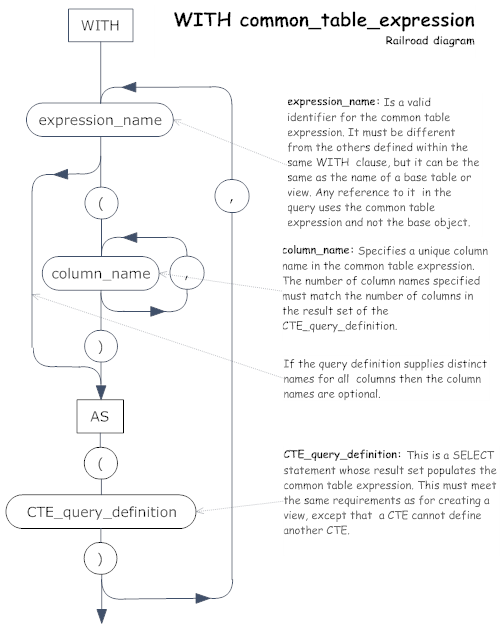
**SQL Server CTE**

The CTE (Common type expression) was introduced into standard SQL in order to simplify various classes of SQL Queries for which a derived table just wasn't suitable.



##### Problem

Many organizations have some type of hierarchy for business processes. When it comes to large organizations, the hierarchy can get very complex and large, so building a hierarchy in a RDBMS is a tedious task. We have to create views, cursors and so on, but using a CTE in SQL Server is a better solution to retrieve hierarchy-based data and in this tip, I will show you how.

##### Solution

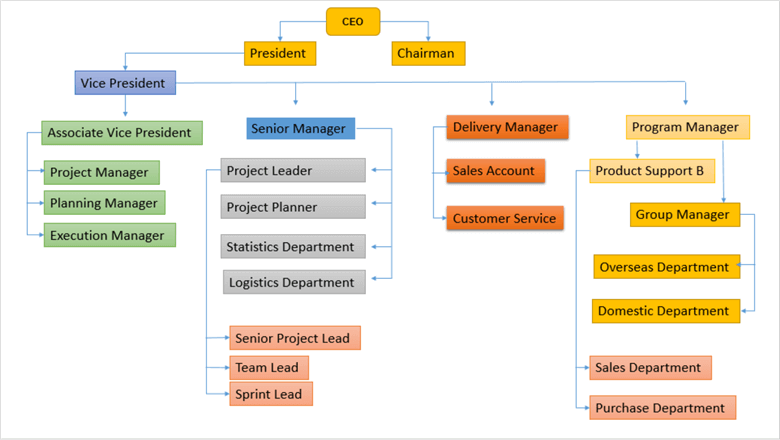
Common Table Expressions (CTE) have two types, recursive and non-recursive. We will see how the recursive CTE works with examples in this tip.

A recursive CTE can be explained in three parts:

* Anchor Query: This is the first statement which is executed. This query will give the base data for the CTE.
* Separator: This is the middle part where in we generally use a UNION ALL and few more operators.
* Recursive Query: This is the main part, this is the CTE query which refers to the same CTE by recursion.

## Let’s us create an example of CTE

Let’s take a scenario of an organization (org) chart. In this example the organization chart would start from "CEO" and end up at the “Purchase Department”.  Each department/person is linked to the predecessor as nodes. Let's see how a CTE can be used to achieve this in SQL Server. We will also touch base on how to use MAXRECURSION when using a CTE.



Let's create a sample table.

IF OBJECT\_ID ('MyDepartment','U') IS NOT NULL

DROP TABLE MyDepartment;

GO

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[MyDepartment]

(

[DepartmentID] [smallint] NOT NULL,

[DepartmentName] [nvarchar](30) NOT NULL,

[ParentID] [nvarchar](40) NULL,

CONSTRAINT [PK\_DepartmentID] PRIMARY KEY CLUSTERED ( [DepartmentID] ASC )

WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON) ON [PRIMARY]

) ON [PRIMARY]

GO

Now the table is created and we can populate the table with values the table “MyDepartment”. The below insert statements can be executed to insert the data into “MyDepartment” table.

