**SQL Server 2008 and 2008 R2 Diagnostic Information Queries**

Posted on [November 18, 2010](http://sqlserverperformance.wordpress.com/2010/11/18/sql-server-2008-and-2008-r2-diagnostic-information-queries-2/) by [Glenn Berry](http://sqlserverperformance.wordpress.com/author/guderian1961/)

Since Microsoft released [SQL Server 2008 SP2 CU1 and SQL Server 2008 SP1 CU11](http://sqlserverperformance.wordpress.com/2010/11/16/sql-server-2008-service-pack-2-cumulative-update-1/) this week, I thought it would be a good excuse to post the latest version of my Diagnostic Information Queries. You need to have VIEW SERVER STATE permission to run many of these queries.

A couple of these queries will not work on SQL Server Denali CTP1, since Microsoft decided to make some breaking changes (boo)!  I have filed a Connect Item on those, but they are “by design” changes. I will list those two in a subsequent post.

-- SQL Server 2008 and R2 Diagnostic Information Queries

-- Glenn Berry

-- November 2010

-- http://sqlserverperformance.wordpress.com/

-- Twitter: GlennAlanBerry

-- Instance level queries \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- SQL and OS Version information for current instance

SELECT @@VERSION AS [SQL Server and OS Version Info];

-- SQL Server 2008 RTM is considered an "unsupported service pack" as of April 13, 2010

-- SQL Server 2008 RTM Builds SQL Server 2008 SP1 Builds SQL Server 2008 SP2 Builds

-- Build Description Build Description Build Description

-- 1600 Gold RTM

-- 1763 RTM CU1

-- 1779 RTM CU2

-- 1787 RTM CU3 --> 2531 SP1 RTM

-- 1798 RTM CU4 --> 2710 SP1 CU1

-- 1806 RTM CU5 --> 2714 SP1 CU2

-- 1812 RTM CU6 --> 2723 SP1 CU3

-- 1818 RTM CU7 --> 2734 SP1 CU4

-- 1823 RTM CU8 --> 2746 SP1 CU5

-- 1828 RTM CU9 --> 2757 SP1 CU6

-- 1835 RTM CU10 --> 2766 SP1 CU7

-- RTM Branch Retired --> 2775 SP1 CU8 --> 4000 SP2 RTM

-- 2789 SP1 CU9

-- 2799 SP1 CU10

-- 2804 SP1 CU11 --> 4266 SP2 CU1

-- SQL Server 2008 R2 Builds

-- Build Description

-- 10.50.1092 August 2009 CTP2

-- 10.50.1352 November 2009 CTP3

-- 10.50.1450 Release Candidate

-- 10.50.1600 RTM

-- 10.50.1702 RTM CU1

-- 10.50.1720 RTM CU2

-- 10.50.1734 RTM CU3

-- 10.50.1746 RTM CU4

-- SQL Server Denali

-- Build Description

-- 11.00.1055 CTP0

-- 11.00.1103 CTP1

-- SQL Azure Builds (most DMV queries don't work on SQL Azure)

-- Build Description

-- 10.25.9200 RTM Service Update 1

-- 10.25.9268 RTM Service Update 2

-- 10.25.9331 RTM Service Update 3

-- 10.25.9386 RTM Service Update 4

-- 10.25.9445 RTM Service Update 5

-- Hardware information from SQL Server 2008

-- (Cannot distinguish between HT and multi-core)

SELECT cpu\_count AS [Logical CPU Count], hyperthread\_ratio AS [Hyperthread Ratio],

cpu\_count/hyperthread\_ratio AS [Physical CPU Count],

physical\_memory\_in\_bytes/1048576 AS [Physical Memory (MB)], sqlserver\_start\_time

FROM sys.dm\_os\_sys\_info;

-- Get configuration values for instance

SELECT name, value, value\_in\_use, [description]

FROM sys.configurations

ORDER BY name ;

