Unleashing CTE's

STEVE STEDMAN

UNLEASHING COMMON TABLE EXPRESSIONS IN SQL SERVER

http://SteveStedman.com
Follow me on Twitter @SqlEmt

Instructor: Steve Stedman

- Developer of the Database Health Project
  - http://DatabaseHealth.SteveStedman.com
- DBA/Consultant/Trainer/Speaker/Writer
  - Been using SQL Server since 1991
  - Taught SQL Server classes at WWU
  - SQL Server consultant
  - Writer contributing to a new SQL Server book – Tribal SQL
- Working at Emergency Reporting as CTO
- Volunteer Firefighter and EMT
- http://SteveStedman.com for more information.

Presentation Overview - CTE

1. Prerequisites
2. What is a Common Table Expression
3. Simple CTE
4. CTE – Subquery Re-use
5. Recursive CTE
6. Multiple CTE
7. CTE Common Uses
8. CTE for Math Geeks
9. CTE Fact or Fiction – Debunking

Awesome

Data + Brains = Awesome

On twitter by @devNambi

Data + Brains + CTE's = More Awesome

On twitter as @SqlEmt

1. Prerequisites

- To get the most value out of this presentation you should:
  - Be familiar with TSQL and able to write queries.
  - Have experience with Subqueries
  - Understand execution plans

2. What is a Common Table Expression?

- Similar to the ease of a temporary table
- Like a Temporary Named Result Set
- Acts like a temporary view, or a run time / inline view, which provides temporary result during runtime
- TRANSACT SQL feature that I wish I had learned about 7 years ago, CTE's were introduced in SQL Server 2005

Why Use Common Table Expressions?

- Simplify your query – test one part at a time
- Recursion
  - Computer Science: When a function calls itself
- Make queries and subqueries more readable

CTE Syntax - WITH

- Queries start with ;WITH not SELECT
- Can be confusing if you are assuming that any query to select data would start with a SELECT

3. Simple CTE Syntax

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

SELECT <column_list>
FROM expression_name;
```

Demo: Simple CTE

```
;WITH deptCTE (id, department, parent) AS
(
  SELECT id, department, parent
    FROM Departments
)

SELECT * FROM deptCTE;
```

More about CTE's

- Can be use to
  - Create a recursive query
  - Simplify a query by using a result set multiple times
  - Self JOIN a subquery
- Each time the CTE is referenced, the CTE query is run again
- If a CTE is not the first statement in a batch it must be proceeded with a semicolon

4. CTE Subquery Re-use

- Simplifies the query
- Does not improve the performance
- More value for large subqueries in that the TSQL is cleaner and easier to read and understand
- Eliminates accidents by duplicating subquery TSQL code

Subquery without CTE

```
SELECT q1.department, q2.department
  FROM  (SELECT id, department, parent
          FROM Departments) as q1
INNER JOIN (SELECT id, department, parent
          FROM Departments) as q2
      ON q1.id = q2.parent
WHERE q1.parent is null;
```

CTE for SubQuery Re-use

```
;WITH deptCTE(id, department, parent) AS
(SELECT id, department, parent
   FROM Departments)
SELECT q1.department, q2.department
  FROM deptCTE q1
 INNER JOIN deptCTE q2 on q1.id = q2.parent
WHERE q1.parent is null;
```

5. Recursive CTE

- It is recursive when the CTE references itself
- Recursion stops when the second SELECT produces no results
- Specify MAXRECURSION
  - Default is 100
  - MAXRECURSION of 0 implies no maximum
- Uses
  - Hierarchical listing of categories
  - Recursive calculations

Demo: Recursive CTE

