**An Introduction to Query Memory**

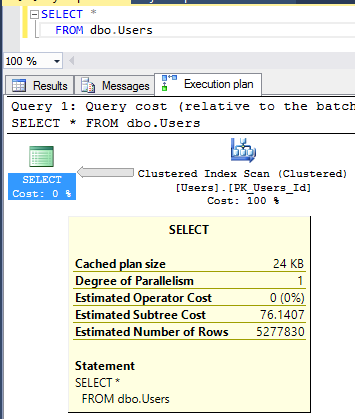
August 5, 2016

[Brent Ozar](https://www.brentozar.com/archive/author/brento/)[Execution Plans](https://www.brentozar.com/archive/category/development/execution-plans/)

FROM: <https://www.brentozar.com/archive/2016/08/memory-grants-week-introduction-query-memory/?utm_source=Brent+Ozar+Unlimited%C2%AE+List&utm_campaign=9831dcb39e-RSS_EMAIL_CAMPAIGN&utm_medium=email&utm_term=0_8e3e861dd9-9831dcb39e-296071565&goal=0_8e3e861dd9-9831dcb39e-296071565&mc_cid=9831dcb39e&mc_eid=b137e91a2b>

Microsoft has been quietly making some amazing improvements for performance tuners in SQL Server 2012, 2014, and 2016. This week, we’re going to introduce you to just how awesome they are. (They being the improvements, not Microsoft. You already knew they were awesome.)

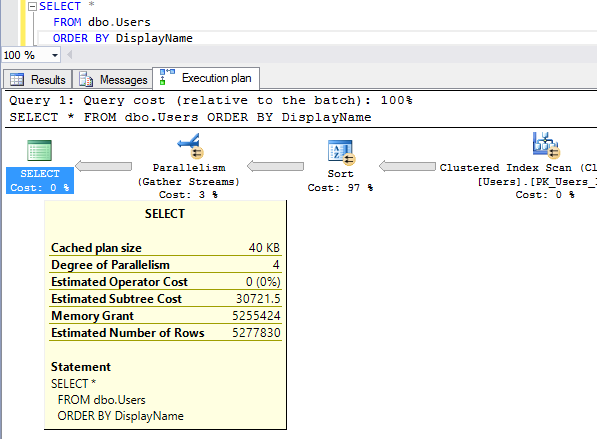
Using the [freely available StackOverflow database](https://www.brentozar.com/archive/2015/10/how-to-download-the-stack-overflow-database-via-bittorrent/), let’s start with a simple query – SELECT \* FROM Users:



Simple query with no ORDER BY

When I hover my mouse over the SELECT in the execution plan, I get a little popup with things like estimated subtree cost, estimated number of rows, and more.

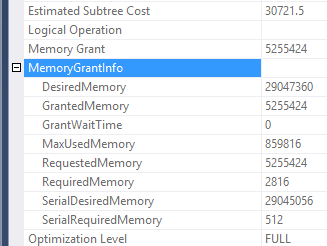
Now let’s add an ORDER BY:



Execution plan with a memory grant

When I hover over the SELECT, there’s a new line in town: Memory Grant. SQL Server needs RAM to sort the list of users before delivering them back to us. It didn’t need memory for the prior query because it was just dumping the data out as fast as it could read it.

You can see more details about the memory grant if you right-click on the SELECT and click Properties:



Memory grant details in the Properties pane

When SQL Server builds your query’s execution plan, it has to guess:

* **Desired Memory** – how much this plan ***wants*** in a perfect scenario, in kilobytes. Cached as part of the cached execution plan, and will remain consistent for all executions of this plan.
* **Granted Memory** – how much ***this*** execution of the query actually got. This can vary between execution to execution depending on the server’s available workspace memory at the time. Yes, dear reader, this is one of the reasons why sometimes your query is slow, and sometimes it’s fast. This value is not cached as part of the plan.
* **Grant Wait Time** – how long ***this*** execution of the query had to wait in order to get its memory. This relates to the [Perfmon counter Memory Grants Pending](https://www.brentozar.com/archive/2006/12/dba-101-using-perfmon-for-sql-performance-tuning/). Not cached, since it varies from execution to execution.
* **Max Used Memory** – how much ***this*** execution used, and also not cached obviously. This is where things start to get interesting.

**SQL Server’s query grant memory estimates are really important.**

**If it estimates too little memory**, then as your query starts to bring back more and more data and it runs out, then you’ll be spilling to TempDB. That’s why we freak out when user-defined functions only estimate 1-100 rows will come back, or when table variables only estimate 1 row will come back.