INSERT INTO [dbo].[MyDepartment] ([DepartmentID],[DepartmentName],[ParentID])

VALUES

('1','CEO',null),

('2','President','1'),

('3','Chairman','1'),

('4','Vice President','2'),

('5','Associate Vice President','4'),

('6','Senior Manager','4'),

('7','Delivery Manager','4'),

('8','Program Manager','4'),

('9','Project Manager','5'),

('10','Planning Manager','5'),

('11','Execution Manager','5'),

('12','Project Leader','6'),

('13','Project Planner','6'),

('14','Senior Project Lead','12'),

('15','Team Lead','12'),

('16','Sprint Lead','12'),

('17','Statistics Department','6'),

('18','Logistics Department','6'),

('19','Sales Account','7'),

('20','Customer Service','7'),

('21','Product Support B','8'),

('22','Sales Department','21'),

('23','Purchase Department','21'),

('24','Group Manager','8'),

('25','Overseas Department','24'),

('26','Domestic Department','24');

go

The ParentID for the top-most tree is set to NULL indicating that there is no parent for this row. Now that we have loaded the data we can code a query to walk the hierarchy using a Common Table Expression.

We need to report on the entire organizational structure under the CEO.  The following recursive code using a CTE will get us the output:

WITH OrgTree (DepartmentID, DepartmentName, ParentID, Tree)

AS

(

SELECT DepartmentID, DepartmentName, ParentID , 0 AS Tree

FROM MyDepartment

WHERE ParentID IS NULL

UNION ALL

SELECT MyDepartment.DepartmentID, MyDepartment.DepartmentName, MyDepartment.ParentID , OrgTree.Tree + 1

FROM MyDepartment

JOIN OrgTree ON MyDepartment.ParentID = OrgTree.DepartmentID

)

SELECT \* FROM OrgTree ORDER BY Tree

Below is the output from the above query execution.

|  |  |  |  |
| --- | --- | --- | --- |
| **DepartmentID** | **DepartmentName** | **ParentID** | **Tree** |
| 1 | CEO | NULL | 0 |
| 2 | President | 1 | 1 |
| 3 | Chairman | 1 | 1 |
| 4 | Vice President | 2 | 2 |
| 5 | Associate Vice President | 4 | 3 |
| 6 | Senior Manager | 4 | 3 |
| 7 | Delivery Manager | 4 | 3 |
| 8 | Program Manager | 4 | 3 |
| 21 | Product Support B | 8 | 4 |
| 24 | Group Manager | 8 | 4 |
| 19 | Sales Account | 7 | 4 |
| 20 | Customer Service | 7 | 4 |
| 12 | Project Leader | 6 | 4 |
| 13 | Project Planner | 6 | 4 |
| 17 | Statistics Department | 6 | 4 |
| 18 | Logistics Department | 6 | 4 |
| 9 | Project Manager | 5 | 4 |
| 10 | Planning Manager | 5 | 4 |
| 11 | Execution Manager | 5 | 4 |
| 14 | Senior Project Lead | 12 | 5 |
| 15 | Team Lead | 12 | 5 |
| 16 | Sprint Lead | 12 | 5 |
| 25 | Overseas Department | 24 | 5 |
| 26 | Domestic Department | 24 | 5 |
| 22 | Sales Department | 21 | 5 |
| 23 | Purchase Department | 21 | 5 |

#### Anchor Query

Creating tables is one piece of it, inserting data is another group in the example. The important part is implementing the CTE using the WITH clause. For our example the name of the CTE is named as “OrgTree”. The first select in the CTE is used to extract the first node of the Tree which is “CEO” and the Parent ID is set as NULL. The below query will get the first node in our example.

