Database Tuning Efforts

# Monitoring

1. NOTE: All gathered statistics will be put into a tracking database and analytics will be gleaned from the same.
2. Which queries are the worst performing
   1. Daily
   2. Weekly
   3. Monthly
   4. sp\_PerfMonitor.sql
   5. PerformanceMonitorForProcs.sql
   6. findWorstPerformers.sql
   7. Record/Store retrieved stats into the Performance Monitoring Cube on a daily basis.
3. Which queries use the most CPU
4. Which queries use the most IO
5. What are the worst performing stored procedures
6. What are the heaviest hit tables
7. Are there any unused indexes
8. Are there any missing indexes
9. What are the worst counters on the server(s)
10. REMOVE OUTER JOINS
    1. This can be easier said than done and depends on how much influence you have in changing table content. One solution is to remove OUTER JOINS by placing placeholder rows in both tables. Say you have the following tables with an OUTER JOIN defined to ensure all data is returned:
    2. UTIL\_FindOuterJoins.sql
11. What PROCS and PROCESSES are causing BLOCKS
    1. UTIL\_MonitorDeadlocks.sql
    2. find\_Locks\_And\_Blocks.sql
12. What period of the day has the heaviest load
    1. What percentage is DB
    2. What percentage is CPU
    3. What percentage is IO
    4. What percentage is Paging
    5. What percentage is Memory Swap
    6. DURING THE HARD HIT PERIODS (refer to previous items)
       1. Which queries use the most CPU
       2. Which queries use the most IO
       3. What are the worst performing stored procedures
       4. What are the heaviest hit tables
       5. Are there any unused indexes
       6. Are there any missing indexes
       7. What are the worst counters on the server(s)
13. Table Growth History
    1. Stored in DFS\_TABLE\_GROWTH\_HISTORY
14. Table Read and Writes
    1. Stored in DFS\_TableReadWrites
15. Table Statistics
    1. Stored in DFS\_TableStats
16. Define Tables that merit review for:
    1. Partitioning
    2. Index partitioning
    3. Physical partitioning to a separate server
       1. Consider disk striping where appropriate
       2. Review:
          1. Vertical partitioning
          2. Horizontal partitioning
    4. memory\_used\_by\_table\_kb: For memory-optimized tables, the memory used in kilobytes, by table Resource: Utilization sys.dm\_db\_xtp\_table\_memory\_stats (Dynamic Management View)
       1. We want to make sure there’s enough room for our data, and SQL Server’s memory-optimized tables elevate memory to the same level of importance as storage. Memory-optimized tables in SQL Server 2016 and beyond can be of any size, as long as they fit within the limits of your system memory.
       2. It’s important to compare the size of your memory-optimized tables with the memory available on your system. Microsoft recommends maintaining enough system memory to accommodate twice the estimated size of the data and indexes within a memory-optimized table. This is not only because you need room for the indexes and data themselves, but also because memory-optimized tables enable concurrent reads and writes by storing several versions of a single row. Since memory-optimized tables can be as large as memory allows, it’s important to leave aside enough resources to support their growth.
       3. Memory-optimized tables are designed to accommodate high-throughput, low-latency transactions. You can monitor the rate of queries to in-memory tables alongside their resource use to see if your use case fits this profile.
    5. Disk usage Space used by data or by indexes in a given table Resource: Utilization sp\_spaceused (Stored Procedure)
       1. If the server is running out of disk space, it’s critical to get notified with enough lead time that you can take action. The stored procedure sp\_spaceused returns the disk usage for a specific table or database.
       2. As data nears capacity, we’ll want to think about the design of the storage. SQL Server lets us configure the way your tables use disk space. It’s possible, for instance, to distribute your data files across multiple disks and assign them to a logical unit, the filegroup. You can use T-SQL statements to declare a filegroup and associate files with it by path. When you declare a table, you can assign it to a filegroup. Queries to the table will read and write data to files in the filegroup. Since files in a filegroup can be local or remote, you can counteract limited system space by adding files from separate drives. And as SQL Server can access multiple disks at once, filegroups can improve performance.