```
;WITH DepartmentCTE(DeptId, Department, Parent, Level) AS
( SELECT id as DeptId, Department, parent, 0 as Level
    FROM Departments
   WHERE parent is NULL
   UNION ALL -- and now for the recursive part
  SELECT d.id as DeptId, d.Department, d.parent,
            DepartmentCTE.Level + 1 as Level
    FROM Departments d
   INNER JOIN DepartmentCTE
         ON DepartmentCTE.DeptId = d.parent)
SELECT *
 FROM DepartmentCTE
ORDER BY parent;
```

Recursive CTE with Tree Path

- Tree Path shows the department and all parent departments.
- Simple to do with a recursive CTE

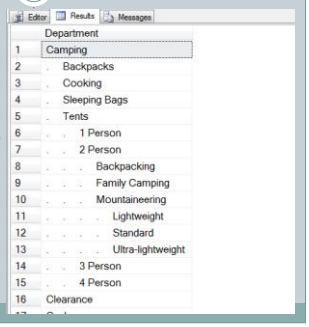
Level	TreePath
0	Camping
1	Camping -> Backpacks
1	Camping -> Cooking
1	Camping -> Sleeping Bags
1	Camping -> Tents
2	Camping -> Tents -> 1 Person
2	Camping -> Tents -> 2 Person
3	Camping -> Tents -> 2 Person -> Backpacking
3	Camping -> Tents -> 2 Person -> Family Camping
3	Camping -> Tents -> 2 Person -> Mountaineering
4	Camping -> Tents -> 2 Person -> Mountaineering -> Lightweight
4	Camping -> Tents -> 2 Person -> Mountaineering -> Standard
4	Camping -> Tents -> 2 Person -> Mountaineering -> Ultra-lightweight
2	Camping -> Tents -> 3 Person
2	Camping -> Tents -> 4 Person
0	Clearance
0	Cycle
1	Cycle -> Bikes
1	Cycle -> Helmets

Demo: Recursive CTE with Tree Path

```
;WITH DepartmentCTE(DeptId, Department, Parent, Level, TreePath) AS
( SELECT id as DeptId, Department, parent, 0 as Level,
            cast(Department as varchar(1024)) as TreePath
    FROM Departments
   WHERE parent is NULL
   UNION ALL -- and now for the recursive part
  SELECT d.id as DeptId, d.Department, d.parent,
            DepartmentCTE.Level + 1 as Level,
            cast(DepartmentCTE.TreePath + ' -> ' +
            cast(d.department as varchar(1024))
            as varchar(1024)) as TreePath
    FROM Departments d
   INNER JOIN DepartmentCTE
         ON DepartmentCTE.DeptId = d.parent)
SELECT *
 FROM DepartmentCTE
ORDER BY TreePath;
```

Recursive CTE with Indendation

- Simple add on to Tree Path query
- Still using Tree Path for sort order
- Using the SQL Server REPLICATE function to indent the category.



	Department
1	Camping
2	. Backpacks
3	. Cooking
4	. Sleeping Bags
5	. Tents
6	. . 1 Person
7	. . 2 Person
8	. . . Backpacking
9	. . . Family Camping
10	. . . Mountaineering
11	. . . . Lightweight
12	. . . . Standard
13	. . . . Ultra-lightweight
14	. . 3 Person
15	. . 4 Person
16	Clearance

Recursive CTE with Indentation

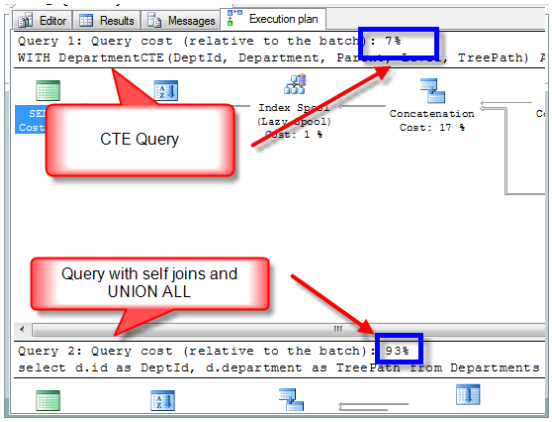
```
;WITH DepartmentCTE(DeptId, Department, Parent, Level, TreePath) AS
( SELECT id as DeptId, Department, parent, 0 as Level,
            cast(Department as varchar(1024)) as TreePath
   FROM Departments
   WHERE parent is NULL
   UNION ALL -- and now for the recursive part
  SELECT d.id as DeptId, d.Department, d.parent,
       DepartmentCTE.Level + 1 as Level,
       cast(DepartmentCTE.TreePath + ' -> ' +
          cast(d.department as varchar(1024))
        as varchar(1024)) as TreePath
   FROM Departments d
   INNER JOIN DepartmentCTE
   ON DepartmentCTE.DeptId = d.parent)
SELECT REPLICATE('.  ', Level) + Department
  FROM DepartmentCTE
 ORDER BY TreePath;
```

Recursive CTE Performance

- Using a CTE for re-use of a subquery does not improve performance.
- CTE Compared to a UNION ALL self join to create 6 levels in the hierarchy has a huge performance difference, plus the CTE version is much easier to read.

Hierarchical Query without CTE

- Many Self Joins with a UNION ALL
- Nested Cursors
- Performance Differences
  - Sample Department query with self joins takes 13 times as long as CTE. 7% compared to 93%.



6. Multiple CTE's In A Single Query

- You can include multiple CTE's by comma seperating them:

