Steve Stedman
Freelance SQL Server Consultant
http://stevestedman.com

Calculating Factorials with a Recursive CTE

What is a Factorial:

The product of an integer and all the integers below it; e.g., factorial four (4!) is equal to 24.

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**

It is simple to do with a recursive function, and actually commonly used as a programming interview question to determine if you understand recursion. For instance, years ago, I was asked the following during an interview at Microsoft.

> **What is recursion? Calculate factorial numbers using a recursive function?**

Which is very easy to answer using C, which was the programming language of choice when I was asked it. But what if you were asked that on an intervie for a SQL Server DBA, or SQL Server Programmer job. Your first answer might be to use a recursive stored procedure, which would be a good answer, but what if you were asked the following, how would you answer it?

> **What is recursion? Using TSQL write a query to calculate factorials without creating any stored procedures or functions.**

Think, think, think.... You want the job, and that is the question that the interviewer is asking you. How do you solve it?

1. First, make sure you understand factorials. If you don't understand what a factorial is you won't get anywhere with this.
2. Then write the query, using a Recursive Common Table Expression.

Here is how I solved it.

First, get started with a simple **CTE** that just calculates the first factorial 1! as shown below. Pretty simple at this point. The result set is correct, the Factorial of 1 is just 1.

Next, I would add the recursive **CTE** query to continue with the factorial calculations beyond 1!, this time going up to 5! When working with recursive **CTE**'s you usually want to have an escape clause to avoid too much recursion. In this example the **CTE** exits at 5.

That's nice, but lets go further. Lets go for 20! Which you will see in the image below caused some problems as shown with an error of "Arithmetic overflow error converting expression to data type int."

Now to fix the overflow. It turns out that Factorial of 20 gets big pretty quick, and an integer only can hold a number up to just over 2 Billion. But a BIGINT can go much further, to just over 9 Quintillion (US). So to fix this we cast the factorial column to be a BIGINT rather than an INT.

INT

Range: -2^31 (-2,147,483,648) to 2^31-1 (2,147,483,647) Space: 4 Bytes

BIGINT

Range: -2^63 (-9,223,372,036,854,775,808) to 2^63-1 (9,223,372,036,854,775,807) Space: 8 Bytes

So here we go with an attempt to get to 20! , which works great.

But how far can we go with the BIGINT 9 Quintillion limit of BIGINT. As it turns out 21! exceeds the size of a BIGINT.

So what now? What if we want to calculate more than 20! using SQL Server? Cast it to a NUMERIC(38,0) which gives us 38 digits to work with as shown here.

Now what???

I haven't been able to find a solution to do math and store the results larger than a NUMERIC(38,0).

Here is the final query.

Although you could create a stored procedure in sql to calculated factorial, the CTE is a much easier way of doing it.

I hope you have found this useful.