Common Table Expression - WITH Statement

Manuela Horvat
Babes-Bolyai University, Cluj-Napoca
E-mail: manuela.petrescu@gmail.com

Daniel Stuparu
Babes-Bolyai University, Cluj-Napoca
E-mail:dstuparu@staff.ubbcluj.ro

Abstract

The databases appeared as a result of human need to store large amounts of data and in order to facilitate the usage of these data, standard sets of interrogations were created. SQL query language won the battle over the other sets. As the query languages evolved, database management systems added more and more instructions and statements. One of the most recent features is the ***Common Table Expressions****, introduced by SQL Server 2005. Microsoft SQL Server 2005 is the first version of SQL Server that supports recursive queries and implements a standard recursive Common Table Expression (known as CTE). Common Table Expressions are new constructs that offer a more readable form of the derived table that can be declared once and referenced multiple times in a query. Moreover, CTEs can be recursively defined, allowing a recursive entity to be enumerated without the need for recursive-stored procedures.*

This paper proposes an incursion in the world of Common Table Expression, in order to find its features, the advantages and disadvantages of using it.

1. Introduction

When working with database data, there might appear the need to operate over a set of data that doesn't inherently exist within the system. For example, some reports need to run over a particular subset of data or filter by results returned by scalar subqueries. Before CTE was introduced, views were used to split complex queries into subqueries. Common Table Expressions offer the same functionality as a view, but are ideal for one-off usages where is not necessary to define a view for the system. A main drawback of the views is the fact that they are permanent objects at the system-level. So, if is needed only a reference to a complex query in a single stored procedure or UDF, other option is to use a derived table. The derived tables have other drawback: they decrease the readability of the query and must be repeated for each use in a statement.

The Common Table Expression is a "temporary result set" that exists only within the scope of a single SQL statement. It allows access to functionality within that single SQL statement that was previously only available through use of functions, temp tables, cursors, etc. A CTE can be created using the WITH Statement.

2. General Description

2.1. General Structure

A CTE is defined using the WITH statement and is made up of an expression name representing the CTE, an optional column list, and a query. After a CTE is defined, it can be referenced like a table or a view in a SELECT, INSERT, UPDATE, or DELETE statement. A CTE can also be used in a CREATE VIEW statement as part of its defining SELECT statement, (recursive CTE). [2]

The basic syntax structure for a CTE is:

```
WITH expr_name[(column_name[,...n])] AS
   ( CTE_query_definition )
```

The use of CTE statements has some advantages like recursion and usually is faster than the previous statements (views and derived tables), but it also has some drawbacks as the CTE's members cannot contain the following clauses or keywords: DISTINCT, GROUP BY, HAVING, TOP, LEFT/RIGHT/OUTER JOIN, reducing by this the complexity of the allowed queries.

2.2. Advantages and Disadvantages

CTEs can be used in a recursively and also in a non-recursive form. The non-recursive form of CTE can be compared to a derived table or to a view and can substitute them, being used in place-of user-defined routines. The creation of the tables required within the nested Select statements simplifies the query building

by allowing them to be developed in separate logical blocks.

The CTE usage has some disadvantages, one of them being the fact that CTE's members cannot use the following clauses or keywords: DISTINCT, GROUP BY, HAVING, TOP, LEFT/RIGHT/OUTER JOIN, limiting by this the types of the queries that can be created and reducing their complexity.

Other disadvantage can be the fact that the recursion is limited as the recursive member can refer to the CTE only once.

There are no major disadvantages in using common table expressions, except those due to lack of implementation: table variables and CTEs cannot be passed as parameters in stored procedures. This feature might be implemented in server future versions.

3. Comparison with Temporary Tables

A temporary table is created and populated on disk, in the system database tempdb—with a session-specific identifier packed onto the name, in order to differentiate between similarly-named #temp tables created from other sessions. The data in this #temp table (in fact, the table itself) is visible only to the current scope (a stored procedure, or a set of nested stored procedures). The table is cleared up automatically when the session that created it ends or when the stored procedure it was created in finishes execution.

The use of temporary tables may cause performance issues due to the locking of the tempdb while the temporary table is being created, due to the I/0 activity involved during the use of the temporary table. Other issues are the potential locking of the tempdb if a transaction is used for the creation and the subsequent operations against the temporary table, and the problems SQL Server has with operations against temporary tables. [4]

A CTE can be created using the WITH statement with the CTE name following it. The temporary tables aren't as convenient. Temporary tables can be populated only after they have been created and not otherwise, whereas in a Common Table Expression, it can be called immediately after stating it. As for calling them, both a CTE and a temporary table can be called by name using the SELECT * statement.[3]

As a conclusion, CTE is superior to the temporary tables due to the performance issues related to the temporary tables and due to the fact that CTE structure is easier to use.

