**SQL Server 2022 Diagnostic Information Queries**

**Glenn Berry**

-- https://urldefense.com/v3/\_\_https://glennsqlperformance.com/\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQSMgAOeFg$

-- https://urldefense.com/v3/\_\_https://sqlserverperformance.wordpress.com/\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQRE-bWfnQ$

-- YouTube: https://urldefense.com/v3/\_\_https://bit.ly/2PkoAM1\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQTnF3Znvg$

-- Twitter: GlennAlanBerry

-- Diagnostic Queries are available here

-- https://urldefense.com/v3/\_\_https://glennsqlperformance.com/resources/\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQT\_6P6jdw$

-- YouTube video demonstrating these queries

-- https://urldefense.com/v3/\_\_https://bit.ly/3aXNDzJ\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQSAyuymow$

-- Please make sure you are using the correct version of these diagnostic queries for your version of SQL Server

-- If you like PowerShell, there is a very useful community solution for running these queries in an automated fashion

-- https://urldefense.com/v3/\_\_https://dbatools.io/\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQRa\_xUmBw$

-- Invoke-DbaDiagnosticQuery

-- https://urldefense.com/v3/\_\_https://docs.dbatools.io/Invoke-DbaDiagnosticQuery\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQQEX\_Xk0g$

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

--\* Copyright (C) 2022 Glenn Berry

--\* All rights reserved.

--\*

--\*

--\* You may alter this code for your own \*non-commercial\* purposes. You may

--\* republish altered code as long as you include this copyright and give due credit.

--\*

--\*

--\* THIS CODE AND INFORMATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF

--\* ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED

--\* TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND/OR FITNESS FOR A

--\* PARTICULAR PURPOSE.

--\*

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- Check the major product version to see if it is SQL Server 2022 CTP 2 or greater

IF NOT EXISTS (SELECT \* WHERE CONVERT(varchar(128), SERVERPROPERTY('ProductMajorVersion')) = '16')

BEGIN

DECLARE @ProductVersion varchar(128) = CONVERT(varchar(128), SERVERPROPERTY('ProductVersion'));

RAISERROR ('Script does not match the ProductVersion [%s] of this instance. Many of these queries may not work on this version.' , 18 , 16 , @ProductVersion);

END

ELSE

PRINT N'You have the correct major version of SQL Server for this diagnostic information script';

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

-- SQL and OS Version information for current instance (Query 1) (Version Info)

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

------

-- SQL Server 2022 Builds

-- Build Description Release Date URL to KB Article

-- 16.0.600.9 CTP 2.0 5/20/2022

-- 16.0.700.4 CTP 2.1 7/27/2022

-- 16.0.900.6 RC0 8/23/2022

-- 16.0.950.9 RC1 9/22/2022

-- 16.0.1000.6 RTM 11/16/2022

-- What's new in SQL Server 2022 (16.x)

-- https://urldefense.com/v3/\_\_https://bit.ly/3MJEjR1\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQRC3eXGQw$

-- How to determine the version, edition and update level of SQL Server and its components

-- https://urldefense.com/v3/\_\_https://bit.ly/2oAjKgW\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQT7JLeIFw$

-- Announcing the Modern Servicing Model for SQL Server

-- https://urldefense.com/v3/\_\_https://bit.ly/2KtJ8SS\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQTyAK91jQ$

-- Update Center for Microsoft SQL Server

-- https://urldefense.com/v3/\_\_https://bit.ly/2pZptuQ\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQQ4HtJmeQ$

-- Download SQL Server Management Studio (SSMS)

-- https://urldefense.com/v3/\_\_https://bit.ly/1OcupT9\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQRGT90GSg$

-- Download and install Azure Data Studio

-- https://urldefense.com/v3/\_\_https://bit.ly/2vgke1A\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQSDFV62VA$

-- SQL Server 2022 Configuration Manager is SQLServerManager16.msc

-- SQL Server troubleshooting (Microsoft documentation resources)

-- https://urldefense.com/v3/\_\_http://bit.ly/2YY0pb1\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQRIcIjbhg$

-- Get socket, physical core and logical core count from the SQL Server Error log. (Query 2) (Core Counts)

-- This query might take a few seconds depending on the size of your error log

EXEC sys.xp\_readerrorlog 0, 1, N'detected', N'socket';

------

-- This can help you determine the exact core counts used by SQL Server and whether HT is enabled or not

-- It can also help you confirm your SQL Server licensing model

-- Be on the lookout for this message "using 40 logical processors based on SQL Server licensing"

-- (when you have more than 40 logical cores) which means grandfathered Server/CAL licensing

-- This query will return no results if your error log has been recycled since the instance was last started

-- Get selected server properties (Query 3) (Server Properties)

SELECT SERVERPROPERTY('MachineName') AS [MachineName],

SERVERPROPERTY('ServerName') AS [ServerName],

SERVERPROPERTY('InstanceName') AS [Instance],

SERVERPROPERTY('IsClustered') AS [IsClustered],

SERVERPROPERTY('ComputerNamePhysicalNetBIOS') AS [ComputerNamePhysicalNetBIOS],

SERVERPROPERTY('Edition') AS [Edition],

SERVERPROPERTY('ProductLevel') AS [ProductLevel], -- What servicing branch (RTM/SP/CU)

SERVERPROPERTY('ProductUpdateLevel') AS [ProductUpdateLevel], -- Within a servicing branch, what CU# is applied

SERVERPROPERTY('ProductVersion') AS [ProductVersion],

SERVERPROPERTY('ProductMajorVersion') AS [ProductMajorVersion],

SERVERPROPERTY('ProductMinorVersion') AS [ProductMinorVersion],

SERVERPROPERTY('ProductBuild') AS [ProductBuild],

SERVERPROPERTY('ProductBuildType') AS [ProductBuildType], -- Is this a GDR or OD hotfix (NULL if on a CU build)

SERVERPROPERTY('ProductUpdateReference') AS [ProductUpdateReference], -- KB article number that is applicable for this build

SERVERPROPERTY('ProcessID') AS [ProcessID],

SERVERPROPERTY('Collation') AS [Collation],

SERVERPROPERTY('IsFullTextInstalled') AS [IsFullTextInstalled],

SERVERPROPERTY('IsIntegratedSecurityOnly') AS [IsIntegratedSecurityOnly],

SERVERPROPERTY('FilestreamConfiguredLevel') AS [FilestreamConfiguredLevel],

SERVERPROPERTY('IsHadrEnabled') AS [IsHadrEnabled],

SERVERPROPERTY('HadrManagerStatus') AS [HadrManagerStatus],

SERVERPROPERTY('InstanceDefaultDataPath') AS [InstanceDefaultDataPath],

SERVERPROPERTY('InstanceDefaultLogPath') AS [InstanceDefaultLogPath],

SERVERPROPERTY('InstanceDefaultBackupPath') AS [InstanceDefaultBackupPath],

SERVERPROPERTY('ErrorLogFileName') AS [ErrorLogFileName],

SERVERPROPERTY('BuildClrVersion') AS [Build CLR Version],

SERVERPROPERTY('IsXTPSupported') AS [IsXTPSupported],

SERVERPROPERTY('IsPolybaseInstalled') AS [IsPolybaseInstalled],

SERVERPROPERTY('IsAdvancedAnalyticsInstalled') AS [IsRServicesInstalled],

SERVERPROPERTY('IsTempdbMetadataMemoryOptimized') AS [IsTempdbMetadataMemoryOptimized],

SERVERPROPERTY('IsServerSuspendedForSnapshotBackup') AS [IsServerSuspendedForSnapshotBackup],

SERVERPROPERTY('SuspendedDatabaseCount') AS [SuspendedDatabaseCount];