```
;WITH firstCTE (query goes here),

secondCTE (second query goes here)

SELECT * FROM firstCTE
  INNER JOIN secondCTE on ....
```

Demo: Multiple CTE

```
;WITH Fnames (Name) AS
(SELECT 'John' UNION Select 'Mary' UNION Select 'Bill'),
Minitials (initial) AS
(SELECT 'A' UNION SELECT 'B' UNION SELECT 'C'),
Lnames (Name) AS
(SELECT 'Anderson' UNION Select 'Hanson' UNION Select
   'Jones')

SELECT F.Name, M.initial, L.Name
  FROM Fnames F
 CROSS JOIN Lnames as L
 CROSS JOIN Minitials m;
```

7. Other Common CTE Uses

- Data paging on a search result
- Information on the dates in a year
- Creating a replacement for a Numbers table
- Breaking up or parsing strings into tables
  - Query String
  - SQL Server connect string
- Simplifying or breaking up a huge query

Data Paging

- To achieve data paging without CTE it usually involves selecting TOP x, then TOP 2x then top 3x, each time taking longer and longer to get to the data that is needed.
- Data paging can be simplified and not a challenge to create with CTE's.
- TSQL 2012 introduces the OFFSET and FETCH keywords which is easier to use than a CTE for data paging, and more efficient.

Data Paging Page 1

	TableName	ColumnName	RowNum
1	Departments	department	1
2	Departments	id	2
3	Departments	parent	3
4	filestream_tombstone_2073058421	oplsn_bOffset	4
5	filestream_tombstone_2073058421	oplsn_fseqno	5
6	filestream_tombstone_2073058421	oplsn_slotid	6
7	filestream_tombstone_2073058421	rowset_guid	7
8	filestream_tombstone_2073058421	status	8
9	filestream_tombstone_2073058421	transaction_sequence_num	9
10	filestream_tombstone_2073058421	file_id	10

Data Paging Page 2

	TableName	ColumnName	RowNum
1	filestream_tombstone_2073058421	filestream_value_name	11
2	filestream_tombstone_2073058421	column_guid	12
3	queue_messages_1977058079	conversation_handle	13
4	queue_messages_1977058079	conversation_group_id	14
5	queue_messages_1977058079	binary_message_body	15
6	queue_messages_1977058079	fragment_size	16
7	queue_messages_1977058079	fragment_bitmap	17
8	queue_messages_1977058079	message_id	18
9	queue_messages_1977058079	message_sequence_number	19
10	queue_messages_1977058079	message_type_id	20

Data Paging Page 3

	TableName	ColumnName	RowNum
1	queue_messages_1977058079	next_fragment	21
2	queue_messages_1977058079	validation	22
3	queue_messages_1977058079	status	23
4	queue_messages_1977058079	service_contract_id	24
5	queue_messages_1977058079	service_id	25
6	queue_messages_1977058079	priority	26
7	queue_messages_1977058079	queuing_order	27
8	queue_messages_2009058193	queuing_order	28
9	queue_messages_2009058193	priority	29
10	queue_messages_2009058193	service_id	30

Demo: Data Paging

