A Performance Troubleshooting Methodology for SQL Server

28 September 2011

FROM: <https://www.simple-talk.com/sql/performance/a-performance-troubleshooting-methodology-for-sql-server/>

When healing a sick SQL Server, you must forget the idea that there could ever be a simple correspondence between symptom and disease: The art of troubleshooting is much more the art of discovering, and assembling, the various pieces of the puzzle so that you have a complete understanding of what is going on inside of a server

[https://secure.gravatar.com/avatar/8945a8723d6fb0079749ff0ed21c7c8b?s=40&d=mm&r=gJonathan Kehayias](https://www.simple-talk.com/author/jonathan-kehayias/)

Knowing where to start is the toughest part of solving a problem. As a Senior Database Administrator, I prided myself on being able to pinpoint the root cause of problems in my servers, and quickly restore services to normal working order. The ability to do this is partly down to a sound knowledge of your SQL Server environment, partly to having the right tools and scripts, and partly to what you learn to look out for, based on hard-earned lessons of the past.

Nailing down a specific methodology for troubleshooting problems with SQL Server is hard because, of course, the exact route taken to solve the problem will depend on the specific nature of the problem and the environment. One of the keys to accurate troubleshooting is not only collecting and examining all of the relevant pieces of information, but also working out what they are telling you, collectively. There is a famous old proverb, recorded in John Heywood’s *Dialogue Containing the Number in Effect of All the Proverbs in the English Tongue*, which sums this up very well:

*I see, yet I cannot see the wood for the trees.*

If you collect and examine individually five separate pieces of performance data, it’s possible that each could send you down a separate path. Viewed as a group, they will likely lead you down the sixth, and correct, path to resolving the issue. If there is one take-away from this article, it should be that focusing on a single piece of information alone can often lead to an incorrect diagnosis of a problem.

What I attempt to offer in this article is not a set of stone tablets, prescribing the exact steps to take to resolve all SQL Server problems, but rather a basic approach and set of tools that have served me well time and again in the six years I’ve spent working with SQL Server, troubleshooting performance problems. It covers a high-level description of my basic approach, followed by more detailed sections on each of my areas of focus, including wait statistics, virtual file statistics, SQL Server-related performance counters, and plan cache analysis.

Defining a Troubleshooting Methodology

Defining a troubleshooting methodology is hard, because the actual methodology that I apply depends entirely on the specific problem that I am trying to troubleshoot for a specific environment. However, my basic approach, and the tools I use, remain constant, regardless of whether the problem is users complaining of slow performance, or if I am just performing a standard server health check.

When I am examining a server for the first time, I need to establish a picture of its general health, and there are a number of items on which I’ll focus in order to obtain this picture. For each piece of information I collect, I’ll be examining it in relation to the previous data points, in order to validate or disprove any previous indicators as to the nature of the problem.

Fairly early on in any analysis, I’ll take a look at the wait statistics, in the sys.dm\_os\_wait\_stats Dynamic Management View (DMV), to identify any major resource waits in the system, at the operating system level. Let’s say I identify very high PAGEIOLATCH\_SH waits, which indicates that sessions are experiencing delays in obtaining a latch for a buffer page. This happens when lots of sessions, or maybe one session in particular, are requesting a lot of data pages that are not available in the buffer pool (and so physical I/O is needed to retrieve them). SQL Server must allocate a buffer page for each one, and place a latch on that page while it’s retrieved from disk. The bottleneck here is disk I/O; the disk subsystem simply can’t return pages quickly enough to satisfy all of the page requests, and so sessions are waiting for latches, and performance is suffering. However, this does not necessarily mean that a slow disk subsystem is the cause of the bottleneck; it may simply be the victim of excessive I/O caused by a problem elsewhere in the system.

At this point, I’ll want to validate this information by examining the virtual file stats, in sys.dm\_io\_virtual\_file\_stats. Specifically, I’ll be looking for evidence of high latency associated with the read and write operations being performed by SQL Server. At the same time, I’ll be drilling deeper into the problem, since the virtual file stats will tell me how much I/O activity is being performed by SQL Server, and how the I/O load is distributed across files and databases on the SQL Server instance. To corroborate this data further, I may also check the values of the Physical Disk\Avg. Disk Reads/sec and Physical Disk\Avg. Disk Writes/sec PerfMon counters. So, at this stage, let’s say I’ve confirmed high latency associated with read and write operations, and found that a particular database is experiencing a very high level of mainly read-based I/O.

My next step will be to investigate the execution statistics for queries against this database, which are maintained in sys.dm\_exec\_query\_stats for the execution plans that are in the plan cache. I’ll identify the queries that have the highest accumulated physical reads, and then review their associated execution plans, looking for any performance tuning opportunities, either by adding missing indexes to the database, or making changes to the SQL code, in order to optimize the way the database engine accesses the data.

It may be that the code is optimized as far as it can be, but a commonly executed reporting query simply needs to read 6 GB of data from the database, for aggregation, as a part of its execution. If most of this data isn’t found in the buffer cache, it will cause high physical I/O, and will account for the high PAGEIOLATCH\_SH waits. At this point, we may need to look at our hardware configuration and see if the actual root of our problem is a lack of memory installed in the server. In order to verify this, I’ll examine the PerfMon memory counters. If I see that the Page Life Expectancy is consistently fluctuating, and the system is experiencing non-zero values for Free List Stalls/sec, and high Lazy Writes/sec, then I can be fairly certain that the buffer pool for the instance is inadequately sized for the amount of data that is being used by the workload. This does not necessarily mean the server needs more memory; it may be that the queries are inefficient and are reading far more data than necessary. To identify the appropriate fix will require further and deeper analysis. This is just one of many possible examples, but it is a real-world example that I have encountered on many occasions while troubleshooting performance problems with SQL Server.

There are a number of points in this troubleshooting process where it would have been very easy to jump to the wrong conclusion regarding the nature of the problem. For example, after reviewing the virtual file statistics and the performance counters for the Physical Disks in the server, it would be easy to conclude that the disk I/O subsystem for the server was inappropriately sized for the amount of work being done, and that additional disks needed to be purchased to handle the disk I/O demands for the server. Unfortunately, scaling up a disk I/O subsystem can be an extremely expensive solution if the problem happens to be a missing index related to a commonly executed query, or buffer pool memory pressure. It is possible that buying a large enough disk configuration will temporarily mask the problem, but since the underlying root cause has not been resolved, you can be sure that the same problem will recur later, as the system continues to grow.

