

Common Table Expressions (Introduction to CTE's)

CTE 的主要作用是：

1. 简化复杂的 JOIN 和 subquery
2. 提供查询结构化数据的能力（比如树状结构）

The Common Table Expressions or CTE's for short are used within SQL Server to simplify complex joins and subqueries, and to provide a means to query hierarchical data such as an organizational chart.

In this article, we'll introduce you to common table expressions, the two types of the CTEs, and their uses. In addition, we'll introduce CTE's overall. Once you're familiar, I highly encourage you to read these articles as well:

- Non recursive CTE's
- Recursive CTE's

Recursive CTE 比较难理解。

Introduction to Common Table Expressions

A CTE (Common Table Expression) is a temporary result set that you can reference within another `SELECT`, `INSERT`, `UPDATE`, or `DELETE` statement. They were introduced in SQL Server version 2005. They are SQL-compliant and part of the ANSI SQL 99 specification.

A CTE always returns a result set. They are used to simplify queries, for example, you could use one to eliminate a derived table from the main query body.

Note: All the examples for this lesson are based on Microsoft SQL Server Management Studio and the AdventureWorks2012 database. You can get started using these free tools using my Guide *Getting Started Using SQL Server*.

What is a CTE or Common Table Expression in SQL Server?

A CTE (Common Table Expression) defines a temporary result set which you can then use in a `SELECT` statement. It becomes a convenient way to manage complicated queries.

Common Table Expressions are defined within the statement using the `WITH` operator. You can define one or more common table expression in this fashion.

Here is a really simple example of one CTE:

```
WITH Employee_CTE (EmployeeNumber, Title)
AS
(SELECT NationalIDNumber,
        JobTitle
 FROM   HumanResources.Employee)
SELECT EmployeeNumber,
        Title
FROM   Employee_CTE
```

Let's break this down a bit.

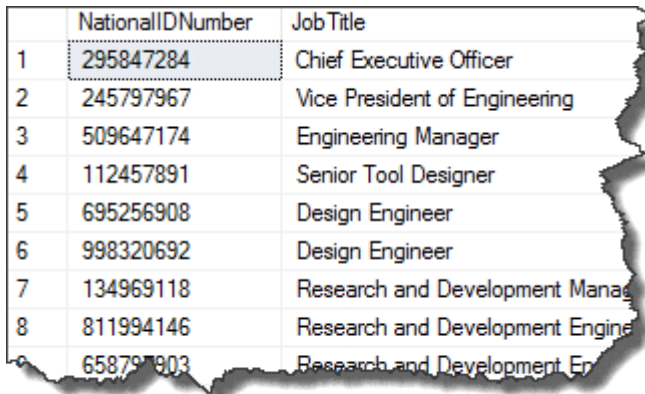
The diagram illustrates the structure of the SQL query. It is divided into two main sections by a horizontal line. The top section, highlighted in blue, contains the CTE definition: `With Employee_CTE (EmployeeNumber, Title) AS (SELECT NationalIDNumber, JobTitle FROM HumanResources.Employee)`. This section is bracketed on the right and labeled "CTE (Common Table Expression)". The bottom section, highlighted in yellow, contains the query that uses the CTE: `SELECT EmployeeNumber, Title FROM Employee_CTE`. This section is also bracketed on the right and labeled "Query Using CTE".

