5 most common causes of SQL Server performance problems

POSTED ON [JANUARY 31, 2018](https://www.wardyit.com/blog/5-common-causes-sql-server-performance-problems/) BY [WARDY IT SOLUTIONS](https://www.wardyit.com/blog/author/wardy-it-solutions/)

**31  
Jan**

<https://www.wardyit.com/blog/5-common-causes-sql-server-performance-problems/>

At WARDY IT Solutions, we’ve lived and breathed SQL Server for the past 14 years. With more than 100 years of SQL Server experience between them, we asked our Data Platform consultants to share their opinions on the most common causes of performance problems and of course, how to avoid them. Happy performance tuning!

Start by checking your wait stats

Now this isn’t a cause but if you hit a performance problem, checking your wait stats is the place to start. SQL Server continuously tracks why execution threads have to wait. This information is vital when you’re trying to pinpoint the root cause of your performance problems. The good news is finding the cause of your performance problem is easy, the bad news is interpreting the information and fixing the issue is often complicated and time consuming but this will point you in the right direction. If your environment is anything like most, the chances are wait stats will point you in one of these five directions:

1. Ensure your TempDB database is configured optimally.

TempDB configuration is a repeat offender when it comes to performance bottlenecks. Look out for multiple data files and check that trace flags 1117 and 1118 are enabled.

*Note: If you’re running SQL Server 2016 or above, trace flags 1117 and 1118 are enabled as standard.*

*Tip:*

* PAGELATCH\_XX wait types are an **indicator** of tempdb contention, further troubleshooting is required
* For machines with up to 8 CPUs, the general rule of thumb is that you should have the same number of tempdb data files as you have logical processors <https://support.microsoft.com/en-us/help/2154845/recommendations-to-reduce-allocation-contention-in-sql-server-tempdb-d>
* Place your tempdb files on dedicated high-speed disks
* Grow your tempdb data files to a desired size
* Set your files to grow equally
* Enable IFI

2. Make sure you’re running index maintenance frequently.

It’s difficult to know how often you should be running index maintenance as it’s dependant on your workload and how quickly your data changes. If you’re unsure what ‘frequently’ means for you, start by checking for index fragmentation on a daily basis. If it takes a week for fragmentation to reach 30% or more, schedule in weekly index maintenance. If they fragment sooner, you should try to schedule a couple of index maintenance sessions a week, if not daily – **this should include updating statistics.**

Tip: If you don’t have an index maintenance plan, try using a tool like [SQL Server Maintenance Solution](https://ola.hallengren.com/) by Ola Hallengren. This free script dynamically determines the indexes that require maintenance. We run this tool in every SQL Server we manage – it’s a great place to start.

3. Implement indexes that provide a benefit to your queries.

This involved creating missing indexes *and* removing poor performing or unutilized indexes. Having a large number of indexes can be detrimental to DML transactions. Appropriate indexing means only creating indexes which have a positive impact to your read performance that outweigh the negative impact on write performance which requires a good understanding of your workload.

This script will help you identify missing indexes:

|  |  |
| --- | --- |
|  | SELECT |
|  | migs.avg\_total\_user\_cost \* (migs.avg\_user\_impact / 100.0) \* (migs.user\_seeks + migs.user\_scans) AS improvement\_measure, |
|  | 'CREATE INDEX [missing\_index\_' + CONVERT (varchar, mig.index\_group\_handle) + '\_' + CONVERT (varchar, mid.index\_handle) |
|  | + '\_' + LEFT (PARSENAME(mid.statement, 1), 32) + ']' |
|  | + ' ON ' + mid.statement |
|  | + ' (' + ISNULL (mid.equality\_columns,'') |
|  | + CASE WHEN mid.equality\_columns IS NOT NULL AND mid.inequality\_columns IS NOT NULL THEN ',' ELSE '' END |
|  | + ISNULL (mid.inequality\_columns, '') |
|  | + ')' |
|  | + ISNULL (' INCLUDE (' + mid.included\_columns + ')', '') AS create\_index\_statement, |
|  | migs.\*, mid.database\_id, mid.[object\_id] |
|  | FROM |
|  | sys.dm\_db\_missing\_index\_groups mig |
|  | INNER JOIN sys.dm\_db\_missing\_index\_group\_stats migs ON migs.group\_handle = mig.index\_group\_handle |
|  | INNER JOIN sys.dm\_db\_missing\_index\_details mid ON mig.index\_handle = mid.index\_handle |
|  | WHERE |
|  | migs.avg\_total\_user\_cost \* (migs.avg\_user\_impact / 100.0) \* (migs.user\_seeks + migs.user\_scans) > 10 |
|  | ORDER BY |
|  | migs.avg\_total\_user\_cost \* migs.avg\_user\_impact \* (migs.user\_seeks + migs.user\_scans) DESC |