Having provided an overview of my basic approach, the following sections will drill a little deeper into the specific areas of focus, such as wait statistics, virtual file statistics, performance counters, and plan cache usage. I’ll explain the information they offer individually, and how all of this information interrelates, to help you assemble a complete understanding of what is going on inside of a server.

**Don’t forget the obvious:**

Just a gentle reminder: before you get yourself all giddy collecting diagnostic data, make sure you’ve checked for obvious problems. If a user reports that their application is “not working properly,” the first thing you should probably do is to ensure that the SQL Server services are actually running on your server. If you open **SQL Server Configuration Manager (SSCM)** and find that the status of the **Database Engine** service, which has a **Service Type** of SQL Server, is **Stopped** then this is very likely the cause of the problem, unless the instance is running in a failover cluster, at which point you need to look at the **Failover Cluster Manager** to identify if the service and its dependent resources are online, and begin troubleshooting why the service fails to start, based on what you find!

Wait Statistics: the Basis for Troubleshooting

One of the first items that I check, when troubleshooting performance problems on a SQL Server, is the wait statistics, which are tracked by the SQLOS during normal operations of any SQL Server.

The SQLOS is a pseudo-operating system that runs as a part of the SQL Server database engine and provides thread scheduling, memory management, and other functions to SQL Server. Normally, such services would, for any processes running inside the operating system, be provided by the operating system. The reason that SQL Server provides its own pseudo-operating system environment is that SQL Server knows how to schedule its tasks better than the Windows operating system does, and the cooperative scheduling that is implemented by the SQLOS allows for higher levels of concurrency than the preemptive scheduling provided by the Windows operating system.

As an example of this, any time that SQL Server has to wait while executing an operation or statement, the time spent waiting is tracked by the SQLOS, as **wait time**. This data is exposed, for each instance of SQL Server installed on a server, in the sys.dm\_os\_wait\_stats DMV. The cause and length of the various waits that SQL Server is experiencing can provide significant insight into the cause of the performance problems, as long as you understand exactly what the wait statistics are telling you, and know how to correlate the wait information with the additional troubleshooting information such as the PerfMon counters, and other DMVs.

One of the reasons that wait statistics is such a good place to begin troubleshooting SQL Server performance problems is that, often times, the specifics of the problem are not well defined by the users, when reporting the problem. More often than not, the description of the problem is limited to, “x, y, or z process is slower than normal, can you fix it?” One of the easiest ways to troubleshoot an unknown problem with performance is to look at where and why SQL Server actually had to wait to continue execution of its various tasks.

Usually, Windows Server and SQL Server patches will have been regularly applied to the server, so you’ll know how long ago the server was restarted, and therefore over what period the statistics have accumulated (unless someone manually cleared them out – see later). Ideally, you’ll want this period to be longer than around two weeks (in order to ensure the stats cover the entire workload), but not so long that the data becomes hard to analyze. In the latter case, you might also consider capturing the values, waiting a period, capturing again and comparing the two.

Diagnosing wait statistics for a single instance of SQL Server is no small task. Often times, the information provided by the wait statistics is only a symptom of the actual problem. To use this wait information effectively, you need to understand the difference between resource (i.e. traceable to a hardware resource) and non-resource waits in the system, and the other outputs provided by SQL Server, in relation to the wait information that is being tracked by the SQL Server instance overall.

As a part of the normal operations of SQL Server, a number of wait conditions exist which are non-problematic in nature and generally expected on the server. These wait conditions can generally be queried from the sys.dm\_os\_waiting\_tasks DMV for the system sessions, as shown in Listing 1.1.

|  |  |
| --- | --- |
| 1  2  3  4  5 | SELECT DISTINCT          wt.wait\_type  FROM    sys.dm\_os\_waiting\_tasks AS wt          JOIN sys.dm\_exec\_sessions AS s ON wt.session\_id = s.session\_id  WHERE   s.is\_user\_process = 0 |

Listing 1.1: Discovering system session waits.

When looking at the wait statistics being tracked by SQL Server, it’s important that these wait types are eliminated from the analysis, allowing the more problematic waits in the system to be identified. One of the things I do as a part of tracking wait information is to maintain a script that filters out the non-problematic wait types, as shown in Listing 1.2.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | SELECT TOP 10          wait\_type ,          max\_wait\_time\_ms wait\_time\_ms ,          signal\_wait\_time\_ms ,          wait\_time\_ms - signal\_wait\_time\_ms AS resource\_wait\_time\_ms ,          100.0 \* wait\_time\_ms / SUM(wait\_time\_ms) OVER ( )                                      AS percent\_total\_waits ,          100.0 \* signal\_wait\_time\_ms / SUM(signal\_wait\_time\_ms) OVER ( )                                      AS percent\_total\_signal\_waits ,          100.0 \* ( wait\_time\_ms - signal\_wait\_time\_ms )          / SUM(wait\_time\_ms) OVER ( ) AS percent\_total\_resource\_waits  FROM    sys.dm\_os\_wait\_stats  WHERE   wait\_time\_ms > 0 -- remove zero wait\_time          AND wait\_type NOT IN -- filter out additional irrelevant waits  ( 'SLEEP\_TASK', 'BROKER\_TASK\_STOP', 'BROKER\_TO\_FLUSH',    'SQLTRACE\_BUFFER\_FLUSH','CLR\_AUTO\_EVENT', 'CLR\_MANUAL\_EVENT',    'LAZYWRITER\_SLEEP', 'SLEEP\_SYSTEMTASK', 'SLEEP\_BPOOL\_FLUSH',    'BROKER\_EVENTHANDLER', 'XE\_DISPATCHER\_WAIT', 'FT\_IFTSHC\_MUTEX',    'CHECKPOINT\_QUEUE', 'FT\_IFTS\_SCHEDULER\_IDLE\_WAIT',    'BROKER\_TRANSMITTER', 'FT\_IFTSHC\_MUTEX', 'KSOURCE\_WAKEUP',    'LAZYWRITER\_SLEEP', 'LOGMGR\_QUEUE', 'ONDEMAND\_TASK\_QUEUE',    'REQUEST\_FOR\_DEADLOCK\_SEARCH', 'XE\_TIMER\_EVENT', 'BAD\_PAGE\_PROCESS',    'DBMIRROR\_EVENTS\_QUEUE', 'BROKER\_RECEIVE\_WAITFOR',    'PREEMPTIVE\_OS\_GETPROCADDRESS', 'PREEMPTIVE\_OS\_AUTHENTICATIONOPS',    'WAITFOR', 'DISPATCHER\_QUEUE\_SEMAPHORE', 'XE\_DISPATCHER\_JOIN',    'RESOURCE\_QUEUE' )  ORDER BY wait\_time\_ms DESC |

Listing 1.2: Finding the top ten cumulative wait events.