CTE Query Definition

The blue portion is the CTE. Notice it contains a query that can be run on its own in SQL. This is called the CTE query definition:

```
SELECT NationalIDNumber,  
       JobTitle  
FROM   HumanResources.Employee
```

When you run it you see results like:



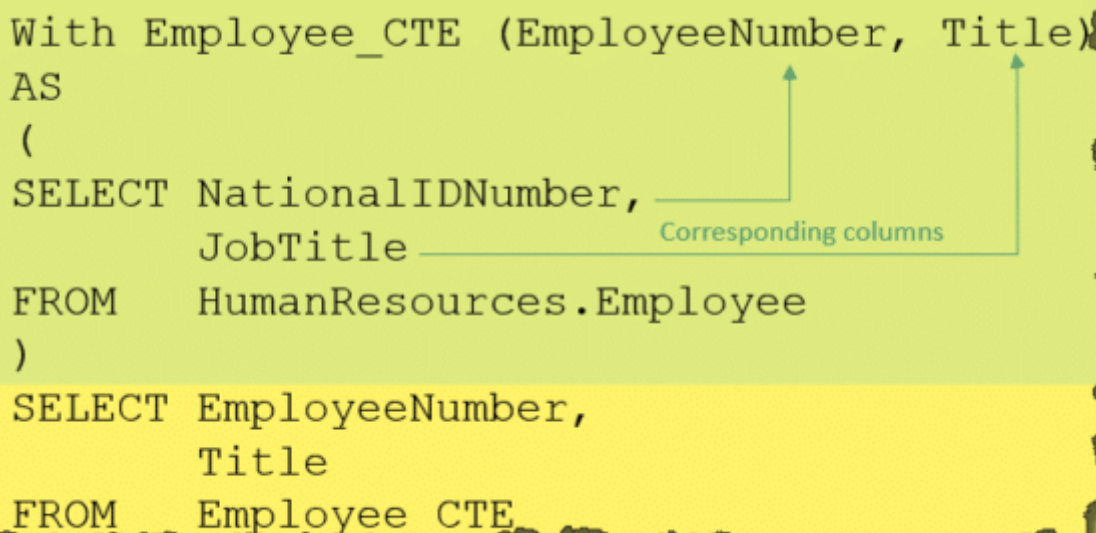
	NationalIDNumber	JobTitle
1	295847284	Chief Executive Officer
2	245797967	Vice President of Engineering
3	509647174	Engineering Manager
4	112457891	Senior Tool Designer
5	695256908	Design Engineer
6	998320692	Design Engineer
7	134969118	Research and Development Manager
8	811994146	Research and Development Engineer
9	658797903	Research and Development Engineer

CTE Query Definition Results

Notice that when we define the CTE we give the result a name as well its columns. In this way a CTE acts like a VIEW. The result and columns are named differently. This allows you to encapsulate complicated query logic with the common table expression.

Now going back to the CTE, notice that the WITH statement. There you'll see the name and columns are defined. These columns correspond to the columns returned from the inner query.

Employee_CTE (EmployeeNumber, Title) 这种语法，乍一看容易理解成函数调用，但实际上 EmployeeNumber 和 Title 表示的是 CTE 查询返回的列的名字。



```
With Employee_CTE (EmployeeNumber, Title)  
AS  
(  
  SELECT NationalIDNumber,  
         JobTitle  
  FROM   HumanResources.Employee  
)  
SELECT EmployeeNumber,  
       Title  
FROM   Employee_CTE
```

CTE Query Definition Column Mappings

Finally notice that our final query references the CTE and columns defined.

From our outer query's perspective all it "sees" is this definition. It isn't concerned how the CTE is constructed, just its name and columns.

As such, the results from the CTE are:

	EmployeeNumber	Title
1	295847284	Chief Executive Officer
2	245797967	Vice President of Engineering
3	509647174	Engineering Manager
4	112457891	Senior Tool Designer
5	695256908	Design Engineer
6	998320692	Design Engineer
7	134969118	Research and Development Manager
8	811994146	Research and Development Engineer
9	658797903	Research and Development Engineer
10	879342154	Research and Development Manager
11	974026903	Senior Tool Designer

Notice the column names, they're based on the those defined in the CTE.

I want to point out that you can define more than one CTE within a WITH statement. This can help you simplify some very complicated queries which are ultimately joined together. Each complicated piece can include in their own CTE which is then referred to and joined outside the WITH clause.

