

# Common Table Expressions

Wayne Snyder, Mariner ([www.mariner-usa.com](http://www.mariner-usa.com))

Common Table Expressions(CTEs) are new to SQL Server 2005. CTEs are defined in the ANSI SQL-99 standard. We need to begin learning about the new features of SQL 2005, and CTEs are an interesting place to start. You will learn about the syntax and how to think about CTEs. **SQL Server 2005 training from Learnkey** ([www.Learnkey.com](http://www.Learnkey.com)) will go into the details of all of the new features of SQL 2005.

CTEs come in 2 forms, nonrecursive and recursive. In this whitepaper we will look only at nonrecursive CTEs. Next time we will do recursive CTEs – it is amazing what you can do with a recursive CTE.

You can think of a nonrecursive CTE (NR-CTE) as a derived table, or a temporary view. Microsoft refers to a CTE as a “*temporary, named result set*”. Most of the NR-CTEs can be re-written as a longer statement which uses derived tables or subqueries. Since a CTE can be referred to multiple times in a query, the basic benefit you get from NR-CTEs is simplicity of syntax in your queries.

The basic syntax for a CTE follows:

```
With CTE_Name (Column_Name_1, Column_Name_2 ... Column_Name_n )
As
(
    cte_Query_Definition
)
--TSQL which uses the CTE
```

Listing 1. Basic Syntax for CTE

Let's take a look at a simple NR-CTE which lists each Manager and his number of Direct Reports.

```
USE AdventureWorks;
GO
WITH DirReps(ManagerID, DirectReports) AS
(
    SELECT ManagerID, COUNT(*)
    FROM HumanResources.Employee AS e
    WHERE ManagerID IS NOT NULL
    GROUP BY ManagerID
)
SELECT ManagerID, DirectReports
FROM DirReps
ORDER BY ManagerID;
GO
```

The Statement produces the result below(Listing 2 - CTE Query Results ). (The list is truncated for brevity.)  
Now let's look at the CTE. The CTE definition begins

```
WITH DirReps(ManagerID, DirectReports) AS  
(  
)
```

DirReps is the name of the CTE. The CTE name may be the same as a table/view name used in the expression, but I suggest you make the CTE Names different. Duplicate names are too confusing - at least for me. If your CTE name is the same as a table name, any reference would point to the CTE name. You will see later that you may include multiple CTEs within the same WITH clause. CTE Names MUST be unique within the same WITH clause.

You may optionally rename the columns which are returned from the CTE, in the same way you can provide column names in a view definition. If the column names are not supplied, the column names used in the select statement are used.

In parenthesis is the CTE query definition. The rules which apply to a NR-CTE query are exactly the same as the rules which apply to the select statement when creating a view. Hopefully, now you can see that the NR-CTE definition is very much like a derived table, and almost exactly like a Create View definition.

The CTE query Definition may not contain:

- COMPUTE or COMPUTE BY
- ORDER BY (except when a TOP clause is specified)
- INTO
- OPTION clause with query hints
- FOR XML
- FOR BROWSE

The T-SQL statement which uses the CTE follows the CTE definition. In this case:

```
SELECT ManagerID, DirectReports  
FROM DirReps  
ORDER BY ManagerID;
```

As you can see the CTE name, DirReps, is used exactly like a table, view or derived table alias. The T-SQL statement which uses the CTE definition can be a Select, Insert, Update, or Delete. The CTE definition's scope is the scope of the query. When the statement is complete the definition is gone.

	ManagerID	DirectReports
1	3	7
2	6	8
3	7	6
4	12	1
5	14	6
6	16	12
7	18	8
8	21	22
9	25	5
10	30	5
11	38	12
12	41	4
13	42	7
14	44	4
15	49	4
16	51	8
17	64	5
18	71	1
19	74	8
20	85	5
21	87	6
22	90	4

Listing 2 - CTE Query Results

Now let's re-write the the query without using a CTE.

```
SELECT ManagerID, COUNT(*) as DirectReports
FROM HumanResources.Employee AS e
WHERE ManagerID IS NOT NULL
GROUP BY ManagerID
ORDER BY ManagerID;
GO
```

For the example above, I would prefer to use the regular query, instead of the NR-CTE for its brevity and ease of understanding.

