**[Aggregation with ROLLUP operator is not fully supported](http://www.sqlservercentral.com/blogs/sherry-lis-bi-corner/2014/12/31/sql-56aggregation-with-rollup-operator-is-not-fully-supported/)**

Posted on 31 December 2014

FROM: <http://www.sqlservercentral.com/blogs/sherry-lis-bi-corner/2014/12/31/sql-56aggregation-with-rollup-operator-is-not-fully-supported/>

ROLLUP, CUBE, and GROUPING SETS Operators have been around since SQL Server 2005. These operators are extensions of the GROUP BY clause.

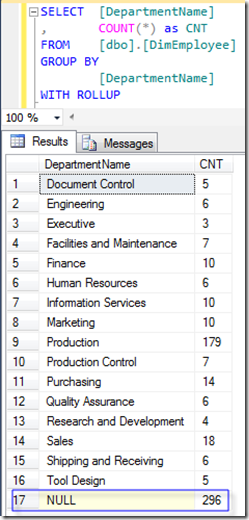
The GROUPING SETS operator might be introduced later in SQL Server 2008.

I only have experience with the ROLLUP operator. My test of GROUPING SET also suggested that it is not fully supported either.

**The ROLLUP operator is useful in generating reports that contain subtotals and totals**

The ROLLUP operator is useful in generating reports that contain subtotals and totals in the same dataset.

This simple GROUP BY query with a ROLLUP operator generates a grand total rows at the end.

[](https://bisherryli.files.wordpress.com/2014/12/pic11.png)

**Dynamic SQL and aggregates with the ROLLUP are commonly used in web reporting project**

You might be wondering where you can actually use this feature. I never used it until I was involved in a web reporting project. To centralize all the data manipulation logic in the database layer, the decision was to write a very flexible stored procedure that meets the following three requirements:

* All metrics calculations must be table-driven.
* Only one stored procedure should be used.
* The stored procedure should return subtotals and grand totals in the same result set.

The first two requirements forced us to generate dynamic SQLs instead of static SQL queries. The last requirement forced us to resort to the **ROLLUP** operator with aggregations.

This all worked well until we had a new requirement for an additional metric, which needs to calculate:

* A percentage of a COUNT for group over a subtotal.

**SUM(COUNT(\*)) OVER() can generate an aggregate count**

My first reaction is to simply add a TOTAL\_CNT to the query using **SUM(COUNT(\*)) OVER()**. Then the PCT\_OF\_TOTAL will be just the CNT over the TOTAL\_CNT.

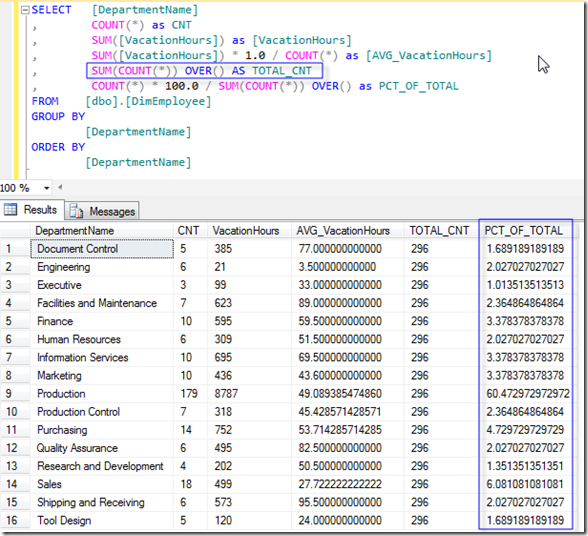
My thought was right on the mark, until I noticed that ROLLUP operator altered two calculations, TOTAL\_CNT and PCT\_OF\_TOTAL.

To try to figure out what exactly went wrong, I tested a simple aggregation query.

Here I added a few things to the first query.

* VacationHours and AVG\_VacationHours: I wanted to make sure that a simple averaging would work always.
* TOTAL\_CNT: I used a **SUM(COUNT(\*)) OVER()** to get the total count over all the DepartmentName.
* PCT\_OF\_TOTAL: this is just the CNT over the TOTAL\_CNT.

