|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  |  |  |  |  |  | |  | |  | | --- | | Performance Methodology | | | | |  | |  |  |  |  |  |  | | |  | | --- | | When it comes to database performance improvements, there is usually an initial focus on hardware to provide short term improvements in performance and scalability. For long term improvements, shifting the focus to application and database design (schema design, indexing, data archiving, efficient data retrieval, stored procedure improvements, reasonable search criteria, etc.) will yield the most effective performance gains in a SQL environment. | | | | |  |  | |  |  |  |  |  |  | |  |  |  |  |  |  | | |  | | --- | | ***A Recommended Methodology***  There are many different ways to approach troubleshooting performance issues with SQL Server. Some DBAs approach it from the server view initially and exclude network, server CPU, memory, and disk issues before drilling into SQL Server. Other DBAs start with SQL Server first and eliminate SQL Server -specific issues before moving out to the server level. If you are looking to adopt a methodology, a recommended one to start with is documented in Jimmy May’s blog:-  <http://blogs.msdn.com/jimmymay/archive/2008/09/01/sql-server-performance-troubleshooting-methodology.aspx>  ***Performance Monitoring for Baseline & Performance Testing***  This section lists the recommended performance counters for both the system and for SQL Server to establish baseline behavior and to watch during testing to help ensure optimal performance of the application. These performance counters are all included in the performance capturing templates. Create a Baseline Now that you have reviewed the counters that will be included in your baseline, it is time to walk through the creation of the baseline chart. The following steps should be performed on your production server during a time of normal server activity.  To create the baseline chart, perform the following steps on your production server during a time of normal server activity:   * Open the Administrative Tools program menu and select Performance. * Double click Performance Logs and Alerts in the left window. Type the name of your baseline in the name field and select "OK". In the Select Counters window that appears, select your first Performance Object, and then select the related Counter. Select "Add". * Repeat this step for each counter you are tracking. For counters that allow you to track by disk, or by instance, and where you have more than one, repeat for each disk number or instance that appears in the Instance list. * In the Select Counters window select "Close". * Set the sample interval that you have decided upon. The time interval that you choose will depend on the amount of data that you want to collect, and the disk space available for that data. A longer sampling interval will allow you to collect less data over a longer time in the same amount of file space. For this reason you should try a short baseline first and reevaluate. (Note that the default value is "every 15 seconds". For a very detailed baseline this is ideal. But, many DBAs will find that this produces more data than they want to store.)   Select the **Log Files** tab and enter the file location in the **"Location"** field. Congratulations! You have scheduled a baseline log.  After the log is created, you can click the **System Monitor** option in the Performance window and then click **View Log File Data** in the right pane (the icon is a gray cylinder and is the fourth icon from the left). By opening the log file in the **Select Log File** window that appears, you will be able to view a graph of the logged results.  Now that you have created your baseline, you can begin to develop your ongoing monitoring solution. Clearly System Monitor will play an important role in your regimen. But what other technologies will you use? The next sections detail the tools at your disposal and the ways in which future System Monitor graphs can be compared to the baseline graph will be explored.   Performance Counters Guidelines While conducting baseline tests or performance tests if any of the following performance counters fall outside the listed tolerance it suggested that the application be further looked into and dissected to see what is causing the counters to go beyond a favorable tolerance.  ***General Performance by Symptom*** Memory ***Memory: Pages/sec:*** 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. It is the sum of Memory\Pages Input/sec and Memory\Pages Output/sec. It is counted in numbers of pages, so it can be compared to other counts of pages, such as Memory\Page Faults/sec, without conversion. It includes pages retrieved to satisfy faults in the file system cache (usually requested by applications) non-cached mapped memory files. (Hard Page fault). A high reading may be symptom of system memory under stress.  ***Memory: Available MBytes:*** A reading of less than 96MB is a symptom of memory pressure and a performance hit to the server.  ***SQL Server: Buffer Manager: Buffer Cache Hit Ratio:*** Total amount of dynamic memory the server is using for the dynamic SQL cache Anything above 90% is acceptable, however this ideally should be as near to 99% as possible depending on the system, namely if OLTP. Anything less than 90% may be indication of memory pressure. Disk I/O To evaluate whether you have a disk array resource bottleneck, you will utilize both of these counters to efficiently diagnose. Alerts should also be in place to notify an engineer of low disk space available, which can be a contributor to this issue as well.  You should utilize both of these counters in unison. If you have a sustained % Disk Idle Time of < 45% and an average disk queue length of greater than 2 per physical disk (spindle), then you can be confident you are experiencing an I/O bottleneck.  ***Physical Disk: Disk Reads/sec:*** The rate at which bytes are transferred from the disk during read operations. Growth of your baseline over time is symptomatic of memory pressure; this should be no higher than 85%.  ***Physical Disk: Disk Writes/sec:*** The rate at which bytes are transferred to the disk during write operations. Growth of your baseline over time is symptomatic of memory pressure; this should be no higher than 85%.  ***Physical Disk: % Disk Time:*** The percentage of elapsed time that the selected disk drive was busy servicing read or writes requests. Spikes are common. Results of >55% over a period of time warrants investigation.  ***Physical Disk: Avg. Disk Queue Length:*** The average number of both read and writes requests that were queued for the selected disk during the sample interval. Results of > than 2 x number of physical disks in volume is not optimal.  ***Physical Disk: % Free Space:*** The amount of free space available on physical disk. This should be no less than 5%  ***Logical Disk: % Free Space:*** The amount of free space available on logical disk. This should be no less than 5%. CPU Utilization ***Processor: Processor Time % (\_Total):*** The percentage of elapsed time that the processor spends to execute a non-Idle thread. This counter is the primary indicator of processor activity, and displays the average percentage of busy time observed during the sample interval. >80% % over a period of time may be an indicator of a CPU bottleneck. Spikes are common.  ***System: Processor Queue Length*:** The number of threads in the processor queue. >2 per CPU is indication of a possible CPU bottleneck.  Network Structure and Bandwidth  ***Network Interface: Bytes Received/sec:*** The rate at which bytes are transferred from the disk during read operations. a lot higher compared to a baseline. A substantial sudden baseline increase may be indication of an external attack; investigation is required.  ***Network Interface: Bytes Sent/sec:*** The rate at which bytes are sent over the network adapter. A sudden increase over baseline may be indication of a large volume of data being accessed. If you cannot explain the sudden increase, investigation is required.  ***Network Interface: Bytes/sec:*** This is the level rate as to traffic that is passed over the network. A substantial sudden baseline increase may be indication of an external attack; investigation is required. (ex. DOS – denial of service – attack)  ***Network Interface: Output Queue Length:*** Number of packets sent over the network adapter that had to wait for transmission. A non-zero value would indicate a faulty NIC or excessive use for current NIC capabilities. User Connections ***SQL Server: General: User Connections:*** This will show the number of user connections, not the number of users concurrently connected to SQL Server. A number of 255 is extraordinary and should be looked at for a corrective action. | | | | |  |  | |  |

|  |  |
| --- | --- |
|  |  |
| |  |  |  | | --- | --- | --- | |  |  |  | | |  | | --- | | Appendix | | |  | |  |  |  | |  | |  | | --- | | ***Appendix A: Explanation of Performance Counters*** System Performance Counters  * **LogicalDisk(\*)\% Idle Time:** The percentage of elapsed time that the selected drive was not busy servicing read or write requests. Inverse spikes are common. * **LogicalDisk(\*)\Avg. Disk Bytes/Read:** The rate at which bytes are transferred from the disk during read operations. a lot higher compared to a baseline. * **LogicalDisk(\*)\Avg. Disk Bytes/Write:** The rate at which bytes are transferred to the disk during write operations. a lot higher compared to a baseline. * **LogicalDisk(\*)\Avg. Disk Queue Length:** The average number of both read and write requests that were queued for the selected disk during the sample interval. > than 2 x number of physical disks in volume. * **LogicalDisk(\*)\Avg. Disk sec/Read:** The average time, in seconds, of a read of data from the disk. * **LogicalDisk(\*)\Avg. Disk sec/Write:** The average time, in seconds, of a write of data to the disk. * **LogicalDisk(\*)\Current Disk Queue Length:** The number of requests outstanding on the disk at the time the performance data is collected. It also includes requests in service at the time of the collection. This is an instantaneous snapshot, not an