As we join those objects to other things in our query, SQL Server may wildly underestimate the amount of work it has to do for the rest of the tables. You can see this in action in [my Watch Brent Tune Queries video](https://www.brentozar.com/sql/watch-brent-tune-queries/) from SQLRally Nordic in Copenhagen.

On the other hand, **when SQL Server estimates too much memory** is required, we have a different problem:

* Your query may wait a long time just to start because that estimated memory isn’t available
* SQL Server may clear out a lot of otherwise-useful memory just to run your query
* Other people may have to wait for memory because a ton got allocated to your query

You can see this one in action in [Kendra’s AdventureWorks query that estimates bazillions of rows](https://www.brentozar.com/archive/2013/08/query-plans-what-happens-when-row-estimates-get-high/). It estimates 10.5 terabytes of data will be handled, when in actuality it’s only 292MB.

**But we’ve been flying blind – until now.**

Until now, it was really hard to troubleshoot these issues. While the plan cache does show us how much memory a query would want, it didn’t show us how much was actually used.

We could try running the query to get the actual plan, but even that didn’t always get us the right answer. If we were suffering from [parameter sniffing problems](https://www.brentozar.com/archive/2013/06/the-elephant-and-the-mouse-or-parameter-sniffing-in-sql-server/), one set of parameters might work fine while the other would produce catastrophic underestimates or overestimates.

We could try using Profiler or Extended Events to catch spills or huge grants as they happened, but…let’s be honest, you’re not doing that. You’re the kind of person who barely finishes reading a blog post, let alone starts firing up a tool before problems happen.

[231](javascript:void(0))

[](https://www.brentozar.com/)

[Brent Ozar](https://www.brentozar.com/archive/author/brento/)

<https://www.brentozar.com>

I make Microsoft SQL Server faster and more reliable. I love teaching, travel, and laughing. I live in Chicago with my wife Erika and my dog Ernie. I'm on an epic life quest to have fun and make a difference. In my spare time, I'm behind a bunch of stuff like GroupBy.org, FirstResponderKit.org, DBAreactions.com, and SQLServerUpdates.com.

[Previous Post Cardinality Estimator Trace Flags and Links](https://www.brentozar.com/archive/2016/08/cardinality-estimator-trace-flags-links/) [Next Post Windows Update Now Delivers SQL Server Cumulative Updates](https://www.brentozar.com/archive/2016/08/windows-update-now-delivers-sql-server-cumulative-updates/)

**4 Comments.** [**Leave new**](https://www.brentozar.com/archive/2016/08/memory-grants-week-introduction-query-memory/?utm_source=Brent+Ozar+Unlimited%C2%AE+List&utm_campaign=9831dcb39e-RSS_EMAIL_CAMPAIGN&utm_medium=email&utm_term=0_8e3e861dd9-9831dcb39e-296071565&goal=0_8e3e861dd9-9831dcb39e-296071565&mc_cid=9831dcb39e&mc_eid=b137e91a2b#respond)

https://secure.gravatar.com/avatar/f39c46571644c9d3f15e4c2818c2f5eb?s=50&d=mm&r=g

**Dennis Post**

[August 8, 2016 10:05 am](https://www.brentozar.com/archive/2016/08/memory-grants-week-introduction-query-memory/?utm_source=Brent+Ozar+Unlimited%C2%AE+List&utm_campaign=9831dcb39e-RSS_EMAIL_CAMPAIGN&utm_medium=email&utm_term=0_8e3e861dd9-9831dcb39e-296071565&goal=0_8e3e861dd9-9831dcb39e-296071565&mc_cid=9831dcb39e&mc_eid=b137e91a2b#comment-2306800)

Great post Brent.

I’ve been checking sys.dm\_exec\_query\_memory\_grants lately, comparing Granted Memory and Max Used Memory. Has been very helpful in finding poor queries. Have seen grants +4 GB that only use <200 MB. What a waste!

[Reply](https://www.brentozar.com/archive/2016/08/memory-grants-week-introduction-query-memory/?replytocom=2306800#respond)

https://secure.gravatar.com/avatar/24eff5b56ffe4f99c353ae7a523b056e?s=50&d=mm&r=g

**Aaron**

[August 10, 2016 3:33 pm](https://www.brentozar.com/archive/2016/08/memory-grants-week-introduction-query-memory/?utm_source=Brent+Ozar+Unlimited%C2%AE+List&utm_campaign=9831dcb39e-RSS_EMAIL_CAMPAIGN&utm_medium=email&utm_term=0_8e3e861dd9-9831dcb39e-296071565&goal=0_8e3e861dd9-9831dcb39e-296071565&mc_cid=9831dcb39e&mc_eid=b137e91a2b#comment-2307415)

We proactively stash the results from dm\_exec \_query\_memory\_grants thoughout Dev and test environments to get this.

