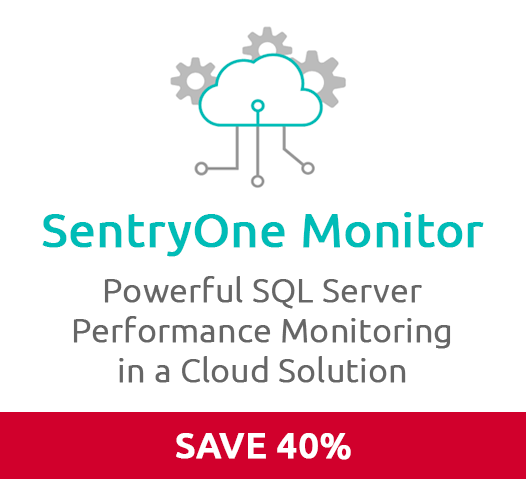
**An approach to index tuning**

March 26, 2020 by [Erin Stellato](https://sqlperformance.com/author/erinsqlskills-com) in [SQL Indexes](https://sqlperformance.com/category/sql-indexes) | [2 Comments](https://sqlperformance.com/2020/03/sql-indexes/an-approach-to-index-tuning-part-1#comments)

FROM: <https://sqlperformance.com/2020/03/sql-indexes/an-approach-to-index-tuning-part-1>

[](https://www.sentryone.com/products/sentryone-platform/sentryone-monitor?utm_source=sqlperformance-monitor-sidebar)

**SentryOne Newsletters**

The SQLPerformance.com bi-weekly newsletter keeps you up to speed on the most recent blog posts and forum discussions in the SQL Server community.

eNews is a bi-monthly newsletter with fun information about SentryOne, tips to help improve your productivity, and much more.

[Subscribe **→**](https://www.sentryone.com/events?utm_source=sqlperformance&utm_medium=banner&utm_campaign=subscribe-sidebar#landing-form)

**Featured Author**

[](https://sqlperformance.com/author/jonathansqlskills-com)Jonathan Kehayias is a Principal Consultant with SQLskills and the youngest MCM ever.

[Jonathan’s Posts **→**](https://sqlperformance.com/author/jonathansqlskills-com)

A performance tuning engagement can end up taking many turns as you work through it – it all depends on what is showing up as the problem and what the data tells you. Some days it lands on a specific query, or set of queries, that can be improved with indexes – either new ones or modifications to existing indexes. One of my favorite parts of tuning is working with indexes and, as I was thinking about this post, I was tempted to label index tuning as an "easier" task… but it really isn't.

I think of index tuning as an art and a science. You have to try and think like the optimizer, and you must understand the table schema and the query (or queries) you're trying to tune. Both of those are data-driven and thus in the category of science. The art component comes into play when you think about the *other* indexes on the table, and *all* the *other* queries that involve the table which could be affected by index changes.

**Step 1 : Identify the query and review the plan**

When I identify a query that could benefit from an index, I immediately get its plan. I often get the Execution Plan from the [plan cache](https://sqlperformance.com/?s=%22plan+cache%22) or [Query Store](https://sqlperformance.com/?s=%22query+store%22), and then use SSMS to get the Execution Plan plus Run-Time Statistics (aka Actual Execution Plan). Many times, the shape of those two plans is the same; but it's not a guarantee, which is why I like to see both.

The plan may have a missing index recommendation, it may have a clustered index scan (or heap scan if there's no clustered index), it may use a nonclustered index but then have a lookup to retrieve additional columns. Fixing each of those issues individually sounds pretty easy. Just add the missing index, right? If there's a scan of a clustered index or heap, create the index that I need for the query and be done? Or if there's an index being used but it goes to the table to get the additional columns, just add the columns to that index?

It's usually not that easy, and even when it is, I still go through the process that I am outlining here.