-- Focus on

-- backup compression default

-- clr enabled (only enable if it is needed)

-- lightweight pooling (should be zero)

-- max degree of parallelism

-- max server memory (MB) (set to an appropriate value)

-- optimize for ad hoc workloads (should be 1)

-- priority boost (should be zero)

-- File Names and Paths for TempDB and all user databases in instance

SELECT DB\_NAME([database\_id])AS [Database Name],

[file\_id], name, physical\_name, type\_desc, state\_desc,

CONVERT( bigint, size/128.0) AS [Total Size in MB]

FROM sys.master\_files

WHERE [database\_id] > 4

AND [database\_id] <> 32767

OR [database\_id] = 2

ORDER BY DB\_NAME([database\_id]);

-- Things to look at:

-- Are data files and log files on different drives?

-- Is everything on the C: drive?

-- Is TempDB on dedicated drives?

-- Are there multiple data files?

-- Calculates average stalls per read, per write, and per total input/output for each database file.

SELECT DB\_NAME(fs.database\_id) AS [Database Name], mf.physical\_name, io\_stall\_read\_ms, num\_of\_reads,

CAST(io\_stall\_read\_ms/(1.0 + num\_of\_reads) AS NUMERIC(10,1)) AS [avg\_read\_stall\_ms],io\_stall\_write\_ms,

num\_of\_writes,CAST(io\_stall\_write\_ms/(1.0+num\_of\_writes) AS NUMERIC(10,1)) AS [avg\_write\_stall\_ms],

io\_stall\_read\_ms + io\_stall\_write\_ms AS [io\_stalls], num\_of\_reads + num\_of\_writes AS [total\_io],

CAST((io\_stall\_read\_ms + io\_stall\_write\_ms)/(1.0 + num\_of\_reads + num\_of\_writes) AS NUMERIC(10,1))

AS [avg\_io\_stall\_ms]

FROM sys.dm\_io\_virtual\_file\_stats(null,null) AS fs

INNER JOIN sys.master\_files AS mf

ON fs.database\_id = mf.database\_id

AND fs.[file\_id] = mf.[file\_id]

ORDER BY avg\_io\_stall\_ms DESC;

-- Helps determine which database files on the entire instance have the most I/O bottlenecks

-- Recovery model, log reuse wait description, log file size, log usage size

-- and compatibility level for all databases on instance

SELECT db.[name] AS [Database Name], db.recovery\_model\_desc AS [Recovery Model],

db.log\_reuse\_wait\_desc AS [Log Reuse Wait Description],

ls.cntr\_value AS [Log Size (KB)], lu.cntr\_value AS [Log Used (KB)],

CAST(CAST(lu.cntr\_value AS FLOAT) / CAST(ls.cntr\_value AS FLOAT)AS DECIMAL(18,2)) \* 100 AS [Log Used %],

db.[compatibility\_level] AS [DB Compatibility Level],

db.page\_verify\_option\_desc AS [Page Verify Option], db.is\_auto\_create\_stats\_on, db.is\_auto\_update\_stats\_on,

db.is\_auto\_update\_stats\_async\_on, db.is\_parameterization\_forced,

db.snapshot\_isolation\_state\_desc, db.is\_read\_committed\_snapshot\_on

FROM sys.databases AS db

INNER JOIN sys.dm\_os\_performance\_counters AS lu

ON db.name = lu.instance\_name

INNER JOIN sys.dm\_os\_performance\_counters AS ls

ON db.name = ls.instance\_name

WHERE lu.counter\_name LIKE N'Log File(s) Used Size (KB)%'

AND ls.counter\_name LIKE N'Log File(s) Size (KB)%'

AND ls.cntr\_value > 0;

-- Things to look at:

-- How many databases are on the instance?

-- What recovery models are they using?

-- What is the log reuse wait description?

-- How full are the transaction logs ?

-- What compatibility level are they on?