In general, when examining wait statistics, I focus on the top waits, according to wait\_time\_ms, and look out for high wait times associated with the following specific wait types:

* **CXPACKET**
  + Often indicates nothing more than that certain queries are executing with parallelism; CXPACKET waits in the server are not an immediate sign of problems, although they may be the symptom of another problem, associated with one of the other high value wait types in the instance.
* **SOS\_SCHEDULER\_YIELD**
  + The tasks executing in the system are yielding the scheduler, having exceeded their quantum, and are having to wait in the runnable queue for other tasks to execute. This may indicate that the server is under CPU pressure.
* **THREADPOOL**
  + A task had to wait to have a worker bound to it, in order to execute. This could be a sign of worker thread starvation, requiring an increase in the number of CPUs in the server, to handle a highly concurrent workload, or it can be a sign of blocking, resulting in a large number of parallel tasks consuming the worker threads for long periods.
* **LCK\_\***
  + These wait types signify that blocking is occurring in the system and that sessions have had to wait to acquire a lock of a specific type, which was being held by another database session. This problem can be investigated further using, for example, the information in the sys.dm\_db\_index\_operational\_stats.
* **PAGEIOLATCH\_\*, IO\_COMPLETION, WRITELOG**
  + These waits are commonly associated with disk I/O bottlenecks, though the root cause of the problem may be, and commonly is, a poorly performing query that is consuming excessive amounts of memory in the server. PAGEIOLATCH\_\* waits are specifically associated with delays in being able to read or write data from the database files. WRITELOG waits are related to issues with writing to log files. These waits should be evaluated in conjunction with the virtual file statistics as well as Physical Disk performance counters, to determine if the problem is specific to a single database, file, or disk, or is instance wide.
* **PAGELATCH\_\***
  + Non-I/O waits for latches on data pages in the buffer pool. A lot of times PAGELATCH\_\* waits are associated with allocation contention issues. One of the best-known allocations issues associated with PAGELATCH\_\* waits occurs in tempdb when the a large number of objects are being created and destroyed in tempdb and the system experiences contention on the Shared Global Allocation Map (SGAM), Global Allocation Map (GAM), and Page Free Space (PFS) pages in the tempdb database.
* **LATCH\_\***
  + These waits are associated with lightweight short-term synchronization objects that are used to protect access to internal caches, but not the buffer cache. These waits can indicate a range of problems, depending on the latch type. Determining the specific latch class that has the most accumulated wait time associated with it can be found by querying the sys.dm\_os\_latch\_stats DMV.
* **ASYNC\_NETWORK\_IO**
  + This wait is often incorrectly attributed to a network bottleneck. In fact, the most common cause of this wait is a client application that is performing row-by-row processing of the data being streamed from SQL Server as a result set (client accepts one row, processes, accepts next row, and so on). Correcting this wait type generally requires changing the client-side code so that it reads the result set as fast as possible, and then performs processing.

These basic explanations of each of the major wait types won’t make you an expert on wait type analysis, but the appearance of any of these wait types high up in the output of Listing 1.2 will certainly help direct your subsequent investigations. For example, if you see PAGEIOLATCH\_\* waits you will probably want to make your next focus the virtual file stats, as explained in the previous example.

Conversely, if the primary wait types in the system are LCK\_\* waits, then you won’t want to waste time looking at the disk I/O configuration, but instead focus on discovering what might be causing blocking inside the databases on the server. When LCK\_\* wait types crop up, I tend to jump immediately into more advanced troubleshooting of that specific problem, and begin looking at blocking in the system using the sys.dm\_exec\_requests DMV, and other methods, rather than strictly adhering to my normal methodology. However I may, depending on what I find, double back to see what other problems are in the system.

After fixing any problem in the server, in order to validate that the problem has indeed been fixed, the wait statistics being tracked by the server can be reset using the code in Listing 1.3.

|  |  |
| --- | --- |
| 1 | DBCC SQLPERF('sys.dm\_os\_wait\_stats', clear) |

Listing 1.3: Clearing the wait statistics on a server.

One of the caveats associated with clearing the wait statistics on the server, is that it will take a period of time for the wait statistics to accumulate to the point that you know whether or not a specific problem has been addressed.

Virtual File Statistics

A common trap in my experience, when using wait statistics as a primary source of troubleshooting data, is that most SQL Servers will demonstrate signs of what looks like a disk I/O bottleneck. Unfortunately, the wait statistics don’t tell you what is causing the I/O to occur, and it’s easy to misdiagnose the root cause.

This is why an examination of the virtual file statistics, alongside the wait statistics, is almost always recommended. The virtual file statistics are exposed through the sys.dm\_io\_virtual\_file\_stats function which, when passed a file\_id (and possibly database\_id), will provide cumulative physical I/O statistics, the number of reads and writes on each data file, and the number of reads and writes on each log file, for the various databases in the instance, from which can be calculated the ratio of reads to writes. This also shows the number of I/O stalls and the stall time associated with the requests, which is the total amount of time sessions have waited for I/O to be completed on the file.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | SELECT  DB\_NAME(vfs.database\_id) AS database\_name ,          vfs.database\_id ,          vfs.FILE\_ID ,          io\_stall\_read\_ms / NULLIF(num\_of\_reads, 0) AS avg\_read\_latency ,          io\_stall\_write\_ms / NULLIF(num\_of\_writes, 0)                                                 AS avg\_write\_latency ,          io\_stall / NULLIF(num\_of\_reads + num\_of\_writes, 0)                                                 AS avg\_total\_latency ,          num\_of\_bytes\_read / NULLIF(num\_of\_reads, 0)                                                 AS avg\_bytes\_per\_read ,          num\_of\_bytes\_written / NULLIF(num\_of\_writes, 0)                                                 AS avg\_bytes\_per\_write ,          vfs.io\_stall ,          vfs.num\_of\_reads ,          vfs.num\_of\_bytes\_read ,          vfs.io\_stall\_read\_ms ,          vfs.num\_of\_writes ,          vfs.num\_of\_bytes\_written ,          vfs.io\_stall\_write\_ms ,          size\_on\_disk\_bytes / 1024 / 1024. AS size\_on\_disk\_mbytes ,          physical\_name  FROM    sys.dm\_io\_virtual\_file\_stats(NULL, NULL) AS vfs          JOIN sys.master\_files AS mf ON vfs.database\_id = mf.database\_id                                         AND vfs.FILE\_ID = mf.FILE\_ID  ORDER BY avg\_total\_latency DESC |

Listing 1.4: Virtual file statistics.