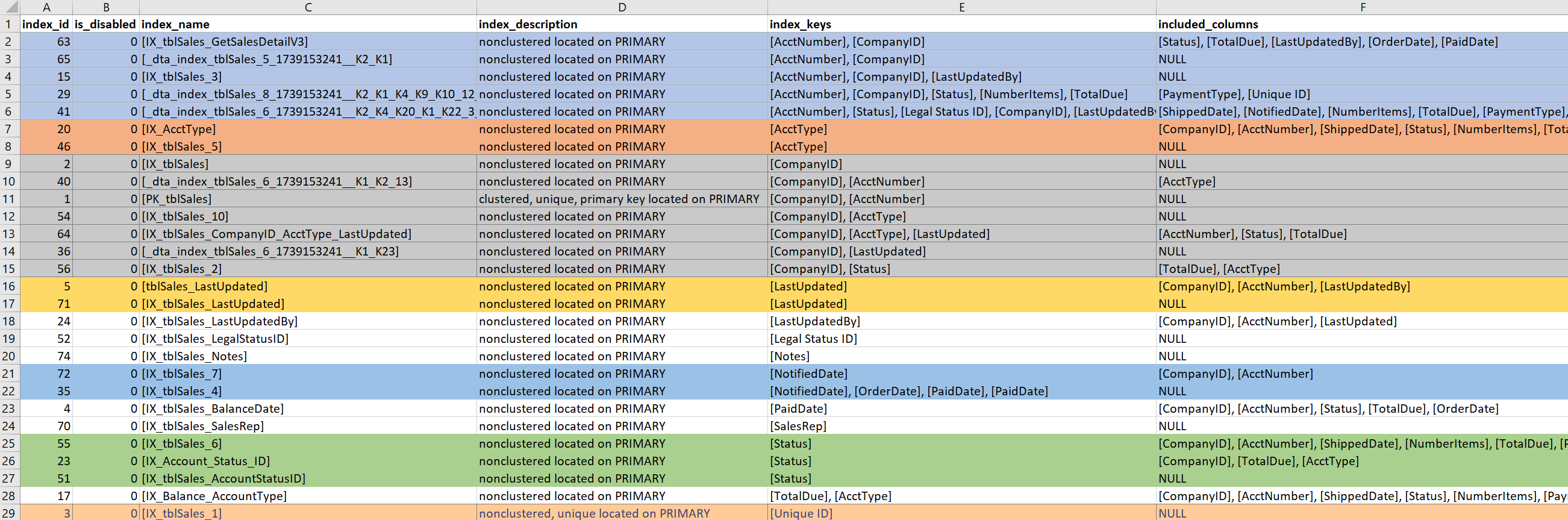
**Step 2 : Determine what table(s) to review**

Now that I have my query, I have to figure out what tables are not indexed properly. In addition to reviewing the plan, I also enable IO and TIME statistics in SSMS. This is probably old-school of me, as execution plans contain more and more information – including duration and IO numbers per operator – with each release, but I like the IO statistics because I can quickly see the reads for each table. For queries that are complex with multiple joins, or sub-queries, or CTEs, or nested views, understanding where the IO and/or time is spent in the query drives where I spend my time. Whenever possible from this point, I take the larger, complex query and pare it down to the part that's causing the biggest problem.

For example, if there's a query that joins to 10 tables and has two sub-queries, the plan (along with IO and duration information) helps me identify where the problem exists. Then I will pull out that part of the query – the problematic table and maybe a couple others to which it joins – and focus on that. Sometimes it's just the sub-query, so I start there.

**Step 3 : Look at existing indexes**

With the query (or part of the query) defined, then I focus on the existing indexes for the tables involved. For this step, I rely on [Kimberly's version of sp\_helpindex](https://www.sqlskills.com/blogs/kimberly/sp_helpindex-v20170228/). I much prefer her version to the standard sp\_helpindex because it also lists INCLUDEd columns and the filter definition (if one exists). Depending on the number of indexes that show up for a table, I will often copy this and paste it into Excel, and then order based on the index key and then the included columns. This lets me find any redundancies quickly.

