aGGREGATES\_OF\_aggregates

T-SQL Tuesday #123: LIFE HACKING BY OPENING WINDOWS

[February 12, 2020](https://jimbabwe.co.za/2020/02/12/t-sql-tuesday-123-life-hacking-by-opening-windows/) [jimbabwe](https://jimbabwe.co.za/author/jimbabwe/) [#sqlcommunity](https://jimbabwe.co.za/category/sqlcommunity/), [Consulting](https://jimbabwe.co.za/category/consulting/), [T-SQL Tuesday](https://jimbabwe.co.za/category/t-sql-tuesday/) [One comment](https://jimbabwe.co.za/2020/02/12/t-sql-tuesday-123-life-hacking-by-opening-windows/#comments)

Reading Time: 5 minutes

FROM: <https://jimbabwe.co.za/2020/02/12/t-sql-tuesday-123-life-hacking-by-opening-windows/>

[](https://blog.waynesheffield.com/wayne/archive/2018/08/t-sql-tuesday-105-invitation-brick-wall/)

January is a month that feels like it lasts forever but somehow it’s already the 2nd Tuesday of February, and another chance to contribute to the #sqlcommunity, and to write about the things that simplify our lives. Thanks, as always, to Adam Machanic [(t|b)](https://blog.waynesheffield.com/wayne/archive/2018/08/t-sql-tuesday-105-invitation-brick-wall/) for starting T-SQL Tuesday, and to Steve Jones ([t](https://twitter.com/way0utwest)|[b](https://voiceofthedba.com/)), for his contribution (and [website](http://www.tsqltuesday.com/)).

I had the great pleasure of meeting Jess Pomfret ([t](https://twitter.com/jpomfret)|[b](https://jesspomfret.com/)|[l](https://www.linkedin.com/in/jpomfret/)) at PASS Summit 2019, and although she is embarrassing me in the SQL Family Fantasy League (English Premier League), I’m still taking up [her challenge](https://jesspomfret.com/t-sql-tuesday-123/) of writing about our life hacks, or the little things that make our lives simpler.

*This could be anything from a keyboard shortcut in SSMS that runs ‘sp\_whoisactive’, to a technique you use to get and stay organised.*

From the T-SQL Tuesday invitation

I have lots of little life hacks, but the one that I want to mention today is **Aggregates of Aggregates using Windowing Functions**. When I present my session on Window Functions, people are frequently wowed when they see this technique, but it’s something I use all the time, and I love showing people how it works.

**Windowing Functions in a minute**

Talking about Windowing Functions at PASS Summit 2018. On of my proudest achievements. Session video is available [here](https://www.pass.org/Learn/Recordings/Listing.aspx) (paid content).

Windowing functions are an underused feature in SQL Server, with myriad uses. The most common problems we solve are running totals, seeing group totals on the same line as individual lines from the group (allowing calculations like subtotal %). I don’t know if there are better solutions, but before Windowing Functions, I used to solve these kind of problems with self joins, or nested queries. The performance of Windowing Functions is significantly better than this approach, and that alone has made my life considerably better.

The more I learn about Windowing Functions, the more often I see use cases where they are useful.

I love talking about, and teaching people to use, Windowing functions to make their lives better, so I’m quite excited to be able to use them as the topic for today’s post.

**Groups of Groups Problems**

Assume I have table of all the **individual visits** to my website with the country of the visitor as one of the columns. I want to count all the visits per country, and on the same row, add the highest number of visits from a single country, and also the total number of visits across all countries.

That all sounds pretty complicated because our query is basically querying at 3 different grouping sets, (per Country, over the group of all the countries, and over the entire dataset), and before Windowing functions, the way I would have solved this problem may have looked like this (there are lots of different solutions, but I think this is easiest to follow).

SELECT  [Country],

        COUNT(1) as [CountryVisitors],

        (SELECT TOP 1 COUNT(1) FROM [Visitors] GROUP BY [Country] ORDER BY count(1) DESC) as [MaxCountryVisitors],

        (SELECT COUNT(1) FROM [Visitors]) as [TotalVisitors]

FROM [Visitors]

GROUP BY [Country]

Now that query is probably ok on a smallish dataset, and there are other ways to do this (cross joins to subqueries etc.) but in my experience Windowing Functions is the cleanest way to do this.

**The two step Windowing Functions solution.**

So this is a solution using Windowing Functions and a Common Table Expression (CTE).