------

-- This gives you a lot of useful information about your instance of SQL Server,

-- such as the ProcessID for SQL Server and your collation

-- Note: Some columns will be NULL on older SQL Server builds

-- SERVERPROPERTY('IsServerSuspendedForSnapshotBackup') is a new option for SQL Server 2022

-- SERVERPROPERTY('SuspendedDatabaseCount') is a new option for SQL Server 2022

-- SERVERPROPERTY (Transact-SQL)

-- https://urldefense.com/v3/\_\_https://bit.ly/2eeaXeI\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQSST79VNQ$

-- Get instance-level configuration values for instance (Query 4) (Configuration Values)

SELECT name, value, value\_in\_use, minimum, maximum, [description], is\_dynamic, is\_advanced

FROM sys.configurations WITH (NOLOCK)

ORDER BY name OPTION (RECOMPILE);

------

-- Focus on these settings:

-- automatic soft-NUMA disabled (should be 0 in most cases)

-- backup checksum default (should be 1)

-- backup compression algorithm

-- backup compression default (should be 1 in most cases)

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

-- cost threshold for parallelism (depends on your workload)

-- lightweight pooling (should be zero)

-- max degree of parallelism (depends on your workload and hardware)

-- max server memory (MB) (set to an appropriate value, not the default)

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

-- priority boost (should be zero)

-- remote admin connections (should be 1)

-- tempdb metadata memory-optimized (0 by default, some workloads may benefit by enabling)

-- sys.configurations (Transact-SQL)

-- https://urldefense.com/v3/\_\_https://bit.ly/2HsyDZI\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQSc62ZPOQ$

-- New in SQL Server 2022 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- ADR Cleaner Thread Count Max number of threads ADR cleaner can assign

-- backup compression algorithm Configure backup compression algorithm

-- Data processed daily limit in TB SQL On-demand data processed daily limit in TB

-- Data processed monthly limit in TB SQL On-demand data processed monthly limit in TB

-- Data processed weekly limit in TB SQL On-demand data processed weekly limit in TB

-- hardware offload config Offload processing to specialized hardware

-- openrowset auto\_create\_statistics Enable or disable auto create statistics for openrowset sources.

-- suppress recovery model errors Return warning instead of error for unsupported ALTER DATABASE SET RECOVERY command

-- Returns a list of all global trace flags that are enabled (Query 5) (Global Trace Flags)

DBCC TRACESTATUS (-1);

------

-- If no global trace flags are enabled, no results will be returned.

-- It is very useful to know what global trace flags are currently enabled as part of the diagnostic process.

-- Common trace flags that should be enabled in most cases

-- TF 3226 - Suppresses logging of successful database backup messages to the SQL Server Error Log

-- https://urldefense.com/v3/\_\_https://bit.ly/38zDNAK\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQQ1NA7hDA$

-- TF 6534 - Enables use of native code to improve performance with spatial data. This is a startup trace flag only

-- https://urldefense.com/v3/\_\_https://bit.ly/2HrQUpU\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQQhIUMHhQ$

-- TF 7745 - Prevents Query Store data from being written to disk in case of a failover or shutdown command

-- https://urldefense.com/v3/\_\_https://bit.ly/2GU69Km\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQTJ8xgzCg$

-- DBCC TRACEON - Trace Flags (Transact-SQL)

-- https://urldefense.com/v3/\_\_https://bit.ly/2FuSvPg\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQRUj8UDtQ$

-- SQL Server Process Address space info (Query 6) (Process Memory)

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

SELECT physical\_memory\_in\_use\_kb/1024 AS [SQL Server Memory Usage (MB)],

locked\_page\_allocations\_kb/1024 AS [SQL Server Locked Pages Allocation (MB)],

large\_page\_allocations\_kb/1024 AS [SQL Server Large Pages Allocation (MB)],

page\_fault\_count, memory\_utilization\_percentage, available\_commit\_limit\_kb,

process\_physical\_memory\_low, process\_virtual\_memory\_low

FROM sys.dm\_os\_process\_memory WITH (NOLOCK) OPTION (RECOMPILE);

------

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

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

-- This indicates that you are not under internal memory pressure

-- If locked\_page\_allocations\_kb > 0, then LPIM is enabled

-- sys.dm\_os\_process\_memory (Transact-SQL)

-- https://urldefense.com/v3/\_\_https://bit.ly/3iUgQgC\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQRq-Cexeg$

-- How to enable the "locked pages" feature in SQL Server 2012

-- https://urldefense.com/v3/\_\_https://bit.ly/2F5UjOA\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQRBcKiAzA$

-- Memory Management Architecture Guide

-- https://urldefense.com/v3/\_\_https://bit.ly/2JKkadC\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQSN\_hrkZA$

-- SQL Server Services information (Query 7) (SQL Server Services Info)

SELECT servicename, process\_id, startup\_type\_desc, status\_desc,

last\_startup\_time, service\_account, is\_clustered, cluster\_nodename, [filename],

instant\_file\_initialization\_enabled

FROM sys.dm\_server\_services WITH (NOLOCK) OPTION (RECOMPILE);

------

-- Tells you the account being used for the SQL Server Service and the SQL Agent Service

-- Shows the process\_id, when they were last started, and their current status

-- Also shows whether you are running on a failover cluster instance, and what node you are running on

-- Also shows whether IFI is enabled

-- sys.dm\_server\_services (Transact-SQL)

-- https://urldefense.com/v3/\_\_https://bit.ly/2oKa1Un\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQTTkDrYPA$

-- Last backup information by database (Query 8) (Last Backup By Database)

SELECT ISNULL(d.[name], bs.[database\_name]) AS [Database], d.recovery\_model\_desc AS [Recovery Model],

d.log\_reuse\_wait\_desc AS [Log Reuse Wait Desc],

CONVERT(DECIMAL(18,2), ds.cntr\_value/1024.0) AS [Total Data File Size on Disk (MB)],

CONVERT(DECIMAL(18,2), ls.cntr\_value/1024.0) AS [Total Log File Size on Disk (MB)],

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

MAX(CASE WHEN bs.[type] = 'D' THEN bs.backup\_finish\_date ELSE NULL END) AS [Last Full Backup],

MAX(CASE WHEN bs.[type] = 'D' THEN CONVERT (BIGINT, bs.compressed\_backup\_size / 1048576 ) ELSE NULL END) AS [Last Full Compressed Backup Size (MB)],

MAX(CASE WHEN bs.[type] = 'D' THEN bs.compression\_algorithm ELSE NULL END) AS [Last Full Backup Compression Algorithm],

MAX(CASE WHEN bs.[type] = 'I' THEN bs.backup\_finish\_date ELSE NULL END) AS [Last Differential Backup],

MAX(CASE WHEN bs.[type] = 'L' THEN bs.backup\_finish\_date ELSE NULL END) AS [Last Log Backup],

MAX(CASE WHEN bs.[type] = 'L' THEN bs.last\_valid\_restore\_time ELSE NULL END) AS [Last Valid Restore Time],

DATABASEPROPERTYEX ((d.[name]), 'LastGoodCheckDbTime') AS [Last Good CheckDB]

FROM sys.databases AS d WITH (NOLOCK)

INNER JOIN sys.master\_files as mf WITH (NOLOCK)

ON d.database\_id = mf.database\_id

LEFT OUTER JOIN msdb.dbo.backupset AS bs WITH (NOLOCK)

ON bs.[database\_name] = d.[name]

AND bs.backup\_finish\_date > GETDATE()- 30

LEFT OUTER JOIN sys.dm\_os\_performance\_counters AS lu WITH (NOLOCK)

ON d.name = lu.instance\_name

LEFT OUTER JOIN sys.dm\_os\_performance\_counters AS ls WITH (NOLOCK)

ON d.name = ls.instance\_name

INNER JOIN sys.dm\_os\_performance\_counters AS ds WITH (NOLOCK)

ON d.name = ds.instance\_name

WHERE d.name <> N'tempdb'

