Common Table Expressions in SQL Server 2005

Common Table Expressions and Recursive Queries

Srinivas Sampath

MVP SQL Server

This is probably one of the best things that can happen to SQL Server. Among the many T-SQL enhancements that have been made in SQL Server 2005, I like it the best and it is something that I've been looking forward to also.

SQL Server 2005 defines the concept of **Common Table Expressions** or CTEs as they are called. The introduction of CTEs is as per the SQL 99 standard and has been included in SQL Server for two primary reasons:

1. Expressive power
2. Recursive queries

If we consider expressive power, remember that in SQL Server 2000, there was a concept called **derived tables**. A derived table is basically a SELECT statement with a name that is specified after the FROM clause of an outer SELECT statement. Here is an example:

```
SELECT au_id, au_lname, au_fname, titleCount
 FROM authors a
    INNER JOIN (
                     SELECT au_id,      COUNT(title_id)
                FROM titleauthor
                GROUP BY au_id
              ) AS TitleCount (authorID, titleCount)
      ON TitleCount.authorID = a.au_id
```

What does this example do? Basically, we are displaying the author details along with a column that displays the number of titles that the author has written. This is done by creating a table on the fly that summarizes this information from the **titleauthor** table. We then join with this table in the INNER JOIN clause based on author ID.

Derived tables help you build tables on the fly that need not be persisted in the database and are very useful. But imagine for a second if in the same query above, you needed to refer to the same **TitleCount** derived table. What would you do? You will have to repeat the definition once more, alias it and then use it and this can get pretty messy to read if you had multiple instances of the same definition within the query. This is where CTEs are useful. Before seeing the definition of CTEs, let us quickly see how you can write the same query above using CTEs.

```
WITH TitleCount (authorID, titleCount) AS
 (
  SELECT au_id, COUNT(title_id)
   FROM titleauthor
   GROUP BY au_id
 )
SELECT au_id, au_lname, au_fname, titleCount
 FROM authors a
   INNER JOIN TitleCount
            ON TitleCount.authorID = a.au_id
```

Note the simplicity in the syntax and how we are referring to the **TitleCount** in the INNER JOIN clause as though it was a normal table. Before we get further on this, let us formally define a CTE and look at its syntax.

Definition of a CTE

Simply put, a CTE is a temporary view defined within the scope of an executing statement. The scope can be either simple SELECT statements or other DML statements also (meaning the CTE can precede SELECT, INSERT, UPDATE or DELETE statements).

CTE Syntax

In its simplified form, the syntax of a CTE is as follows:

```
WITH cte_name (optional column list) AS
 (
  Cte_query
 )
 statement that uses the above CTE
```

Note that the above syntax is for defining a single CTE. You can also define multiple CTEs in the context of a single statement and if so, you will need to separate each CTE by a comma. The following example shows a query using multiple CTEs

```
WITH TitleCount (authorID, titleCount) AS
 (
  SELECT au_id, COUNT(title_id)
   FROM titleauthor
        GROUP BY au_id
 ),
 Top3Authors (authorID, titleCount) AS
 (
  SELECT TOP 3
         authorID, titleCount
   FROM TitleCount
   ORDER BY titleCount DESC
 )
SELECT
      a.au_id, a.au_lname, a.au_fname,
      t.titleCount,
                honorMessage = CASE WHEN EXISTS ( SELECT authorID FROM Top3Authors
                                                  WHERE Top3Authors.authorID = a.au_id
                                                                                    )
                               THEN 'Top Author'
                                                                       ELSE NULL END
 FROM authors a
   INNER JOIN TitleCount t
           ON t.authorID = a.au_id
```

What does this query do? It defines a CTE called **TitleCount** which provides a list of authors and their title count. It then defines another CTE called **Top3Authors** that builds a list of the top 3 authors based on the first defined CTE. We then have the SELECT statement that uses the 2 CTEs to provide a list of authors and an honor message for those identified as the top 3 authors.

Note that by defining multiple CTEs, you can incrementally build on the earlier CTEs or define new results that are then used later on. Note however, that you cannot create a CTE that uses **forward-reference** (a CTE that is yet to be defined).

CTEs bring us the chance to create much more complex queries while retaining a much simpler syntax. They also can lessen the administrative burden of creating and testing views for situations where the view will not be resused. In my next article I'll look at recursive queries and how CTEs make use of them.