4. Comparison with Derived Tables

A derived table can be created by simply moving a SELECT statement inside of a FROM clause surrounded by parenthesis and an alias. The CTEs in SQL Server 2005 improves the derived table concept from SQL 2000. These concepts are quite similar; the main difference is that the derived tables are used only in simple queries. The derived tables also have two main drawbacks: they can be used only once, and they cannot be referred by name outside of the statement where were used. Also, the derived tablemake a query more difficult to read and thus to maintain. CTEs can be used for complex statements, and also for simple queries (a big plus regarding to derived tables). In the following example a CTEs are rewritten as derived tables [5]:

```
SELECT * FROM
(
   SELECT Amount, ( Amount * .95) AS Total
      FROM itemlist.item
)
DerivedTable
```

Like a derived table, a CTE lasts only for the duration of a query but, in contrast to a derived table, a CTE can be referenced multiple times in the same query.

5. Recursion and CTE

One of the greatest features offered by common table expression is the advantage of being able to reference itself, offering the method to create a recursive CTE. A CTE is recursive when an initial CTE is repeatedly executed to return subsets of data until the complete result set is obtained. Using recursive CTE, recursive queries can be used for hierarchical data retrieving in SQL Server 2005. In earlier versions of SQL Server, a recursive query usually requires the usage of temporary tables, cursors, and logic to control the flow of the recursive steps.[6]

The basic formula for creating a recursive CTE is as follows:

- Create the query that returns the top level (this is the anchor member)
- Write a recursive query (this is the recursive member)
- UNION the first query with the recursive part.
- The ending condition on the recursive algorithm is the case where no rows will be returned (the termination check).

A recursive CTE can return multiple rows and consists of three elements:

- Invocation of the routine - The first invocation of the recursive CTE consists of one or more *CTE_query_definitions* joined by UNION ALL, UNION, EXCEPT, or INTERSECT operators. Because these query definitions form the base result set of the CTE structure, they are referred to as anchor members.
- Recursive invocation of the routine - The recursive invocation includes one or more *CTE_query_definitions* joined by UNION ALL operators that reference the CTE itself. These query definitions are referred to as recursive members.
- Termination check - The termination check is implicit; recursion stops when no rows are returned from the previous invocation.

The following guidelines apply to using a recursive CTE:

- It must contain both an anchor member and a recursive member;
- Both members must have the same number of columns and the columns belonging to both members must have matching data types.

Please also notice:

- All columns returned by the recursive CTE are nullable regardless of the nullability of the columns returned by the participating SELECT statements.
- A view that contains a recursive common table expression cannot be used to update data.
- Tables on remote servers may be referenced in the CTE. If the remote server is referenced in the recursive member of the CTE, a spool is created for each remote table so the tables can be repeatedly accessed locally.

5.1. Multiple Anchor Members

Any entry that does not reference the CTE is considered an anchor member. In the previous example only one anchor was used , but CTE supports multiple anchors. An example of multiple anchor usage is: assuming that a database contains a table Monkeys with some fields like MonkeyId, Name, MotherId, FatherId, etc, and using a recursive query with MotherID and FatherId fields as anchors, a genealogy can be retrieved.

5.2. MaxRecursion

An incorrectly composed recursive CTE may cause an infinite loop. For example, if the recursive member query definition returns the same values for both the parent and child columns, an infinite loop is created. When testing the results of a recursive query, the number of recursion levels allowed for a specific statement can be limited by using the MAXRECURSION option and a value between 0 and 32,767 in the OPTION clause of the INSERT, UPDATE, DELETE, or SELECT statement. The server-wide default is 100, and when 0 is specified, no limit is applied. Only one MAXRECURSION value can be specified per statement. If the MAXRECURSION limit is reached, an exception is thrown and the execution is interrupted.

6. Testing Scenario

The following performance tests were executed in order to investigate the advantages and disadvantages of CTE usage. The technical data, the I/O access number and the execution time are obtained using SQLProfiler. The database used for testing contains only two tables, the first test scenario is for a nonrecursive statement - executes a simple join between the two tables., and the second one has a hierarchical structure in order to simulate recursion.

TBL_Organigrama [codunitate int , nivelunitate int , codunitateierarhicsuperior int]

Script 1. Using WITH statement:

```
WITH Unitate_Parinte AS (
        SELECT codunitateierarhicsup as cip
        FROM TBL_organigrama
        WHERE nivelunitate  in (1, 2 ,3, 4, 5)
)
SELECT codunitate, denumire, nivelunitate
        FROM    TBL_Unitate    u    INNER    JOIN
Unitate_Parinte up on u.codunitate = up.cip
```

Script 2. Using sub-select statement:

```
SELECT codunitate, denumire, nivelunitate
        FROM TBL_Unitate u
        INNER JOIN
        (SELECT codunitateierarhicsup as cip
                FROM TBL_Organigrama
                WHERE nivelunitate in (1, 2 ,3,
                4, 5)
        )
up ON u.codunitate = up.cip
```