[**view raw**](https://gist.github.com/anonymous/9a139dcb673353b01ace5a355a1f9419/raw/d6ae9e0a44c1c5d2f6796452d04c87bcbb75df27/Missing%20Indexes.sql)[**Missing Indexes.sql**](https://gist.github.com/anonymous/9a139dcb673353b01ace5a355a1f9419#file-missing-indexes-sql) hosted with  by [**GitHub**](https://github.com/)

And this script will find poor performing indexes:

|  |  |
| --- | --- |
|  | /\*============================================================================ |
|  | File: Index - Rarely Used Indexes |
|  |  |
|  | Summary: Sample stored procedure that lists rarely-used indexes. Because the number and type of accesses are |
|  | tracked in dmvs, this procedure can find indexes that are rarely useful. Because the cost of these indexes |
|  | is incurred during maintenance (e.g. insert, update, and delete operations), the write costs of rarely-used |
|  | indexes may outweigh the benefits. |
|  |  |
|  | sp\_help tblPasswordHistory |
|  | sp\_helptext fnt\_currency\_user |
|  | select top 10 \* from tblPasswordHistory |
|  |  |
|  | Date: 2008 |
|  |  |
|  | Versions: 2005, 2008, 2012 |
|  | ------------------------------------------------------------------------------ |
|  | Written by Ben DeBow, SQLHA |
|  |  |
|  | For more scripts and sample code, check out |
|  | http://www.SQLHA.com |
|  |  |
|  | THIS CODE AND INFORMATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF |
|  | ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED |
|  | TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND/OR FITNESS FOR A |
|  | PARTICULAR PURPOSE. |
|  | ============================================================================\*/ |
|  |  |
|  | /\* Create a temporary table to hold our data, since we're going to iterate through databases \*/ |
|  | IF OBJECT\_ID('tempdb..#Results') IS NOT NULL |
|  | DROP TABLE #Results; |
|  |  |
|  | CREATE TABLE [dbo].#Results( |
|  | [Server Name] [nvarchar](128) NULL, |
|  | [DB Name] [nvarchar](128) NULL, |
|  | [source] [varchar](10) NOT NULL, |
|  | [objectname] [nvarchar](128) NULL, |
|  | [object\_id] [int] NOT NULL, |
|  | [indexname] [sysname] NULL, |
|  | [data\_compression] [varchar](24) NOT NULL, |
|  | [index\_id] [int] NOT NULL, |
|  | [rowcnt] [bigint] NULL, |
|  | [datapages] [bigint] NULL, |
|  | [is\_unique] [bit] NULL, |
|  | [count] [int] NULL, |
|  | [user\_seeks] [bigint] NOT NULL, |
|  | [user\_scans] [bigint] NOT NULL, |
|  | [user\_lookups] [bigint] NOT NULL, |
|  | [user\_updates] [bigint] NOT NULL, |
|  | [total\_usage] [bigint] NOT NULL, |
|  | [%Reads] [bigint] NULL, |
|  | [%Writes] [bigint] NULL, |
|  | [%Seeks] [bigint] NULL, |
|  | [%Scans] [bigint] NULL, |
|  | [%Lookups] [bigint] NULL, |
|  | [%Updates] [bigint] NULL, |
|  | [last\_user\_scan] [datetime] NULL, |
|  | [last\_user\_seek] [datetime] NULL, |
|  | [run\_time] [datetime] NOT NULL |
|  | ) ON [PRIMARY] |
|  | EXECUTE sys.sp\_MSforeachdb |
|  | 'USE [?]; |
|  | declare @dbid int |
|  | select @dbid = db\_id() |
|  | INSERT INTO #Results |
|  | SELECT @@SERVERNAME AS [Server Name] |
|  | , db\_name() AS [DB Name] |
|  | , ''Usage Data'' ''source'' |
|  | , objectname=object\_name(s.object\_id) |
|  | , s.object\_id, indexname=i.name |
|  | , data\_compression\_desc, i.index\_id |
|  | , s2.rowcnt, sa.total\_pages, is\_unique |