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

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

AND ds.counter\_name LIKE N'Data File(s) Size (KB)%'

AND ls.cntr\_value > 0

GROUP BY ISNULL(d.[name], bs.[database\_name]), d.recovery\_model\_desc, d.log\_reuse\_wait\_desc, d.[name],

CONVERT(DECIMAL(18,2), ds.cntr\_value/1024.0),

CONVERT(DECIMAL(18,2), ls.cntr\_value/1024.0),

CAST(CAST(lu.cntr\_value AS FLOAT) / CAST(ls.cntr\_value AS FLOAT) AS DECIMAL(18,2)) \* 100

ORDER BY d.recovery\_model\_desc, d.[name] OPTION (RECOMPILE);

------

-- This helps you spot runaway transaction logs and other issues with your backup schedule

-- Get detailed accelerator status information (Query 9) (Accelerator Status)

SELECT accelerator, accelerator\_desc, config, config\_in\_use , mode, mode\_desc,

mode\_reason, mode\_reason\_desc, accelerator\_hardware\_detected,

accelerator\_library\_version, accelerator\_driver\_version

FROM sys.dm\_server\_accelerator\_status WITH (NOLOCK) OPTION (RECOMPILE);

------

-- This shows which accelerators are present and their detailed status information

-- sys.dm\_server\_accelerator\_status (Transact-SQL)

-- https://urldefense.com/v3/\_\_https://bit.ly/3B6Fczw\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQTMJMki\_w$

-- How to Enable Intel QAT Backup Compression in SQL Server 2022

-- https://urldefense.com/v3/\_\_https://bit.ly/3Cudwpy\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQTThkPAfg$

-- Get SQL Server Agent jobs and Category information (Query 10) (SQL Server Agent Jobs)

SELECT sj.name AS [Job Name], sj.[description] AS [Job Description],

sc.name AS [CategoryName], SUSER\_SNAME(sj.owner\_sid) AS [Job Owner],

sj.date\_created AS [Date Created], sj.[enabled] AS [Job Enabled],

sj.notify\_email\_operator\_id, sj.notify\_level\_email, h.run\_status,

RIGHT(STUFF(STUFF(REPLACE(STR(h.run\_duration, 7, 0), ' ', '0'), 4, 0, ':'), 7, 0, ':'),8) AS [Last Duration - HHMMSS],

CONVERT(DATETIME, RTRIM(h.run\_date) + ' ' + STUFF(STUFF(REPLACE(STR(RTRIM(h.run\_time),6,0),' ','0'),3,0,':'),6,0,':')) AS [Last Start Date]

FROM msdb.dbo.sysjobs AS sj WITH (NOLOCK)

INNER JOIN

(SELECT job\_id, instance\_id = MAX(instance\_id)

FROM msdb.dbo.sysjobhistory WITH (NOLOCK)

GROUP BY job\_id) AS l

ON sj.job\_id = l.job\_id

INNER JOIN msdb.dbo.syscategories AS sc WITH (NOLOCK)

ON sj.category\_id = sc.category\_id

INNER JOIN msdb.dbo.sysjobhistory AS h WITH (NOLOCK)

ON h.job\_id = l.job\_id

AND h.instance\_id = l.instance\_id

ORDER BY CONVERT(INT, h.run\_duration) DESC, [Last Start Date] DESC OPTION (RECOMPILE);

------

--run\_status

-- Value Status of the job execution

-- 0 = Failed

-- 1 = Succeeded

-- 2 = Retry

-- 3 = Canceled

-- 4 = In Progress

-- Gives you some basic information about your SQL Server Agent jobs, who owns them and how they are configured

-- Look for Agent jobs that are not owned by sa

-- Look for jobs that have a notify\_email\_operator\_id set to 0 (meaning no operator)

-- Look for jobs that have a notify\_level\_email set to 0 (meaning no e-mail is ever sent)

--

-- MSDN sysjobs documentation

-- https://urldefense.com/v3/\_\_https://bit.ly/2paDEOP\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQRpiVswGA$

-- SQL Server Maintenance Solution (Ola Hallengren)

-- https://urldefense.com/v3/\_\_https://bit.ly/1pgchQu\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQRRhg-81A$

-- You can use this script to add default schedules to the standard Ola Hallengren Maintenance Solution jobs

-- https://urldefense.com/v3/\_\_https://bit.ly/3ane0gN\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQTYZkZw8w$

-- Get SQL Server Agent Alert Information (Query 11) (SQL Server Agent Alerts)

SELECT name, event\_source, message\_id, severity, [enabled], has\_notification,

delay\_between\_responses, occurrence\_count, last\_occurrence\_date, last\_occurrence\_time

FROM msdb.dbo.sysalerts WITH (NOLOCK)

ORDER BY name OPTION (RECOMPILE);

------

-- Gives you some basic information about your SQL Server Agent Alerts

-- (which are different from SQL Server Agent jobs)

-- Read more about Agent Alerts here: https://urldefense.com/v3/\_\_https://bit.ly/2v5YR37\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQQyeY6jbw$

-- Host information (Query 12) (Host Info)

SELECT host\_platform, host\_distribution, host\_release,

host\_service\_pack\_level, host\_sku, os\_language\_version,

host\_architecture

FROM sys.dm\_os\_host\_info WITH (NOLOCK) OPTION (RECOMPILE);

------

-- host\_release codes (only valid for Windows)

-- 10.0 is either Windows 10, Windows Server 2016 or Windows Server 2019

-- 6.3 is either Windows 8.1 or Windows Server 2012 R2

-- 6.2 is either Windows 8 or Windows Server 2012

-- host\_sku codes (only valid for Windows)

-- 4 is Enterprise Edition

-- 7 is Standard Server Edition

-- 8 is Datacenter Server Edition

-- 10 is Enterprise Server Edition

-- 48 is Professional Edition

-- 161 is Pro for Workstations

-- 1033 for os\_language\_version is US-English

-- SQL Server 2022 requires Windows Server 2016 or newer

-- Hardware and Software Requirements for Installing SQL Server

-- https://urldefense.com/v3/\_\_https://bit.ly/2y3ka5L\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQSuZNWQYw$

-- Using SQL Server in Windows 8 and later versions of Windows operating system

-- https://urldefense.com/v3/\_\_https://bit.ly/2F7Ax0P\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQRtpYThlA$

-- SQL Server NUMA Node information (Query 13) (SQL Server NUMA Info)

SELECT osn.node\_id, osn.node\_state\_desc, osn.memory\_node\_id, osn.processor\_group, osn.cpu\_count, osn.online\_scheduler\_count,

osn.idle\_scheduler\_count, osn.active\_worker\_count,

osmn.pages\_kb/1024 AS [Committed Memory (MB)],

osmn.locked\_page\_allocations\_kb/1024 AS [Locked Physical (MB)],

CONVERT(DECIMAL(18,2), osmn.foreign\_committed\_kb/1024.0) AS [Foreign Commited (MB)],

osmn.target\_kb/1024 AS [Target Memory Goal (MB)],

osn.avg\_load\_balance, osn.resource\_monitor\_state

FROM sys.dm\_os\_nodes AS osn WITH (NOLOCK)

INNER JOIN sys.dm\_os\_memory\_nodes AS osmn WITH (NOLOCK)

ON osn.memory\_node\_id = osmn.memory\_node\_id

WHERE osn.node\_state\_desc <> N'ONLINE DAC' OPTION (RECOMPILE);

------

-- Gives you some useful information about the composition and relative load on your NUMA nodes

-- You want to see an equal number of schedulers on each NUMA node

-- Watch out if SQL Server 2022 Standard Edition has been installed

-- on a physical or virtual machine with more than four sockets or more than 24 physical cores

-- sys.dm\_os\_nodes (Transact-SQL)