17. Latches
    1. [Microsoft](http://technet.microsoft.com/en-us/library/aa224727(v=SQL.80).aspx) describes latches as “very lightweight, short-term synchronization objects protecting actions that need not be locked for the life of a transaction”. What is very important to remember about latches is that they are not locks! Even though they share some of the same types as locks (like SH or EX) they do not play a role in isolating you transaction. There are a lot of different type of latches SQL Server uses, you can look them up by querying the sys.dm\_os\_wait\_stats DMV for all Buffer related latches and sys.dm\_os\_latch\_stats for all non-buffer related latches:
    2. Measure the PAGEIOLATCH\_SH waits over history
    3. Measure the PAGEIOLATCH\_EX waits over history
    4. findWaitsAndLatches.sql
18. What objects are causing the highest Wait Statistics
    1. Why
    2. Proposed tuning
    3. 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.
19. Critical performance counters to collect:
    1. CLASS: Access Methods\Full Scans/sec
    2. CLASS: Access Methods\Index Searches/sec
    3. CLASS: Buffer Manager\Lazy Writes/sec
    4. CLASS: Buffer Manager\Page life expectancy
    5. CLASS: Buffer Manager\Free list stalls/sec
    6. CLASS: General Statistics\Processes Blocked
    7. CLASS: General Statistics\User Connections
    8. CLASS: Locks\Lock Waits/sec
    9. CLASS: Locks\Lock Wait Time (ms)
    10. CLASS: Memory Manager\Memory Grants Pending
    11. CLASS: SQL Statistics\Batch Requests/sec
    12. CLASS: SQL Statistics\SQL Compilations/sec
    13. CLASS: SQL Statistics\SQL Re-Compilations/sec
    14. HOW TO GATHER:
        1. UTIL\_GatherPerformanceStats.sql
        2. 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.
        3. 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.
20. 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 (DTA) or the Missing Indexes feature to determine if there are missing indexes in the database, resulting in excess I/O operations.
21. 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.
22. 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.
23. 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.
24. 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.
25. 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
    1. sys.dm\_exec\_requests, sys.dm\_exec\_sessions and
    2. sys.dm\_os\_waiting\_tasks DMVs to troubleshoot the problems further.
26. 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.
27. 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.
28. Monitor Plan cache Usage
    1. MonitorPlanCacheUsage.sql
29. Gather and store historical plan execution and use the performance indicators to determine where “shifts”, either positive or negative are taking place.
30. Find all recursive views
    1. RecursiveViewCTE.sql
31. Find and identify embedded sub-selects
32. Evaluate the use of joins
33. Evaluate the use of CTE
    1. FindCTE\_usage.sql
34. Evaluate the use of Table Variables
    1. FindTableVar\_usage.sql
35. Evaluate the use of Temp Tables
    1. Monitor TEMPDB activity
    2. Monitor TEMPDB utilization stats
36. Database growth
    1. Proc: UTIL\_MonitorDBGrowth
    2. UTIL\_DB\_Growth.sql
    3. Store: DFS\_DB\_GrowthData
37. Server/Database Workload
    1. UTIL\_MonitorWorkload.sql
    2. Store: DFS\_Workload
    3. DMV\_SQLServerWorkloadStats.sql
38. Table growth
    1. C:\Users\wdale\Documents\SQL Server Management Studio\proc\_TABLE\_GROWTH\_HISTORY.sql
    2. Store: DFS\_TableGrowthHistory
39. Individual table activity
    1. UTIL\_GetAllDBTblStats.sql
    2. Store: DFS\_TableStats
40. Monitor Index Fragmentation
    1. UTIL\_FindFragmentedIndexesAndGenRebuildSQL.sql
    2. RebuildIndexIfFragmented.sql
    3. sp\_FindDependencies.sql
41. When needed, locate all table dependencies for troubleshooting
    1. sp\_FindDependencies.sql
42. Required Jobs in DFSAnalytics
    1. JOB\_UTIL\_CPU\_BoundQry
    2. JOB\_UTIL\_IO\_BoundQry
    3. JOB\_UTIL\_MonitorWorkload
    4. JOB\_UTIL\_TxMonitorIDX
    5. JOB\_UTIL\_TxMonitorTableStats