|  | , (select count(\*) |
|  | from sys.indexes r |
|  | where r.object\_id = s.object\_id) ''count'' |
|  | , user\_seeks, user\_scans, user\_lookups, user\_updates, user\_seeks + user\_scans + user\_lookups + user\_updates AS [total\_usage] |
|  | , CAST(CAST(user\_seeks + user\_scans + user\_lookups AS DEC(12,2))/CAST(REPLACE((user\_seeks + user\_scans + user\_lookups + user\_updates), 0, 1) AS DEC(12,2)) \* 100 AS DEC(5,2)) [%Reads] |
|  | , CAST(CAST(user\_updates AS DEC(12,2))/CAST(REPLACE((user\_seeks + user\_scans + user\_lookups + user\_updates), 0, 1) AS DEC(12,2)) \* 100 AS DEC(5,2)) [%Writes] |
|  | , CAST(CAST(user\_seeks AS DEC(12,2))/CAST(REPLACE((user\_seeks + user\_scans + user\_lookups + user\_updates), 0, 1) AS DEC(12,2)) \* 100 AS DEC(5,2)) [%Seeks] |
|  | , CAST(CAST(user\_scans AS DEC(12,2))/CAST(REPLACE((user\_seeks + user\_scans + user\_lookups + user\_updates), 0, 1) AS DEC(12,2)) \* 100 AS DEC(5,2)) [%Scans] |
|  | , CAST(CAST(user\_lookups AS DEC(12,2))/CAST(REPLACE((user\_seeks + user\_scans + user\_lookups + user\_updates), 0, 1) AS DEC(12,2)) \* 100 AS DEC(5,2)) [%Lookups] |
|  | , CAST(CAST(user\_updates AS DEC(12,2))/CAST(REPLACE((user\_seeks + user\_scans + user\_lookups + user\_updates), 0, 1) AS DEC(12,2)) \* 100 AS DEC(5,2)) [%Updates] |
|  | , last\_user\_scan |
|  | , last\_user\_seek |
|  | , getdate() run\_time |
|  | from sys.dm\_db\_index\_usage\_stats s |
|  | join sys.indexes i on i.object\_id = s.object\_id |
|  | and i.index\_id = s.index\_id |
|  | join sysindexes s2 on i.object\_id = s2.id |
|  | and i.index\_id = s2.indid |
|  | join sys.partitions sp on i.object\_id = sp.object\_id |
|  | and i.index\_id = sp.index\_id |
|  | join sys.allocation\_units sa on sa.container\_id = sp.hobt\_id |
|  | where objectproperty(s.object\_id, ''IsUserTable'') = 1 |
|  | and database\_id = @dbid' |
|  |  |
|  | EXECUTE sys.sp\_MSforeachdb |
|  | 'USE [?]; |
|  |  |
|  | declare @dbid int |
|  |  |
|  | select @dbid = db\_id() |
|  |  |
|  | INSERT INTO #Results |
|  | SELECT @@SERVERNAME |
|  | , db\_name() |
|  | , ''NA'' |
|  | , object\_name(i.object\_id) |
|  | , o.object\_id |
|  | , i.name |
|  | , data\_compression\_desc |
|  | , i.index\_id |
|  | , s2.rowcnt |
|  | , sa.total\_pages |
|  | , is\_unique |
|  | , (select count(\*) |
|  | from sys.indexes r |
|  | where r.object\_id = i.object\_id) ''count'' |
|  | , 0 |
|  | , 0 |
|  | , 0 |
|  | , 0 |
|  | , 0 |
|  | , 0 |
|  | , 0 |
|  | , 0 |
|  | , 0 |
|  | , 0 |
|  | , 0 |
|  | , 0 |
|  | , 0 |
|  | , getdate() |
|  | FROM sys.indexes i |
|  | JOIN sys.objects o |
|  | ON i.object\_id = o.object\_id |
|  | join sysindexes s2 on i.object\_id = s2.id |
|  | and i.index\_id = s2.indid |
|  | join sys.partitions sp on i.object\_id = sp.object\_id |
|  | and i.index\_id = sp.index\_id |
|  | join sys.allocation\_units sa on sa.container\_id = sp.hobt\_id |
|  | WHERE OBJECTPROPERTY(o.object\_id,''IsUserTable'') = 1 |
|  | AND i.index\_id NOT IN ( |
|  | SELECT s.index\_id |
|  | FROM sys.dm\_db\_index\_usage\_stats s |
|  | WHERE s.object\_id = i.object\_id |
|  | AND i.index\_id = s.index\_id |
|  | AND database\_id = @dbid) |
|  | --AND i.index\_id NOT IN (0,1)' |
|  |  |
|  | SELECT \* |
|  | FROM #Results |
|  | WHERE [DB Name] NOT IN ('MASTER', 'msdb', 'MODEL', 'TEMPDB') |