You can also put the CTE into a view if you wish.

```
Create View DirectReports as
WITH DirReps(ManagerID, DirectReports) AS
(
    SELECT ManagerID, COUNT(*)
    FROM HumanResources.Employee AS e
    WHERE ManagerID IS NOT NULL
    GROUP BY ManagerID
)
SELECT ManagerID, DirectReports
FROM DirReps
```

```
GO
Select * From DirectReports ORDER BY ManagerID;
```

As you can see, the NR-CTE is really not much different than using a derived table. However if you find yourself in a situation where you would have to create the derived table several times in a query, using a CTE will be more succinct.

Let's take a look at another example. We wish to have a listing of each Employee and the number of Orders they have taken for 2004 and 2003(Listing 3 Employee Order History.).

	EmployeeID	OrderYear	NumberOfOrders	OrderYear	NumberOfOrders
1	268	2004	10	2003	14
2	275	2004	86	2003	169
3	276	2004	82	2003	166
4	277	2004	97	2003	181
5	278	2004	49	2003	88
6	279	2004	81	2003	158
7	280	2004	12	2003	17
8	281	2004	52	2003	93
9	282	2004	57	2003	81
10	283	2004	41	2003	69
11	284	2004	9	2003	25
12	285	2004	91	2003	173

Listing 3 – Employee Order History.

One way we might write the query to produce this would be to make a derived table for the employeeid, orderyear and number of Orders for 2004.

```
SELECT SalesPersonID,
       COUNT(*) as NumberOfOrders,
       YEAR(OrderDate)as OrderYear
FROM Sales.SalesOrderHeader as TargetYear
GROUP BY SalesPersonID, YEAR(OrderDate)
WHERE TargetYear.OrderYear = 2004
```

Then do the same to get the 2003 numbers. Notice they are the same. We will use WHERE clauses later to limit the rows which are returned.

```
SELECT SalesPersonID,
       COUNT(*) as NumberOfOrders,
       YEAR(OrderDate)as OrderYear
FROM Sales.SalesOrderHeader as PriorYear
GROUP BY SalesPersonID, YEAR(OrderDate)
WHERE PriorYear.OrderYear = 2003
```

Then we will use both the above queries as derived tables in the complete query. We do this by joining the derived tables. The Employee table is also joined to look up the EmployeeID from the SalePersonId. To simplify maintenance we also change the year selection for the PriorYear derived table:

```

SELECT E.EmployeeID,
       TargetYear.OrderYear,
       TargetYear.NumberOfOrders,
       PriorYear.OrderYear,
       PriorYear.NumberOfOrders
FROM HumanResources.Employee AS E
INNER JOIN
    (SELECT SalesPersonID,
           COUNT(*) as NumberOfOrders,
           YEAR(OrderDate)as OrderYear
     FROM Sales.SalesOrderHeader
     GROUP BY SalesPersonID, YEAR(OrderDate)
    ) AS TargetYear
ON E.EmployeeID = TargetYear.SalesPersonID
LEFT OUTER JOIN
    (SELECT SalesPersonID,
           COUNT(*) as NumberOfOrders ,
           YEAR(OrderDate) as OrderYear
     FROM Sales.SalesOrderHeader
     GROUP BY SalesPersonID, YEAR(OrderDate)
    ) as PriorYear
ON E.EmployeeID = PriorYear.SalesPersonID
AND PriorYear.OrderYear = TargetYear.OrderYear-1
WHERE TargetYear.OrderYear = 2004
ORDER BY E.EmployeeID;