[](https://sqlperformance.com/wp-content/uploads/2020/03/word-image.png)

Based on the example output above, there are seven indexes that start with CompanyID, five that start with AcctNumber, and some other potential redundancies. While it seems ideal to only have *one* index that leads on a particular column (e.g. CompanyID), for some query patterns that isn't enough.

When I'm looking at existing indexes, it's very easy to go down a rabbit-hole. I look at the output above and immediately start asking why there are seven indexes that start with CompanyID, and I want to know who created them, and why, and for what query. But… if my problematic query doesn't use CompanyID, should I care? Yes… because in general I'm there to improve performance, and if that means looking at other indexes on the table along the way, then so be it. But this is where it's easy to lose track of time (and true purpose).

If my problematic query needs an index that leads on PaidDate, I only have to deal with one existing index. If my problematic query needs an index that leads on AcctNumber, it gets tricky. When existing indexes sort of cover a query, and I'm looking to expand an index (add more columns) or consolidate (merge two or maybe three indexes into one), then I have to dig in.

**Step 4 : Index Usage Stats**

I find that a lot of people do not capture index usage stats on an ongoing basis. This is unfortunate, because I find the data helpful when deciding which indexes to keep, and which to drop or merge. In the case where I don't have historical usage stats, I at least check to see how the usage looks currently (since the last service restart):

