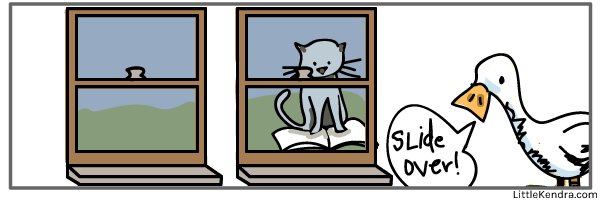
<https://www.littlekendra.com/2016/02/02/sliding-window-table-partitioning-what-to-decide-before-you-automate/>

**Sliding Window Table Partitioning: What to Decide Before You Automate**

By Kendra Little, February 2, 2016 [**3**](https://www.littlekendra.com/2016/02/02/sliding-window-table-partitioning-what-to-decide-before-you-automate/#comments)

[](http://littlekendra.com/2016/02/02/sliding-window-table-partitioning-what-to-decide-before-you-automate/sliding-window-partitioning/)

Before you do all the work to map out a complex sliding window table partitioning scheme for your SQL Server tables, here’s the top five questions I’d think through carefully:

**1) Do You Need Table Partitioning?**

Some folks think they need partitioning for performance– but it really shines as a data management feature. Just because you’ve got tables with millions of rows in them doesn’t necessarily mean that partitioning will make queries faster. Make sure you’ve worked through traditional indexing and query re-writes first. Partitioning is lots of work, so don’t skip this question.

Bonus: traditional indexes don’t require Enterprise Edition like partitioning. And even if you do eventually go down the partitioning route, you’re going to need to be familiar with the top queries that hit the table for tuning later, anyway.

**2) Do You Need a Sliding Window?**

With “sliding window” table partitioning, you regularly add new boundary points and partitions and “switch in” recent data. Similarly,  you’re regularly switching out older data and removing older boundary points.

Even with regular data loads, some patterns work well with a “rotating log buffer” pattern, which doesn’t require constantly modifying your partition function and scheme. [Read more about it on Thomas Kejser’s blog here](http://kejser.org/table-pattern-rotating-log-ring-buffer/).

**3) Filegroups: Will You Benefit from Lots of Them?**

Many of us have written tutorials showing sliding window partitioning where each partition sits on its own filegroup. When you add new boundary points, you add new filegroups. That’s extra scripting, and extra places where things can go wrong. Do you really need it?

There’s a few benefits to using lots of filegroups, but not everyone can take advantage of them (or needs them):

* **Filegroup level restore can be useful during disasters**. With Enterprise Edition (which you need for partitioning anyway), you must first bring the PRIMARY filegroup online, but can bring later filegroups on individually, even while queries are running. This means you could bring more recent/critical partitions online first if it’s permissible for you to have partial availability for that table. That’s not always allowed, as some reports might show incorrect data.
* **DBCC CHECKFILEGROUP can be run against individual filegroups.** DBCC CHECKDB *can’t* be run against individual partitions. If your table is very large and running CHECKDB against the whole thing is time prohibitive, this might be useful. A workaround if you don’t do this is to restore the whole database regularly to a secondary server just for CHECKDB. (I submitted a suggestion to add partition awareness to [DBCC CHECKTABLE back in 2010, but it was closed as “wont’ fix”.](https://connect.microsoft.com/SQLServer/feedback/details/566600/dbcc-checktable-should-support-checking-integrity-for-individual-partitions) You can still vote it up – suggestions can be reopened.)
* **Older filegroups can be migrated to slower storage**. This isn’t used very commonly anymore, because if you move the filegroup yourself then you have to take that filegroup offline and move the data. If you use SAN storage, your admin can frequently migrate the data to different storage online, but more and more SANs are dynamically keeping more active segments of data on faster disk and “tiering” less frequently accessed data to slower storage automatically. You don’t typically have to use filegroups to get a SAN to do that anymore.

Important: whatever you decide, I encourage you to not put everything in the PRIMARY filegroup. If you’ve got enough data that you need partitioning, you should have *at least* one additional filegroup for managing restores in worst case scenarios.

**4) Error Handling: What Happens When The Jobs Fail?**

Map this out before you write the code. When will the jobs run, and what should happen if they fail? Should someone be engaged? What tools will they need, and when is the Service Level Agreement for when the process has to be complete? You’ll need lots of details on this to make sure your automation and documentation meet the bar.

**4b) Are You Using WAIT\_AT\_LOW\_PRIORITY?**

I’m cheating on the numbering a little, but this is related to job failure. SQL Server 2014 added two really nice features for table partitioning:

* The ability to reindex an individual partition online
* The WAIT\_AT\_LOW\_PRIORITY options to dictate what happens if blocking occurs during an index operation (including a switch)

These two features are a huge reason to be on SQL Server 2014 or higher for any table partitioning implementation.

If you can get to SQL Server 2016, it adds [partition level truncation](https://msdn.microsoft.com/en-us/library/ms177570.aspx). (Yay!)

**5) Are  You Monitoring to Make Sure Automation Isn’t Disabled?**

Every sliding window partitioning implementation should have a secondary job or monitoring system checking state with a really obvious name, like “DBA – Critical Partition Monitoring.” The job should use logic to check and make sure that the right number of boundary points exist.

It’s very, very common for the jobs which automate table partitioning to be disabled, and accidentally not re-enabled. That leads to [“lumpy” partitions which are a big pain to fix](http://littlekendra.com/2016/01/26/partitioned-tables-missing-boundary-points/). Don’t let it happen to you.

**Are You A Partitioning Veteran? Add Your Tips in the Comments!**

I’d love to hear what you’ve learned from your implementations, too.