What I’m primarily looking at in the results is patterns of activity on the file, whether heavy-read or heavy-write, and at the average latency associated with the I/O, as this will direct further investigation and possible solutions.

If the data and log files are on a shared disk array in the server, and the calculated avg\_total\_latency is the same across all of the databases, and higher than what is acceptable for the specific workload, then the problem may be that the workload has outgrown the disk I/O subsystem.

However, if the server hosts a database that is used for archiving data to slower storage, for year-on-year reporting, then it may be that having PAGEIOLATCH\_\* waits in the database is entirely normal, and the io\_stall information for the specific database files may lead us to determine that the waits are most likely attributable to the archiving process. This highlights the fact that it helps to have a sound knowledge of the underlying configuration and type of workload for the server, while you’re troubleshooting the problem.

If a particular file is subject to very heavy read activity (for example a ratio of 10:1, or higher, for the read:write ratio), and is showing high average latency, then I may recommend a RAID change for the disk array, for example from RAID 10 to RAID 5, offering more spindles to share the read I/O.

Hopefully, this discussion has highlighted the key element of effective troubleshooting, which is that you need to examine many “data points” together, in order to arrive at a true diagnosis. The discovery of I/O pressure, revealed by high I/O-related waits, could be caused by inadequate capacity or configuration of the disk subsystem, but its root cause is actually more likely to lie elsewhere, such as in a memory bottleneck in the buffer pool, or excessive index and/or table scans due to poorly written queries (covered in the *Plan Cache Usage* section) and a lack of indexing.

Performance Counters

Many articles, white papers, and blog posts on the Internet attempt to provide detailed lists of the important performance counters that should be monitored for SQL Server instances, along with general guidelines for acceptable values for these counters. However, if you try to collect and analyze the values for all of the available counters, you’ll quickly find it an overwhelming task.

Personally, at least in the initial stages of my investigation, I rely on a small subset of counters, directly related to SQL Server. At a more advanced stage in the troubleshooting process, I may also begin collecting Windows counters, in order to verify the information that I already have, or to help isolate an edge case problem to a specific cause.

One of my favorite tools, when I get to the point that I need to collect a larger subset of counters, collecting information from Windows as well as SQL Server, is the **Performance Analysis of Logs** (PAL) tool, which has been made available by Microsoft for free on [http://pal.codeplex.com](http://pal.codeplex.com/).

The tool provides built-in templates that can be exported to create a **Performance Collector Set** in Windows, each set containing the key counters for a specific product. It includes a template for SQL Server 2005 and 2008. The greatest benefit of this tool is that it also has built-in threshold templates that can be used to process the performance counter data after it has been collected. These can be used to produce a detailed report, breaking down the data into time slices and so automating the analysis of the data into periods of time and activity. If you want to know more about all of the counters related to SQL Server performance, what they mean, and what Microsoft currently says the threshold values for those counters are, I would recommend downloading the tool and taking a look at all the information contained in the SQL Server threshold file.

Nevertheless, the counters I investigate initially are limited to those related to specific areas of SQL Server, and are ones that have proven themselves over the years to provide information critical to determining how to continue with the troubleshooting process. The counters are all available from within SQL Server through the sys.dm\_os\_performance\_counters DMV and can be queried using T-SQL alone.