-- https://urldefense.com/v3/\_\_https://bit.ly/2pn5Mw8\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQRxbF1pog$

-- How to Balance SQL Server Core Licenses Across NUMA Nodes

-- https://urldefense.com/v3/\_\_https://bit.ly/3i4TyVR\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQTFwZ\_2QQ$

-- Good basic information about OS memory amounts and state (Query 14) (System Memory)

SELECT total\_physical\_memory\_kb/1024 AS [Physical Memory (MB)],

available\_physical\_memory\_kb/1024 AS [Available Memory (MB)],

total\_page\_file\_kb/1024 AS [Page File Commit Limit (MB)],

total\_page\_file\_kb/1024 - total\_physical\_memory\_kb/1024 AS [Physical Page File Size (MB)],

available\_page\_file\_kb/1024 AS [Available Page File (MB)],

system\_cache\_kb/1024 AS [System Cache (MB)],

system\_memory\_state\_desc AS [System Memory State]

FROM sys.dm\_os\_sys\_memory WITH (NOLOCK) OPTION (RECOMPILE);

------

-- You want to see "Available physical memory is high" for System Memory State

-- This indicates that you are not under external memory pressure

-- Possible System Memory State values:

-- Available physical memory is high

-- Physical memory usage is steady

-- Available physical memory is low

-- Available physical memory is running low

-- Physical memory state is transitioning

-- sys.dm\_os\_sys\_memory (Transact-SQL)

-- https://urldefense.com/v3/\_\_https://bit.ly/2pcV0xq\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQTkRyUQMQ$

-- You can skip the next two queries if you know you don't have a clustered instance

-- Get information about your cluster nodes and their status (Query 15) (Cluster Node Properties)

-- (if your database server is in a failover cluster)

SELECT NodeName, status\_description, is\_current\_owner

FROM sys.dm\_os\_cluster\_nodes WITH (NOLOCK) OPTION (RECOMPILE);

------

-- Knowing which node owns the cluster resources is critical

-- Especially when you are installing Windows or SQL Server updates

-- You will see no results if your instance is not clustered

-- Get information about any AlwaysOn AG cluster this instance is a part of (Query 16) (AlwaysOn AG Cluster)

SELECT cluster\_name, quorum\_type\_desc, quorum\_state\_desc

FROM sys.dm\_hadr\_cluster WITH (NOLOCK) OPTION (RECOMPILE);

------

-- You will see no results if your instance is not using AlwaysOn AGs

-- Good overview of AG health and status (Query 17) (AG Status)

SELECT ag.name AS [AG Name], ar.replica\_server\_name, ar.availability\_mode\_desc, adc.[database\_name],

drs.is\_local, drs.is\_primary\_replica, drs.synchronization\_state\_desc, drs.is\_commit\_participant,

drs.synchronization\_health\_desc, drs.recovery\_lsn, drs.truncation\_lsn, drs.last\_sent\_lsn,

drs.last\_sent\_time, drs.last\_received\_lsn, drs.last\_received\_time, drs.last\_hardened\_lsn,

drs.last\_hardened\_time, drs.last\_redone\_lsn, drs.last\_redone\_time, drs.log\_send\_queue\_size,

drs.log\_send\_rate, drs.redo\_queue\_size, drs.redo\_rate, drs.filestream\_send\_rate,

drs.end\_of\_log\_lsn, drs.last\_commit\_lsn, drs.last\_commit\_time, drs.database\_state\_desc

FROM sys.dm\_hadr\_database\_replica\_states AS drs WITH (NOLOCK)

INNER JOIN sys.availability\_databases\_cluster AS adc WITH (NOLOCK)

ON drs.group\_id = adc.group\_id

AND drs.group\_database\_id = adc.group\_database\_id

INNER JOIN sys.availability\_groups AS ag WITH (NOLOCK)

ON ag.group\_id = drs.group\_id

INNER JOIN sys.availability\_replicas AS ar WITH (NOLOCK)

ON drs.group\_id = ar.group\_id

AND drs.replica\_id = ar.replica\_id

ORDER BY ag.name, ar.replica\_server\_name, adc.[database\_name] OPTION (RECOMPILE);

-- You will see no results if your instance is not using AlwaysOn AGs

-- SQL Server 2016 – It Just Runs Faster: Always On Availability Groups Turbocharged

-- https://urldefense.com/v3/\_\_https://bit.ly/2dn1H6r\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQQ-Nlijog$

-- Hardware information from SQL Server 2022 (Query 18) (Hardware Info)

SELECT cpu\_count AS [Logical CPU Count], scheduler\_count,

(socket\_count \* cores\_per\_socket) AS [Physical Core Count],

socket\_count AS [Socket Count], cores\_per\_socket, numa\_node\_count,

physical\_memory\_kb/1024 AS [Physical Memory (MB)],

max\_workers\_count AS [Max Workers Count],

affinity\_type\_desc AS [Affinity Type],

sqlserver\_start\_time AS [SQL Server Start Time],

DATEDIFF(hour, sqlserver\_start\_time, GETDATE()) AS [SQL Server Up Time (hrs)],

virtual\_machine\_type\_desc AS [Virtual Machine Type],

softnuma\_configuration\_desc AS [Soft NUMA Configuration],

sql\_memory\_model\_desc,

container\_type\_desc

FROM sys.dm\_os\_sys\_info WITH (NOLOCK) OPTION (RECOMPILE);

------

-- Gives you some good basic hardware information about your database server

-- Note: virtual\_machine\_type\_desc of HYPERVISOR does not automatically mean you are running SQL Server inside of a VM

-- It merely indicates that you have a hypervisor running on your host

-- sys.dm\_os\_sys\_info (Transact-SQL)

-- https://urldefense.com/v3/\_\_https://bit.ly/2pczOYs\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQTCZCUMVw$

-- Soft NUMA configuration was a new column for SQL Server 2016

-- OFF = Soft-NUMA feature is OFF

-- ON = SQL Server automatically determines the NUMA node sizes for Soft-NUMA

-- MANUAL = Manually configured soft-NUMA

-- Configure SQL Server to Use Soft-NUMA (SQL Server)

-- https://urldefense.com/v3/\_\_https://bit.ly/2HTpKJt\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQThHx-SYA$

-- sql\_memory\_model\_desc values (Added in SQL Server 2016 SP1)

-- CONVENTIONAL

-- LOCK\_PAGES

-- LARGE\_PAGES

-- Get System Manufacturer and model number from SQL Server Error log (Query 19) (System Manufacturer)

EXEC sys.xp\_readerrorlog 0, 1, N'Manufacturer';

------

-- This can help you determine the capabilities and capacities of your database server

-- Can also be used to confirm if you are running in a VM

-- This query might take a few seconds if you have not recycled your error log recently

-- This query will return no results if your error log has been recycled since the instance was started

-- Get BIOS date from Windows Registry (Query 20) (BIOS Date)

EXEC sys.xp\_instance\_regread N'HKEY\_LOCAL\_MACHINE', N'HARDWARE\DESCRIPTION\System\BIOS', N'BiosReleaseDate';

------

-- Helps you understand whether the main system BIOS is up to date, and the possible age of the hardware

-- Not as useful for virtualization

-- Does not work on Linux

-- Get processor description from Windows Registry (Query 21) (Processor Description)

EXEC sys.xp\_instance\_regread N'HKEY\_LOCAL\_MACHINE', N'HARDWARE\DESCRIPTION\System\CentralProcessor\0', N'ProcessorNameString';

------

-- Gives you the model number and rated clock speed of your processor(s)

-- Your processors may be running at less than the rated clock speed due

-- to the Windows Power Plan or hardware power management

-- Does not work on Linux

-- You can use CPU-Z to get your actual CPU core speed and a lot of other useful information

-- https://urldefense.com/v3/\_\_https://bit.ly/QhR6xF\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQT39eU7Jw$