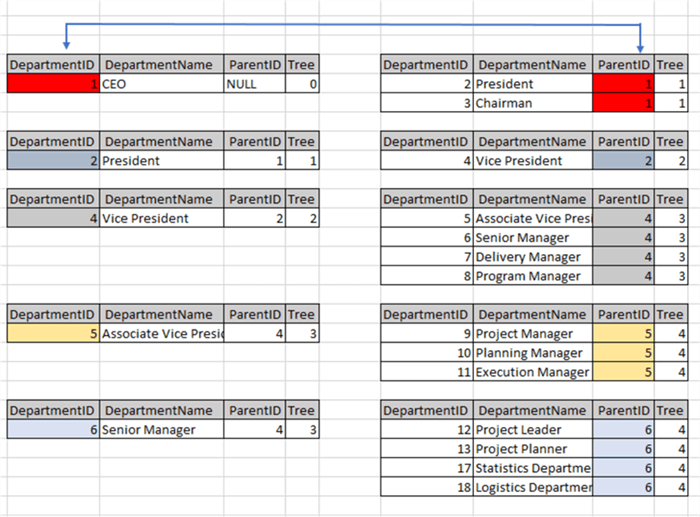
SELECT DepartmentID, DepartmentName, ParentID , 0 AS Tree

FROM MyDepartment

WHERE ParentID IS NULL

|  |  |  |  |
| --- | --- | --- | --- |
| **DepartmentID** | **DepartmentName** | **ParentID** | **Tree** |
| 1 | CEO | NULL | 0 |

Below I explain the data.  The left side has the parent data and the right side has the child data. If you notice DepartmentID 1 (left side) is the parent for DepartmentID (right side) 2 and 3. If you look further each DepartmentID is connected with the ParentID in the child table. Below is the image representation of the query that was generated above. The arrow connects the DepartmentID and ParentID for our reference.



#### Separator and Recursive Query

The next section is the INNER JOIN combining the CTE where in recursion comes into picture, which intern refers to itself. The INNER JOIN retrieves the data by splitting the “MyDepartment” table into two parts using the OrgTree CTE and does a join and creates the CTE for us to query the CTE.

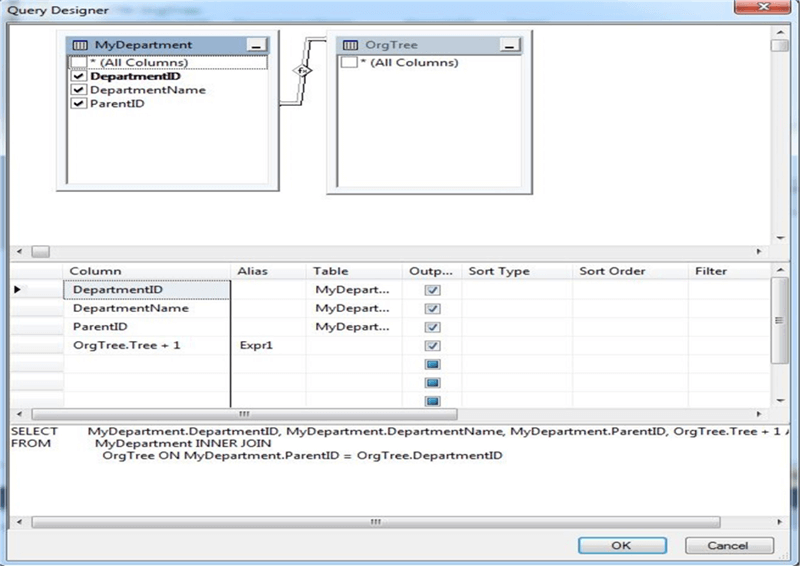
SELECT MyDepartment.DepartmentID, MyDepartment.DepartmentName, MyDepartment.ParentID , OrgTree.Tree + 1

FROM MyDepartment

JOIN OrgTree ON MyDepartment.ParentID = OrgTree.DepartmentID

#### Query Designer

The below Query Designer screen print is available for us to see the query in the Designer. The right-side is the CTE - “OrgTree”, since CTE will be created after execution the Query Designer does not show the columns, if you notice the “MyDepartment” table has the column and the INNER JOIN reference.



