Copyright © 2015 by The Learning House.

All rights reserved. No part of these materials may be reproduced, distributed, or transmitted in any form or by any means, including photocopying, recording, or other electronic or mechanical methods, without the prior written permission of The Learning House. For permission requests, write to The Learning House, addressed “Attention: Permissions Coordinator,” at the address below.

The Learning House

427 S. 4th Street #300

Louisville KY 40202

Common Table Expressions (CTEs)

.NET Cohort

Coding Bootcamp

SOFTWARE GUILD

Lesson Goals

- Learn different techniques to generate temporary objects for more complicated queries (derived and calculated fields)
- Learn the Common Table Expression syntax to improve readability of temporary objects

SOFTWARE GUILD

Why Do We Care?

When we query our database, sometimes we need to operate over a set of data that does not already exist in the system.

We already know we can use Views and Derived Tables, but they have some disadvantages.

SOFTWARE GUILD

Disadvantages

Views

Views are permanent objects at the system level, so creating views that are only used by a single script or stored procedure creates clutter.

Derived Tables

Derived tables, being wrapped in other queries, can be hard to read, especially when multiple derived tables are present in the same query.

SOFTWARE GUILD

Advantages of CTEs

CTEs are a more readable form of a derived table that can be declared once and referenced multiple times in the same query block.

CTEs can also be recursively defined, which grants easier access to recursive logic.

SOFTWARE GUILD

A Simple CTE

Try the following:

```
WITH ProductAndCategoryNamesOverTenDollars (ProductName, CategoryName, UnitPrice) AS
(
   SELECT
      p.ProductName,
      c.CategoryName,
      p.UnitPrice
   FROM Products p
      INNER JOIN Categories c ON
         c.CategoryID = p.CategoryID
   WHERE p.UnitPrice > 10.0
)

SELECT *
FROM ProductAndCategoryNamesOverTenDollars
ORDER BY CategoryName ASC, UnitPrice ASC, ProductName ASC
```

SOFTWARE GUILD

Break it Down

WITH starts a CTE

CTE must have a name

and a list of fields

```
WITH ProductAndCategoryNamesOverTenDollars (ProductName, CategoryName, UnitPrice) AS
(
    SELECT
        p.ProductName,
        c.CategoryName,
        p.UnitPrice
    FROM Products p
        INNER JOIN Categories c ON
            c.CategoryID = p.CategoryID
    WHERE p.UnitPrice > 10.0
)

SELECT *
FROM ProductAndCategoryNamesOverTenDollars
ORDER BY CategoryName ASC, UnitPrice ASC, ProductName ASC
```

The query to create the temporary results goes in paranthesis (just like a sproc)

We have to use it after declaring it though... it doesn't stick around

SOFTWARE GUILD

The list of fields can alias

```
WITH ProductAndCategoryNamesOverTenDollars (Banana, Apple, Peach) AS
(
   SELECT
      p.ProductName,
      c.CategoryName,
      p.UnitPrice
   FROM Products p
      INNER JOIN Categories c ON
         c.CategoryID = p.CategoryID
   WHERE p.UnitPrice > 10.0
)

SELECT *
FROM ProductAndCategoryNamesOverTenDollars
ORDER BY Banana ASC, Apple ASC, Peach ASC
```

SOFTWARE GUILD

It's optional if not aliasing

```
WITH ProductAndCategoryNamesOverTenDollars AS
(
   SELECT
      p.ProductName,
      c.CategoryName,
      p.UnitPrice
   FROM Products p
      INNER JOIN Categories c ON
         c.CategoryID = p.CategoryID
   WHERE p.UnitPrice > 10.0
)

SELECT *
FROM ProductAndCategoryNamesOverTenDollars
ORDER BY CategoryName ASC, UnitPrice ASC, ProductName ASC
```

SOFTWARE GUILD

What Should We Do With This Power?

What if we wanted to show a list of categories, how many products they have, and only the products that have a unit price over $10.00?

SOFTWARE GUILD

Multiple CTEs!

Just comma separate them.

New concept: we can put a subquery inside a select statement. Be careful with this though, because it will run *n* extra queries, where *n* is the number of rows in the table.

```
WITH CategoryAndNumberOfProducts AS
(
   SELECT
      CategoryID,
      CategoryName,
      (SELECT COUNT(1) FROM Products p
       WHERE p.CategoryID = c.CategoryID) as NumberOfProducts
   FROM Categories c
),
ProductsOverTenDollars AS
(
   SELECT
      ProductID,
      CategoryID,
      ProductName,
      UnitPrice
   FROM Products p
   WHERE UnitPrice > 10.0
)

SELECT c.CategoryName, c.NumberOfProducts,
      p.ProductName, p.UnitPrice
FROM ProductsOverTenDollars p
   INNER JOIN CategoryAndNumberOfProducts c ON
      p.CategoryID = c.CategoryID
ORDER BY NumberOfProducts DESC, CategoryName,
    UnitPrice DESC, ProductName
```

Recursion

Inside a CTE, we can use a UNION statement to merge the base case (top-level) and the recursive step (drilling down) in order to provide query data in hierarchical format.

This is a very advanced technique. If you don't get it, don't fret, because most people don't.

This is a query to show the hierarchy levels of management in Northwind.

```
WITH EmployeeHierarchy AS
(
    -- Base case
    SELECT
        EmployeeID,
        LastName,
        FirstName,
        ReportsTo,
        1 as HierarchyLevel
    FROM Employees
    WHERE ReportsTo IS NULL

    UNION ALL

    -- Recursive step
    SELECT
        e.EmployeeID,
        e.LastName,
        e.FirstName,
        e.ReportsTo,
        eh.HierarchyLevel + 1 AS HierarchyLevel
    FROM Employees e
        INNER JOIN EmployeeHierarchy eh ON
            e.ReportsTo = eh.EmployeeID
)

SELECT *
FROM EmployeeHierarchy
ORDER BY HierarchyLevel, LastName, FirstName
```

When Do I Use CTEs?

- For recursive queries
- For a complicated one-off where I don’t want to add a view to the system
- To enable grouping by a column that is derived from a sub-select
- When I need to reference a query result multiple times in the same statement

SOFTWARE GUILD