-- Clear Wait Stats

-- DBCC SQLPERF('sys.dm\_os\_wait\_stats', CLEAR);

-- Isolate top waits for server instance since last restart or statistics clear

WITH Waits AS

(SELECT wait\_type, wait\_time\_ms / 1000. AS wait\_time\_s,

100. \* wait\_time\_ms / SUM(wait\_time\_ms) OVER() AS pct,

ROW\_NUMBER() OVER(ORDER BY wait\_time\_ms DESC) AS rn

FROM sys.dm\_os\_wait\_stats

WHERE wait\_type NOT IN ('CLR\_SEMAPHORE','LAZYWRITER\_SLEEP','RESOURCE\_QUEUE','SLEEP\_TASK'

,'SLEEP\_SYSTEMTASK','SQLTRACE\_BUFFER\_FLUSH','WAITFOR', 'LOGMGR\_QUEUE','CHECKPOINT\_QUEUE'

,'REQUEST\_FOR\_DEADLOCK\_SEARCH','XE\_TIMER\_EVENT','BROKER\_TO\_FLUSH','BROKER\_TASK\_STOP','CLR\_MANUAL\_EVENT'

,'CLR\_AUTO\_EVENT','DISPATCHER\_QUEUE\_SEMAPHORE', 'FT\_IFTS\_SCHEDULER\_IDLE\_WAIT'

,'XE\_DISPATCHER\_WAIT', 'XE\_DISPATCHER\_JOIN', 'SQLTRACE\_INCREMENTAL\_FLUSH\_SLEEP'))

SELECT W1.wait\_type,

CAST(W1.wait\_time\_s AS DECIMAL(12, 2)) AS wait\_time\_s,

CAST(W1.pct AS DECIMAL(12, 2)) AS pct,

CAST(SUM(W2.pct) AS DECIMAL(12, 2)) AS running\_pct

FROM Waits AS W1

INNER JOIN Waits AS W2

ON W2.rn <= W1.rn

GROUP BY W1.rn, W1.wait\_type, W1.wait\_time\_s, W1.pct

HAVING SUM(W2.pct) - W1.pct < 99; -- percentage threshold

-- Common Significant Wait types with BOL explanations

-- \*\*\* Network Related Waits \*\*\*

-- ASYNC\_NETWORK\_IO Occurs on network writes when the task is blocked behind the network

-- \*\*\* Locking Waits \*\*\*

-- LCK\_M\_IX Occurs when a task is waiting to acquire an Intent Exclusive (IX) lock

-- LCK\_M\_IU Occurs when a task is waiting to acquire an Intent Update (IU) lock

-- LCK\_M\_S Occurs when a task is waiting to acquire a Shared lock

-- \*\*\* I/O Related Waits \*\*\*

-- ASYNC\_IO\_COMPLETION Occurs when a task is waiting for I/Os to finish

-- IO\_COMPLETION Occurs while waiting for I/O operations to complete.

-- This wait type generally represents non-data page I/Os. Data page I/O completion waits appear

-- as PAGEIOLATCH\_\* waits

-- PAGEIOLATCH\_SH Occurs when a task is waiting on a latch for a buffer that is in an I/O request.

-- The latch request is in Shared mode. Long waits may indicate problems with the disk subsystem.

-- PAGEIOLATCH\_EX Occurs when a task is waiting on a latch for a buffer that is in an I/O request.

-- The latch request is in Exclusive mode. Long waits may indicate problems with the disk subsystem.

-- WRITELOG Occurs while waiting for a log flush to complete.

-- Common operations that cause log flushes are checkpoints and transaction commits.

-- PAGELATCH\_EX Occurs when a task is waiting on a latch for a buffer that is not in an I/O request.

-- The latch request is in Exclusive mode.

-- BACKUPIO Occurs when a backup task is waiting for data, or is waiting for a buffer in which to store data

-- \*\*\* CPU Related Waits \*\*\*

-- SOS\_SCHEDULER\_YIELD Occurs when a task voluntarily yields the scheduler for other tasks to execute.