## Memory Bottleneck Analysis

PERFMON

Object: - Memory

Counter: - Available Mbytes

Preferred Value: - > 20MB

Description: -

Reference: - KB 889654

Object: - Memory

Counter: - Free System Page Table Entries

Preferred Value: - > 7000

Description: - Free System Page Table Entries is the number of page table entries not currently in use by the system. If < 7000, consider removing /3GB.

Reference: - KB 311901

Object: - Memory

Counter: - Pages/Sec

Preferred Value: - < 50

Description: - Pages/sec is the rate at which pages are read from or written to disk to resolve hard page faults. This counter is a primary indicator of the kinds of faults that cause system-wide delays.

Reference: - Monitoring and Tuning Your Server

Object: - Memory

Counter: - Pages Input/Sec

Preferred Value: - < 10

Description: - Pages Input/sec is the rate at which pages are read from disk to resolve hard page faults.

Reference: - KB 889654

Object: - Paging File

Counter: - %Usage

Preferred Value: - < 70%

Description: - The amount of the Page File instance in use in percent.

Reference: - KB 889654

Object: - Paging File

Counter: - %Usage

Preferred Value: - < 70%

Description: - The peak usage of the Page File instance in percent.

Reference: - KB 889654

Object: - CLASS: Buffer Manager

Counter: - Page Life Expectancy

Preferred Value: - > 300

Description: - This performance monitor counter tells you, on average, how long data pages are staying in the buffer. If this value gets below 300 seconds, this is a potential indication that your SQL Server could use more memory in order to boost performance.

Reference: -

Object: - CLASS: Buffer Manager

Counter: - Lazy Writes/Sec

Preferred Value: - < 20

Description: - This counter tracks how many times a second that the Lazy Writer process is moving dirty pages from the buffer to disk in order to free up buffer space. Generally speaking, this should not be a high value, say more than 20 per second or so. Ideally, it should be close to zero. If it is zero, this indicates that your SQL Server's buffer cache is plenty big and SQL Server doesn't have to free up dirty pages, instead waiting for this to occur during regular checkpoints. If this value is high, then a need for more memory is indicated.

Reference: -

Object: - CLASS: Buffer Manager

Counter: - Checkpoint Pages/Sec

Preferred Value: - This value is relative, it varies from server to server, we need to compare the average to a base line capture to tell if the value is high or low.

Description: - When a checkpoint occurs, all dirty pages are written to disk. This is a normal procedure and will cause this counter to rise during the checkpoint process. What you don't want to see is a high value for this counter over time. This can indicate that the checkpoint process is running more often than it should, which can use up valuable server resources. If this has a high figure (and this will vary from server to server), consider adding more RAM to reduce how often the checkpoint occurs, or consider increasing the "recovery interval" SQL Server configuration setting.

Reference: -

Object: - CLASS: Buffer Manager

Counter: - Page reads/sec

Preferred Value: - < 90

Description: - Number of physical database page reads issued. 80 – 90 per second is normal, anything that is above indicates indexing or memory constraint.

Reference: -

Object: - CLASS: Buffer Manager

Counter: - Page writes/sec

Preferred Value: - < 90

Description: - Number of physical database page writes issued. 80 – 90 per second is normal, anything more we need to check the lazy writer/sec and checkpoint counters, if these counters are also relatively high then, it’s memory constraint.

Reference: -

Object: - CLASS: Buffer Manager

Counter: - Free pages

Preferred Value: - > 640

Description: - Total number of pages on all free lists.

Reference: -

Object: - CLASS: Buffer Manager

Counter: - Stolen pages

Preferred Value: - Varies. Compare with baseline

Description: - Number of pages used for miscellaneous server purposes (including procedure cache).