#### MAXRECURSION

When it comes to using a CTE one of the problems faced is an infinite loop while forming the CTE. In general, when the parent and child query returns the same or equal value, the CTE may go into an infinite loop and the transaction may go into an infinite loop. To avoid this there is an option clause which can be used at the end of CTE SELECT command with the key word MAXRECURSION and the row count. Using 0 has no restriction, but in our example I have used a value of 10.

WITH OrgTree (DepartmentID, DepartmentName, ParentID, Tree)

AS

(

SELECT DepartmentID, DepartmentName, ParentID, 0 AS Tree

FROM MyDepartment

WHERE ParentID IS NULL

UNION ALL

SELECT MyDepartment.DepartmentID, MyDepartment.DepartmentName, MyDepartment.ParentID, OrgTree.Tree + 1

FROM MyDepartment

JOIN OrgTree ON MyDepartment.ParentID = OrgTree.DepartmentID

)

SELECT \* FROM OrgTree OPTION (MAXRECURSION 10)

##### Example Queries

Try these example queries to find the data and see if you can come up with other scenarios and how to query the data.

-- return everyone under Program Manager (ParentID = 8)

WITH OrgTree (DepartmentID, DepartmentName, ParentID, Tree)

AS

(

SELECT DepartmentID, DepartmentName, ParentID , 0 AS Tree

FROM MyDepartment

WHERE ParentID = 8

UNION ALL

SELECT MyDepartment.DepartmentID, MyDepartment.DepartmentName, MyDepartment.ParentID , OrgTree.Tree + 1

FROM MyDepartment

JOIN OrgTree ON MyDepartment.ParentID = OrgTree.DepartmentID

)

SELECT \* FROM OrgTree;

-- return Vice President (DepartmentID = 4) and direct reports (ParentID = 4)

WITH OrgTree (DepartmentID, DepartmentName, ParentID, Tree)

AS

(

SELECT DepartmentID, DepartmentName, ParentID , 0 AS Tree

FROM MyDepartment

WHERE DepartmentID = 4

UNION ALL

SELECT MyDepartment.DepartmentID, MyDepartment.DepartmentName, MyDepartment.ParentID , OrgTree.Tree + 1

FROM MyDepartment

JOIN OrgTree ON MyDepartment.ParentID = OrgTree.DepartmentID

WHERE MyDepartment.ParentID = 4

)

SELECT \* FROM OrgTree;

-- return everyone above Senior Manager (DepartmentID = 6)

WITH OrgTree(DepartmentName,ParentID,ReportsTo)AS

(

SELECT T1.DepartmentName,T2.DepartmentID,T2.DepartmentName

FROM MyDepartment T1

INNER JOIN MyDepartment T2 ON T1.ParentID=T2.DepartmentID

WHERE T1.DepartmentID=6

UNION ALL

SELECT OT.ReportsTo,T2.DepartmentID,T2.DepartmentName

FROM OrgTree OT

INNER JOIN MyDepartment T1 ON OT.ParentID=T1.DepartmentID

INNER JOIN MyDepartment T2 ON T1.ParentID=T2.DepartmentID

)

SELECT \* FROM OrgTree;

-- return list with of people with no direct reports

WITH OrgTree(ParentID, DepartmentID, DepartmentName, DepartmentLevel) AS

(

SELECT ParentID, DepartmentID, DepartmentName, 0 AS DepartmentLevell

FROM MyDepartment

WHERE ParentID IS NULL

UNION ALL

SELECT e.ParentID, e.DepartmentID, e.DepartmentName, DepartmentLevel + 1

FROM MyDepartment AS e

INNER JOIN OrgTree AS d ON e.ParentID = d.DepartmentID

)

SELECT \* FROM OrgTree WHERE DepartmentLevel = 5;