SQL Server Common Table Expressions

STEVE STEDMAN

EVERYTHING YOU EVER WANTED TO KNOW ABOUT COMMON TABLE EXPRESSIONS

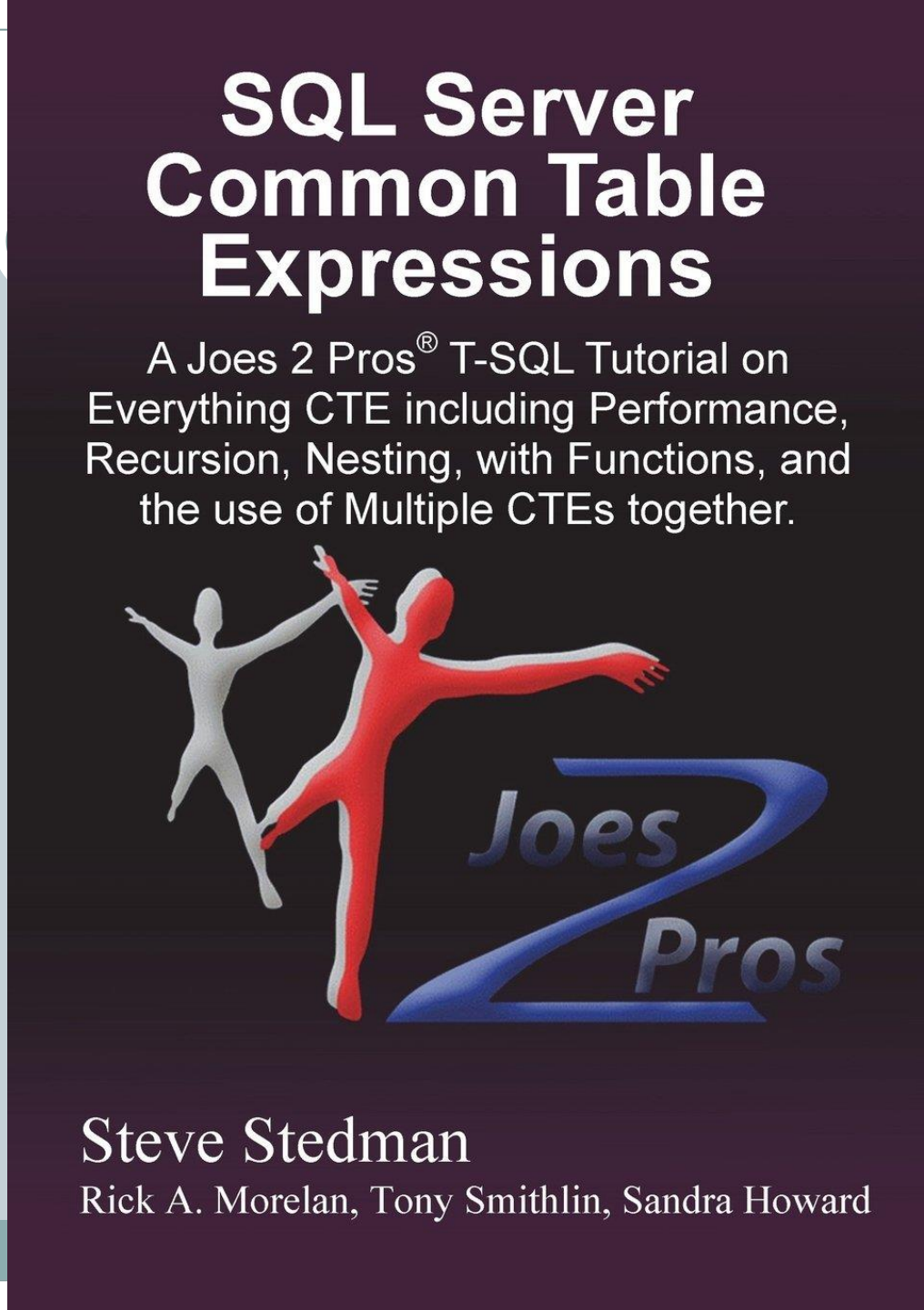
http://SteveStedman.com
Follow me on Twitter @SqlEmt

About Steve Stedman

- DBA/Consultant/Trainer/Speaker/Writer
  - Been using SQL Server since 1990 (SQL Server 1.0 for OS/2)
  - Taught SQL Server classes at WWU
  - SQL Server consultant
- Developer of the Database Health Project
  - http://DatabaseHealth.SteveStedman.com
- Working at Emergency Reporting as CTO
- Volunteer Firefighter and EMT
- http://SteveStedman.com for more information.