SELECT

DB\_NAME(ius.database\_id),

OBJECT\_NAME(i.object\_id) [TableName],

i.name [IndexName],

ius.database\_id,

i.object\_id,

i.index\_id,

ius.user\_seeks,

ius.user\_scans,

ius.user\_lookups,

ius.user\_updates

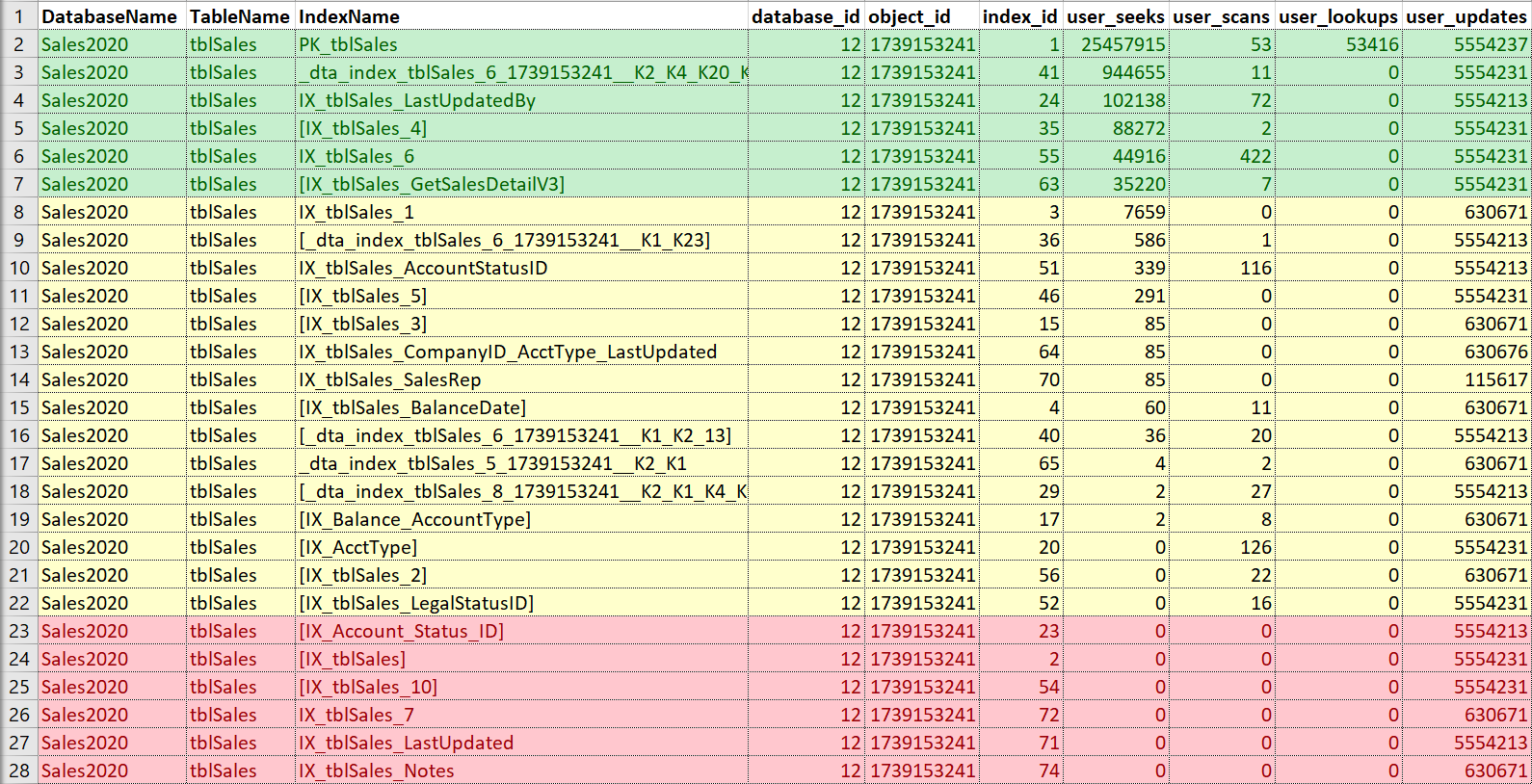
FROM sys.indexes i

INNER JOIN sys.dm\_db\_index\_usage\_stats ius

ON ius.index\_id = i.index\_id AND ius.object\_id = i.object\_id

WHERE ius.database\_id = DB\_ID(N'Sales2020')

AND i.object\_id = OBJECT\_ID('dbo.tblSales');

[](https://sqlperformance.com/wp-content/uploads/2020/03/word-image-1.png)

Again, I like to put this into Excel, sort by seeks and then scans, and also take note of updates. For this example, the indexes in red are those with no seeks, scans, or lookups… only updates. Those are candidates for being disabled and potentially dropped, if they are truly not used (again, having usage history would help here). The indexes in green are definitely being used, I want to keep those (though perhaps in some cases they could be tweaked). The ones in yellow… some are kind of being used, some are barely being used. Again, history would be helpful here, or context from others — sometimes an index may be crucial for a report or process that doesn't run all the time.

If I'm just looking to modify or add a new index, versus true clean-up and consolidation, then I am mostly concerned with any indexes that are similar to what I want to add or change. However, I will make sure to point out the usage information to the customer and, if time permits, assist with the overall indexing strategy for the table.

**What's Next?**

We're not done! This is part 1 of my approach to index tuning, and my next installment will list out the rest of my steps. In the meantime, if you're not capturing index usage stats, that's something you can put in place using the query above, or another variation. I would recommend capturing usage stats for all user databases, not just a specific table and database as I've done above, so modify the predicate as necessary. And finally, as part of that scheduled job to snapshot that information to a table, don't forget another step to clean up the table after data has been there for a while (I keep it for at least six months; some might say a year is necessary).

**2 thoughts on “An approach to index tuning – Part 1”**

1. *Chris Wood* says:

[March 27, 2020 at 6:23 PM](https://sqlperformance.com/2020/03/sql-indexes/an-approach-to-index-tuning-part-1#comment-163476)

Time Ford and Louis Davidson produced a book that shows the index usage stats and how to use them. I adopted that soon after I read the book.

[Reply](https://sqlperformance.com/2020/03/sql-indexes/an-approach-to-index-tuning-part-1#comment-163476)

* 1. *Erin Stellato* says:

[March 27, 2020 at 8:01 PM](https://sqlperformance.com/2020/03/sql-indexes/an-approach-to-index-tuning-part-1#comment-163477)

Chris-

That's a great resource! I only wrote two paragraphs, but you can write pages about digging into usage stats and really making the most of them.

Thanks for reading!

Erin

[Reply](https://sqlperformance.com/2020/03/sql-indexes/an-approach-to-index-tuning-part-1#comment-163477)