-- During this wait the task is waiting for its quantum to be renewed.

-- THREADPOOL Occurs when a task is waiting for a worker to run on.

-- This can indicate that the maximum worker setting is too low, or that batch executions are taking

-- unusually long, thus reducing the number of workers available to satisfy other batches.

-- CX\_PACKET Occurs when trying to synchronize the query processor exchange iterator

-- You may consider lowering the degree of parallelism if contention on this wait type becomes a problem

-- Signal Waits for instance

SELECT CAST(100.0 \* SUM(signal\_wait\_time\_ms) / SUM (wait\_time\_ms) AS NUMERIC(20,2))

AS [%signal (cpu) waits],

CAST(100.0 \* SUM(wait\_time\_ms - signal\_wait\_time\_ms) / SUM (wait\_time\_ms) AS NUMERIC(20,2))

AS [%resource waits]

FROM sys.dm\_os\_wait\_stats;

-- Signal Waits above 10-15% is usually a sign of CPU pressure

-- Get CPU Utilization History for last 144 minutes (in one minute intervals)

-- This version works with SQL Server 2008 and SQL Server 2008 R2 only

DECLARE @ts\_now bigint = (SELECT cpu\_ticks/(cpu\_ticks/ms\_ticks)FROM sys.dm\_os\_sys\_info);

SELECT TOP(144) SQLProcessUtilization AS [SQL Server Process CPU Utilization],

SystemIdle AS [System Idle Process],

100 - SystemIdle - SQLProcessUtilization AS [Other Process CPU Utilization],

DATEADD(ms, -1 \* (@ts\_now - [timestamp]), GETDATE()) AS [Event Time]

FROM (

SELECT record.value('(./Record/@id)[1]', 'int') AS record\_id,

record.value('(./Record/SchedulerMonitorEvent/SystemHealth/SystemIdle)[1]', 'int')

AS [SystemIdle],

record.value('(./Record/SchedulerMonitorEvent/SystemHealth/ProcessUtilization)[1]',

'int')

AS [SQLProcessUtilization], [timestamp]

FROM (

SELECT [timestamp], CONVERT(xml, record) AS [record]

FROM sys.dm\_os\_ring\_buffers

WHERE ring\_buffer\_type = N'RING\_BUFFER\_SCHEDULER\_MONITOR'

AND record LIKE N'%<SystemHealth>%') AS x

) AS y

ORDER BY record\_id DESC;

-- Good basic information about memory amounts and state

SELECT total\_physical\_memory\_kb, available\_physical\_memory\_kb,

total\_page\_file\_kb, available\_page\_file\_kb,

system\_memory\_state\_desc

FROM sys.dm\_os\_sys\_memory;

-- You want to see "Available physical memory is high"

-- SQL Server Process Address space info

--(shows whether locked pages is enabled, among other things)

SELECT physical\_memory\_in\_use\_kb,locked\_page\_allocations\_kb,

page\_fault\_count, memory\_utilization\_percentage,

available\_commit\_limit\_kb, process\_physical\_memory\_low,

process\_virtual\_memory\_low

FROM sys.dm\_os\_process\_memory;

-- You want to see 0 for process\_physical\_memory\_low

-- You want to see 0 for process\_virtual\_memory\_low

-- Page Life Expectancy (PLE) value for default instance

SELECT cntr\_value AS [Page Life Expectancy]

FROM sys.dm\_os\_performance\_counters

WHERE OBJECT\_NAME = N'MSSQL$SQL\_CAPAX\_02:Buffer Manager' -- Modify this if you have named instances

AND counter\_name = N'Page life expectancy';

-- PLE is a good measurement of memory pressure.

-- Higher PLE is better. Below 300 is generally bad.

-- Watch the trend, not the absolute value.