-- You can learn more about processor selection for SQL Server by following this link

-- https://urldefense.com/v3/\_\_https://bit.ly/2F3aVlP\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQTz4fUOZg$

-- Get CPU vectorization level from SQL Server Error log (Query 22) (CPU Vectorization Level)

IF EXISTS (SELECT \* WHERE CONVERT(VARCHAR(2), SERVERPROPERTY('ProductMajorVersion')) = '16')

BEGIN

-- Get CPU Description from Registry (only works on Windows)

DROP TABLE IF EXISTS #ProcessorDesc;

CREATE TABLE #ProcessorDesc

(RegValue NVARCHAR(50), RegKey NVARCHAR(100));

INSERT INTO #ProcessorDesc (RegValue, RegKey)

EXEC sys.xp\_instance\_regread N'HKEY\_LOCAL\_MACHINE', N'HARDWARE\DESCRIPTION\System\CentralProcessor\0', N'ProcessorNameString';

DECLARE @ProcessorDesc NVARCHAR(100) = (SELECT RegKey FROM #ProcessorDesc);

-- Get CPU Vectorization Level from SQL Server Error Log

DROP TABLE IF EXISTS #CPUVectorizationLevel;

CREATE TABLE #CPUVectorizationLevel

(LogDateTime DATETIME, ProcessInfo NVARCHAR(12), LogText NVARCHAR(200));

INSERT INTO #CPUVectorizationLevel (LogDateTime, ProcessInfo, LogText)

EXEC sys.xp\_readerrorlog 0, 1, N'CPU vectorization level';

DECLARE @CPUVectorizationLevel NVARCHAR(200) = (SELECT LogText FROM #CPUVectorizationLevel);

-- Get TF15097 Status

DROP TABLE IF EXISTS #TraceFlagStatus;

CREATE TABLE #TraceFlagStatus

(TraceFlag smallint, TFStatus tinyint, TFGlobal tinyint, TFSession tinyint);

INSERT INTO #TraceFlagStatus (TraceFlag, TFStatus, TFGlobal, TFSession)

EXEC ('DBCC TRACESTATUS (15097, -1) WITH NO\_INFOMSGS');

DECLARE @TraceStatus tinyint = (SELECT TFStatus FROM #TraceFlagStatus);

-- Return relevant results

SELECT SERVERPROPERTY('ProductVersion') AS [Product Build], SERVERPROPERTY('Edition') AS [Edition],

@ProcessorDesc AS [Processor Description],

@CPUVectorizationLevel AS [CPU Vectorization Level], @TraceStatus AS [TF 15097 Status];

DROP TABLE IF EXISTS #ProcessorDesc;

DROP TABLE IF EXISTS #CPUVectorizationLevel;

DROP TABLE IF EXISTS #TraceFlagStatus;

END

------

-- Note: TF 15097 enables AVX-512 support for SQL Server 2022 (16.x) Preview and later (if your CPU supports it)

-- If you see AVX-512 in the CPU vectorization level results, you should consider enabling TF 15097

-- AVX-512 support only works in Enterprise Edition

-- Here are some CPU families that have good AVX-512 support:

-- Intel Ice Lake and later

-- AMD EYPC Genoa and later

-- Get information on location, time and size of any memory dumps from SQL Server (Query 23) (Memory Dump Info)

SELECT [filename], creation\_time, size\_in\_bytes/1048576.0 AS [Size (MB)]

FROM sys.dm\_server\_memory\_dumps WITH (NOLOCK)

ORDER BY creation\_time DESC OPTION (RECOMPILE);

------

-- This will not return any rows if you have

-- not had any memory dumps (which is a good thing)

-- sys.dm\_server\_memory\_dumps (Transact-SQL)

-- https://urldefense.com/v3/\_\_https://bit.ly/2elwWll\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQS9N2aqcw$

-- Look at Suspect Pages table (Query 24) (Suspect Pages)

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

event\_type, error\_count, last\_update\_date

FROM msdb.dbo.suspect\_pages WITH (NOLOCK)

ORDER BY database\_id OPTION (RECOMPILE);

------

-- event\_type value descriptions

-- 1 = 823 error caused by an operating system CRC error

-- or 824 error other than a bad checksum or a torn page (for example, a bad page ID)

-- 2 = Bad checksum

-- 3 = Torn page

-- 4 = Restored (The page was restored after it was marked bad)

-- 5 = Repaired (DBCC repaired the page)

-- 7 = Deallocated by DBCC

-- Ideally, this query returns no results. The table is limited to 1000 rows.

-- If you do get results here, you should do further investigation to determine the root cause

-- Manage the suspect\_pages Table

-- https://urldefense.com/v3/\_\_https://bit.ly/2Fvr1c9\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQRmylHKog$

-- Read most recent entries from all SQL Server Error Logs (Query 25) (Error Log Entries)

DROP TABLE IF EXISTS #ErrorLogFiles;

CREATE TABLE #ErrorLogFiles

([Archive #] INT,[Date] NVARCHAR(25),[Log File Size (Byte)]INT)

INSERT INTO #ErrorLogFiles

([Archive #],[Date],[Log File Size (Byte)])

EXEC master.sys.xp\_enumerrorlogs;

DROP TABLE IF EXISTS #SQLErrorLog\_AllLogs;

CREATE TABLE #SQLErrorLog\_AllLogs

(LogDate DATETIME ,ProcessInfo NVARCHAR(12), LogText NVARCHAR(4000))

DECLARE @i INT = 0;

DECLARE @sql NVARCHAR(200) = N'';

DECLARE @logCount INT = (SELECT COUNT(\*) FROM #ErrorLogFiles);

WHILE (@i < @logCount)

BEGIN

IF(@i in (SELECT [Archive #] FROM #ErrorLogFiles))

BEGIN

SET @sql = N'INSERT INTO #SQLErrorLog\_AllLogs (LogDate, ProcessInfo, LogText)

EXEC master.sys.sp\_readerrorlog ' + CAST(@i AS NVARCHAR(2)) + N';'

EXEC master.sys.sp\_executesql @sql;

END

SET @i += 1;

END

SELECT TOP(1000)LogDate, ProcessInfo, LogText

FROM #SQLErrorLog\_AllLogs WITH (NOLOCK)

ORDER BY LogDate DESC OPTION (RECOMPILE);

DROP TABLE IF EXISTS #ErrorLogFiles;

DROP TABLE IF EXISTS #SQLErrorLog\_AllLogs;

GO

------

-- Get number of data files in tempdb database (Query 26) (TempDB Data Files)

EXEC sys.xp\_readerrorlog 0, 1, N'The tempdb database has';

------

-- Get the number of data files in the tempdb database

-- 4-8 data files that are all the same size is a good starting point

-- This query will return no results if your error log has been recycled since the instance was last started

-- File names and paths for all user and system databases on instance (Query 27) (Database Filenames and Paths)

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

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

is\_percent\_growth, growth,

CONVERT(bigint, growth/128.0) AS [Growth in MB],

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

FROM sys.master\_files WITH (NOLOCK)

ORDER BY DB\_NAME([database\_id]), [file\_id] OPTION (RECOMPILE);

------

-- Things to look at:

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

-- Is everything on the C: drive?

-- Is tempdb on dedicated drives?

-- Is there only one tempdb data file?

-- Are all of the tempdb data files the same size?

-- Are there multiple data files for user databases?

-- Is percent growth enabled for any files (which is bad)?