Reference: -

Object: - CLASS: Buffer Manager

Counter: - Buffer Cache hit ratio

Preferred Value: - > 90%

Description: - Percentage of pages that were found in the buffer pool without having to incur a read from disk.

Reference: -

Object: - CLASS: Buffer Manager

Counter: - Target Server Memory(KB)

Preferred Value: -

Description: - Total amount of dynamic memory the server can consume.

Reference: -

Object: - CLASS: Buffer Manager

Counter: - Total Server Memory(KB)

Preferred Value: -

Description: - Total amount of dynamic memory (in kilobytes) that the server is using currently

Reference: -

Disk Bottleneck Analysis

Object: - PhysicalDisk

Counter: - Avg. Disk Sec/Read

Preferred Value: - < 8ms

Description: - Measure of disk latgency. Avg. Disk sec/Read is the average time, in seconds, of a read of data from the disk.

More Info:

Reads or non cached Writes

Excellent < 08 Msec ( .008 seconds )

Good < 12 Msec ( .012 seconds )

Fair < 20 Msec ( .020 seconds )

Poor > 20 Msec ( .020 seconds )

Cached Writes Only

Excellent < 01 Msec ( .001 seconds )

Good < 02 Msec ( .002 seconds )

Fair < 04 Msec ( .004 seconds )

Poor > 04 Msec ( .004 seconds

Reference: -

Object: - PhysicalDisk

Counter: - Avg. Disk sec/Write

Preferred Value: - < 8ms (non cached) < 1ms (cached)

Description: - Measure of disk latency. Avg. Disk sec/Write is the average time, in seconds, of a write of data to the disk.

Reference: -

Object: - PhysicalDisk

Counter: - Avg. Disk Read Queue Length

Preferred Value: - < 2 \* spindles

Description: - Avg. Disk Read Queue Length is the average number of read requests that were queued for the selected disk during the sample interval.

More Info:

< (2+ no of spindles) Excellent

< (2\*no of spindles) Good

< (3\* no of spindles) Fair

Reference - Whitepaper “Performance Monitoring in Windows 2003: Best Practices” by Ben W. Christenbury

Note: If the disk has say 20 disk and it is RAID 10 then no. of spindles = 20/2 = 10. If it is RAID 5 then the no. of spindles = no of disks = 20.

Reference: -

Object: - PhysicalDisk

Counter: - Avg. Disk Write Queue Length

Preferred Value: - < 2 \* spindles

Description: - Avg. Disk Write Queue Length is the average number of write requests that were queued for the selected disk during the sample interval.

Reference: -

Object: - CLASS: Buffer Manager

Counter: - Page reads/sec

Preferred Value: - < 90

Description: - Number of physical database page reads issued. 80 – 90 per second is normal, anything that is above indicates indexing or memory constraint.

Reference: -

Object: - CLASS: Buffer Manager

Counter: - Page writes/sec

Preferred Value: - < 90

Description: - Number of physical database page writes issued. 80 – 90 per second is normal, anything more we need to check the lazy writer/sec and checkpoint counters, if these counters are also relatively high then, it’s memory constraint.

Reference: -

Object: - CLASS: Buffer Manager

Counter: - Free pages

Preferred Value: - > 640

Description: - Total number of pages on all free lists.

Reference: -

Object: - CLASS: Buffer Manager

Counter: - Stolen pages

Preferred Value: - Varies. Compare with baseline

Description: - Number of pages used for miscellaneous server purposes (including procedure cache).

Reference: -

Object: - CLASS: Buffer Manager

Counter: - Buffer Cache hit ratio

Preferred Value: - > 90%

Description: - Percentage of pages that were found in the buffer pool without having to incur a read from disk.

Reference: -

Processor Bottleneck Analysis

Object: - Processor

Counter: - %Processor Time

Preferred Value: - < 80%

Description: - % Processor Time is the percentage of elapsed time that the processor spends to execute a non-Idle thread.