Common Table Expressions Book

- Published May 2013
- Available at Amazon.com and at Joes2Pros.com
- Printed and Kindle are both available now



Prerequisites

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

Presentation Overview - CTE

1. What is a Common Table Expression
2. Simple CTE
3. CTE instead of a Derived Table
4. Recursive CTE
5. Multiple CTEs in a Query
6. CTE Common Uses
7. Manipulating Data with a CTE
8. CTE for Math Geeks

1. What is a Common Table Expression?

- A type of a virtual table
- Similar to the ease of a temporary table
- Sort of like a derived table
- Like a temporary named result set
- Acts like a temporary view, or a run time view

Availability of CTEs

- TRANSACT SQL feature that I wish I had learned about 8 years ago, CTE's were introduced in SQL Server 2005
- All versions of SQL Server since SQL 2005 and all variations of SQL Server support CTEs
- CTEs are also available in SQL Azure.

Why Use Common Table Expressions?

- Simplify your query – test one part at a time
- Recursion
  - Computer Science: When a function calls itself
  - SQL Server: When a query calls itself
- Make derived table queries more readable
- Alternative to a temp table or a table variable

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
- The scope of the CTE is confined to a single query
- A CTE just seems a little weird, until you master the syntax

2. Simple CTE Syntax

;WITH

2. Simple CTE Syntax

;WITH **expression_name**

2. Simple CTE Syntax

;WITH expression_name **[(column_name[,...n])]**

2. Simple CTE Syntax

;WITH expression_name [(column_name[,...n])]

AS

2. Simple CTE Syntax

;WITH expression_name [(column_name[,...n])]

AS

(CTE_query_definition)

2. Simple CTE Syntax

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

SELECT <column_list>
FROM expression_name;
```

Demo: Simple CTE

;WITH **departmentsCTE**

Demo: Simple CTE

;WITH departmentsCTE **(id, department, parent)**

Demo: Simple CTE

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

Demo: Simple CTE

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

SELECT *
    FROM departmentsCTE;
```

Demo

Reminder

- If a CTE is not the first statement in a batch it must be proceeded with a semicolon

3. CTE Instead of a Derived Table

- Simplifies the query – allows for clean code
- Does not improve the performance
- More value for large derived table queries in that the TSQL is cleaner and easier to read and understand
- Eliminates accidents by duplicating derived table queries TSQL code

Derived Table Without a 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;
```

Steps to Convert a Derived Table to a CTE

1. Find the first occurrence of the derived table query to be broken out. Create a name for it and add “CTE” to the name.
2. Copy the derived table definition, including the parentheses, and leave the new name as the placeholder.
3. Paste the query, copied earlier, above the SELECT statement.
4. At the top of the query add the CTE declaration using the same name from step 1.
5. Find all other occurrences of the same derived table query and replace them with the CTE name.
6. Clean up the formatting and test the query.

CTE for Derived Table 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;
```

CTE Instead of a Derived Table Summary

- Most derived tables can be easily converted to a CTE
- Copy and paste errors can be reduced by using a CTE
- Using a CTE doesn't improve the performance over a similar query written with derived tables
- For a CTE that is referenced multiple times the CTE query is not reused, it is executed multiple times

4. Recursive CTE

- Considered recursive when the CTE references itself
- Recursion stops
  - When the second SELECT produces no results
  - Or specify MAXRECURSION
- Uses
  - Hierarchical listing of categories
  - Recursive calculations
  - Much, much more…

Recursive Terminology

- Anchor Query
  - Start the recursion
- Recursive Query
  - The part that repeats
- MAXRECURSION
  - The number of times to repeat the recursive query
  - Default is 100
  - MAXRECURSION of 0 implies no maximum

Recursion Overview

- Sum the numbers from 1 to 10 without recursion

55 = 10 + 9 + 8 + 7 + 6 + 5 + 4 +3 + 2 + 1

- Sum the numbers from 1 to 10 recursively

55 = 10 + (sum of numbers 1 to 9)
55 = 10 + (9 + (sum of numbers 1 to 8))
55 = 10 + (9 + (8 + (sum of numbers 1 to 7)))

Eventually we get to:

55 = 10 + (9 + (8 + (7 + (6 + (5 + (4 + (3 + (2 + 1)))))))))

Example of How a Recursive CTE Works

1. Select some starting set of data from table A.
2. Join that starting set of data to table A.
3. For the results from step 2, join that to Table A.
4. Repeat until there are no more items produced by the join.