-- Drive information for all fixed drives visible to the operating system (Query 28) (Fixed Drives)

SELECT fixed\_drive\_path, drive\_type\_desc,

CONVERT(DECIMAL(18,2), free\_space\_in\_bytes/1073741824.0) AS [Available Space (GB)]

FROM sys.dm\_os\_enumerate\_fixed\_drives WITH (NOLOCK) OPTION (RECOMPILE);

------

-- This shows all of your drives, not just LUNs with SQL Server database files

-- New in SQL Server 2017

-- sys.dm\_os\_enumerate\_fixed\_drives (Transact-SQL)

-- https://urldefense.com/v3/\_\_https://bit.ly/2EZoHLj\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQT66FXPYg$

-- Volume info for all LUNS that have database files on the current instance (Query 29) (Volume Info)

SELECT DISTINCT vs.volume\_mount\_point, vs.file\_system\_type, vs.logical\_volume\_name,

CONVERT(DECIMAL(18,2), vs.total\_bytes/1073741824.0) AS [Total Size (GB)],

CONVERT(DECIMAL(18,2), vs.available\_bytes/1073741824.0) AS [Available Size (GB)],

CONVERT(DECIMAL(18,2), vs.available\_bytes \* 1. / vs.total\_bytes \* 100.) AS [Space Free %],

vs.supports\_compression, vs.is\_compressed,

vs.supports\_sparse\_files, vs.supports\_alternate\_streams

FROM sys.master\_files AS f WITH (NOLOCK)

CROSS APPLY sys.dm\_os\_volume\_stats(f.database\_id, f.[file\_id]) AS vs

ORDER BY vs.volume\_mount\_point OPTION (RECOMPILE);

------

-- Shows you the total and free space on the LUNs where you have database files

-- Being low on free space can negatively affect performance

-- sys.dm\_os\_volume\_stats (Transact-SQL)

-- https://urldefense.com/v3/\_\_https://bit.ly/2oBPNNr\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQRNjyokjg$

-- Drive level latency information (Query 30) (Drive Level Latency)

SELECT tab.[Drive], tab.volume\_mount\_point AS [Volume Mount Point],

CASE

WHEN num\_of\_reads = 0 THEN 0

ELSE (io\_stall\_read\_ms/num\_of\_reads)

END AS [Read Latency],

CASE

WHEN num\_of\_writes = 0 THEN 0

ELSE (io\_stall\_write\_ms/num\_of\_writes)

END AS [Write Latency],

CASE

WHEN (num\_of\_reads = 0 AND num\_of\_writes = 0) THEN 0

ELSE (io\_stall/(num\_of\_reads + num\_of\_writes))

END AS [Overall Latency],

CASE

WHEN num\_of\_reads = 0 THEN 0

ELSE (num\_of\_bytes\_read/num\_of\_reads)

END AS [Avg Bytes/Read],

CASE

WHEN num\_of\_writes = 0 THEN 0

ELSE (num\_of\_bytes\_written/num\_of\_writes)

END AS [Avg Bytes/Write],

CASE

WHEN (num\_of\_reads = 0 AND num\_of\_writes = 0) THEN 0

ELSE ((num\_of\_bytes\_read + num\_of\_bytes\_written)/(num\_of\_reads + num\_of\_writes))

END AS [Avg Bytes/Transfer]

FROM (SELECT LEFT(UPPER(mf.physical\_name), 2) AS Drive, SUM(num\_of\_reads) AS num\_of\_reads,

SUM(io\_stall\_read\_ms) AS io\_stall\_read\_ms, SUM(num\_of\_writes) AS num\_of\_writes,

SUM(io\_stall\_write\_ms) AS io\_stall\_write\_ms, SUM(num\_of\_bytes\_read) AS num\_of\_bytes\_read,

SUM(num\_of\_bytes\_written) AS num\_of\_bytes\_written, SUM(io\_stall) AS io\_stall, vs.volume\_mount\_point

FROM sys.dm\_io\_virtual\_file\_stats(NULL, NULL) AS vfs

INNER JOIN sys.master\_files AS mf WITH (NOLOCK)

ON vfs.database\_id = mf.database\_id AND vfs.file\_id = mf.file\_id

CROSS APPLY sys.dm\_os\_volume\_stats(mf.database\_id, mf.[file\_id]) AS vs

GROUP BY LEFT(UPPER(mf.physical\_name), 2), vs.volume\_mount\_point) AS tab

ORDER BY [Overall Latency] OPTION (RECOMPILE);

------

-- Shows you the drive-level latency for reads and writes, in milliseconds

-- Latency above 30-40ms is usually a problem

-- These latency numbers include all file activity against all SQL Server

-- database files on each drive since SQL Server was last started

-- sys.dm\_io\_virtual\_file\_stats (Transact-SQL)

-- https://urldefense.com/v3/\_\_https://bit.ly/3bRWUc0\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQTJHFLd8w$

-- sys.dm\_os\_volume\_stats (Transact-SQL)

-- https://urldefense.com/v3/\_\_https://bit.ly/33thz2j\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQSwlN\_9zQ$

-- Calculates average latency per read, per write, and per total input/output for each database file (Query 31) (IO Latency by File)

SELECT DB\_NAME(fs.database\_id) AS [Database Name], CAST(fs.io\_stall\_read\_ms/(1.0 + fs.num\_of\_reads) AS NUMERIC(10,1)) AS [avg\_read\_latency\_ms],

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

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

CONVERT(DECIMAL(18,2), mf.size/128.0) AS [File Size (MB)], mf.physical\_name, mf.type\_desc, fs.io\_stall\_read\_ms, fs.num\_of\_reads,

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

io\_stall\_queued\_read\_ms AS [Resource Governor Total Read IO Latency (ms)], io\_stall\_queued\_write\_ms AS [Resource Governor Total Write IO Latency (ms)]

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

INNER JOIN sys.master\_files AS mf WITH (NOLOCK)

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

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

ORDER BY avg\_io\_latency\_ms DESC OPTION (RECOMPILE);

------

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

-- This can help you decide whether certain LUNs are overloaded and whether you might

-- want to move some files to a different location or perhaps improve your I/O performance

-- These latency numbers include all file activity against each SQL Server

-- database file since SQL Server was last started

-- sys.dm\_io\_virtual\_file\_stats (Transact-SQL)

-- https://urldefense.com/v3/\_\_https://bit.ly/3bRWUc0\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQTJHFLd8w$

-- Look for I/O requests taking longer than 15 seconds in the six most recent SQL Server Error Logs (Query 32) (IO Warnings)

DROP TABLE IF EXISTS #IOWarningResults;

CREATE TABLE #IOWarningResults(LogDate datetime, ProcessInfo sysname, LogText nvarchar(1000));

INSERT INTO #IOWarningResults

EXEC xp\_readerrorlog 0, 1, N'taking longer than 15 seconds';

INSERT INTO #IOWarningResults

EXEC xp\_readerrorlog 1, 1, N'taking longer than 15 seconds';

INSERT INTO #IOWarningResults

EXEC xp\_readerrorlog 2, 1, N'taking longer than 15 seconds';

INSERT INTO #IOWarningResults

EXEC xp\_readerrorlog 3, 1, N'taking longer than 15 seconds';

INSERT INTO #IOWarningResults

EXEC xp\_readerrorlog 4, 1, N'taking longer than 15 seconds';

INSERT INTO #IOWarningResults

EXEC xp\_readerrorlog 5, 1, N'taking longer than 15 seconds';

SELECT LogDate, ProcessInfo, LogText

FROM #IOWarningResults

ORDER BY LogDate DESC;

DROP TABLE IF EXISTS #IOWarningResults;

------

-- Finding 15 second I/O warnings in the SQL Server Error Log is useful evidence of

-- poor I/O performance (which might have many different causes)

-- Look to see if you see any patterns in the results (same files, same drives, same time of day, etc.)