Here is an example using of TWO CTE's, it's a simple example, but it shows how two CTE's are defined, and then used in an INNER JOIN

```
WITH PersonCTE (BusinessEntityID, FirstName, LastName)
AS (SELECT Person.BusinessEntityID,
          FirstName,
          LastName
     FROM Person.Person
     WHERE LastName LIKE 'C%'),
PhoneCTE (BusinessEntityID, PhoneNumber)
AS (SELECT BusinessEntityID,
          PhoneNumber
```

一个语句可以有多条 CTE (with 子句), 并且
它们的查询结果可以互相 JOIN。

```

FROM Person.PersonPhone)

SELECT FirstName,
       LastName,
       PhoneNumber
FROM   PersonCTE
INNER JOIN
PhoneCTE
ON PersonCTE.BusinessEntityID = PhoneCTE.BusinessEntityID;

```

The first common table expression is colored **green**, the second **blue**. As you can see from the SELECT statement the CTE's are joined as if they were tables.

Hopefully you can see that as your queries become more complicated, CTE's can become a really useful way to separate operations; therefore, simplify your final query.

为什么使用 CTE ?

Why Do you need CTE's?

1. 可读性。CTE 比复杂的 JOIN 和 subquery 可读性强
2. 作为 View 的替代。CTE 跟 view 非常类似，可以看作是一次性的 view。假如你没有权限创建 view，你可以使用 CTE
3. 递归。查询结构化数据时需要递归能力
4. 避开普通 SELECT 语句的限制，比如 referencing itself (也

There are several reasons why you may want to use a CTE over other methods.

Some of them include: 就是第 3 点的递归)，以及在 GROUP BY 中使用非决定性函数（？？）
5. 使用 RANKING 功能

- **Readability** – CTE's promote readability. Rather than lump all you query logic into one large query, create several CTE's, which are the combined later in the statement. This lets you get the chunks of data you need and combine them in a final SELECT.
- **Substitute for a View** – You can substitute a CTE for a view. This is handy if you don't have permissions to create a view object or you don't want to create one as it is only used in this one query.
- **Recursion** – Use CTE's do create recursive queries, that is queries that can call themselves. This is handy when you need to work on hierarchical data such as organization charts.
- **Limitations** – Overcome SELECT statement limitations, such as referencing itself (recursion), or performing GROUP BY using non-deterministic functions.
- **Ranking** – Whenever you want to use ranking function such as ROW_NUMBER(), RANK(), NTILE() etc.