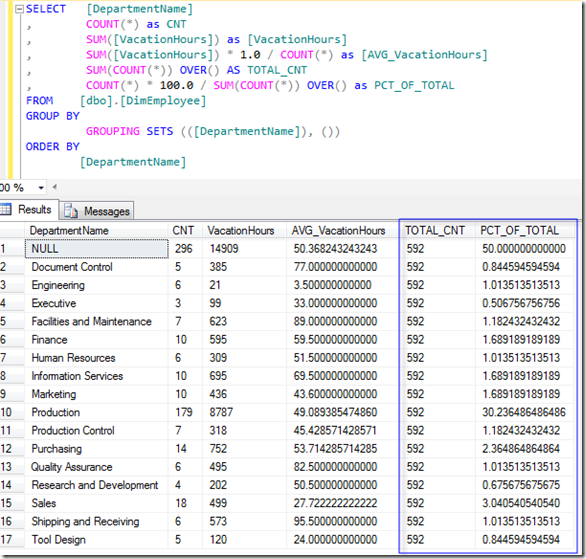
This query has no ROLLUP or GROUPING SETS. All numbers look good including the TOTAL\_CNT and PCT\_OF\_TOTAL

**[](https://bisherryli.files.wordpress.com/2014/12/pic22.png)**

**GROUPING SETS and ROLLUP messed up the SUM(COUNT(\*)) OVER()**

Next, I added the GROUPING SETS operator.

with the GROUPING SETS operator, a grand total row was added to the result set. The count and the average are all correct, but the TOTAL\_CNT and the PCT\_OF\_TOTAL has been messed up.

[[](https://bisherryli.files.wordpress.com/2014/12/pic31.png)](https://bisherryli.files.wordpress.com/2014/12/pic31.png)

Looking closely, you can see that the TOTAL\_CNT was doubled. You might be tempted to try to correct it by dividing by 2 to get it right. But adding more hierarchies to the aggregation generates even more confusing results.

**ROLLUP operator does not support the SUM() OVER()**

I was quite disappointed that the ROLLUP operator does not support the SUM() OVER(). This basically means that I’d need to write a lot more inefficient code to achieve this. With the dynamic queries in the midst, the coding will not be as efficient and elegant.

The closest example I could find is from this SQL Magazine article from 2012 by D. Yitzhak, in which only examples of ranking functions with the ROLLUP, CUBE, and GROUPING SETS operators were given.

[Take Advantage of SQL Server 2012’s Window Functionality](http://sqlmag.com/sql-server-2012/take-advantage-sql-server-2012s-window-functionality)

**GROUPING SETS is supposed to supersede ROLLUP**

In the last query, I used the GROUPING SETS instead of the ROLLUP operator. This is because GROUPING SETS is a newer additions to SQL Server, and is supposed to supersede ROLLUP. According to many blogs, the WITH ROLLUP and WITH CUBE features will be removed in future versions of SQL Server. But I have not seen any official statement from Microsoft.

For more on GROUPING SETS in SQL Server 2008 R2, check out these TechNet article:

* [Using GROUP BY with ROLLUP, CUBE, and GROUPING SETS](http://technet.microsoft.com/en-us/library/bb522495(v=sql.105).aspx)
* [GROUPING SETS Equivalents](http://msdn.microsoft.com/en-us/library/bb510427(v=sql.105).aspx)

**OVER Clause**

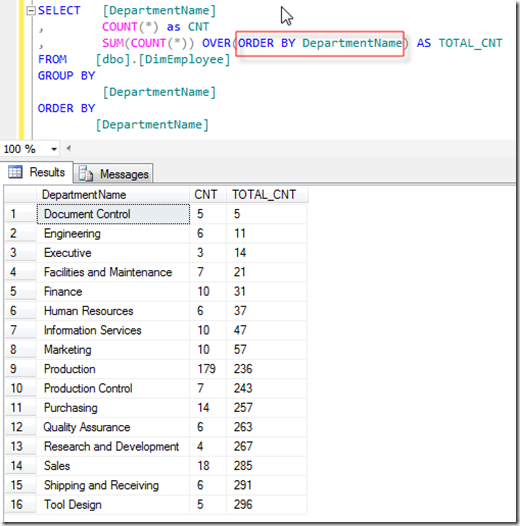
OVER Clause has been around in SQL Server for a long time. If you have used any of the following types of functions, you are already familiar with it:

* Ranking functions
* Aggregate functions
* Analytic functions
* NEXT VALUE FOR function

The OVER clause defines a window within a query result set. A window function then computes a value for each row in the window. You can use the OVER clause with functions to compute aggregated values such as

* moving averages
* cumulative aggregates
* running totals
* top N per group results

In the following example, adding an ORDER BY clause to the **SUM(COUNT(\*)) OVER()** generates running totals.

[](https://bisherryli.files.wordpress.com/2014/12/pic4.png)

For more information, check out this MSDN SQL Server 2014 article:

[OVER Clause (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms189461.aspx)