-- Buffer cache hit ratio for default instance

SELECT (a.cntr\_value \* 1.0 / b.cntr\_value) \* 100.0 AS [Buffer Cache Hit Ratio]

FROM sys.dm\_os\_performance\_counters AS a

INNER JOIN (SELECT cntr\_value, [OBJECT\_NAME], instance\_name

FROM sys.dm\_os\_performance\_counters

WHERE counter\_name = N'Buffer cache hit ratio base'

AND [OBJECT\_NAME] = N'SQLServer:Buffer Manager') AS b -- Modify this if you have named instances

ON a.[OBJECT\_NAME] = b.[OBJECT\_NAME]

AND a.instance\_name = b.instance\_name

WHERE a.counter\_name = N'Buffer cache hit ratio'

AND a.[OBJECT\_NAME] = N'MSSQL$SQL\_CAPAX\_02:Buffer Manager'; -- Modify this if you have named instances

-- Shows the percentage that SQL Server is finding requested data in memory

-- A higher percentage is better than a lower percentage

-- Watch the trend, not the absolute value.

-- Memory Clerk Usage for instance

-- Look for high value for CACHESTORE\_SQLCP (Ad-hoc query plans)

SELECT TOP(20) [type], [name], SUM(single\_pages\_kb) AS [SPA Mem, Kb]

FROM sys.dm\_os\_memory\_clerks

GROUP BY [type], [name]

ORDER BY SUM(single\_pages\_kb) DESC;

-- CACHESTORE\_SQLCP SQL Plans These are cached SQL statements or batches that aren't in

-- stored procedures, functions and triggers

-- CACHESTORE\_OBJCP Object Plans These are compiled plans for stored procedures,

-- functions and triggers

-- CACHESTORE\_PHDR Algebrizer Trees An algebrizer tree is the parsed SQL text that

-- resolves the table and column names

use master

go

-- Find single-use, ad-hoc queries that are bloating the plan cache

SELECT TOP(100) [text], cp.size\_in\_bytes

FROM sys.dm\_exec\_cached\_plans AS cp

CROSS APPLY sys.dm\_exec\_sql\_text(plan\_handle)

WHERE cp.cacheobjtype = N'Compiled Plan'

AND cp.objtype = N'Adhoc'

AND cp.usecounts = 1

ORDER BY cp.size\_in\_bytes DESC;

-- Gives you the text and size of single-use ad-hoc queries that waste space in the plan cache

-- Enabling 'optimize for ad hoc workloads' for the instance can help (SQL Server 2008 and 2008 R2 only)

-- Enabling forced parameterization for the database can help, but test first!

-- Database specific queries \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- \*\*\*\* Switch to a user database \*\*\*\*\*

USE kplus;

GO

-- Individual File Sizes and space available for current database

SELECT name AS [File Name] , physical\_name AS [Physical Name], size/128.0 AS [Total Size in MB],

size/128.0 - CAST(FILEPROPERTY(name, 'SpaceUsed') AS int)/128.0 AS [Available Space In MB], [file\_id]

FROM sys.database\_files;

-- Look at how large and how full the files are and where they are located

-- Make sure the transaction log is not full!!

-- Top Cached SPs By Execution Count (SQL 2008)

SELECT TOP(100) p.name AS [SP Name], qs.execution\_count,

ISNULL(qs.execution\_count/DATEDIFF(Second, qs.cached\_time, GETDATE()), 0) AS [Calls/Second],

qs.total\_worker\_time/qs.execution\_count AS [AvgWorkerTime], qs.total\_worker\_time AS [TotalWorkerTime],

qs.total\_elapsed\_time, qs.total\_elapsed\_time/qs.execution\_count AS [avg\_elapsed\_time],

qs.cached\_time

FROM sys.procedures AS p

INNER JOIN sys.dm\_exec\_procedure\_stats AS qs

ON p.[object\_id] = qs.[object\_id]

WHERE qs.database\_id = DB\_ID()

ORDER BY qs.execution\_count DESC;