One of the challenges with querying the raw performance counter data directly is that some of the performance counters are cumulative ones, increasing in value as time progresses, and analysis of the data requires capturing two snapshots of the data and then calculating the difference between the snapshots. The query in Listing 1.5 performs the snapshots and calculations automatically, allowing the output to be analyzed directly. There are other performance counters, not considered in Listing 1.5, which have a secondary, associated base counter by which the main counter has to be divided to arrive at its actual value.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124 | DECLARE @CounterPrefix NVARCHAR(30)  SET @CounterPrefix = CASE      WHEN @@SERVICENAME = 'MSSQLSERVER'      THEN 'SQLServer:'      ELSE 'MSSQL$'+@@SERVICENAME+':'      END;      -- Capture the first counter set  SELECT CAST(1 AS INT) AS collection\_instance ,        [OBJECT\_NAME] ,        counter\_name ,        instance\_name ,        cntr\_value ,        cntr\_type ,        CURRENT\_TIMESTAMP AS collection\_time  INTO #perf\_counters\_init  FROM sys.dm\_os\_performance\_counters  WHERE ( OBJECT\_NAME = @CounterPrefix+'Access Methods'           AND counter\_name = 'Full Scans/sec'        )        OR ( OBJECT\_NAME = @CounterPrefix+'Access Methods'             AND counter\_name = 'Index Searches/sec'        )        OR ( OBJECT\_NAME = @CounterPrefix+'Buffer Manager'             AND counter\_name = 'Lazy Writes/sec'        )        OR ( OBJECT\_NAME = @CounterPrefix+'Buffer Manager'        AND counter\_name = 'Page life expectancy'        )        OR ( OBJECT\_NAME = @CounterPrefix+'General Statistics'             AND counter\_name = 'Processes Blocked'        )        OR ( OBJECT\_NAME = @CounterPrefix+'General Statistics'             AND counter\_name = 'User Connections'        )        OR ( OBJECT\_NAME = @CounterPrefix+'Locks'             AND counter\_name = 'Lock Waits/sec'        )        OR ( OBJECT\_NAME = @CounterPrefix+'Locks'             AND counter\_name = 'Lock Wait Time (ms)'        )        OR ( OBJECT\_NAME = @CounterPrefix+'SQL Statistics'             AND counter\_name = 'SQL Re-Compilations/sec'        )        OR ( OBJECT\_NAME = @CounterPrefix+'Memory Manager'             AND counter\_name = 'Memory Grants Pending'        )        OR ( OBJECT\_NAME = @CounterPrefix+'SQL Statistics'             AND counter\_name = 'Batch Requests/sec'        )        OR ( OBJECT\_NAME = @CounterPrefix+'SQL Statistics'             AND counter\_name = 'SQL Compilations/sec'  )    -- Wait on Second between data collection  WAITFOR DELAY '00:00:01'    -- Capture the second counter set  SELECT CAST(2 AS INT) AS collection\_instance ,         OBJECT\_NAME ,         counter\_name ,         instance\_name ,         cntr\_value ,         cntr\_type ,         CURRENT\_TIMESTAMP AS collection\_time  INTO #perf\_counters\_second  FROM sys.dm\_os\_performance\_counters  WHERE ( OBJECT\_NAME = @CounterPrefix+'Access Methods'        AND counter\_name = 'Full Scans/sec'        )        OR ( OBJECT\_NAME = @CounterPrefix+'Access Methods'             AND counter\_name = 'Index Searches/sec'        )        OR ( OBJECT\_NAME = @CounterPrefix+'Buffer Manager'             AND counter\_name = 'Lazy Writes/sec'        )        OR ( OBJECT\_NAME = @CounterPrefix+'Buffer Manager'             AND counter\_name = 'Page life expectancy'        )        OR ( OBJECT\_NAME = @CounterPrefix+'General Statistics'             AND counter\_name = 'Processes Blocked'        )        OR ( OBJECT\_NAME = @CounterPrefix+'General Statistics'             AND counter\_name = 'User Connections'        )        OR ( OBJECT\_NAME = @CounterPrefix+'Locks'             AND counter\_name = 'Lock Waits/sec'        )        OR ( OBJECT\_NAME = @CounterPrefix+'Locks'             AND counter\_name = 'Lock Wait Time (ms)'        )        OR ( OBJECT\_NAME = @CounterPrefix+'SQL Statistics'             AND counter\_name = 'SQL Re-Compilations/sec'        )        OR ( OBJECT\_NAME = @CounterPrefix+'Memory Manager'             AND counter\_name = 'Memory Grants Pending'        )        OR ( OBJECT\_NAME = @CounterPrefix+'SQL Statistics'             AND counter\_name = 'Batch Requests/sec'        )        OR ( OBJECT\_NAME = @CounterPrefix+'SQL Statistics'             AND counter\_name = 'SQL Compilations/sec'  )    -- Calculate the cumulative counter values  SELECT  i.OBJECT\_NAME ,          i.counter\_name ,          i.instance\_name ,          CASE WHEN i.cntr\_type = 272696576            THEN s.cntr\_value - i.cntr\_value            WHEN i.cntr\_type = 65792 THEN s.cntr\_value          END AS cntr\_value  FROM #perf\_counters\_init AS i    JOIN  #perf\_counters\_second AS s      ON i.collection\_instance + 1 = s.collection\_instance        AND i.OBJECT\_NAME = s.OBJECT\_NAME        AND i.counter\_name = s.counter\_name        AND i.instance\_name = s.instance\_name  ORDER BY OBJECT\_NAME    -- Cleanup tables  DROP TABLE #perf\_counters\_init  DROP TABLE #perf\_counters\_second |

Listing 1.5: SQL Server performance counters.

The performance counters collected by this script are:

* **SQLServer:Access Methods\Full Scans/sec**
* **SQLServer:Access Methods\Index Searches/sec**
* **SQLServer:Buffer Manager\Lazy Writes/sec**
* **SQLServer:Buffer Manager\Page life expectancy**
* **SQLServer:Buffer Manager\Free list stalls/sec**
* **SQLServer:General Statistics\Processes Blocked**
* **SQLServer:General Statistics\User Connections**
* **SQLServer:Locks\Lock Waits/sec**
* **SQLServer:Locks\Lock Wait Time (ms)**
* **SQLServer:Memory Manager\Memory Grants Pending**
* **SQLServer:SQL Statistics\Batch Requests/sec**
* **SQLServer:SQL Statistics\SQL Compilations/sec**
* **SQLServer:SQL Statistics\SQL Re-Compilations/sec**

The two **Access Methods** counters provide information about the ways that tables are being accessed in the database. The most important one is the Full Scans/sec counter, which can give us an idea of the number of index and table scans that are occurring in the system.

If the disk I/O subsystem is the bottleneck (which, remember, is most often caused by pressure placed on it by a problem elsewhere) and this counter is showing that there are scans occurring, it may be a sign that there are missing indexes, or inefficient code in the database. How many scans are problematic? It depends entirely on the size of the objects being scanned and the type of workload being run. In general, I want the number of Index Searches/sec to be higher than the number of Full Scans/sec by a factor of 800-1000. If the number of Full Scans/sec is too high, I use the [Database Engine Tuning Advisor](http://msdn.microsoft.com/en-us/library/ms173494.aspx) (DTA) or the [Missing Indexes](http://msdn.microsoft.com/en-us/library/ms345524.aspx) feature to determine if there are missing indexes in the database, resulting in excess I/O operations.

The **Buffer Manager** and **Memory Manager** counters can be used, as a group, to identify if SQL Server is experiencing memory pressure. The values of the Page Life Expectancy, Free List Stalls/sec, and Lazy Writes/sec counters, when correlated, will validate or disprove the theory that the buffer cache is under memory pressure.

A lot of online references will tell you that if the Page Life Expectancy (PLE) performance counter drops lower than 300, which is the number of seconds a page will remain in the data cache, then you have memory pressure. However, this guideline value for the PLE counter was set at a time when most SQL Servers only had 4 GB of RAM, and the data cache portion of the buffer pool was generally 1.6 GB. In modern servers, where it is common for SQL Servers to have 32 GB or more of installed RAM, and a significantly larger data cache, having 1.6 GB of data churn through that cache every 5 minutes is not necessarily a significant event.

In short, the appropriate value for this counter depends on the size of the SQL Server data cache, and a fixed value of 300 no longer applies. Instead, I evaluate the value for the PLE counter based on the installed memory in the server. To do this, I take the base counter value of 300 presented by most resources, and then determine a multiple of this value based on the configured buffer cache size, which is the ‘max server memory’ sp\_configure option in SQL Server, divided by 4 GB. So, for a server with 32 GB allocated to the buffer pool, the PLE value should be at least (32/4)\*300 = 2400.