Demo: Recursive CTE

```
;WITH DepartmentCTE(DeptId, Department, Parent, Level) AS
```

Step 1. Declare the CTE and Columns

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
```

Step 2 – Add the Anchor Query

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
```

Step 3 – Add the UNION ALL to connect to the recursive query

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)
```

Step 4 – Add the recursive Query

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
```

Step 1. Declare the CTE and Columns

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
```

Step 2 – Add the Anchor Query

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

Step 3 – Add the UNION ALL to connect to the recursive query

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 + ' -> ' +
                  d.department as varchar(1024)) as TreePath
   FROM Departments d
  INNER JOIN DepartmentCTE
       ON DepartmentCTE.DeptId = d.parent)
```

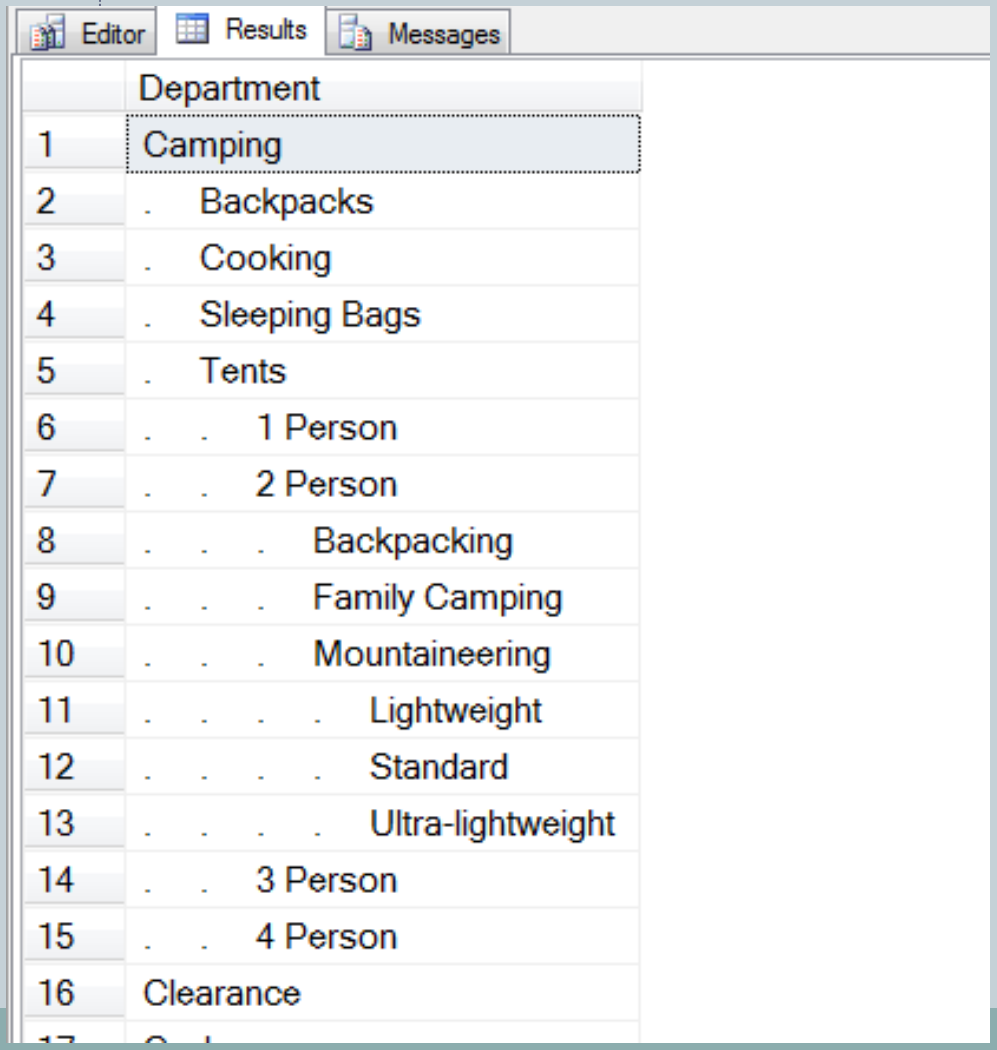
Step 4 – Add the recursive Query

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 + ' -> ' +
                      d.department 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 Indentation