Reference: -

Object: - Processor

Counter: - %Privileged Time

Preferred Value: - < 30% of Total %Processor Time

Description: - % Privileged Time is the percentage of elapsed time that the process threads spent executing code in privileged mode.

Reference: -

Object: - Process (sqlservr)

Counter: - %Processor Time

Preferred Value: - < 80%

Description: -

Reference: -

Object: - Process (sqlservr)

Counter: - %Privileged Time

Preferred Value: - < 30% of %Processor Time (sqlservr)

Description: - Note: Divide the value by number of processors

Reference: -

Object: - System

Counter: - Context Switches/sec

Preferred Value: - < 3000

Description: - 1500 – 3000 per processor Excellent – Fair

> 6000 per processor Poor

Upper limit is about 40,000 at 90 % CPU per CPU

NOTE: Remember to divide by number of processors

Reference: -

Object: - System

Counter: - Processor Queue Length

Preferred Value: - < 4 per CPU

Description: - For standard servers with long Quantums

<= 4 per CPU Excellent

< 8 per CPU Good

< 12 per CPU Fair

Reference: -

Object: - CLASS: Access Methods

Counter: - Full Scans / sec

Preferred Value: - < 1

Description: - If we see high CPU then we need to invistigate this counter, otherwise if the full scan are on small tables we can ignore this counter. Values greater than 1 or 2 indicates that we are having table / Index page scans. We need to analyze how this can be avoided.

Reference: -

Object: - CLASS: Access Methods

Counter: - Worktables Created/Sec

Preferred Value: - < 20

Description: - Number of worktables created in tempdb per second. Worktables are used for queries that use various spools (table spool, index spool, etc).

Reference: -

Object: - CLASS: Access Methods

Counter: - Workfiles Created/Sec

Preferred Value: - < 20

Description: - Number of work files created per second. Tempdb workfiles are used in processing hash operations when the amount of data being processed is too big to fit into the available memory. They may be able to reduce this number by making the queries more efficient by adding/changing indexes, adding additional memory, etc.

Reference: -

Object: - CLASS: Access Methods

Counter: - Page Splits/sec

Preferred Value: - < 20

Description: - Interesting counter that can lead us to our table / index design. This value needs to be low as possible. If you find out that the number of page splits is high, consider increasing the fillfactor of your indexes. An increased fillfactor helps to reduce page splits because there is more room in data pages before it fills up and a page split has to occur.

Reference: -

## Overall SQL Server Bottleneck Analysis

Object: - CLASS: General Statistics

Counter: - User Connections

Preferred Value: -

Description: - The number of users currently connected to the SQL Server.

Reference: -

Object: - CLASS: General Statistics

Counter: - Logins/sec

Preferred Value: - < 2

Description: - > 2 per second indicates that the application is not correctly using connection pooling.

Reference: -

CLASS: - CLASS: General Statistics

Counter: - Logouts/sec

Preferred Value: - < 2

Description: - > 2 per second indicates that the application is not correctly using connection pooling.

Reference: -

CLASS: - CLASS: SQL Statistics

Counter: - Batch Requests/Sec

Preferred Value: - < 1000

Description: - Over 1000 batch requests per second indicate a very busy SQL Server.

Reference: -

CLASS: - CLASS: SQL Statistics

Counter: - SQL Compilations/sec

Preferred Value: - < 10% of the number of Batch Requests / sec

Description: - The number of times per second that SQL Server compilations have occurred. This value needs to be as low as possible. If you see a high value such as over 100, then it’s an indication that there are lots or adhoc queries that are running, might cause CPU

Reference: -

CLASS: - CLASS: SQL Statistics

Counter: - SQL Re-Compilations/sec

Preferred Value: - < 10% of the number of SQL Compilations/sec

Description: - This needs to be nil in our system as much as possible. A recompile can cause deadlocks and compile locks that are not compatible with any locking type.

Reference: -

CLASS: - CLASS: Latches

Counter: - Average Latch Wait Time (ms)

Preferred Value: - < 300

Description: - Average latch wait time (milliseconds) for latch requests that had to wait.