average over the time interval. Multi-spindle disk devices can have multiple requests that are active at one time, but other concurrent requests are awaiting service. This counter might reflect a transitory high or low queue length, but if there is a sustained load on the disk drive, it is likely that this will be consistently high. Requests experience delays proportional to the length of this queue minus the number of spindles on the disks. For good performance, this difference should average less than two. * **LogicalDisk(\*)\Free Megabytes:** Displays the unallocated space, in megabytes, on the disk drive in megabytes. One megabyte is equal to 1,048,576 bytes. * **Memory\Available Mbytes:** The amount of physical memory, in Megabytes, immediately available for allocation to a process or for system use. It is equal to the sum of memory assigned to the standby (cached), free and zero page lists. * **Memory\Cache Bytes:** Size, in bytes, of the portion of the system file cache which is currently resident and active in physical memory. The Cache Bytes and Memory\System Cache Resident Bytes counters are equivalent. This counter displays the last observed value only; it is not an average. * **Memory\Commit Limit:** The amount of virtual memory that can be committed without having to extend the paging file(s). It is measured in bytes. Committed memory is the physical memory which has space reserved on the disk paging files. There can be one paging file on each logical drive). If the paging file(s) are be expanded, this limit increases accordingly. This counter displays the last observed value only; it is not an average. * **Memory\Committed Bytes:** The amount of committed virtual memory, in bytes. Committed memory is the physical memory which has space reserved on the disk paging file(s). There can be one or more paging files on each physical drive. This counter displays the last observed value only; it is not an average. * **Memory\Free System Page Table Entries:** The number of page table entries not currently in used by the system. This counter displays the last observed value only; it is not an average. It is only relevant in x86 systems. <5000 is a symptom of virtual memory constraints, namely on x86 machines. This does NOT APPLY to Windows 2008 it has a new kernel with no realistic limit on system PTEs. * **Memory\Pages/sec:** 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. It is the sum of Memory\Pages Input/sec and Memory\Pages Output/sec. It is counted in numbers of pages, so it can be compared to other counts of pages, such as Memory\\Page Faults/sec, without conversion. It includes pages retrieved to satisfy faults in the file system cache (usually requested by applications) non-cached mapped memory files. (Hard Page fault). Spikes may result from backups, restores, checkpoints, BCP or DTS/DTSx tasks, and such spikes may be safely ignored, if they don’t represent a trend in the system. * **Memory\Page Faults/sec:** The average number of pages faulted per second. It is measured in number of pages faulted per second because only one page is faulted in each fault operation, hence this is also equal to the number of page fault operations. This counter includes both hard faults (those that require disk access) and soft faults (where the faulted page is found elsewhere in physical memory.) Most processors can handle large numbers of soft faults without significant consequence. However, hard faults, which require disk access, can cause significant delays.– evaluate soft and hard pages. * **Network Interface(\*)\Bytes Received/sec** * **Network Interface(\*)\Bytes Sent/sec** * **Network Interface(\*)\Current Bandwidth:** Is the estimate of the current bandwidth of the network interface in bits per second (BPS). For interfaces that do not vary in bandwidth or for those where no accurate estimation can be made, this value is the nominal bandwidth. * **Network Interface(\*)\Output Queue Length:** Output Queue Length is the length of the output packet queue (in packets). If this is longer than two, there are delays and the bottleneck should be found and eliminated, if possible. * **Paging File(\*)\% Usage:** The amount of the Page File instance in use in percent. * **PhysicalDisk(\*)\% Disk Time:** The percentage of elapsed time that the selected disk drive was busy servicing read or write requests. Results of >55% over a period of time warrants investigation. * **PhysicalDisk(\*)\Avg. Disk Queue Length:** the average number of both read and write requests that were queued for the selected disk during the sample interval. * **PhysicalDisk(\*)\Avg. Disk sec/Read:** The average time, in seconds, of a read of data from the disk. * **PhysicalDisk(\*)\Avg. Disk sec/Write:** The average time, in seconds, of a write of data to the disk. * **PhysicalDisk(\*)\Current Disk Queue Length** * **Process(\*)\% Processor Time:** Percentage of elapsed time that all of process threads used the processor to execution instructions. An instruction is the basic unit of execution in a computer, a thread is the object that executes instructions, and a process