(February 13th, 2017)

If you know someone who would benefit from being an Insider, feel free to forward this PDF to them so they can sign up here.

Quick Tips for our Insider friends!

Hey Insiders,

This newsletter is coming to you from Redmond, where "Common-Sense Boy" (me) was attacked and beaten by bamboo last week. We had a major snowfall (for Western WA) and I was shaking snow off some bamboo when I managed to stab myself in the eye with a stick, resulting in a corneal abrasion. Wow – does that hurt! I was on Percocet the rest of the week and you can read the various episodes of "Percocet Paul" on my Facebook page. Although apparently entertaining, Kimberly's especially glad that I'm done with pain killers!

The latest Pluralsight course we've published is Erin's *SQL Server: Introduction to Query Store*, which you can read about here.

We're looking forward to seeing a bunch of you this year in our classes in Chicago in April/May, our classes in Bellevue in July/August, and our Spring SQLintersection conference in Orlando in May. See here for the class schedule and here for SQLintersection details. Note that IEPTO1 in Chicago is already sold out and IE0 only has three seats remaining. I'm particularly pleased that we have a **three-day class on using PowerShell** to administer SQL Server, taught by industry-expert, MCM, and MVP, Ben Miller.

Even though we're not teaching any Immersion Events in Europe this year, Kimberly and I will both be **presenting at SQLSaturday #620 in Dublin in June**.

And even if you can't join us in person, we're still taking requests for remote sessions for this year. We have 50 scheduled so far; if you'd like one of us to present for your user group, check out my blog post here.

The latest book I've finished is Alison Weir's *The Life of Elizabeth I*. I actually listened to most of it during the week using Amazon's Audible service, where if you sign up you get two free books and can cancel at any time. I might keep it on for use on long flights and when doing things like soldering. Anyway, the book is excellent, as are all of Weir's Tudor-era histories. It covers Elizabeth's life and reign from when she becomes queen in 1558 to her death in 1603, and provided a good balance to Weir's *Mary, Queen of Scots, and the Murder of Lord Darnley*, which I read back in 2007. It's very readable and provides a wealth of detail without becoming dry and dull. Highly recommended!

Note: you can get all the prior Insider newsletters here.

The Curious Case of…

This section of the newsletter explains problems we've found on client systems; they might be something you're experiencing too.

I had another random email last week via my waits library from someone investigating the *ACCESS_METHODS_DATASET_PARENT* latch on their system.

This latch wait occurs when threads in a parallel scan need to request the next range to scan and access a coordinating object to do so. In this case, the person was also seeing *CXPACKET* waits (perfectly normal and expected to see with that latch) and was perplexed because their instance 'max degree of parallelism' configuration option was set to 1.

They wanted to know under what circumstances a query could still result in these two symptoms when parallelism was turned off.

The answer is that instance MAXDOP can be overridden by anyone with any privilege level, using the query syntax like *OPTION (MAXDOP 8)*.

There are two ways to stop *anyone* from running in parallel:

- Set the instance 'cost threshold for parallelism' value to be as high as it can go, so no queries will ever qualify as too expensive for serial execution
- Run all queries with a Resource Governor workload group that has its *MAX_DOP* set to 1 (as this cannot be overridden)

Bottom line: if you're experiencing symptoms of parallelism when you don't expect it, check for code that's overriding the instance setting. I'll do a blog post this week that shows a way to do that using Extended Events.

Paul's Ponderings

(As I was off work all last week, I didn't get a chance to write an editorial for the newsletter, and I'm still trying not to stare at a screen over the weekend. I was going to write about tempdb contention and temp tables, so instead I'm re-running an editorial from 2011 on the same subject with comments. Enjoy!)

Over the last year we've helped a few clients track down tempdb space hogs with automated monitoring built around the *sys.dm_db_task_space_usage* DMV to identify procedures and ad hoc code that are misusing temp tables and generating query plans that cause memory spills to tempdb from, for example, large sorts and joins where table indexing is incorrect.

(2017: This is a perennial problem, and if anything it's even more prevalent nowadays and causes tempdb allocation bitmap contention as more processors are vying for access to the allocation bitmaps in memory for temp table page allocation changes.)

In this newsletter I'd like to describe some of the misuses of temp tables that I've seen while investigating some of the tempdb space hogs. Don't get me wrong though – temp tables are great – when they're used efficiently.

There are three main problems we see:

- Over-population of temp tables
- Incorrect indexing (or none where there should be one/some) on temp tables
- Using a temp table where none is required

The first problem involves creating a temp table using a *SELECT ... INTO #temptable* construct and pulling in far more data into the temp table than is necessary.

The most common thing we see is pulling lots of user table columns into the temp table, where some of the columns are not used ever again in subsequent code. This is a HUGE waste of I/O and CPU resources (extracting the columns from the user table in the first place) and a horrible waste of tempdb space (storing the columns in the temp table). I've seen code pulling in large *varchar* columns (without reason) and with million-plus row datasets…

(2017: This is mitigated somewhat in later versions as bulk operations in tempdb won't actually hit the disk (as usually happens from 'eager writing' during bulk operations) so there's less of an I/O load.)