- 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

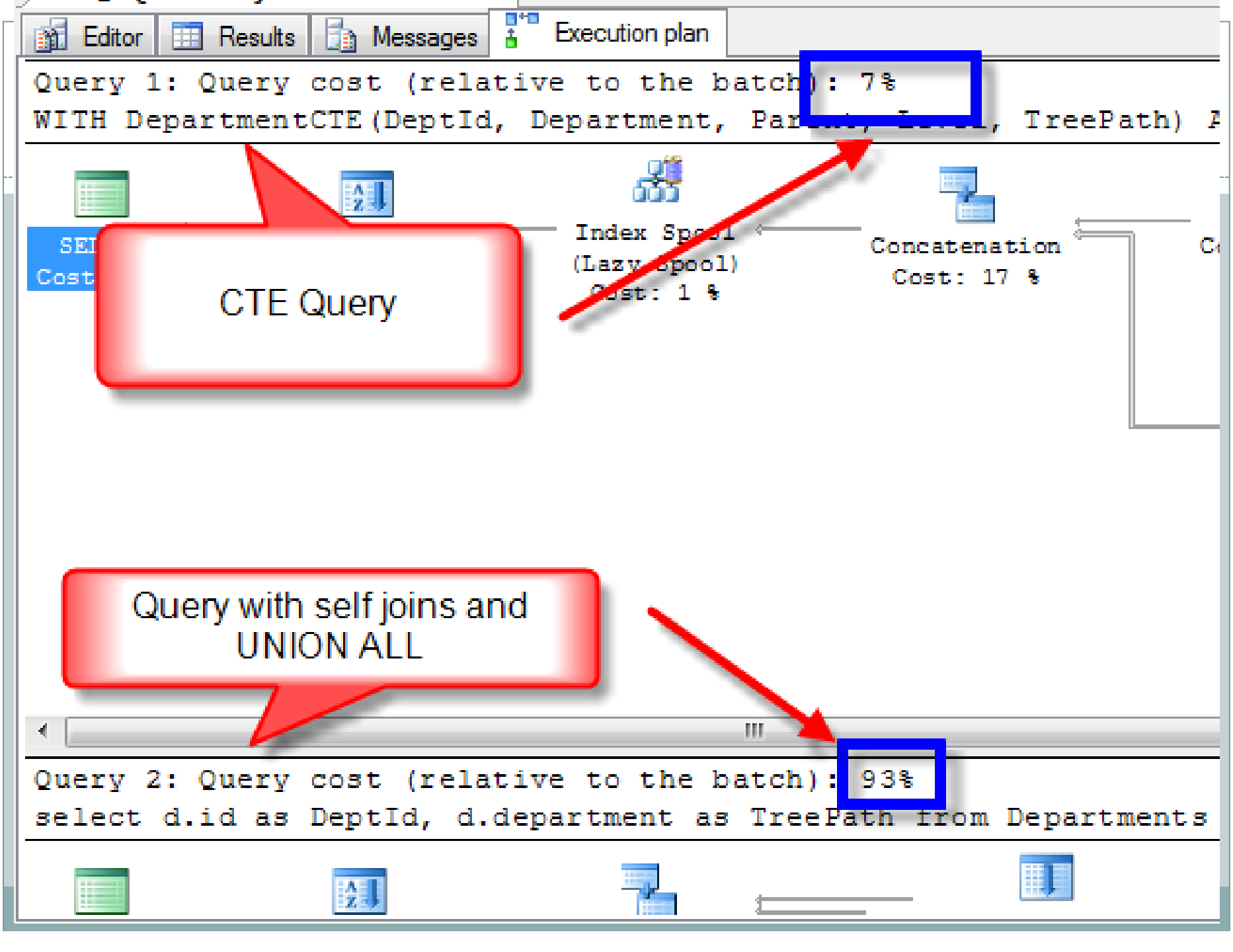
```
;WITH DepartmentCTE(DeptId, Department, Parent, Level, TreePath) AS
( SELECT id as DeptId, Department, parent, 0 as Level,
            cast(Department as varchar(1024)) as TreePath
   FROM Departments
  WHERE parent is NULL
  UNION ALL -- and now for the recursive part
 SELECT d.id as DeptId, d.Department,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


Editor
Results
Messages
Execution plan
Query 1: Query cost (relative to the batch): 7%
WITH DepartmentCTE(DeptId, Department, Par
Index Spool
(Lazy Spool)
Concatenation
Cost: 17 %
CTE Query
Query with self joins and
UNION ALL
Query 2: Query cost (relative to the batch): 93%
select d.id as DeptId, d.department as TreePath from Departments

5. 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 ....
```

Steps to add a Second CTE

1. Add a comma at the end of the first CTE, after the closing parentheses.
2. After the comma, on the next line, declare the name of the new CTE.
3. After the name of the new CTE add the optional columns declaration.
4. Add the AS keyword followed by opening and closing parentheses.
5. Inside of the parentheses add the new CTE query.
6. Call the CTE query from the outer SELECT statement.