is the object created when a program is run. Code executed to handle some hardware interrupts and trap conditions are included in this count. Results of >80% over a period of time indicate CPU pressure. * **Process(\*)\IO Data Operations/sec:** The rate at which the process is issuing read and write I/O operations. This counter counts all I/O activity generated by the process to include file, network and device I/Os. * **Process(\*)\IO Other Operations/sec:** The rate at which the process is issuing I/O operations that are neither read nor write operations (for example, a control function). This counter counts all I/O activity generated by the process to include file, network and device I/Os. * **Process(\*)\IO Read Operations/sec:** The rate at which the process is issuing read I/O operations. This counter counts all I/O activity generated by the process to include file, network and device I/Os. * **Process(\*)\IO Write Operations/sec:** The rate at which the process is issuing write I/O operations. This counter counts all I/O activity generated by the process to include file, network and device I/Os. * **Process(\*)\Virtual Bytes:** Virtual Bytes is the current size, in bytes, of the virtual address space the process is using. Use of virtual address space does not necessarily imply corresponding use of either disk or main memory pages. Virtual space is finite, and the process can limit its ability to load libraries. * **Process(sqlservr)\% Privileged Time:** Percentage of elapsed time that the process threads spent executing code in privileged mode. When a Windows system service is called, the service will often run in privileged mode to gain access to system-private data. Such data is protected from access by threads executing in user mode. Calls to the system can be explicit or implicit, such as page faults or interrupts. Unlike some early operating systems, Windows uses process boundaries for subsystem protection in addition to the traditional protection of user and privileged modes. Some work done by Windows on behalf of the application might appear in other subsystem processes in addition to the privileged time in the process. Unless the processes are graphics-intensive or I/O-intensive such as file and print services, most applications should not be processing much work in kernel mode. Privileged mode corresponds to the percentage of time the processor spends on execution of Microsoft Windows kernel commands, such as processing of SQL Server I/O requests. If this counter is consistently high (> 20%) when the Physical Disk counters are high, consider installing a faster or more efficient disk subsystem. * **Processor(\*)\% DPC Time:** Percentage of time that the processor spent receiving and servicing deferred procedure calls (DPCs) during the sample interval. DPCs are interrupts that run at a lower priority than standard interrupts. % DPC Time is a component of % Privileged Time because DPCs are executed in privileged mode. They are counted separately and are not a component of the interrupt counters. This counter displays the average busy time as a percentage of the sample time. * **Processor(\*)\% Interrupt Time:** Time the processor spends receiving and servicing hardware interrupts during sample intervals. This value is an indirect indicator of the activity of devices that generate interrupts, such as the system clock, the mouse, disk drivers, data communication lines, network interface cards and other peripheral devices. These devices normally interrupt the processor when they have completed a task or require attention. Normal thread execution is suspended during interrupts. Most system clocks interrupt the processor every 10 milliseconds, creating a background of interrupt activity. suspends normal thread execution during interrupts. This counter displays the average busy time as a percentage of the sample time. * **Processor(\*)\% Privileged Time:** percentage of elapsed time that the process threads spent executing code in privileged mode. When a Windows system service in called, the service will often run in privileged mode to gain access to system-private data. Such data is protected from access by threads executing in user mode. Calls to the system can be explicit or implicit, such as page faults or interrupts. Unlike some early operating systems, Windows uses process boundaries for subsystem protection in addition to the traditional protection of user and privileged modes. Some work done by Windows on behalf of the application might appear in other subsystem processes in addition to the privileged time in the process. If this counter is consistently high (> 20%) when the Physical Disk counters are high, consider installing a faster or more efficient disk subsystem. * **Processor(\*)\% Processor Time:** Percentage of elapsed time that the processor spends to execute a non-Idle thread. It is calculated by measuring the percentage of time that the processor spends executing the idle thread and then subtracting that value from 100%. (Each processor has an idle thread that consumes cycles when no other threads are ready to