-- Tells you which cached stored procedures are called the most often

-- This helps you characterize and baseline your workload

-- Top Cached SPs By Avg Elapsed Time (SQL 2008)

SELECT TOP(25) p.name AS [SP Name],

cast(qs.total\_elapsed\_time/qs.execution\_count/1000.0 as decimal(20,2)) AS [avg\_elapsed\_time\_ms],

cast(qs.total\_elapsed\_time/1000.0 as decimal(20,2)) as total\_elapsed\_time\_ms,

qs.execution\_count, ISNULL(qs.execution\_count/DATEDIFF(Second, qs.cached\_time,

GETDATE()), 0) AS [Calls/Second],

cast(qs.total\_worker\_time/qs.execution\_count/1000.0 as decimal(20,2)) AS [AvgWorkerTime],

cast(qs.total\_worker\_time/1000.0 as decimal(20,2)) AS [TotalWorkerTime\_ms], qs.cached\_time

FROM sys.procedures AS p

INNER JOIN sys.dm\_exec\_procedure\_stats AS qs

ON p.[object\_id] = qs.[object\_id]

WHERE qs.database\_id = DB\_ID()

ORDER BY 2 DESC;

-- This helps you find long-running cached stored procedures

-- Top Cached SPs By Total Worker time (SQL 2008). Worker time relates to CPU cost

SELECT TOP(25) p.name AS [SP Name], qs.total\_worker\_time AS [TotalWorkerTime],

qs.total\_worker\_time/qs.execution\_count AS [AvgWorkerTime], qs.execution\_count,

ISNULL(qs.execution\_count/DATEDIFF(Second, qs.cached\_time, GETDATE()), 0) AS [Calls/Second],

qs.total\_elapsed\_time, qs.total\_elapsed\_time/qs.execution\_count

AS [avg\_elapsed\_time], qs.cached\_time

FROM sys.procedures AS p

INNER JOIN sys.dm\_exec\_procedure\_stats AS qs

ON p.[object\_id] = qs.[object\_id]

WHERE qs.database\_id = DB\_ID()

ORDER BY qs.total\_worker\_time DESC;

-- This helps you find the most expensive cached stored procedures from a CPU perspective

-- You should look at this if you see signs of CPU pressure

-- Top Cached SPs By Total Logical Reads (SQL 2008). Logical reads relate to memory pressure

SELECT TOP(25) p.name AS [SP Name], qs.total\_logical\_reads AS [TotalLogicalReads],

qs.total\_logical\_reads/qs.execution\_count AS [AvgLogicalReads],qs.execution\_count,

ISNULL(qs.execution\_count/DATEDIFF(Second, qs.cached\_time, GETDATE()), 0) AS [Calls/Second],

qs.total\_elapsed\_time, qs.total\_elapsed\_time/qs.execution\_count

AS [avg\_elapsed\_time], qs.cached\_time

FROM sys.procedures AS p

INNER JOIN sys.dm\_exec\_procedure\_stats AS qs

ON p.[object\_id] = qs.[object\_id]

WHERE qs.database\_id = DB\_ID()

ORDER BY qs.total\_logical\_reads DESC;

-- This helps you find the most expensive cached stored procedures from a memory perspective

-- You should look at this if you see signs of memory pressure

-- Top Cached SPs By Total Physical Reads (SQL 2008). Physical reads relate to disk I/O pressure

SELECT TOP(25) p.name AS [SP Name],qs.total\_physical\_reads AS [TotalPhysicalReads],

qs.total\_physical\_reads/qs.execution\_count AS [AvgPhysicalReads], qs.execution\_count,

qs.total\_logical\_reads,qs.total\_elapsed\_time, qs.total\_elapsed\_time/qs.execution\_count

AS [avg\_elapsed\_time], qs.cached\_time

FROM sys.procedures AS p

INNER JOIN sys.dm\_exec\_procedure\_stats AS qs

ON p.[object\_id] = qs.[object\_id]

WHERE qs.database\_id = DB\_ID()

ORDER BY qs.total\_physical\_reads, qs.total\_logical\_reads DESC;

-- This helps you find the most expensive cached stored procedures from a read I/O perspective

-- You should look at this if you see signs of I/O pressure or of memory pressure

-- Top Cached SPs By Total Logical Writes (SQL 2008).