|  |  |
|  | DROP TABLE #Results; |
|  |  |
|  | /\* |
|  | declare @dbid int |
|  |  |
|  | select @dbid = db\_id() |
|  |  |
|  | SELECT @@SERVERNAME AS [Server Name] |
|  | , db\_name() AS [DB Name] |
|  | , 'Usage Data' 'source' |
|  | , objectname=object\_name(s.object\_id) |
|  | , s.object\_id |
|  | , indexname=i.name |
|  | , data\_compression\_desc |
|  | , i.index\_id |
|  | , s2.rowcnt |
|  | , sa.total\_pages |
|  | , is\_unique |
|  | , (select count(\*) |
|  | from sys.indexes r |
|  | where r.object\_id = s.object\_id) 'count' |
|  | , user\_seeks |
|  | , user\_scans |
|  | , user\_lookups |
|  | , user\_updates |
|  | , user\_seeks + user\_scans + user\_lookups + user\_updates AS [total\_usage] |
|  | , CAST(CAST(user\_seeks AS DEC(12,2))/CAST(REPLACE((user\_seeks + user\_scans + user\_lookups + user\_updates), 0, 1) AS DEC(12,2)) \* 100 AS DEC(5,2)) [% Seeks] |
|  | , CAST(CAST(user\_scans AS DEC(12,2))/CAST(REPLACE((user\_seeks + user\_scans + user\_lookups + user\_updates), 0, 1) AS DEC(12,2)) \* 100 AS DEC(5,2)) [% Scans] |
|  | , CAST(CAST(user\_lookups AS DEC(12,2))/CAST(REPLACE((user\_seeks + user\_scans + user\_lookups + user\_updates), 0, 1) AS DEC(12,2)) \* 100 AS DEC(5,2)) [% Lookups] |
|  | , CAST(CAST(user\_updates AS DEC(12,2))/CAST(REPLACE((user\_seeks + user\_scans + user\_lookups + user\_updates), 0, 1) AS DEC(12,2)) \* 100 AS DEC(5,2)) [% Updates] |
|  | , last\_user\_scan |
|  | , last\_user\_seek |
|  | , getdate() run\_time |
|  | from sys.dm\_db\_index\_usage\_stats s |
|  | join sys.indexes i on i.object\_id = s.object\_id |
|  | and i.index\_id = s.index\_id |
|  | join sysindexes s2 on i.object\_id = s2.id |
|  | and i.index\_id = s2.indid |
|  | join sys.partitions sp on i.object\_id = sp.object\_id |
|  | and i.index\_id = sp.index\_id |
|  | join sys.allocation\_units sa on sa.container\_id = sp.hobt\_id |
|  | where objectproperty(s.object\_id, 'IsUserTable') = 1 |
|  | and database\_id = @dbid |
|  | --and 'etblHistory' = object\_name(s.object\_id) |
|  | UNION ALL |
|  | SELECT @@SERVERNAME AS [Server Name] |
|  | , db\_name() AS [DB Name] |
|  | , 'NA' |
|  | , objectname = object\_name(o.object\_id) |
|  | , o.object\_id |
|  | , indexname = i.name |
|  | , i.index\_id |
|  | , s2.rowcnt |
|  | , sa.total\_pages |
|  | , is\_unique |
|  | , data\_compression\_desc |
|  | , (select count(\*) |
|  | from sys.indexes r |
|  | where r.object\_id = i.object\_id) 'count' |
|  | , 0 |
|  | , 0 |
|  | , 0 |
|  | , 0 |
|  | , 0 |
|  | , 0 |
|  | , 0 |
|  | , 0 |
|  | , 0 |
|  | , 0 |
|  | , 0 |
|  | , getdate() run\_time |
|  | FROM sys.indexes i |
|  | JOIN sys.objects o |
|  | ON i.object\_id = o.object\_id |
|  | join sysindexes s2 on i.object\_id = s2.id |
|  | and i.index\_id = s2.indid |
|  | join sys.partitions sp on i.object\_id = sp.object\_id |
|  | and i.index\_id = sp.index\_id |
|  | join sys.allocation\_units sa on sa.container\_id = sp.hobt\_id |
|  | WHERE OBJECTPROPERTY(o.object\_id,'IsUserTable') = 1 |
|  | AND i.index\_id NOT IN ( |
|  | SELECT s.index\_id |
|  | FROM sys.dm\_db\_index\_usage\_stats s |
|  | WHERE s.object\_id = i.object\_id |
|  | AND i.index\_id = s.index\_id |
|  | AND database\_id = @dbid) |
|  | --AND i.index\_id NOT IN (0,1) |
|  | order by last\_user\_scan, last\_user\_seek |
|  | \*/ |