run). This counter is the primary indicator of processor activity, and displays the average percentage of busy time observed during the sample interval. It should be noted that the accounting calculation of whether the processor is idle is performed at an internal sampling interval of the system clock (10ms). On today’s fast processors, % Processor Time can therefore underestimate the processor utilization as the processor may be spending a lot of time servicing threads between the system clock sampling intervals. Workload based timer applications are one example of applications which are more likely to be measured inaccurately as timers are signaled just after the sample is taken. Results of >80% over a period of time indicate CPU pressure. * **Processor(\*)\% User Time:** Percentage of elapsed time the processor spends in the user mode. User mode is a restricted processing mode designed for applications, environment subsystems, and integral subsystems. The alternative, privileged mode, is designed for operating system components and allows direct access to hardware and all memory. The operating system switches application threads to privileged mode to access operating system services. This counter displays the average busy time as a percentage of the sample time. * **Processor(\*)\DPC Rate:** Percentage of time that the processor spent receiving and servicing deferred procedure calls (DPCs) during the sample interval. DPCs are interrupts that run at a lower priority than standard interrupts. % DPC Time is a component of % Privileged Time because DPCs are executed in privileged mode. They are counted separately and are not a component of the interrupt counters. This counter displays the average busy time as a percentage of the sample time. When this value is high and Processor\% Processor Time is also high, further investigation is warranted. * **System\Context Switches/sec:** Combined rate at which all processors on the computer are switched from one thread to another. Context switches occur when a running thread voluntarily relinquishes the processor, is preempted by a higher priority ready thread, or switches between user-mode and privileged (kernel) mode to use an Executive or subsystem service. It is the sum of Thread\\Context Switches/sec for all threads running on all processors in the computer and is measured in numbers of switches. There are context switch counters on the System and Thread objects. This counter displays the difference between the values observed in the last two samples, divided by the duration of the sample interval. High context switching is only a problem if overall CPU is high as well. As a general rule, context switching rates of greater than 5,000 per second per processor are considered a warning. If context switching rates exceed 10,000 per second per processor, then there is a constraint. * **System\Processor Queue Length:** number of threads in the processor queue. Unlike the disk counters, this counter counters, this counter shows ready threads only, not threads that are running. There is a single queue for processor time even on computers with multiple processors. Therefore, if a computer has multiple processors, you need to divide this value by the number of processors servicing the workload. A sustained processor queue of less than 10 threads per processor is normally acceptable, dependent of the workload, but should be usually 0 (zero). Note that due to the way in which this counter is collected, ignore this counter and alerts for it when collected from a virtual computer.  SQL Server Performance Counters  * **Access Methods\Forwarded Records/sec:** Number of records fetched through forwarded record pointers. Forwarded records occur when a data record in a heap increases in size and the records current page does not have the space to store the size increase. The record is moved to a new location, becoming a forwarded record, and the forwarding record is left in the original location to point to the real location of the record. The forwarded record points back to the forwarding record in case its location ever needs to change again. * **Access Methods\FreeSpace Scans/sec:** Number of scans per second that were initiated to search for free space within pages already allocated to an allocation unit to insert or modify record fragments. Each scan may find multiple pages. FreeSpace Scans are due to inserts into heaps that require SQL Server to perform freespace scans to identify pages with free space to insert rows. Freespace scans are an additional I/O expense for inserts and can possibly cause contention on the GAM, SGAM, and PFS pages when many spids are inserting. The solution is often to evaluate clustered index for base tables. * **Access Methods\Full Scans/sec:** Number of full scans on base tables or indexes. High values indicate that we may be having performance issues due to table / index page scans. If we see high CPU and / or low Page Life Expectancy (PLE) then we need to investigate this counter; however, if 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. * **Access Methods\Index