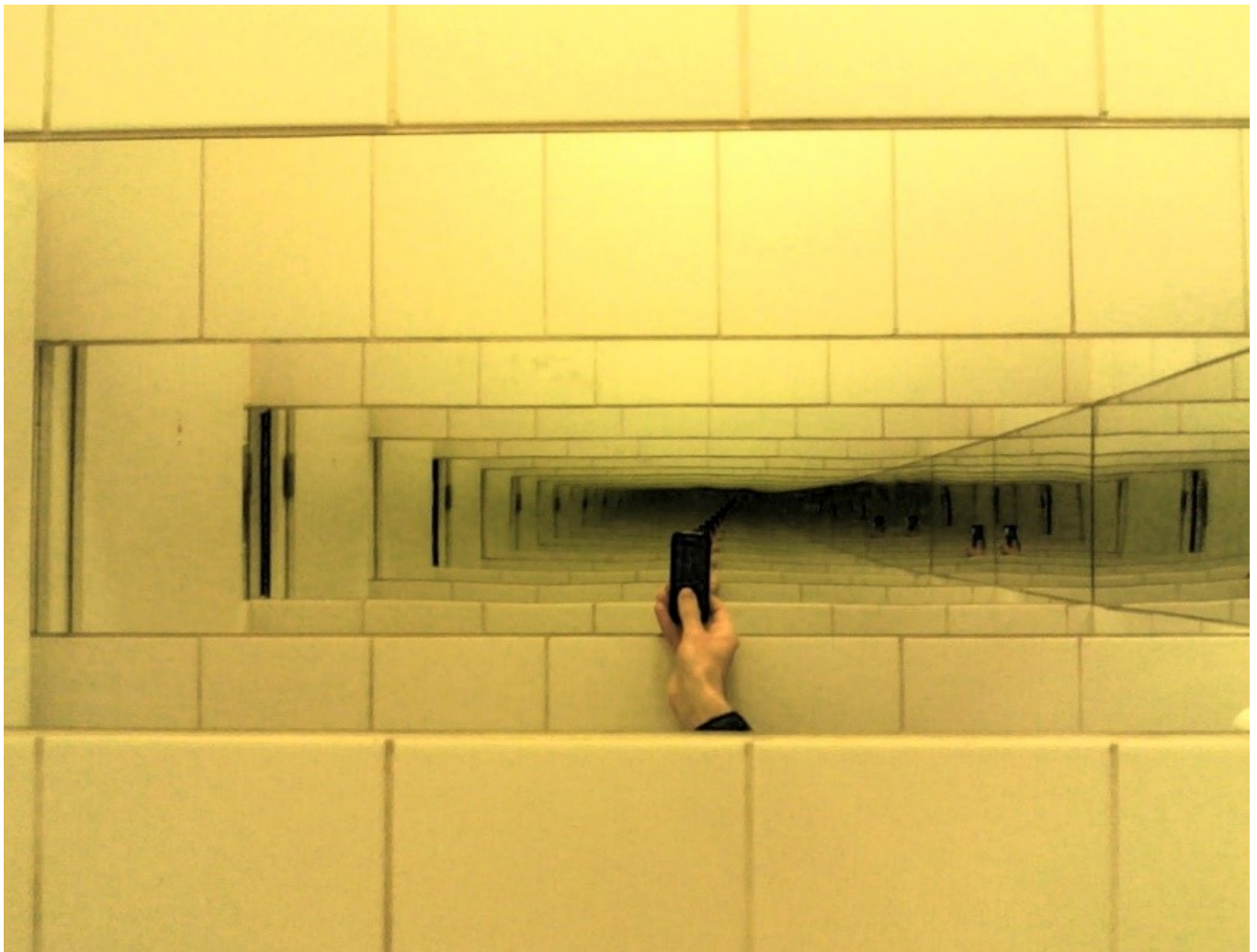
Types of CTE's

Common Table Expressions can be placed into two broad categories: Recursive CTE's and Non-Recursive CTE's.

Recursive CTE's are common table expressions that reference themselves.

Recursion can be a pretty difficult topic to grasp, I really didn't get it until I took a LISP class way back in 1986, but hopefully I can explain it to you.

We'll go deep into recursion in a separate post, but for now let me introduce to you recursion using this diagram:



Here you see picture of opposing mirrors. Due to the reflection, it becomes a picture in a picture.

Recursive queries are like that.

As a recursive query is run, it repeatedly runs on a subset of the data. A recursive query is basically a query that calls itself. At some point there is an end condition, so it doesn't call itself indefinitely.

In a way when you look into the picture you can imagine each picture in a picture is the picture calling itself. However, unlike the "infinite reflection" in the mirrors, there comes a point where a recursive query encounters the end condition and stop calling itself.

At that point, the recursion starts to unwind, collect and calculate data as it reviews each successive result.

Non-Recursive CTE's, as the name implies, are don't use recursion. They don't reference themselves. They are easier to understand so we'll look at them first in detail in the next article within this series.

Conclusion

Hopefully you now have an appreciation of what CTE's are and why we may want to use them. In the next two article we'll go into much greater detail on CTE's, and when to use them.

Until then, I would recommend reviewing my articles on [joins](#) and [subqueries](#) as we'll draw upon these types of queries for our examples.