If the PLE is consistently below this value, and the server is experiencing high Lazy Writes/sec, which are page flushes from the buffer cache outside of the normal CHECKPOINT process, then the server is most likely experiencing data cache memory pressure, which will also increase the disk I/O being performed by the SQL Server. At this point, the Access Methods counters should be investigated to determine if excessive table or index scans are being performed on the SQL Server.

The General Statistics\Processes Blocked, Locks\Lock Waits/sec, and Locks\Lock Wait Time (ms) counters provide information about blocking in the SQL Server instance, at the time of the data collection. If these counters return a value other than zero, over repeated collections of the data, then blocking is actively occurring in one of the databases and we can use tools such as the Blocked Process Report in SQL Trace, or the sys.dm\_exec\_requests, sys.dm\_exec\_sessions and sys.dm\_os\_waiting\_tasks DMVs to troubleshoot the problems further.

The three SQL Statistics counters provide information about how frequently SQL Server is compiling or recompiling an execution plan, in relation to the number of batches being executed against the server. The higher the number of SQL Compilations/sec in relation to the Batch Requests/sec, the more likely the SQL Server is experiencing an ad hoc workload that is not making optimal using of plan caching. The higher the number of SQL Re-Compilations/sec in relation to the Batch Requests/sec, the more likely it is that there is an inefficiency in the code design that is forcing a recompile of the code being executed in the SQL Server. In either case, investigation of the Plan Cache, as detailed in the next section, should identify why the server has to consistently compile execution plans for the workload.

The Memory Manager\Memory Grants Pending performance counter provides information about the number of processes waiting on a workspace memory grant in the instance. If this counter has a high value, SQL Server may benefit from additional memory, but there may be query inefficiencies in the instance that are causing excessive memory grant requirements, for example, large sorts or hashes that can be resolved by tuning the indexing or queries being executed.

Plan Cache Usage

In my experience, the Plan Cache in SQL Server 2005 and 2008 is one of the most underused assets for troubleshooting performance problems in SQL Server. As a part of the normal execution of batches and queries, SQL Server tracks the accumulated execution information for each of the plans that is stored inside of the plan cache, up to the point where the plan is flushed from the cache as a result of DDL operations, memory pressure, or general cache maintenance. The execution information stored inside of the plan cache can be found in the sys.dm\_exec\_query\_stats DMV as shown in the example query in Listing 1.6. This query will list the top ten statements based on the average number of physical reads that the statements performed as a part of their execution.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | SELECT TOP 10          execution\_count ,          statement\_start\_offset AS stmt\_start\_offset ,          sql\_handle ,          plan\_handle ,          total\_logical\_reads / execution\_count AS avg\_logical\_reads ,          total\_logical\_writes / execution\_count AS avg\_logical\_writes ,          total\_physical\_reads / execution\_count AS avg\_physical\_reads ,          t.TEXT  FROM    sys.dm\_exec\_query\_stats AS s          CROSS APPLY sys.dm\_exec\_sql\_text(s.sql\_handle) AS t  ORDER BY avg\_physical\_reads DESC |

Listing 1.6: SQL Server execution statistics.

The information stored in the plan cache can be used to identify the most expensive queries based on physical I/O operations for reads and for writes, or based on different criteria, depending on the most problematic type of I/O for the instance, discovered as a result of previous analysis of the wait statistics and virtual file statistics.

Additionally, the sys.dm\_exec\_query\_plan() function can be cross-applied using the plan\_handle column from the sys.dm\_exec\_query\_stats DMV to get the execution plan that is stored in the plan cache. By analyzing these plans, we can identify problematic operations that are candidates for performance tuning.

**Query performance tuning:**

A full discussion of query performance tuning is beyond the scope of this book. In fact, several notable books have been written on this topic alone, including [*“SQL Server 2008 Query Performance Tuning Distilled”*](http://www.amazon.com/Server-Performance-Tuning-Distilled-Experts/dp/1430219025) and ([*“Inside Microsoft SQL Server 2008: T-SQL Querying”*](http://www.amazon.com/Inside-Microsoft%C3%82%C2%AE-SQL-Server%C3%82%C2%AE-2008/dp/0735626030).

The information in the sys.dm\_exec\_query\_stats DMV can also be used to identify the statements that have taken the most CPU time, the longest execution time, or that have been executed the most frequently.

In SQL Server 2008, two additional columns, query\_hash and query\_plan\_hash, were added to the sys.dm\_exec\_query\_stats DMV. The query\_hash is a hash over the statement text to allow similar statements to be aggregated together. The query\_plan\_hash is a hash of the query plan shape that allows queries with similar execution plans to be aggregated together. Together, they allow the information contained in this DMV to be aggregated for ad hoc workloads, in order to determine the total impact of similar statements that have different compiled literal values.

Summary

This article has outlined my basic approach to investigating performance problems in SQL Server. This approach is more or less the same, regardless of whether it is a server I know well, or one I’m investigating for the first time, with no prior knowledge of the health and configuration of the SQL Server instance it houses. Based on the information gathered using this methodology, more advanced diagnosis of the identified problem areas can be performed.

The most important point that I want to stress is that no single piece of information in SQL Server should be used to pinpoint any specific problem. The art of taming an unruly SQL Server is the art of assembling the various pieces of the puzzle so that you have a complete understanding of what is going on inside of a server. If you focus only on what is immediately in front of you, you will, in most cases, miss the most important item, which is the true root cause of a particular problem in SQL Server.

This article is adapted, with kind permission of the authors, from the book [“Troubleshooting SQL Server: A guide for the Accidental DBA”](http://www.sqlservercentral.com/articles/books/76296/). It describes in depth the tools and practical techniques for troubleshooting many of the most common causes of SQL Server problems, including high CPU usage, memory mismanagement, missing indexes, blocking, deadlocking, full transactions logs and accidentally-lost data.

For more articles like this, [sign up to the fortnightly Simple-Talk newsletter](https://www.simple-talk.com/account/sign-up).