-- Logical writes relate to both memory and disk I/O pressure

SELECT TOP(25) p.name AS [SP Name], qs.total\_logical\_writes AS [TotalLogicalWrites],

qs.total\_logical\_writes/qs.execution\_count AS [AvgLogicalWrites], qs.execution\_count,

ISNULL(qs.execution\_count/DATEDIFF(Second, qs.cached\_time, GETDATE()), 0) AS [Calls/Second],

qs.total\_elapsed\_time, qs.total\_elapsed\_time/qs.execution\_count AS [avg\_elapsed\_time],

qs.cached\_time

FROM sys.procedures AS p

INNER JOIN sys.dm\_exec\_procedure\_stats AS qs

ON p.[object\_id] = qs.[object\_id]

WHERE qs.database\_id = DB\_ID()

ORDER BY qs.total\_logical\_writes DESC;

-- This helps you find the most expensive cached stored procedures from a write I/O perspective

-- You should look at this if you see signs of I/O pressure or of memory pressure

-- Lists the top statements by average input/output usage for the current database

use master

go

SELECT TOP(50) OBJECT\_NAME(qt.objectid) AS [SP Name],

(qs.total\_logical\_reads + qs.total\_logical\_writes) /qs.execution\_count AS [Avg IO],

SUBSTRING(qt.[text],qs.statement\_start\_offset/2,

(CASE

WHEN qs.statement\_end\_offset = -1

THEN LEN(CONVERT(nvarchar(max), qt.[text])) \* 2

ELSE qs.statement\_end\_offset

END - qs.statement\_start\_offset)/2) AS [Query Text]

FROM sys.dm\_exec\_query\_stats AS qs

CROSS APPLY sys.dm\_exec\_sql\_text(qs.sql\_handle) AS qt

WHERE qt.[dbid] = DB\_ID()

ORDER BY [Avg IO] DESC;

-- Helps you find the most expensive statements for I/O by SP

-- Possible Bad NC Indexes (writes > reads)

SELECT OBJECT\_NAME(s.[object\_id]) AS [Table Name], i.name AS [Index Name], i.index\_id,

user\_updates AS [Total Writes], user\_seeks + user\_scans + user\_lookups AS [Total Reads],

user\_updates - (user\_seeks + user\_scans + user\_lookups) AS [Difference]

FROM sys.dm\_db\_index\_usage\_stats AS s WITH (NOLOCK)

INNER JOIN sys.indexes AS i WITH (NOLOCK)

ON s.[object\_id] = i.[object\_id]

AND i.index\_id = s.index\_id

WHERE OBJECTPROPERTY(s.[object\_id],'IsUserTable') = 1

AND s.database\_id = DB\_ID()

AND user\_updates > (user\_seeks + user\_scans + user\_lookups)

AND i.index\_id > 1

ORDER BY [Difference] DESC, [Total Writes] DESC, [Total Reads] ASC;

-- Look for indexes with high numbers of writes and zero or very low numbers of reads

-- Consider your complete workload

-- Investigate further before dropping an index

-- Missing Indexes current database by Index Advantage

SELECT user\_seeks \* avg\_total\_user\_cost \* (avg\_user\_impact \* 0.01) AS [index\_advantage],

migs.last\_user\_seek, mid.[statement] AS [Database.Schema.Table],

mid.equality\_columns, mid.inequality\_columns, mid.included\_columns,

migs.unique\_compiles, migs.user\_seeks, migs.avg\_total\_user\_cost, migs.avg\_user\_impact