```
;WITH TablesAndColumns AS (
  SELECT OBJECT_NAME(sc.object_id) AS TableName,
         name AS ColumnName,
         row_number()
         OVER (ORDER BY object_name(sc.object_id))
         AS Row
  FROM sys.columns sc )
SELECT *
  FROM TablesAndColumns
  WHERE Row BETWEEN (@pageNum - 1) * @pageSize + 1
                AND @pageNum * @pageSize ;
```

Demo: SQL Server 2012 Data Paging

```
SELECT OBJECT_NAME(sc.object_id) AS TableName,
       name AS ColumnName
  FROM sys.columns sc
  ORDER BY TableName
OFFSET (@pageNum - 1) * @pageSize ROWS
  FETCH NEXT @pageSize ROWS ONLY;
```

•An alternative to CTE's if you are using SQL Server 2012

Information on the dates in a year

```
;WITH Dates as (
  SELECT cast('2011-01-01' as date) as CalendarDate
    UNION ALL
  SELECT dateadd(day , 1, CalendarDate) AS CalendarDate
    FROM Dates
  WHERE dateadd (day, 1, CalendarDate) < '2012-01-01'
)
SELECT
  CalendarDate,
...
  CalendarYear=year(CalendarDate),
DayOfWeek=datepart(weekday, CalendarDate)
FROM Dates
OPTION (MAXRECURSION 366);
```

Creating a replacement for a Numbers table

```
;WITH Numbers (N) AS
( SELECT 1
  UNION ALL
  SELECT 1 + N FROM Numbers
  WHERE N < 1000
)
 SELECT N
  FROM Numbers
 OPTION (MAXRECURSION 0);
```

Breaking up or parsing strings into tables

- Query String
  - Key1=Value1&Key2=Value2&Key3=Value3
- SQL Server connect string
  - server=myserver;user id=sa;password=asdfasdfasdasdffjfjfj

Simplifying huge queries

- Whether you like it or not, eventually you will end up with a really huge query
- CTE can be used to break up the huge query into smaller components that might be easier to understand than the one huge query

8. CTE For Math Geeks

- CTE Fibonacci sequence
- CTE Factorial

Fibonacci sequence

- 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144 …
- By definition, the first two numbers in the Fibonacci sequence are 0 and 1, and each subsequent number is the sum of the previous two.

Demo: Fibonacci Sequence

```
;WITH Fibonacci (PrevN, N) AS
( SELECT 0, 1
  UNION ALL
 SELECT N, PrevN + N
   FROM Fibonacci
  WHERE N < 1000000000)
 SELECT PrevN as Fibo
  FROM Fibonacci
 OPTION (MAXRECURSION 0);
```

Factorial

- The *factorial* of a positive integer n, written n!, is the product of all the positive integers from 1 up to and including n
- Example:

1! = 1

2! = 1 * 2 = 2

3! = 1 * 2 * 3 = 6

4! = 1 * 2 * 3 * 4 = 24

Demo: Factorial