Reference: -

## Transaction Management

CLASS: - CLASS: Locks

Counter: - Number of Deadlocks/sec

Preferred Value: - < 1

Description: - The number of lock requests that resulted in a deadlock.

Reference: -

CLASS: - CLASS: Locks

Counter: - Lock Requests/sec

Preferred Value: - < 1000

Description: - Number of requests for a type of lock per second. Lock requests/sec > 1000 indicates that the queries are accessing large number of rows, the next step is to review high read queries. If you also see high Avg. Wait time, then it’s an indication of blocking, then review the blocking script output.

Reference: -

CLASS: - CLASS: Locks

Counter: - Average Wait Time (ms)

Preferred Value: - < 500

Description: - This is the average wait time in milliseconds to acquire a lock. Lower the value the better it is. If the value goes higher then 500, there may be blocking going on; we need to run blocker script to identify blocking.

Reference: -

|  |  |  |  |
| --- | --- | --- | --- |
| DETAIL ANALYSIS SQL Performance Counters | | | |
| **Object** | **Counter** | **Preferred Value** | **Description** |
| CLASS: Access Methods | Forwarded Records/sec | < 10 per 100 Batch Requests/Sec | Rows with varchar columns can experience expansion when varchar values are updated with a longer string.  In the case where the row cannot fit in the existing page, the row migrates and access to the row will traverse a pointer.  This only happens on heaps (tables without clustered indexes). Evaluate clustered index for heap tables.  In cases where clustered indexes cannot be used, drop non-clustered indexes, build a clustered index to reorg pages and rows, drop the clustered index, and then recreate non-clustered indexes. |
| CLASS: Access Methods | Full Scans / sec | (Index Searches/sec)/(Full Scans/sec) > 1000 | This counter monitors the number of full scans on base tables or indexes. Values greater than 1 or 2 indicate that we are having table / Index page scans. If we see high CPU then we need to investigate this counter, otherwise if the full scans are on small tables we can ignore this counter.  A few of the main causes of high Full Scans/sec are • Missing indexes • Too many rows requested Queries with missing indexes or too many rows requested will have a large number of logical reads and an increased CPU time. |
| CLASS: Access Methods | Index Searches/sec | (Index Searches/sec)/(Full Scans/sec) > 1000 | Number of index searches. Index searches are used to start range scans, single index record fetches, and to reposition within an index. Index searches are preferable to index and table scans.  For OLTP applications, optimize for more index searches and less scans (preferably, 1 full scan for every 1000 index searches). Index and table scans are expensive I/O operations. |
| CLASS: Access Methods | **Page Splits/sec** | < 20 per 100 Batch Requests/Sec | Number of page splits per second that occur as the result of overflowing index pages. Interesting counter that can lead us to our table / index design. This value needs to be low as possible. If you find out that the number of page splits is high, consider increasing the fillfactor of your indexes. An increased fillfactor helps to reduce page splits because there is more room in data pages before it fills up and a page split has to occur.  Note that this counter also includes the new page allocations as well and doesn’t necessarily pose a problem.  The other place we can confirm the page splits that involve data or index rows moves are the fragmented indexes on page splits. |
| CLASS: Buffer Manager | **Buffer Cache hit ratio** | > 90% | This counter indicates how often SQL Server goes to the buffer, not the hard disk, to get data. The higher this ratio, the less often SQL Server has to go to the hard disk to fetch data, and performance overall is boosted. Unlike many of the other counters available for monitoring SQL Server, this counter averages the Buffer Cache Hit Ratio from the time the last instance of SQL Server was restarted. In other words, this counter is not a real-time measurement, but an average of all the days since SQL Server was last restarted. In OLTP applications, this ratio should exceed 90-95%. If it doesn't, then you need to add more RAM to your server to increase performance. In OLAP applications, the ratio could be much less because of the nature of how OLAP works. In any case, more RAM should increase the performance of SQL Server OLAP activity. |
