Using CTEs Instead of Derived Tables

Steve dives into using CTEs instead of derived tables.

1. What the benefit of CTEs
2. Do CTEs over derived tables benefit performance?
3. And a quick demo explanation

Transcription of video:

Now let's take a look at some more complex examples of how to use a CTE instead of a derived table. Now use a CTE instead of a derived table, it's great to simplify the query and it allows for a lot cleaner code. And it does not change the performance of things at all. If you have a speedy query that's being run as a drive table, it's going to be speedy, when we run it as a CTE. If you have a really slow performance issue query, that's run as a drive table, it's going to run just as slow as a CTE, there's no magic happening there that's going to either improve or decrease the performance in any way.

When we get to larger queries where you've got lots and lots of derived tables or sub queries, the CTE can add a lot more value there in making it easier to read. And we'll take a look at some examples of that. It can also eliminate accidents by duplicating a drive table queries in the SQL code. I've seen queries where you might have the same sub query or drive table duplicated two or three times. And then somebody comes back later and changes the code and forgets to change one of them. And things don't work quite right, by wrapping that as a CTE. Kind of like if you wrapped it with a view, it's only in one place, and you don't have the issue of having to duplicate code that needs to be patched.

So here's an example of a derived table without a CTE, where the parts shown in red are two drive tables that are aliased as q1 and q2. And they're being used inside of a joint statement, selecting

from one doing an inner join to the other. Just a pretty simple query there. But to convert that to a CTE, the way that we would do it, find the first occurrence of the derived table, create a name for it and add CTE to the name.

Now, this is one of those things that there's some debate out there over whether you should put CTE in the name of your your CTE results or not. And I'm not a big fan of like naming views view underscore or procedures proc underscore, but I think that's silly. But the difference is, those are things that are kept around permanently, where a CTE, oftentimes, I'm going to tack the word CTE on the end of it, or the letter CT on the end of it, so that I don't have a conflict or that I ended up stomping on a table name somewhere. And I copy the derived table definition, that's just what's inside of the sub query and including the parentheses and leave the new names as the placeholder. I then paste the query copied earlier above the select statement. And then at the top of the query at the CTE declaration, using the same name from step one, and then find all the occurrences of the same derived table query and replace them with the CTE name. And then clean up the formatting, to make it look nice and test the query. And then you end up with something like this, where the part that's shown in red is only written once but the depart the part in blue that shown down below is referenced multiple times accessing the same data.

Let's take a look at a demo of how to do similar thing to that. CTE instead of a drive table, so here's an example of the derived table where we have a sub query here that ends up being the derived table and another sub query here, that's been INNER JOIN to they happen to be the exact same query. And when we run it, we're getting a list of all the top level departments joined to all their sub departments.

So let's take a look at how we convert that into a CTE? Get these both, so we can see them here. So we start by taking that inner that sub query that drive table like this, and we paste it in above the query with the parentheses just like it's done here. We then everywhere that that was referenced in the table, we give it a name here, I'm going to call it department CTE department CTE. And then we add the CTE declaration at the top with departments CTE with these columns as and then we highlight it and we run the table, run the query, and we're going to run both of them here together. And when we run both of them, you can see that we get the same result sets between the CTE and the derived table version of things.

Okay, so that's not exactly rocket science at this point. And it doesn't really make a huge difference

there. But we're using a very simple example here. Where it gets interesting is when those queries are much more complicated than what we're looking at here.

A few notes to think about here regarding a CTEs instead of a derived tables, kind of summarizing a few things. Most derived tables can be very easily converted to a CTE just like we did in that example. You can seriously reduce the amount of copy and paste errors by using a CTE, if you're in the case of having multiple references have that, in that you only have the code referenced in one place rather than two copies of the same sub query. Using the CTE does not change the performance at all over a similar query written with derived tables, they perform exactly the same. For a CTE that is referenced multiple times. That CTE query is not reused. It is executed multiple times. So just like in the derived table example, where if you had two sub queries, they each got run once. In the CTE example, two references in the CTE means that the same CTE is going to get run multiple times.