More often it helps identify where stats are stale than true memory hogs though, which is useful too.

Now might be a good time to mention the query memory cap, which is a maximum of 20% of maximum work space memory from dm\_os\_performance\_counters (in itself a max of 75% of target server memory). It took some digging for me to find why all large memory queries maxed out at the same value. Min query memory is a more obvious setting of course in sp\_configure.

Oh don’t forget resource gov might influence that too.

[Reply](https://www.brentozar.com/archive/2016/08/memory-grants-week-introduction-query-memory/?replytocom=2307415#respond)

https://secure.gravatar.com/avatar/3e321be49cd9779491740f3ab91dcf5a?s=50&d=mm&r=g

**Cody**

[September 1, 2016 12:50 am](https://www.brentozar.com/archive/2016/08/memory-grants-week-introduction-query-memory/?utm_source=Brent+Ozar+Unlimited%C2%AE+List&utm_campaign=9831dcb39e-RSS_EMAIL_CAMPAIGN&utm_medium=email&utm_term=0_8e3e861dd9-9831dcb39e-296071565&goal=0_8e3e861dd9-9831dcb39e-296071565&mc_cid=9831dcb39e&mc_eid=b137e91a2b#comment-2312684)

This is an interesting post and thanks for writing it.

I have a good feeling that my comment will elicit the responses of “this goes beyond the scope of a blog post” and “sp\_BlitzCache can do this”. But I’m going to give my feedback anyway because I guess other people will come along and read it over time too. Besides, scouring through 2,800 odd lines of SQL code trying to reverse engineer a concept isn’t easy 

My feedback is that I’m left wanting a bit more about how to this fits in with the ecosystem, and how to use it (beyond just viewing it in a plan).

So far it seems:  
\* Performance counters for memory grants outstanding is still a go-to indicator for memory issues (you blogged about it here <https://www.brentozar.com/archive/2013/09/how-to-prove-your-sql-server-needs-more-memory-video/>)  
\* If you observed issues you would probably jump into sys.dm\_exec\_query\_memory\_grants to see an exactly-at-this-moment snapshot of what’s going on. But it will be gone in an instant. Aaron above says he stashes them but I’m not sure how much he grabs, on what kind of schedule, and what other analysis of thresholds is done on it.  
\* And afterwards you can scan the plan cache for these new fields to pick up old issues on some kind of threshold. It’s not clear if it’s a good idea to stash this also or if it’s likely not going to be worth it. It’s not clear if it would replace stashing sys.dm\_exec\_query\_memory\_grants.

Definitely outside of the scope of a blog post but if others want to write more about it then please do and (maybe) share the links.

[Reply](https://www.brentozar.com/archive/2016/08/memory-grants-week-introduction-query-memory/?replytocom=2312684#respond)

https://secure.gravatar.com/avatar/247101c7b54fc4e628923e5a10a2bc3e?s=50&d=mm&r=g

**JEff**

[March 7, 2017 3:25 pm](https://www.brentozar.com/archive/2016/08/memory-grants-week-introduction-query-memory/?utm_source=Brent+Ozar+Unlimited%C2%AE+List&utm_campaign=9831dcb39e-RSS_EMAIL_CAMPAIGN&utm_medium=email&utm_term=0_8e3e861dd9-9831dcb39e-296071565&goal=0_8e3e861dd9-9831dcb39e-296071565&mc_cid=9831dcb39e&mc_eid=b137e91a2b#comment-2344875)

I ran into issues recently with pending memory grants on our large reporting warehouse designed through a vendor, our main vendr, Whom I will not name(Healthcare). One note is that Vendor recommended in SQL 2014 we run the DB in 2012 mode. The machine with 3TB and 2.7TB assigned to SQL would max out memory and have many memory grants waiting all the time. Causing so much pain for those end users waiting for their reports.  
In looking I found that we have some reports that are ran in multiple for different departments. Each of these runs were requested and being reserved close to 470GB of RAM. OF note when I ran in 120 CE using the option it would request only around 80gb, so not even comparable as was the query costs. After looking and figuring out the max\_used\_memory by these queries above I ended up turning to resource governor to limit the max memory grants to 10% available. This cleared all our grants and since the queries that I limited memory too never actually used that much we still ran fine with no spill to tempdb, well besides one runaway query that needed to be fixed.  
All This, of course, is until I can get the vendor to prove to me and show me why we cant run in 2014 and use the new 120 CE. I really only want to use the Resource Gov as a stopgap and only enable it in the morning and disable it well before our ETLs. That is one use of figuring out and using memory grants granted and max\_used memory. At least now with much better stats on these I was able on my end to figure what was reserving and then not using my memory.