| CLASS: Buffer Manager | Free list stalls/sec | < 2 | Free list stalls/sec is the frequency with which requests for available database pages are suspended because no buffers are available. Free list stall rates of 3 or 4 per second indicate too little SQL memory available. |
| CLASS: Buffer Manager | Free pages | > 640 | Total number of pages on all free lists. |
| CLASS: Buffer Manager | Lazy Writes/Sec | < 20 | This counter tracks how many times a second that the Lazy Writer process is moving dirty pages from the buffer to disk in order to free up buffer space. Generally speaking, this should not be a high value, say more than 20 per second or so.  Ideally, it should be close to zero. If it is zero, this indicates that your SQL Server's buffer cache is plenty big and SQL Server doesn't have to free up dirty pages, instead waiting for this to occur during regular checkpoints. If this value is high, then a need for more memory is indicated. |
| CLASS: Buffer Manager | **Page Life Expectancy** | > 300 | This performance monitor counter tells you, on average, how long data pages are staying in the buffer. If this value gets below 300 seconds, this is a potential indication that your SQL Server could use more memory in order to boost performance. |
| CLASS: Buffer Manager | Page lookups/sec | (Page lookups/sec) / (Batch Requests/sec) < 100 | Number of requests to find a page in the buffer pool. When the ratio of page lookups to batch requests is much greater than 100, this is an indication that while query plans are looking up data in the buffer pool, these plans are inefficient. Identify queries with the highest amount of logical I/O's and tune them. |
| CLASS: Buffer Manager | Page reads/sec | < 90 | Number of physical database page reads issued. 80 – 90 per second is normal, anything that is above indicates indexing or memory constraint. |
| CLASS: Buffer Manager | Page writes/sec | < 90 | Number of physical database page writes issued. 80 – 90 per second is normal, anything more we need to check the lazy writer/sec and checkpoint counters, if these counters are also relatively high then, it’s memory constraint. |
| CLASS: General Statistics | **Logins/sec** | < 2 | > 2 per second indicates that the application is not correctly using **connection pooling**. |
| CLASS: General Statistics | **Logouts/sec** | < 2 | > 2 per second indicates that the application is not correctly using connection pooling. |
| CLASS: General Statistics | **User Connections** | See Description | The number of users currently connected to the SQL Server.    **Note**: It is recommended to review this counter along with “Batch Requests/Sec”.   A surge in “user connections” may result in a surge of “Batch Requests/Sec”.  So if there is a disparity (one going up and the other staying flat or going down), then that may be a cause for concern. With a blocking problem, for example, you might see user connections, lock waits and lock wait time all increase while batch requests/sec decreases. |
| CLASS: Latches | **Latch Waits/sec** | (Total Latch Wait Time) / (Latch Waits/Sec) < 10 | This is the number of latch requests that could not be granted immediately. In other words, these are the amount of latches, in a one second period that had to wait. |
| CLASS: Latches | **Total Latch Wait Time (ms)** | (Total Latch Wait Time) / (Latch Waits/Sec) < 10 | This is the total latch wait time (in milliseconds) for latch requests in the last second |
| CLASS: Locks | Lock Wait Time (ms) | See Description” | Total wait time (milliseconds) for locks in the last second.    **Note:** For “Lock Wait Time” it is recommended to look beyond the Avg value.  Look for any peaks that are close (or exceeds) to a wait of 60 sec.   Though this counter counts how many total milliseconds SQL Server is  waiting on locks during the last second, but the counter actually records  at the end of locking event.  So most probably the peaks represent one huge locking event.  If those events exceeds more than 60seconds then they may have extended blocking and could be an issue. In such cases, thoroughly analyze the blocking script output. Some applications are written for timing out after 60 seconds and that’s not acceptable response for those applications. |
| CLASS: Locks | **Lock Waits/sec** | 0 | This counter reports how many times users waited to acquire a lock over the past second.  Note that while you are actually waiting on the lock that this is not reflected in this counter—it gets incremented only when you “wake up” after waiting on the lock. If this value is nonzero then it is an indication that there is at least some level of blocking occurring.  If you combine this with the **Lock Wait Time**counter, you can get some idea of how long the blocking lasted.  A zero value for this counter can definitively prove out blocking as a potential cause; a nonzero value will require looking at other information to determine whether it is significant. |