Searches/sec:** Number of index searches. Index searches are used to start range scans, single index record fetches, and to reposition within an index (known as scan point revalidations). * **Access Methods\Page Splits/sec:** Number of page splits per second that occur as a result of overflowing index pages. When a record is inserted into an index, it must be inserted in order. If the data page is full, the page splits in order to maintain the appropriate order. A high value for this counter may warrant the consideration of a lower fill factor. This value should be as low as possible. Heavily fragmented indexes may be the result of high page splits/sec. * **Access Methods\Scan Point Revalidations/sec:** Number of times the scan point had to be revalidated to continue the scan. When a range scan occurs there is an optimization process that occurs where the pages are marked as satisfied with the WHERE predicate that does the range scan. Instead of scanning through each and every row in the page, it does not keep an exclusive lock on those pages; instead it just keeps a mark on it and continues with rest of the scan. If one or more rows in the page are modified by update or a delete operation, the update or delete process will notify the scan to recheck the page to see if the page is still valid for the range scan. * **Access Methods\Workfiles Created/sec:** Number of work files created per second. Work files are used to store temporary results for hash joins and hash aggregated when the amount of data being processed is too big to fit into the available SQL Server memory. * **Access Methods\Worktables Created/sec:** Number of work tables created per second. Work tables are temporary objects and are used to store results for query spool, LOB variables, and cursors. Typically, this number is less than 200. * **Access Methods\Worktables from Cache Ratio:** Percentage of work tables created where the initial two pages of the work table were not allocated but were immediately available from the work table cache. * **Buffer Manager\Buffer cache hit ratio:** Percentage of pages that were found in the buffer pool without having to incur a read from disk. * **Buffer Manager\Checkpoint pages/sec:** Number of pages flushed by checkpoint or other operations that require all dirty pages to be flushed. * **Buffer Manager\Free pages:** Tot This counter measures the total number of pages on all free lists. The more free pages that are available then the less often the lazy writer will have to fire, keeping pages in the buffer pool longer. Usually, this threshold should not be below than 640 pages (5MB), which is already a low value in itself. * **Buffer Manager\Lazy writes/sec:** This counter records the number of buffers written each second by the buffer manager's lazy write process, thus tracking 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. This process is where the dirty, aged buffers are removed from the buffer by a system process that frees the memory up for other uses. A dirty, aged buffer is one that has changes and needs to be written to the disk. High value on this counter possibly indicates I/O issues or even SQL Server memory problems. Generally, this should not be a high value, and it should be close to zero. If it is zero, this indicates that your SQL Server's buffer cache is big enough 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. * **Buffer Manager\Page life expectancy:** Number of seconds a page will stay in the buffer pool without references. If Page life expectancy becomes low SQL Server will attempt physical reads from disk into the buffer pool to honor requests. Requests from physical disk will take considerably longer causing higher disk costs. Generally, this counter should not be below 1000s. * **Buffer Manager\Page lookups/sec:** Number of requests to find a page in the buffer pool. When the ratio of page lookups to batch requests is greater than 100, this is an indication that while query plans are looking up data in the buffer pool, these plans are inefficient. * **Buffer Manager\Page reads/sec:** Number of physical database page reads issued. 80 to 90 per second is normal, anything that is above indicates indexing or memory constraint. * **Buffer Manager\Page writes/sec:** Number of physical database page writes issued. 80 to 90 per second is normal, anything above, check the Lazy Writer/sec and Checkpoint Pages/sec counter, if these counters are relatively high then, this indicates a memory constraint. * **Buffer Manager: Stolen Page Count:** Number of pages used for miscellaneous server purposes (including procedure cache). – up to SQL Server 2008R2 * **Buffer Node: Lazy writes/sec:** Number of lookup requests from this node which were satisfied from a specific NUMA node. Ideally, it should be close to zero. * **Databases\Log Cache Hit Ratio:** Percentage of log cache reads that were satisfied from the log cache. * **Database\Log Growths:** Total number of log growths for a specific database. * **Database\Log Shrinks:** Total number of log shrinks for a specific database. * **General Statistics\Logins/sec:** Total