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;
```

Nested CTE's

- Russian Dolls

- A Nested CTE query can only reference itself or CTE queries declared earlier in the query.

Nested CTE Example

```
;WITH cte0 AS
(select 1 as num)

, cte1 AS
(SELECT * FROM cte0)

, cte2 AS
(SELECT * FROM cte1)

SELECT *
  FROM cte2;
```

6. Other Common CTE Uses

- Data paging on a search result (Chapter 7 in the CTE Book)
- 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 DatesCTE as (
   SELECT cast('2011-01-01' as date) as CalendarDate
     UNION ALL
   SELECT dateadd(day , 1, CalendarDate) AS CalendarDate
     FROM DatesCTE
   WHERE dateadd (day, 1, CalendarDate) < '2012-01-01'
)
SELECT
   CalendarDate,
...
   CalendarYear=year(CalendarDate),
DayOfWeek=datepart(weekday, CalendarDate)
FROM DatesCTE
OPTION (MAXRECURSION 366);
```

Creating a replacement for a Numbers table

```
;WITH NumbersCTE (N) AS
( SELECT 1
   UNION ALL
   SELECT 1 + N
      FROM NumbersCTE
  WHERE N < 1000
)
 SELECT N
   FROM NumbersCTE
 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

7. Manipulating Data with a CTE

- Update
- Delete
- Insert

Update

- When it is run against the CTE the UPDATE changes the base tables inside of the CTE.
- Update works with a single base table CTE.
- Update does work with multiple base tables as long as only one base table is being changed.
- Update doesn’t work if there are no base tables.

Update Example – Single Base Table CTE

```
;WITH departmentsCTE(id, department, parent) AS
(
  SELECT id, department, parent
    FROM Departments
)
UPDATE DepartmentsCTE
   SET department = 'Bike Locks'
 WHERE id = 11;

SELECT *   FROM Departments;
```

Update Example – No Base Table CTE

```
;WITH NumbersCTE (N) AS
( SELECT 1
   UNION ALL
  SELECT 1 + N FROM NumbersCTE
   WHERE N < 1000
)
UPDATE NumbersCte
   SET N = N + 1
OPTION (MAXRECURSION 1000);
```

- **Throws an error**

Delete

- The CTE syntax does not allow for a DELETE statement to be used in any of the queries inside of the CTE
- DELETE statement can run in an outer query.
- The DELETE statement effects the records that were produced by the CTE
- Deleting from a CTE gets very interesting…
  - A DELETE from the outside query of a CTE will delete from the table inside of the CTE

Delete Example

```
WITH departmentsCTE(id, department, parent) AS
(
  SELECT id, department, parent
    FROM Departments
)
DELETE FROM departmentsCTE
 WHERE parent = 1;
```

Where Delete Doesn't Work with a CTE

- A CTE with multiple base tables doesn't support the delete syntax.

```
;WITH CustomerCTE AS
( SELECT c.*
    FROM Customer AS c
   INNER JOIN SalesInvoice AS si
           ON si.CustomerID = c.CustomerID
   WHERE c.LastName like 'Williams')
DELETE
 FROM CustomerCTE;
```

Editor | Messages

```
Msg 4405, Level 16, State 1, Line 1
View or function 'CustomerCTE' is not updatable because
the modification affects multiple base tables.
```

100 %

Insert

- The insert statement can be used to insert into a CTE when the CTE references a single base table

Insert - Demo