The second test: Script 3. Using recursive WITH statement:

```
WITH Explorer AS(
        SELECT codunitate AS cu,
                codunitateierarhicsup AS cis
        FROM TBL_Organigrama
        WHERE codunitateierarhicsup = 1
                AND nivelunitate in (1)

        UNION ALL
```

```
                SELECT    codunitate    AS    cu,
                codunitateierarhicsup AS cis
                FROM TBL_Organigrama MN
                INNER JOIN Explorer MJ
                ON     MN.codunitateierarhicsup=
                MJ.cu
        )

SELECT TBL_Unitate.codunitate,
        TBL_Unitate.denumire,
        TBL_Unitate.marcaunitate,
        TBL_Unitate.nivelunitate
FROM TBL_Unitate
INNER JOIN Explorer oo
ON oo.cu = TBL_Unitate.codunitate
ORDER BY denumire
```

Script 4. Using sub-select statements for emulate the recursion:

```
SELECT TBL_Unitate.codunitate,
        TBL_Unitate.denumire,
        TBL_Unitate.marcaunitate,
        TBL_Unitate.nivelunitate
FROM TBL_Unitate
INNER JOIN(
        SELECT o1.codunitate
        FROM TBL_Organigrama o1
        WHERE o1.codunitateierarhicsup=1
                and o1.nivelunitate in (1)
        UNION ALL
        ...
        SELECT o5.codunitate
        FROM TBL_Organigrama o1
        ...
        INNER JOIN TBL_Organigrama o4
        ON                          o3.codunitate=
o4.codunitateierarhicsup
        INNER   JOIN   TBL_Organigrama   o5   ON
o4.codunitate = o5.codunitateierarhicsup
        WHERE o1.codunitateierarhicsup = 1 and
o1.nivelunitate in (1,2,3,4,5)
)
oo ON oo.codunitate = TBL_Unitate.codunitate
ORDER BY denumire
```

The results of executing these four statements on database can be summarized in the below table. The results were obtained for a number of 100.000 records – the first two lines and 1.000.000 records - the last two lines:

	Script1	**Script2**	**Script3**	**Script4**
Reads (no)	**1662**	1662	**1446050**	313900
Duration (ms)	**263**	296	**4313**	2203
Reads (no)	**18111**	18111	**19085082**	76393
Duration (ms)	**1956**	1960	**68610**	31080

Table 1. Testing results

Some conclusions can be extracted from the test results:

- In case of using CTEs in statements that do not imply recursion (script 1), the tests duration is almost the same with sub-select statements (script 2), because the SQL Server make the same execution plan for both; But CTEs improve the readability of the statements.
- As these two statements have the same execution plan, the number of disk reads is the same, fact confirmed by the table above;

- In the case of data that must be hierarchically accessed, CTEs's (Script 3) behavior is less performant than the corresponding sub-selects instruction.(Script 4). According to the execution results, it seems that CTEs makes about 100 times more disk reads than sub-select statement and the execution time is about two times increased. This behavior is detected when the database has more than 1 million records.

- The execution plan generated by the SQL Server for CTEs is much simple than the sub-select statement case.

- A disadvantage of using CTEs (Script 3) in a recursive scenario is that it covers all data from table regardless the level of details, so these extra operations increased the total execution time.
- The advantage is the dimension of the scripts and stronger logics, so the modification of a CTE statement needs less attention and less time that the sub-selects.

7. Conclusion

CTE's provide a way to make writing T-SQL much more readable when compared to scenarios that use complex derived tables within a query or referencing a view whose definition is external to the T-SQL batch.

With CTEs there is no need of formally declaring the column types, the column names are enought, as the types are assumed from the query definition. The syntax is for sure lighter.

The recursive CTEs can be used when the data dynamics of the table is reduced, or when the table's data have a number of hierarchical levels that cannot be covered in the classical manner, using sub-select joins.

So CTE's also provide an improved tool to address the struggles involved in using recursive algorithms. Whether non-recursive or recursive CTE's are used, the CTE's can help address many common

development scenarios and can improve readability without sacrificing performance.

8. References

[1] "WITH common_table_expression (Transact-SQL)", *SQL Server 2005 Books Online*, September 2007

[2] "Using Common Table Expressions", *SQL Server 2005 Books Online*, September 2007

[3] Steve Manik, "Working With Temporary Views Using Common Table Expression in SQL Server 2005", *http://www.sql-server-performance.com/articles, 25* May 2005, pp.2

[4] Randy Dyess, "Are SQL Server Temp Tables Really Necessary?" *http://www.sql-server-performance.com/articles",* Dec 2003

[5] Steve Manik, „Working With Temporary Views Using Common Table Expression in SQL Server 2005", *http://www.sql-server-performance.com,*

[6] „Recursive Queries Using Common Table Expressions", *SQL Server 2005 Books Online*, September 2005