number of logins started per second. Login and logout rates should be approximately the same. A login rate higher than the logout rate suggests that the server is not in a steady state, or that applications are not correctly using connection pooling. This could result in an increased load on the server. Verify if the .NET connection string is using the pooling=true e connection reset=true parameters. If so, a profiler trace with the Audit login and Audit logout Events would reveal the usage of *sp\_reset\_connection* stored procedure, which is used by SQL Server to support remote stored procedure calls in a transaction. This stored procedure also causes Audit Login and Audit Logout events to fire when a connection is reused from a connection pool. Also, the EventSubClass column in the trace will show if the connections are being pooled or not. Therefore focus the comparison only on the rate of non-pooled Logins and Logouts, as pooled connections will be reflected in the Logins/sec counter, but not on the Logouts/sec counter. * **General Statistics\Logouts/sec:** Total number of logouts started per second. * **General Statistics\Temp Tables Creation Rate:** Number of temporary tables/table variables created/sec. * **General Statistics\Temp Tables For Destruction:** Number of temporary tables/table variables waiting to be destroyed by the cleanup system thread. * **General Statistics\User Connections:** Number of users connected to the system. * **Latches\Latch Waits/sec:** Number of latch requests that could not be granted immediately and had to wait before being granted. * **Latches\Total Latch Wait Time (ms):** Total latch wait time (milliseconds) for latch requests that had to wait in the last second. If the total latch wait time is above 500 milliseconds per each second on average, your SQL Server may be spending too much time waiting on the various latches. It could also be facing resource contention as a result. * **Locks(\*)\Average Wait Time (ms):** The average amount of wait time (milliseconds) for each lock request that resulted in a wait. * **Locks(\*)\Lock Requests/sec:** Number of new locks and lock conversions requested from the lock manager. This value should not be greater than 50% of the number of Batch Requests/sec. * **Locks(\*)\Lock Timeouts/sec:** Number of lock requests that timed out. This includes requests for NOWAIT locks. * **Locks(\*)\Lock Wait Time (ms):** Total wait time (milliseconds) for locks in the last second. Although a sustained average of 500 or more milliseconds can indicate that your SQL Server is spending too much time waiting for locks, also watch for peaks that exceed 60 seconds for extended blocking for a given workload in your system. * **Locks(\*)\Lock Waits/sec:** Number of lock requests that could not be satisfied immediately and required the caller to wait before being granted the lock. * **Locks(\*)\Number of Deadlocks/sec:** Number of lock requests that resulted in a deadlock. * **Memory Manager\Memory Grants Pending:** Current number of processes waiting for a workspace memory grant. Memory Grants Pending records the number of connections that are waiting for memory before they can begin processing a memory intensive query such as a sort or hash operation. Connections that wait in this state for a long enough time will eventually get an 8645 error (A time out occurred while waiting for memory resources to execute the query. Rerun the query). A spid waiting in this state will have a waittype of 0x0040 (RESOURCE\_SEMAPHORE) in sysprocesses. If this counter remains above zero for any significant amount of time then you will need to track down what queries are doing sorts/hashes and run them through Index Tuning Wizard to see if they can get a more efficient plan. * **Memory Manager\Free Memory (KB):** Amount of memory the server is currently not using (equivalent to Buffer Manager: Free Pages in previous versions)– from SQL Server 2012. * **Memory Manager\Stolen Server Memory (KB):** Amount of memory the server is currently using for the purposes other than the database pages (equivalent to Buffer Manager: Stolen Pages in previous versions) – from SQL Server 2012. * **Memory Manager\Target Server Memory(KB):** Ideal amount of memory the server is willing to consume. * **Memory Manager\Total Server Memory (KB):** Total amount of dynamic memory the server is currently consuming. * **SQL Statistics\Batch Requests/sec:** Number of SQL batch requests received by server. * **SQL Statistics\SQL Compilations/sec:** Number of SQL compilations. This value needs to be as low as possible. If you see a high value such as over 100, then it is an indication that there are many ad-hoc queries that are running, might cause CPU usage, solution is to re-write these ad-hoc as stored procedure or use *sp\_executesql*. Set for baseline and should remain at this level or improve. * **SQL Statistics\SQL Re-Compilations/sec:** Number of SQL re-compiles. This needs to be 0 as much as possible. A recompile can cause deadlocks and compile locks that are not compatible with any locking type. | | | |  |  |  | |  |