```
;WITH Factorial (N, Factorial) AS
( SELECT 1, cast(1 as BIGINT)
  UNION ALL
 SELECT N + 1, (N + 1) * Factorial
   FROM Factorial
  WHERE N < 20
)
 SELECT N, Factorial
  FROM Factorial
 OPTION (MAXRECURSION 0);
```

Frequent CTE Questions - INSERT

- Can I use an INSERT statement inside of the CTE.
- **NO-** you are only allowed to do SELECT statements. But it can be used outside of the CTE in the normal query.

Frequent CTE Questions: Execution

- Does a query that JOINs a CTE to itself execute the CTE query once or twice:
- **TWICE.** To confirm write a CTE, JOIN several times, look at the execution plan.

Frequent CTE Questions: View

- How does the performance of a CTE compare to the performance of a view?
- The question assumes that we are not doing a recursive CTE, since you can't do recursion with an view.
- **They have similar performance.**

In Review

- What is a Common Table Expression
- Simple CTE
- CTE – Subquery Re-use
- Recursive CTE
- Multiple CTE
- CTE Common Uses
- CTE for Math Geeks

More Information

- Follow me on Twitter
  - @SqlEmt
- Visit my website
  - http://SteveStedman.com/
  - http://DatabaseHealth.SteveStedman.com
- Send me an email:
  - Steve@SteveStedman.com
- Download Slides and Sample TSQL
  - http://stevestedman.com/speaking/

CTE – Fact or Fiction

STEVE STEDMAN

DEBUNKING COMMON MYTHS ABOUT COMMON TABLE EXPRESSIONS

1. CTE Executions

As a named result set, the CTE is only run once even if it is referenced multiple times in a query.

True or False?

FALSE The CTE is executed once for EACH time that it is referenced in a query.

1. CTE Executions Explained

```
;WITH deptCTE(id, department, parent) AS
(SELECT id, department, parent
   FROM Departments)
SELECT q1.department, q2.department
 FROM deptCTE q1
 INNER JOIN deptCTE q2 on q1.id = q2.parent
 WHERE q1.parent is null;
```

- In this example the deptCTE is executed twice

2. CTEs are proprietary

CTEs are proprietary to Microsoft SQL Server.

True or False?

FALSE Common Table Expressions are supported by several major database platforms, among them PostgreSQL, DB2, Oracle and SQL Server, defined in SQL-99 spec

3. CTE and Hierarchical Queries

CTEs are a great way to create recursive hierarchical queries.

True or False?

TRUE Recursive hierarchical queries are easy to write with a CTE. CTE's save time, are easy to follow, and work great for hierarchical data.

5. Database Versions

SQL Server only supports CTE's on SQL Server Enterprise Edition 2008R2 and newer.

True or False?

FALSE Common Table Expressions have been supported since SQL Server 2005 and are available in all versions.

6. Stored Procedures and Functions

CTEs can be defined in user-defined routines, such as functions, stored procedures, triggers, or views.

True or False?

TRUE Common Table Expressions can be defined and used inside of stored procedures and functions.

7. CTEs and Nesting

CTEs can be nested and one CTE can reference an earlier CTE.

True or False?

TRUE Common Table Expressions can be nested. Just define multiple CTE's and reference an earlier CTE from a later one.

8. Indexing CTEs

Indexes can be added to CTEs to boost performance.

True or False?

FALSE A Common Table Expression is a temporary, "inline" view - you cannot add an index to a CTE.

9. VIEW vs CTE

Which performs better, a non-recursive CTE or a VIEW?

They are the same.

- The big gain is the recursive CTE, which you can't achieve with a view.

10. CTE's and Data Paging

CTE's are a great way to do Data Paging for a result grid.

True or False

It Depends.....

SQL Server 2012 has the new OFFSET and FETCH clause on select statements, which is easier than CTE's. For 2005, 2008 and 2008R2 the CTE is the best option.

11. CTE's performance

Recursive CTE's perform the same as other pseudo recursive solutions?

True or False

FALSE.....

More Information

- Follow me on Twitter
  - @SqlEmt
- Database Health Project
  - http://DatabaseHealth.SteveStedman.com
- Visit my website
  - http://stevestedman.com/
- Send me an email:
  - Steve@SteveStedman.com
- Download Slides and Sample TSQL
  - http://stevestedman.com/speaking/