WITH [cte\_CountryVisits]

AS

(

select [Country],

COUNT(1) as CountryVisits

from [Visitors]

group by [Country]

)

SELECT [Country],

CountryVisits,

MAX(CountryVisits) OVER () as [MaxCountryVisitors],

SUM(CountryVisits) OVER () as [TotalVisitors]

FROM [cte\_CountryVisits]

**WHOA, what is going on there?!**

I know what you’re thinking. James, you’re crazy, it’s *not possible to use AGGREGATE FUNCTIONS without a GROUP BY*. I used to think that too, but then someone pointed out that if **ROW\_NUMBER()**, **RANK()** and **DENSE\_RANK()** can be aware of the entire dataset, why shouldn’t other functions be able to do that.

Well, it turns out, **they can**! And that’s exactly what a Windowing Function is, a function that opens a window to all, or a part of, the results set of the query and performs an operation over that result set.

*You see, if you sprinkle a little bit of magic* ***OVER()*** *the statement, you can do things that seem impossible.*

James McGillivray. Proponent of bad puns.

The “magic ingredient” in Windowing Functions is the [OVER clause](https://docs.microsoft.com/en-us/sql/t-sql/queries/select-over-clause-transact-sql?view=sql-server-ver15), which indicates to the query that you wish to open the window to a greater part of the dataset. There are ways to control exactly what the extended dataset is, but we won’t be looking at those today.

So back to the query. We have two statements with OVER clauses in them.

MAX(CountryVisits) OVER ()

SUM(CountryVisits) OVER ()

Now both of these statements are doing the same thing. Using the OVER clause without any addition clauses (inside the brackets), tells the query to look at all rows of the results set, and perform the aggregation OVER the entire set. So the SUM statement will sum up all the CountryVisits values in the results set, and the MAX will return the largest CountryVisits value across the entire set. Similarly, a ROW\_NUMBER() function and an empty OVER() clause would have looked over the entire set and returned the order of the specific row.

Once you’re happy with basic Windowing Functions, this pattern is extremely simple and comfortable to use. But it would be nice to do this in a single query…

**Aggregates of Aggregates**

The reason we used the CTE in the last query, was to first perform the grouping and aggregation (COUNT(1)) on the query. But now that we know you can perform aggregation over a larger dataset, why couldn’t we have the [TotalVisitors] like this?

SELECT [Country],

COUNT(1) as [CountryVisitors],

COUNT(1) OVER () as [TotalVisitors]

FROM [Visitors]

GROUP BY [Country]

Well the answer is pretty simple. The COUNT(1) function will now be performed OVER() the results set, and will therefore return the total number of rows in the results set, i.e. the number of distinct Countries in our dataset. That’s useful, but it’s not what we’re looking for.

And we should have been able to see that it wouldn’t get us the right answer, because if we’d tried to think of a syntax for the MAX, we don’t really have a column to put inside the MAX function. Which is why we have to use an aggregate of an aggregate.

select [Country],

COUNT(1) as [CountryVisitors],

MAX(COUNT(1)) OVER () as [MaxCountryVisitors],

SUM(COUNT(1)) OVER () as [TotalVisitors]

from [Visitors]

group by [Country]

I’m pretty sure I can already hear your eyes rolling, when you see this query. Only a BI Dev could think that you could write MAX(COUNT(1)) and expect it to work. Well, I’m happy to report that I was BI Dev, and I absolutely expect that query to work.

Why?! Because once we use the OVER clause, we can access an aggregation from the original GROUP BY as if it were a column in our returned dataset. Because it very well could be (just like in the CTE). So it’s absolutely possible to write an aggregate of an aggregate. It’s even possible to partition the results, to create running totals, and generally do everything you would otherwise achieve using Windowing Functions.

This seems a little strange, and when I started using this, I frequently made mistakes with my grouping, and especially my partitioning and ordering. But as you use it more and more, you’ll find more and more situations in which it helps.

*Windowing Function ALL THE THINGS!*

Olivia Allard ([t](https://twitter.com/dboliviaa/)|[l](https://www.linkedin.com/in/olivia-allard-33a928109/))

**TL;DR**

Although counter-intuitive, it is possible to write a NESTED AGGREGATION in a T-SQL query by using a Windowing Function. This technique is a very useful way to add information to GROUP BY queries from the rest of the result set without having intermediate tables or using CTEs.