[**view raw**](https://gist.github.com/anonymous/fc9581931234ffe5cbbfe11565992ffe/raw/01f03d4bbffa80b339feebb7a919fe605a25e2c9/Rarely%20Used%20Indexes.sql)[**Rarely Used Indexes.sql**](https://gist.github.com/anonymous/fc9581931234ffe5cbbfe11565992ffe#file-rarely-used-indexes-sql) hosted with  by [**GitHub**](https://github.com/)

4. Check your most expensive queries and stored procedures.

Identify which queries and stored procs have the most IO, CPU usage and are taking the longest to run and see if there’s room for improvement. Start by working on those that will give you the best performance improvement for the least amount of effort. There aren’t many of us who enjoy this and it’s not always possible but it’s certainly worth a check.

|  |  |
| --- | --- |
|  | SELECT |
|  | TOP 10 SUBSTRING(qt.TEXT, (qs.statement\_start\_offset/2)+1, |
|  | ((CASE qs.statement\_end\_offset |
|  | WHEN -1 THEN DATALENGTH(qt.TEXT) |
|  | ELSE qs.statement\_end\_offset |
|  | END - qs.statement\_start\_offset)/2)+1), |
|  | qs.execution\_count, |
|  | qs.total\_logical\_reads, qs.last\_logical\_reads, |
|  | qs.total\_logical\_writes, qs.last\_logical\_writes, |
|  | qs.total\_worker\_time, |
|  | qs.last\_worker\_time, |
|  | qs.total\_elapsed\_time/1000000 total\_elapsed\_time\_in\_S, |
|  | qs.last\_elapsed\_time/1000000 last\_elapsed\_time\_in\_S, |
|  | qs.last\_execution\_time, |
|  | qp.query\_plan |
|  | FROM |
|  | sys.dm\_exec\_query\_stats qs |
|  | CROSS APPLY sys.dm\_exec\_sql\_text(qs.sql\_handle) qt |
|  | CROSS APPLY sys.dm\_exec\_query\_plan(qs.plan\_handle) qp |
|  | ORDER BY |
|  | qs.total\_logical\_reads DESC -- logical reads |
|  | -- ORDER BY qs.total\_logical\_writes DESC -- logical writes |
|  | -- ORDER BY qs.total\_worker\_time DESC -- CPU time |

[**view raw**](https://gist.github.com/anonymous/d1a514aa8a72882823a22c51e42ad89d/raw/9a8c82fe7df0dadd7ce773f08f10ee8c2e68e093/Top10MostExpensiveQueries.sql)[**Top10MostExpensiveQueries.sql**](https://gist.github.com/anonymous/d1a514aa8a72882823a22c51e42ad89d#file-top10mostexpensivequeries-sql) hosted with  by [**GitHub**](https://github.com/)

5. Monitor your performance counters

Even if you manage to solve your immediate issue by following the steps above, we recommend monitoring your performance counters to isolate the deeper issues. Doing this on a regular basis will ensure you have a targeted performance tuning approach and will help you spot issues before they escalate. Monitoring your counters when your environment is healthy will give you a good baseline.

If you’re not a [Virtual DBA](https://www.wardyit.com/virtual-dba/) customer, you could try using a free tool like the Performance Analysis Log (PAL). PAL enables you to produce a template for collecting your performance counters, collect 24 hours worth of data and produce a report. It uses generic thresholds to give you an indication of why you might have performance issues. The report includes stats on every counter making it time consuming to work through. It helps if you know your environment so you can focus on the most critical areas. [Learn more about using the PAL tool](https://blogs.technet.microsoft.com/askperf/2009/04/10/two-minute-drill-performance-analysis-of-logs-tool-pal/) and or [download PAL from GitHub](https://github.com/clinthuffman/PAL).

SQL Server Healthcheck

If these tips don’t help you or if you don’t have the time to investigate the underlying issue, it may be worth considering a SQL Server Healthcheck. During a healthcheck, our Data Platform consultants carry out a comprehensive examination of your SQL Server environment. We’ll highlight areas of concern and provide actionable recommendations to improve the performance and availability of your SQL Server infrastructure. You can download an example of our [SQL Server Healthchek report & work plan here](http://hub.wardyit.com/download-sql-server-healthcheck-report-work-plan-example).