| CLASS: Locks | Number of Deadlocks/sec | < 1 | The number of lock requests that resulted in a deadlock. |
| CLASS: Memory Manager | **Total Server Memory(KB)** | See Description | The Total Server Memory is the current amount of memory that SQL Server is using.  If this counter is still growing the server has not yet reached its steady-state, and it is still trying to populate the cache and get pages loaded into memory.  Performance will likely be somewhat slower during this time since more disk I/O is required at this stage.  This behavior is normal.  Eventually Total Server Memory should approximate Target Server Memory. |
| CLASS: SQL Statistics | **Batch Requests/Sec** | See Description | This counter measures the number of batch requests that SQL Server receives per second, and generally follows in step to how busy your server's CPUs are. Generally speaking, over 1000 batch requests per second indicates a very busy SQL Server, and could mean that if you are not already experiencing a CPU bottleneck, that you may very well soon. **Of course, this is a relative number, and the bigger your hardware, the more batch requests per second SQL Server can handle**. From a network bottleneck approach, a typical 100Mbs network card is only able to handle about 3000 batch requests per second. If you have a server that is this busy, you may need to have two or more network cards, or go to a 1Gbs network card.    **Note**: Sometimes low batch requests/sec can be misleading.  If there were a SQL statements/sec counter, this would be a more accurate measure of the amount of SQL Server activity.  For example, an application may call only a few stored procedures yet each stored procedure does lot of work.  In that case, we will see a low number for batch requests/sec but each stored procedure (one batch) will execute many SQL statements that drive CPU and other resources.  As a result, many counter thresholds based on the number of batch requests/sec will seem to identify issues because the batch requests on such a server are unusually low for the level of activity on the server.    We cannot conclude that a SQL Server is not active simply by looking at only batch requests/sec.  Rather, you have to do more investigation before deciding there is no load on the server.  If the average number of batch requests/sec is below 5 and other counters (such as SQL Server processor utilization) confirm the absence of significant activity, *then there is not enough of a load to make any recommendations or identify issues regarding scalability.* |
| CLASS: SQL Statistics | **SQL Compilations/sec** | < 10% of the number of Batch Requests/Sec | The number of times per second that SQL Server compilations have occurred. This value needs to be as low as possible. If you see a high value such as over 100, then it’s an indication that there are lots of adhoc queries that are running, might cause CPU usage, solution is to re-write these adhoc as stored procedure or use sp\_executeSQL. |
| CLASS: SQL Statistics | **SQL Re-Compilations/sec** | < 10% of the number of SQL Compilations/sec | This needs to be nil in our system as much as possible. A recompile can cause deadlocks and compile locks that are not compatible with any locking type. |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| | CUSTOMER\_ID | CUSTOMER\_NAME | | --- | --- | | 1 | John Doe | | 2 | Mary Jane | | 3 | Peter Pan | | 4 | Joe Soap | | | CUSTOMER\_ID | SALES\_PERSON | | --- | --- | | NULL | Newbee Smith | | 2 | Oldie Jones | | 1 | Another Oldie | | NULL | Greenhorn | |

The solution is to add a placeholder row in the customer table and update all NULL values in the sales table to the placeholder key.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| | CUSTOMER\_ID | CUSTOMER\_NAME | | --- | --- | | 0 | NO CUSTOMER | | 1 | John Doe | | 2 | Mary Jane | | 3 | Peter Pan | | 4 | Joe Soap | | | CUSTOMER\_ID | SALES\_PERSON | | --- | --- | | 0 | Newbee Smith | | 2 | Oldie Jones | | 1 | Another Oldie | | 0 | Greenhorn | |

Not only have you removed the need for an OUTER JOIN you have also standardised how sales people with no customers are represented. Other developers will not have to write statements such as ISNULL(customer\_id, “No customer yet”).