FROM sys.dm\_db\_missing\_index\_group\_stats AS migs WITH (NOLOCK)

INNER JOIN sys.dm\_db\_missing\_index\_groups AS mig WITH (NOLOCK)

ON migs.group\_handle = mig.index\_group\_handle

INNER JOIN sys.dm\_db\_missing\_index\_details AS mid WITH (NOLOCK)

ON mig.index\_handle = mid.index\_handle

WHERE mid.database\_id = DB\_ID() -- Remove this to see for entire instance

ORDER BY index\_advantage DESC;

-- Look at last user seek time, number of user seeks to help determine source and importance

-- SQL Server is overly eager to add included columns, so beware

-- Do not just blindly add indexes that show up from this query!!!

-- Breaks down buffers used by current database by object (table, index) in the buffer cache

SELECT OBJECT\_NAME(p.[object\_id]) AS [ObjectName],

p.index\_id, COUNT(\*)/128 AS [Buffer size(MB)], COUNT(\*) AS [BufferCount],

p.data\_compression\_desc AS [CompressionType]

FROM sys.allocation\_units AS a

INNER JOIN sys.dm\_os\_buffer\_descriptors AS b

ON a.allocation\_unit\_id = b.allocation\_unit\_id

INNER JOIN sys.partitions AS p

ON a.container\_id = p.hobt\_id

WHERE b.database\_id = CONVERT(int,DB\_ID())

AND p.[object\_id] > 100

GROUP BY p.[object\_id], p.index\_id, p.data\_compression\_desc

ORDER BY [BufferCount] DESC;

-- Tells you what tables and indexes are using the most memory in the buffer cache

-- Get Table names, row counts, and compression status for clustered index or heap

SELECT OBJECT\_NAME(object\_id) AS [ObjectName],

SUM(Rows) AS [RowCount], data\_compression\_desc AS [CompressionType]

FROM sys.partitions

WHERE index\_id < 2 --ignore the partitions from the non-clustered index if any

AND OBJECT\_NAME(object\_id) NOT LIKE 'sys%'

AND OBJECT\_NAME(object\_id) NOT LIKE 'queue\_%'

AND OBJECT\_NAME(object\_id) NOT LIKE 'filestream\_tombstone%'

GROUP BY object\_id, data\_compression\_desc

ORDER BY SUM(Rows) DESC;

-- Gives you an idea of table sizes, and possible data compression opportunities

-- When were Statistics last updated on all indexes?

SELECT o.name, i.name AS [Index Name],

STATS\_DATE(i.[object\_id], i.index\_id) AS [Statistics Date],

s.auto\_created, s.no\_recompute, s.user\_created, st.row\_count

FROM sys.objects AS o WITH (NOLOCK)

INNER JOIN sys.indexes AS i WITH (NOLOCK)

ON o.[object\_id] = i.[object\_id]

INNER JOIN sys.stats AS s WITH (NOLOCK)

ON i.[object\_id] = s.[object\_id]

AND i.index\_id = s.stats\_id

INNER JOIN sys.dm\_db\_partition\_stats AS st WITH (NOLOCK)

ON o.[object\_id] = st.[object\_id]

AND i.[index\_id] = st.[index\_id]

WHERE o.[type] = 'U' and st.row\_count > 0

ORDER BY STATS\_DATE(i.[object\_id], i.index\_id) ASC;

-- Helps discover possible problems with out-of-date statistics

-- Also gives you an idea which indexes are most active

**Possibly related posts: (automatically generated)**

* [**New Cumulative Updates for SQL Server 2005 and SQL Server 2008 R2**](http://sqlserverperformance.wordpress.com/2010/10/20/new-cumulative-updates-for-sql-server-2005-and-sql-server-2008-r2/)
* [**What Build of SQL Server 2008 Are You On?**](http://sqlserverperformance.wordpress.com/2009/05/21/what-build-of-sql-server-2008-are-you-on/)