```

Notice that the derived tables are the same, but have to be written twice. This makes the query longer, and more likely to be buggy. We can shorten this by using a NR-CTE. Make the derived table the NR-CTE. Then write a SQL Query which joins the CTE to itself using aliases. Notice the introduction of the local variable. You may refer to local variables in a CTE or the query which uses the CTE.

```

DECLARE @Year int
SELECT @Year=2004
;WITH Sales_CTE (SalesPersonID, NumberOfOrders, OrderYear)
AS
(
    SELECT SalesPersonID,
           COUNT(*),
           YEAR(OrderDate)
     FROM Sales.SalesOrderHeader
     GROUP BY SalesPersonID, YEAR(OrderDate)
)
SELECT E.EmployeeID,
       TargetYear.OrderYear,
       TargetYear.NumberOfOrders,
       PriorYear.OrderYear,
       PriorYear.NumberOfOrders
FROM HumanResources.Employee AS E
INNER JOIN Sales_CTE AS TargetYear
ON E.EmployeeID = TargetYear.SalesPersonID
LEFT OUTER JOIN SALES_CTE PriorYear
ON E.EmployeeID = PriorYear.SalesPersonID
AND PriorYear.OrderYear = TargetYear.OrderYear-1
WHERE TargetYear.OrderYear = @Year

```

```
ORDER BY E.EmployeeID;
```

Also take note of the semicolon prior to the definition of the CTE. Take the semicolon out and run this CTE. You will get the error:

```
Msg 319, Level 15, State 1, Line 228
Incorrect syntax near the keyword 'with'. If this statement is a common table expression
or an xmlnamespaces clause, the previous statement must be terminated with a semicolon.
```

Whenever you use a CTE in a batch with other statements in front of it, the last statement in front of the CTE must be terminated with a semicolon. Perhaps it is a good practice to always prefix CTEs with a semicolon.

The last item we will see is an example of multiple queries defined in the same CTE expression. This particular example gives us the names of the top 5 and bottom 5 sales people. You can use either `Sales_CTE` or `TopSellers` or both in your query. Note that the `TopSellers` query also refers to `Sales_CTE`. One CTE query can refer to a query defined physically above it in the expression, but can NOT refer to a query defined after it. The queries do not HAVE to refer to one another, they may be totally independent.

```
DECLARE @Year int
SELECT @Year=2004
;WITH Sales_CTE (SalesPersonID, ContactID, NumberOfOrders,
OrderYear)
AS
(
    SELECT SalesPersonID,
        ContactID,
        COUNT(*),
        YEAR(OrderDate)
    FROM Sales.SalesOrderHeader
    WHERE YEAR(OrderDate) = @Year
    GROUP BY SalesPersonID, ContactID, YEAR(OrderDate)
),
TopSellers(FullName, ContactID, NumberOfOrders, OrderYear, Class)
AS
(SELECT TOP 5 Firstname + ' ' + Lastname,
    SCTE.ContactID,
    NumberOfOrders,
    OrderYear,
    'TopSeller'
FROM Person.Contact c
INNER JOIN Sales_CTE AS SCTE
    ON C.ContactID = SCTE.ContactID
ORDER BY NumberofOrders Desc
UNION ALL
SELECT TOP 5 Firstname + ' ' + Lastname,
    SCTE.ContactID,
    NumberOfOrders,
    OrderYear,
    'LowSeller'
FROM Person.Contact c
INNER JOIN Sales_CTE AS SCTE
    ON C.ContactID = SCTE.ContactID
```

```
ORDER BY NumberofOrders Asc
)

SELECT * from TopSellers
```

That's it for NR-CTEs. When they make the query easier to understand – use them. You may find that you use them when you need to take the results of a Group By, and join those results to another table.

Next time we will look at Recursive CTEs. Recursive CTEs is where the REAL POWER is. If you have ever tried to navigate a parent child relationship, hierarchy or network – you know how painful it can be, especially if you do not know how deep the hierarchy goes, or if it is ragged. Recursive CTEs, will do a depth first traversal of a hierarchy so easily. You are going to be pleased and surprised.

See you next time!

Wayne Snyder is a long time Learnkey author ([www.learnkey.com](http://www.learnkey.com)). Learnkey's SQL 2000 Administration Series won the SQL Server Magazine's Best Web Based Training in 2004. He is an author for The SQL Server Standard and SQL Server Magazine, an MCT, SQL Server MVP, and Managing Consultant for Mariner([www.mariner-usa.com](http://www.mariner-usa.com)). Wayne has worked with SQL since its first release, and currently focuses on SQL Business Intelligence.

Watch for LearnKey's SQL 2005 release!