The other facet of over-population of temp tables is pulling in too many rows. For instance, if your code is interested in what happened over the last 12 months, you don't need to pull in all the data from the last ten years. Not only will it be bloating the temp table, it will also drastically slow down the query operations.

(2017: One of the first clients I worked for, back in 2010, had a process that was calculating school grades for the previous month, but pulling in all grade and class data for the previous *15* years into the temp tables… unsurprisingly, they were running out of tempdb space!)

The key to better performance is making sure your selection/projection is as focused as possible. To limit your selection, use an effective *WHERE* clause. To limit your projection, list only the necessary columns in your select list.

The second problem involves either creating indexes before populating the table (so that no statistics are generated) or creating a bunch of inappropriate indexes that are not used. The most common example we see is creating a single-column nonclustered index for each temp table column that is subsequently used in a multi-table join. Those are just taking up space for no use whatsoever. Temp tables *DO* need indexes (preferably after load) but as with any form of query tuning – *only* the RIGHT indexes. Consider creating permanent tables that mimic what's going on in your temporary objects and then using DTA to see if it has recommendations. While DTA's not perfect, it's often WAY better than guessing.

(2017: Another problem can be creating temp tables without any indexes at all; often you're missing a wonderful opportunity – if done correctly!)

The final problem is when a temp table is used when it is not needed. The SQL Server query optimizer is a fabulous beast and is very good at figuring out the most efficient way to execute most queries. If you choose to take some of the query operation and pre-calculate it into a temp table, sometimes you're causing more harm than good. Any time you populate a temp table you're forcing SQL Server to materialize the complete set of results of whatever query you ran. This can really limit SQL Server's ability to produce a pipeline of data flowing efficiently through a query plan and making use of parallelism and collapsing data flows when possible.

While it's true that you might be able to do better than the optimizer sometimes, don't expect that it's the case all the time. Don't just go straight to using temp tables, give the optimizer a chance – and, make sure to retest your code/expectations around service packs and hot fixes as these may have eliminated the need for temp tables as well.

In one recent example I struggled to figure out why a temp table was even being used. It contributed up to 50GB of tempdb space for a query that was run many times per day. Taking the temp table creation code and embedding it as a derived table in the main query completely removed the tempdb usage and took the query from 17 minutes down to a few seconds. Derived tables are a great form of query hint – and always something to try before moving to temp tables.

Summary: if possible, limit the amount of data that temp tables hold, create appropriate indexes ***after*** the temp table is populated (so that the statistics are correct/up-to-date), and make sure that using a temp table is actually more efficient than just letting SQL Server process the whole query in one go.

(2017: Another thing to consider from SQL Server 2014 onwards is to try moving the temp table to be an In-Memory OLTP table…)

Call to action: take a look at your temp table usage. You may be surprised to find a lot of tempdb space and CPU resources being consumed by inappropriate temp table usage and

incorrect temp table indexing. Or if you don't have time and you're suffering from horrible tempdb problems, give us a call to help you. (2017: no change here ☺)

Video Demo

This video is taken from Erin's latest Pluralsight course, *SQL Server: Introduction to Query Store*, and shows how you can use the Query Store to test query plan changes when moving to the new cardinality estimator.

The video is about 5.5 minutes long and you can get it in MOV format here, with the demo code here.

Enjoy!

SQLskills Offerings

The classes for both Spring and Summer 2017 are available for registration!

To help your boss understand the importance of focused, technical training, we've also added a few items to help you justify spending your training dollars with us:

- Letter to your boss explaining why SQLskills training is worthwhile
- Community blog posts about our classes
- Immersion Event FAQ

Upcoming Immersion Events

Chicago, IL, April/May 2017

- **IE0**: Immersion Event for Junior/Accidental DBAs
  - April 24-26 ****3 SEATS REMAINING****
- **IEPTO1**: Immersion Event on Performance Tuning and Optimization – Part 1
  - April 24-28 ****SOLD OUT****
- **IESSIS1**: Immersion Event on Learning SQL Server Integration Services
  - April 24-28
- **IEBI**: Immersion Event on Business Intelligence
  - May 1-5
- **IEPTO2**: Immersion Event on Performance Tuning and Optimization – Part 2
  - May 1-5
- **IEPS**: Immersion Event on Powershell
  - May 8-10 ****NEW** class**
- **IEPDS**: Immersion Event on Practical Data Science
  - May 8-12

- **IEHADR**: Immersion Event on High Availability and Disaster Recovery
    - May 8-12

Bellevue, WA, July/August 2017

- **IEPTO1**: Immersion Event on Performance Tuning and Optimization – Part 1
    - July 31-August 4
- **IEPTO2**: Immersion Event on Performance Tuning and Optimization – Part 2
    - August 7-11

Click here for the main Immersion Event Calendar page that allows you to drill through to each class for more details and registration links.

Summary

We hope you've enjoyed this issue - we really enjoy putting these together.

If there is anything else you're interested in, we'd love to hear from you - drop us a line.

Thanks,
Paul and Kimberly

Paul@SQLskills.com and Kimberly@SQLskills.com