**Tags:** [Performance](https://www.simple-talk.com/tag/performance/), [Performance Counters](https://www.simple-talk.com/tag/performance-counters/), [SQL](https://www.simple-talk.com/tag/sql/), [SQL Home](https://www.simple-talk.com/tag/sql-home/), [SQL Server](https://www.simple-talk.com/tag/sql-server/), [Troubleshooting](https://www.simple-talk.com/tag/troubleshooting/), [Wait Statistics](https://www.simple-talk.com/tag/wait-statistics/)

* **144915** views
* Rate

[Total: 192    Average: 4.6/5]

* + [Printer friendly version](https://www.simple-talk.com/sql/performance/a-performance-troubleshooting-methodology-for-sql-server/?view=print)

[](https://www.simple-talk.com/author/jonathan-kehayias/)

[Jonathan Kehayias](https://www.simple-talk.com/author/jonathan-kehayias/)

Jonathan Kehayias is currently employed as a Principal Consultant and Trainer for SQLskills, one of the best-known and most respected SQL Server training and consulting companies in the world. Jonathan is a SQL Server MVP and one of the few Microsoft Certified Masters for SQL Server 2008, outside of Microsoft. Jonathan frequently blogs about SQL Server, presents sessions at PASS Summit, SQLBits, SQL Connections and local SQL Saturday events, and has remained a top answerer of questions on the MSDN SQL Server Database Engine forum since 2007. Jonathan is a performance tuning expert for both SQL Server and hardware, and has architected complex systems as a developer, business analyst, and DBA. He also has extensive development (T-SQL, C#, and ASP.Net), hardware and virtualization design expertise, Windows expertise, Active Directory experience, and IIS administration experience. Outside of SQL Server, Jonathan is also a Drill Sergeant in the US Army Reserves and is married with two young children. On most nights he can be found at the playground, in a swimming pool, or at the beach with his kids. Jonathan can be found online as [@SQLPoolBoy](https://www.twitter.com/SQLPoolBoy) on Twitter, or through his blog (<http://sqlskills.com/blogs/jonathan>)

[View all articles by Jonathan Kehayias](https://www.simple-talk.com/author/jonathan-kehayias/)

* *Yagnesh*

About the book  
Hi Jonathan,

It is really a good chapter and I am very excited about your book.

I am really looking forward to have this.

* *Robert Matthew Cook*

Nothing short of amazing  
Thank you so much Jonathan! This article may be the best thing I have read all year.

* *Andreas Etelkozi*

Very good, clear article  
Jonathan,

Thank you for a superb article!  
For sure I will invest in your book!

I highly respect your publications and user dialogues, I can always learn a lot from those, and when not learning then organizing my understanding and knowledge!

Thank you,

Andreas

* *Thomas Pullen*

thanks for a  
.. superb article!

* *Mario Broodbakker*

virtual file stats  
great article, I’ve just one remark though. I believe (and verified by comparing with timing from filemon and wait time from the os\_waitstats), that the iostall time is not the ‘total amount of time sessions have waited for I/O to be completed on the file’ as you mention in your article.  
But it’s the time it took to complete the reads/writes themselves: io time.  
Which in fact makes this view an interesting addition next to the session wait stats..

* *Mario Broodbakker*

SLEEP\_TASK  
..and while I am at it..(sorry)  
Sleep\_task is not so ‘irrelevant’ as it sounds..(it’s in the ‘irrelevant’ filter of the waitstats query)  
Unfortunately it’s a bit of a missused ‘wait’ by the sql server devs. Sometimes it’s indeed not relevant for performance analysis, but there are occasions that it can point to waits for hash and sort operations, and has a direct relation to your query response time.  
Ramesh Meyyappan from sqlworkshops.com shows it very well in one of his webcasts (number 3)

* *Jonathan Kehayias*

RE: virtual file stats  
See my blog post that shows using Extended Events that not all I/O operations incur wait\_time or have a wait type set for them that would be tracked in sys.dm\_os\_wait\_stats.

<http://sqlskills.com/blogs/jonathan/post/TSQL-Tuesday-11-Physical-IOe28099s-Done28099t-Always-Accumulate-Wait-Times.aspx>

If you compare the output of the query provided for virtual file stats with your I/O latency averages from PerfMon you will see that they are nearly identical.

* *Jonathan Kehayias*

RE: SLEEP\_TASK  
SLEEP\_TASK waits can occur for a number of problems in the system that is very true, but it is not a resource wait, which is what we are focusing on as stated in the paragraph preceding the code example. I am aware of Ramesh’s videos, I am one of the people he sarcastically refers to as an “expert” when talking about SLEEP\_TASK, of course out of context for the example he showed which was focused towards resource waits as well.

If you have large hash or sort spills that write through tempdb, then SLEEP\_TASK would have significance, but as a non-resource type of wait and a sign of problems in code, indexing, or design. There are other waits that could be considered important in the list that are being filtered as well, just not from a resource standpoint. There are a few people that would add CXPACKET to the list because it isn’t a problematic wait\_type either, it is a symptom of another problem as well.

* *Marios Philippopoulos*

Comment on the Virtual-File Stats query  
Hi Jonathan,

Thank you for this article, very informative. I will certainly be getting your book as well!

One comment I had on the virtual-file stats query above, when you calculate the total avg latency for reads and writes:

…  
io\_stall\_write\_ms / NULLIF(num\_of\_writes + num\_of\_writes, 0) AS avg\_total\_latency ,  
…

That should be:

…  
(io\_stall\_read\_ms + io\_stall\_write\_ms) / NULLIF(num\_of\_reads + num\_of\_writes, 0) AS avg\_total\_latency ,  
…

Thanks for all the online articles and blogs you have posted over the years, I have found them very useful in my day-to-day work.

Regards,  
Marios Philippopoulos

* *Marios Philippopoulos*

Comment on the Virtual-File Stats query  
Hi Jonathan,

Thank you for this article, very informative. I will certainly be getting your book as well!

One comment I had on the virtual-file stats query above, when you calculate the total avg latency for reads and writes:

…  
io\_stall\_write\_ms / NULLIF(num\_of\_writes + num\_of\_writes, 0) AS avg\_total\_latency ,  
…

That should be:

…  
(io\_stall\_read\_ms + io\_stall\_write\_ms) / NULLIF(num\_of\_reads + num\_of\_writes, 0) AS avg\_total\_latency ,  
…

Thanks for all the online articles and blogs you have posted over the years, I have found them very useful in my day-to-day work.