```
;WITH departmentsCTE(id, department, parent) AS
(
  SELECT id, department, parent
    FROM Departments
)
INSERT INTO DepartmentsCTE
          VALUES (99, 'xyz', 1);
```

9. 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: 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

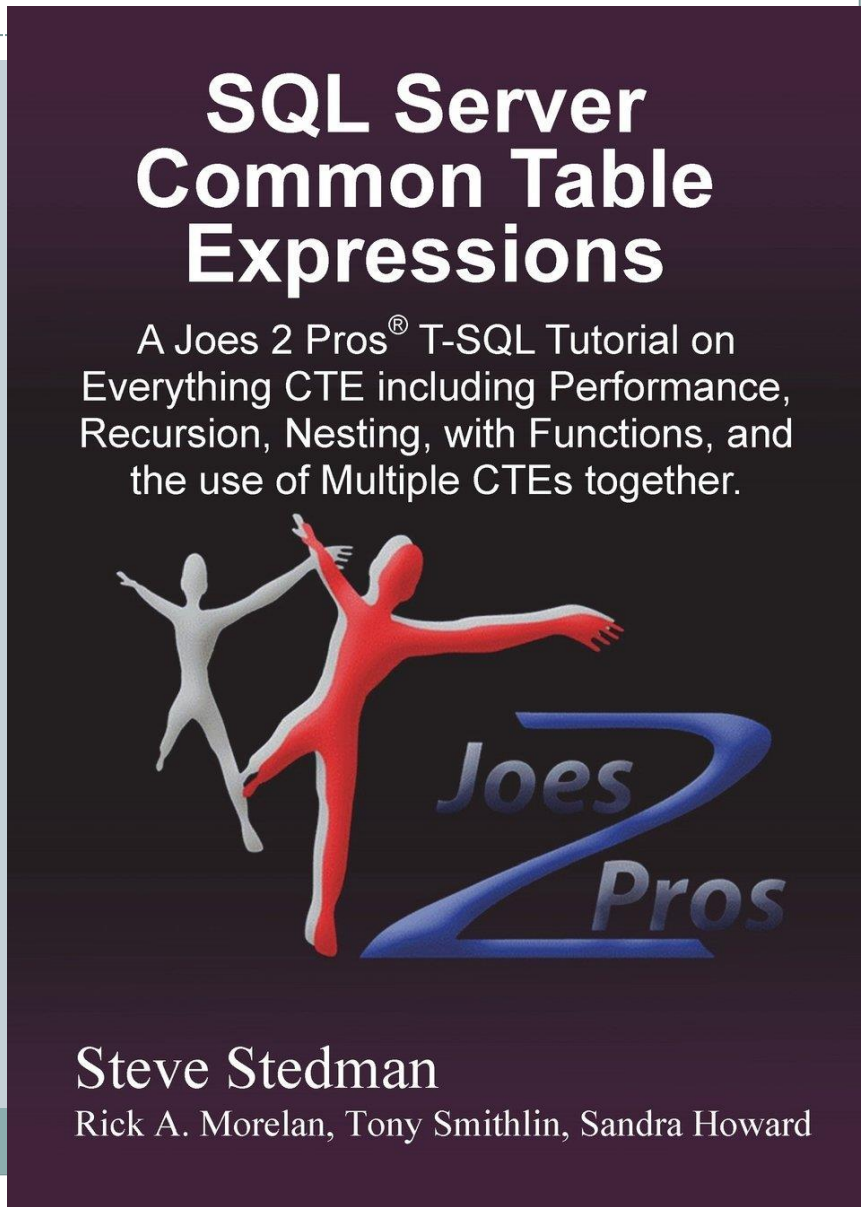
1. What is a Common Table Expression
2. Simple CTE
3. CTE instead of a Derived Table
4. Recursive CTE
5. Multiple CTEs in a Query
6. CTE Common Uses
7. Manipulating Data with a CTE
8. CTE for Math Geeks

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/

Common Table Expressions Book

- Published May 2013
- Available at Amazon.com and at Joes2Pros.com
- Printed and Kindle versions are both available now



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.....

12. CTE's and TempDB

CTE's are similar to Temp Tables or Table Variables in their use of TempDB?

True or False

FALSE Temp Tables and Table Variables both use TempDB, CTE's do not……

- See my blog posting for all the details on this one.
  - http://stevestedman.com/?p=2053
  - It is more than we have time to prove today.

13. Data Paging

An alternative to a CTE would be to use the ROW_NUMBER function in the WHERE clause to filter the results.

True or False?

FALSE ROW_NUMBER can be used to get the current row number in the result set, but it is a windowing function, and windowing functions are not allowed to be used in the WHERE clause.

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/

Common Table Expressions Book

- Published May 2013
- Available at Amazon.com and at Joes2Pros.com
- Kindle version available soon