query. Before executing the entire query, look closely at the expression field. The EmpLevel expressions in the Anchor query will hard-code the numeral 1 (for Sally Smith’s level), while the EmpLevel expressions in the Recursive query will hard-code the numeral 2 for everyone else.

Your query should now look like the screenshot below:



The two new expression fields were a helpful step. In fact, they show the correct EmpLevel information for Sally Smith and for the people at Level 2 (i.e., Adams, Bender, Brown, Kennson, Lonning and Osako). However, the 2 is just a hard-coded placeholder to help visualize your next step. Lisa Kendall and several other employees need to be at Level 3.

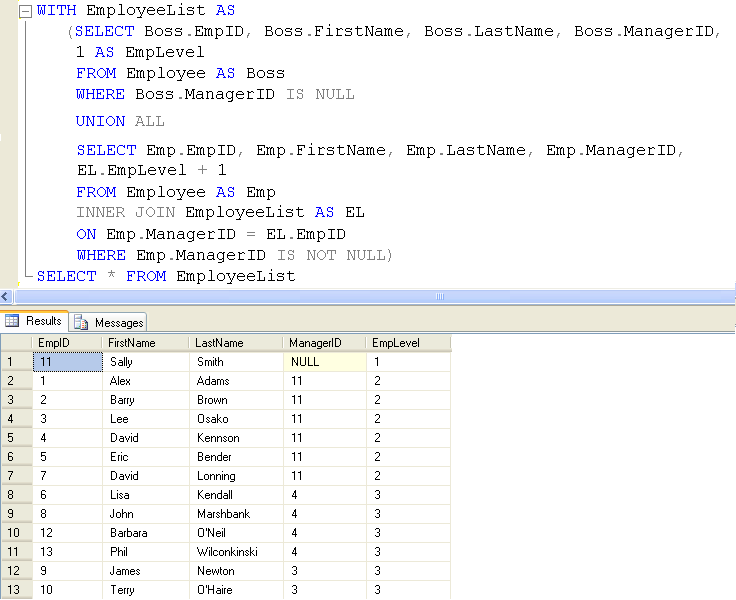
Ideally you would like to make the expression dynamic by replacing “2 AS EmpLevel” with the expression “EmpLevel + 1”.

Add a self-referencing INNER JOIN statement

Let’s take a moment and recognise why this is not going to work quite so simply. The idea to increment EmpLevel in the recursive query of the CTE is on the right track. Unfortunately, the recursive query is trying to reference a field called EmpLevel but can’t find one, since it has only been materialized in the result set of the Anchor query and does not yet exist in the recursive set.

How can you materialize the EmpLevel field for the recursive query? *We can use the CTE for this!* Remember, a recursive CTE requires an INNER JOIN to connect the recursive query to the CTE itself. Go ahead and write an INNER JOIN statement binding the recursive query ‘Emp’ to the CTE ‘EmployeeList AS EL’ ON Emp.ManagerID = EL.EmpID.

Your query should now look like the screenshot below:



***Success!*** You can now see that Sally is at the first level, Alex is at the second level and Lisa appears at the third level. Since a CTE can reference itself, the ‘Emp’ recursive query can now access the EmpLevel field materialized in the EmployeeList CTE.