Regards,  
Marios Philippopoulos

* *Mario Broodbakker*

(re) RE: virtual file stats  
Jonathan, thanks for the link to your blog post, I will further comment on there. But you say in your article: “This also shows the number of I/O stalls and the stall time associated with the requests, which is the total amount of time sessions have waited for I/O to be completed on the file.”

The latter is simply not the case, as you pointed out yourself: it’s the IO time, which is (most of the time) different from session pageio wait time.  
Simply because sessions don’t have to wait for all pageio to be finished in order to continue to work.

* *Mario Broodbakker*

(re) RE: Sleep\_task  
that’s an interesting viewpoint.  
Your paper (and query) is about a ‘Performance troubleshooting methodology’ right? Not about ‘resource or non-resource’ waits. As long as as ‘wait’ has a direct impact on your response time it’s not a wait that should be disregarded as irrelevant I would think.

* *Jonathan Kehayias*

(re) RE: virtual file stats  
Marios,

The definition of io\_stall comes from the Books Online <http://msdn.microsoft.com/en-us/library/ms190326.aspx> and is consistent with the other Microsoft white papers that define what this value means. Not every IO is going to stall and not every IO is going to cause user waits to accumulate that is true.

This article is an adaptation of Chapter 1 from the book that I wrote with SimpleTalk that is currently being printed. The book is about troubleshooting the most common problems with SQL Server as a DBA, it isn’t about tuning code. From a server standpoint, SLEEP\_TASK is never the root cause of the major performance problems in a SQL Server, and neither are the other waits that have been filtered out as irrelevant in the query. Perhaps the article should have been adapted to better frame the intent of the information as being a chapter from the upcoming book and what the purpose of the book is. Have a look at the description of the book, which is in the link at the bottom of the article and the information and method being described will make more sense.

* *Joe Noll*

Thanks for sharing  
Jonathan,

Great article, thanks for sharing and congrats on the new book!

Your wife has to be a saint based on how busy you have been.

* *Dirk Hondong*

looking forward to the book  
Hi Jonathan,

what a great article and a good appetizer for your forthcoming book.  
I´m looking forward to it

* *Anonymous*

Great!  
Great! Thank you Jonathan.

* *Bekets*

Question  
Hi Jonathan,

I’ve really enjoyed reading this piece of work well demonstrated and written. Thanks for publishing it.  
I just have one question which is based on the following statement in your article and I quote;

"When I am examining a server for the first time, I need to establish a picture of its general health, and there are a number of items on which I’ll focus in order to obtain this picture. For each piece of information I collect, I’ll be examining it in relation to the previous data points, in order to validate or disprove any previous indicators as to the nature of the problem."

What about if you do not have data against which to compare data that you have collected? What can you do?  
This is quite common for example, you are asked to investigate performance issues on a server for which the previous DBA didn’t collect and set a baseline for measuring and comparing performance.

Many thanks,

Peter

Join Simple Talk

Join over 200,000 Microsoft professionals, and get full, free access to technical articles, our twice-monthly Simple Talk newsletter, and free SQL tools.

[Sign up](https://www.simple-talk.com/account/sign-up)

[Devops, Continuous Delivery & Database Lifecycle Management](https://www.simple-talk.com/collections/database-lifecycle-management-patterns-practices-library/)

Visit our library of articles to discover the patterns and practices you need to move towards more agile methods of database delivery.

Find out how to automate the process of building, testing and deploying your database changes to reduce risk and speed up the delivery cycle.

Top rated recent articles in Performance

* [Query Store and In-Memory OLTP](https://www.simple-talk.com/sql/database-administration/query-store-memory-oltp/)

by Enrico van de Laar

[**2**](https://www.simple-talk.com/sql/database-administration/query-store-memory-oltp/#comments)

* [SQL Server Statistics Basics](https://www.simple-talk.com/sql/performance/sql-server-statistics-basics/)

by Robert Sheldon

[**2**](https://www.simple-talk.com/sql/performance/sql-server-statistics-basics/#comments)

Related articles

Also in Performance

[T-SQL Window Function Speed Phreakery: The FIFO Stock Inventory Problem](https://www.simple-talk.com/sql/performance/t-sql-window-function-speed-phreakery-the-fifo-stock-inventory-problem/)

Sometimes, in the quest for raw SQL performance, you are forced to sacrifice legibility and maintainability of your code, unless you then document your code lavishly. Phil Factor's SQL Speed Phreak challenge produced some memorable code, but can SQL features introduced since then help to produce code that performs as well and is also easy to understand? Aunty Kathi investigates.… [Read more](https://www.simple-talk.com/sql/performance/t-sql-window-function-speed-phreakery-the-fifo-stock-inventory-problem/)

Also in SQL

[Relational Algebra and its implications for NoSQL databases](https://www.simple-talk.com/opinion/opinion-pieces/relational-algebra-and-its-implications-for-nosql-databases/)

With the rise of NoSQL databases that are exploiting aspects of SQL for querying, and are embracing full transactionality, is there a danger of the data-document model's hierarchical nature causing a fundamental conflict with relational theory? We asked our relational expert, Hugh Bin-Haad to expound a difficult area for database theorists.… [Read more](https://www.simple-talk.com/opinion/opinion-pieces/relational-algebra-and-its-implications-for-nosql-databases/)

Also in SQL Home

[Grant Fritchey's SQL Server Howlers](https://www.simple-talk.com/sql/database-administration/grant-fritcheys-sql-server-howlers/)

We decided to ask various well-known SQL Server people to write about their favourite SQL Server Howlers. These are the common misunderstandings about how SQL Server works that end in tears, and plaintive forum questions. Grant Fritchey opens the series with some of his favourite howlers.… [Read more](https://www.simple-talk.com/sql/database-administration/grant-fritcheys-sql-server-howlers/)

Also in SQL Server

[SQL Server System Functions: The Basics](https://www.simple-talk.com/sql/t-sql-programming/sql-server-system-functions-the-basics/)

Every SQL Server Database programmer needs to be familiar with the System Functions. These range from the sublime (such as @@rowcount or @@identity) to the ridiculous (IsNumeric()) Robert Sheldon provides an overview of the most commonly used of them.… [Read more](https://www.simple-talk.com/sql/t-sql-programming/sql-server-system-functions-the-basics/)