-- Diagnostics in SQL Server help detect stalled and stuck I/O operations

-- https://urldefense.com/v3/\_\_https://bit.ly/2qtaw73\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQSG\_wU2XQ$

-- Resource Governor Resource Pool information (Query 33) (RG Resource Pools)

SELECT pool\_id, [Name], statistics\_start\_time,

min\_memory\_percent, max\_memory\_percent,

max\_memory\_kb/1024 AS [max\_memory\_mb],

used\_memory\_kb/1024 AS [used\_memory\_mb],

target\_memory\_kb/1024 AS [target\_memory\_mb],

min\_iops\_per\_volume, max\_iops\_per\_volume

FROM sys.dm\_resource\_governor\_resource\_pools WITH (NOLOCK)

OPTION (RECOMPILE);

------

-- sys.dm\_resource\_governor\_resource\_pools (Transact-SQL)

-- https://urldefense.com/v3/\_\_https://bit.ly/2MVU0Vy\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQTfJJYsuQ$

-- Recovery model, log reuse wait description, log file size, log usage size (Query 34) (Database Properties)

-- and compatibility level for all databases on instance

SELECT db.[name] AS [Database Name], SUSER\_SNAME(db.owner\_sid) AS [Database Owner],

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

db.recovery\_model\_desc AS [Recovery Model],

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

CONVERT(DECIMAL(18,2), ds.cntr\_value/1024.0) AS [Total Data File Size on Disk (MB)],

CONVERT(DECIMAL(18,2), ls.cntr\_value/1024.0) AS [Total Log File Size on Disk (MB)],

CONVERT(DECIMAL(18,2), lu.cntr\_value/1024.0) AS [Log File Used (MB)],

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

db.page\_verify\_option\_desc AS [Page Verify Option], db.user\_access\_desc, db.state\_desc, db.containment\_desc,

db.is\_mixed\_page\_allocation\_on,

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, db.is\_auto\_close\_on, db.is\_auto\_shrink\_on,

db.target\_recovery\_time\_in\_seconds, db.is\_cdc\_enabled, db.is\_published, db.is\_distributor, db.is\_sync\_with\_backup,

db.group\_database\_id, db.replica\_id, db.is\_memory\_optimized\_enabled, db.is\_memory\_optimized\_elevate\_to\_snapshot\_on,

db.delayed\_durability\_desc, db.is\_query\_store\_on,

db.is\_temporal\_history\_retention\_enabled, db.is\_accelerated\_database\_recovery\_on,

db.is\_data\_retention\_enabled, db.is\_ledger\_on, db.is\_change\_feed\_enabled,

db.is\_master\_key\_encrypted\_by\_server, db.is\_encrypted, de.encryption\_state, de.percent\_complete, de.key\_algorithm, de.key\_length

FROM sys.databases AS db WITH (NOLOCK)

LEFT OUTER JOIN sys.dm\_os\_performance\_counters AS lu WITH (NOLOCK)

ON db.name = lu.instance\_name

LEFT OUTER JOIN sys.dm\_os\_performance\_counters AS ls WITH (NOLOCK)

ON db.name = ls.instance\_name

LEFT OUTER JOIN sys.dm\_os\_performance\_counters AS ds WITH (NOLOCK)

ON db.name = ds.instance\_name

LEFT OUTER JOIN sys.dm\_database\_encryption\_keys AS de WITH (NOLOCK)

ON db.database\_id = de.database\_id

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

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

AND ds.counter\_name LIKE N'Data File(s) Size (KB)%'

AND ls.cntr\_value > 0

ORDER BY db.[name] OPTION (RECOMPILE);

------

-- sys.databases (Transact-SQL)

-- https://urldefense.com/v3/\_\_https://bit.ly/2G5wqaX\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQQG22UrDQ$

-- sys.dm\_os\_performance\_counters (Transact-SQL)

-- https://urldefense.com/v3/\_\_https://bit.ly/3kEO2JR\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQR2mWv\_fw$

-- sys.dm\_database\_encryption\_keys (Transact-SQL)

-- https://urldefense.com/v3/\_\_https://bit.ly/3mE7kkx\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQRwAFe8oA$

-- 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 the databases on?

-- What is the Page Verify Option? (should be CHECKSUM)

-- Is Auto Update Statistics Asynchronously enabled?

-- What is target\_recovery\_time\_in\_seconds? (should be 60 for user databases)

-- Is Delayed Durability enabled?

-- Make sure auto\_shrink and auto\_close are not enabled!

-- is\_mixed\_page\_allocation\_on is a new property for SQL Server 2016. Equivalent to TF 1118 for a user database

-- SQL Server 2016: Changes in default behavior for autogrow and allocations for tempdb and user databases

-- https://urldefense.com/v3/\_\_https://bit.ly/2evRZSR\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQT-jaf-dQ$

-- A non-zero value for target\_recovery\_time\_in\_seconds means that indirect checkpoint is enabled

-- If the setting has a zero value it indicates that automatic checkpoint is enabled

-- Changes in SQL Server 2016 Checkpoint Behavior

-- https://urldefense.com/v3/\_\_https://bit.ly/2pdggk3\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQT7VJ7\_Ug$

-- Missing Indexes for all databases by Index Advantage (Query 35) (Missing Indexes All Databases)

SELECT CONVERT(decimal(18,2), migs.user\_seeks \* migs.avg\_total\_user\_cost \* (migs.avg\_user\_impact \* 0.01)) AS [index\_advantage],

CONVERT(nvarchar(25), migs.last\_user\_seek, 20) AS [last\_user\_seek],

mid.[statement] AS [Database.Schema.Table],

COUNT(1) OVER(PARTITION BY mid.[statement]) AS [missing\_indexes\_for\_table],

COUNT(1) OVER(PARTITION BY mid.[statement], mid.equality\_columns) AS [similar\_missing\_indexes\_for\_table],

mid.equality\_columns, mid.inequality\_columns, mid.included\_columns, migs.user\_seeks,

CONVERT(decimal(18,2), migs.avg\_total\_user\_cost) AS [avg\_total\_user\_,cost], migs.avg\_user\_impact,

REPLACE(REPLACE(LEFT(st.[text], 255), CHAR(10),''), CHAR(13),'') AS [Short Query Text]

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

INNER JOIN sys.dm\_db\_missing\_index\_group\_stats\_query AS migs WITH(NOLOCK)

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

CROSS APPLY sys.dm\_exec\_sql\_text(migs.last\_sql\_handle) AS st

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

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

ORDER BY index\_advantage DESC OPTION (RECOMPILE);

------

-- Getting missing index information for all of the databases on the instance is very useful

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

-- Also look at avg\_user\_impact and avg\_total\_user\_cost to help determine importance

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

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

-- Håkan Winther has given me some great suggestions for this query

-- SQL Server Index Design Guide

-- https://urldefense.com/v3/\_\_https://bit.ly/2qtZr4N\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQR-eGK5Bw$

-- Get VLF Counts for all databases on the instance (Query 36) (VLF Counts)

SELECT [name] AS [Database Name], [VLF Count]

FROM sys.databases AS db WITH (NOLOCK)

CROSS APPLY (SELECT file\_id, COUNT(\*) AS [VLF Count]

FROM sys.dm\_db\_log\_info(db.database\_id)

GROUP BY file\_id) AS li

ORDER BY [VLF Count] DESC OPTION (RECOMPILE);

------

-- High VLF counts can affect write performance to the log file

-- and they can make full database restores and crash recovery take much longer

-- Try to keep your VLF counts under 200 in most cases (depending on log file size)

-- sys.dm\_db\_log\_info (Transact-SQL)

-- https://urldefense.com/v3/\_\_https://bit.ly/3jpmqsd\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQSbq0uLuA$

-- sys.databases (Transact-SQL)

-- https://urldefense.com/v3/\_\_https://bit.ly/2G5wqaX\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQQG22UrDQ$

-- SQL Server Transaction Log Architecture and Management Guide

-- https://urldefense.com/v3/\_\_https://bit.ly/2JjmQRZ\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQSku5\_Ufg$

-- VLF Growth Formula (SQL Server 2022)

-- If the log growth increment is less than 1/8th the current size of the log

-- Then: 1 new VLF

-- Otherwise:

-- Up to 64MB: 1 new VLF

-- 64MB to 1GB: 8 new VLFs

-- More than 1GB: 16 new VLFs

-- Virtual Log Files (VLFs)

-- https://urldefense.com/v3/\_\_https://bit.ly/3TN6en1\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQQyAgsz2A$

-- Get CPU utilization by database (Query 37) (CPU Usage by Database)

WITH DB\_CPU\_Stats

AS

(SELECT pa.DatabaseID, DB\_Name(pa.DatabaseID) AS [Database Name], SUM(qs.total\_worker\_time/1000) AS [CPU\_Time\_Ms]

FROM sys.dm\_exec\_query\_stats AS qs WITH (NOLOCK)

CROSS APPLY (SELECT CONVERT(int, value) AS [DatabaseID]

FROM sys.dm\_exec\_plan\_attributes(qs.plan\_handle)

WHERE attribute = N'dbid') AS pa

GROUP BY DatabaseID)

SELECT ROW\_NUMBER() OVER(ORDER BY [CPU\_Time\_Ms] DESC) AS [CPU Rank],

[Database Name], [CPU\_Time\_Ms] AS [CPU Time (ms)],

CAST([CPU\_Time\_Ms] \* 1.0 / SUM([CPU\_Time\_Ms]) OVER() \* 100.0 AS DECIMAL(5, 2)) AS [CPU Percent]

FROM DB\_CPU\_Stats

WHERE DatabaseID <> 32767 -- ResourceDB

ORDER BY [CPU Rank] OPTION (RECOMPILE);

------

-- Helps determine which database is using the most CPU resources on the instance

-- Note: This only reflects CPU usage from the currently cached query plans

-- sys.dm\_exec\_query\_stats (Transact-SQL)

-- https://urldefense.com/v3/\_\_https://bit.ly/32tHCGH\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQQ5AgoCzQ$

-- sys.dm\_exec\_plan\_attributes (Transact-SQL)

-- https://urldefense.com/v3/\_\_https://bit.ly/35iP2hV\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQSt2weVEw$

-- Get I/O utilization by database (Query 38) (IO Usage By Database)

WITH Aggregate\_IO\_Statistics

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

CAST(SUM(num\_of\_bytes\_read + num\_of\_bytes\_written) / 1048576 AS DECIMAL(12, 2)) AS [ioTotalMB],

CAST(SUM(num\_of\_bytes\_read ) / 1048576 AS DECIMAL(12, 2)) AS [ioReadMB],

CAST(SUM(num\_of\_bytes\_written) / 1048576 AS DECIMAL(12, 2)) AS [ioWriteMB]

FROM sys.dm\_io\_virtual\_file\_stats(NULL, NULL) AS [DM\_IO\_STATS]

GROUP BY database\_id)

SELECT ROW\_NUMBER() OVER (ORDER BY ioTotalMB DESC) AS [I/O Rank],

[Database Name], ioTotalMB AS [Total I/O (MB)],

CAST(ioTotalMB / SUM(ioTotalMB) OVER () \* 100.0 AS DECIMAL(5, 2)) AS [Total I/O %],

ioReadMB AS [Read I/O (MB)],

CAST(ioReadMB / SUM(ioReadMB) OVER () \* 100.0 AS DECIMAL(5, 2)) AS [Read I/O %],

ioWriteMB AS [Write I/O (MB)],

CAST(ioWriteMB / SUM(ioWriteMB) OVER () \* 100.0 AS DECIMAL(5, 2)) AS [Write I/O %]

FROM Aggregate\_IO\_Statistics

ORDER BY [I/O Rank] OPTION (RECOMPILE);

------

-- Helps determine which database is using the most I/O resources on the instance

-- These numbers are cumulative since the last service restart

-- They include all I/O activity, not just the nominal I/O workload

-- sys.dm\_io\_virtual\_file\_stats (Transact-SQL)

-- https://urldefense.com/v3/\_\_https://bit.ly/3bRWUc0\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQTJHFLd8w$

-- Get total buffer usage by database for current instance (Query 39) (Total Buffer Usage by Database)

-- This may take some time to run on a busy instance with lots of RAM

WITH AggregateBufferPoolUsage

AS

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

CAST(COUNT\_BIG(\*) \* 8/1024.0 AS DECIMAL (15,2)) AS [CachedSize],

COUNT(page\_id) AS [Page Count],

AVG(read\_microsec) AS [Avg Read Time (microseconds)]

FROM sys.dm\_os\_buffer\_descriptors WITH (NOLOCK)

GROUP BY DB\_NAME(database\_id))

SELECT ROW\_NUMBER() OVER(ORDER BY CachedSize DESC) AS [Buffer Pool Rank], [Database Name],

CAST(CachedSize / SUM(CachedSize) OVER() \* 100.0 AS DECIMAL(5,2)) AS [Buffer Pool Percent],

[Page Count], CachedSize AS [Cached Size (MB)], [Avg Read Time (microseconds)]

FROM AggregateBufferPoolUsage

ORDER BY [Buffer Pool Rank] OPTION (RECOMPILE);

------

-- Tells you how much memory (in the buffer pool)

-- is being used by each database on the instance

-- sys.dm\_os\_buffer\_descriptors (Transact-SQL)

-- https://urldefense.com/v3/\_\_https://bit.ly/36s7aFo\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQTVJfn5UQ$

-- Get tempdb version store space usage by database (Query 40) (Version Store Space Usage)

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

reserved\_page\_count AS [Version Store Reserved Page Count],

reserved\_space\_kb/1024 AS [Version Store Reserved Space (MB)]

FROM sys.dm\_tran\_version\_store\_space\_usage WITH (NOLOCK)

ORDER BY reserved\_space\_kb/1024 DESC OPTION (RECOMPILE);

------

-- sys.dm\_tran\_version\_store\_space\_usage (Transact-SQL)

-- https://urldefense.com/v3/\_\_https://bit.ly/2vh3Bmk\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQRjtXqeDg$

-- Clear Wait Stats with this command

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

-- Isolate top waits for server instance since last restart or wait statistics clear (Query 41) (Top Waits)

WITH [Waits]

AS (SELECT wait\_type, wait\_time\_ms/ 1000.0 AS [WaitS],

(wait\_time\_ms - signal\_wait\_time\_ms) / 1000.0 AS [ResourceS],

signal\_wait\_time\_ms / 1000.0 AS [SignalS],

waiting\_tasks\_count AS [WaitCount],

100.0 \* wait\_time\_ms / SUM (wait\_time\_ms) OVER() AS [Percentage],

ROW\_NUMBER() OVER(ORDER BY wait\_time\_ms DESC) AS [RowNum]

FROM sys.dm\_os\_wait\_stats WITH (NOLOCK)

WHERE [wait\_type] NOT IN (

N'BROKER\_EVENTHANDLER', N'BROKER\_RECEIVE\_WAITFOR', N'BROKER\_TASK\_STOP',

N'BROKER\_TO\_FLUSH', N'BROKER\_TRANSMITTER', N'CHECKPOINT\_QUEUE',

N'CHKPT', N'CLR\_AUTO\_EVENT', N'CLR\_MANUAL\_EVENT', N'CLR\_SEMAPHORE', N'CXCONSUMER',

N'DBMIRROR\_DBM\_EVENT', N'DBMIRROR\_EVENTS\_QUEUE', N'DBMIRROR\_WORKER\_QUEUE',

N'DBMIRRORING\_CMD', N'DIRTY\_PAGE\_POLL', N'DISPATCHER\_QUEUE\_SEMAPHORE',

N'EXECSYNC', N'FSAGENT', N'FT\_IFTS\_SCHEDULER\_IDLE\_WAIT', N'FT\_IFTSHC\_MUTEX',

N'HADR\_CLUSAPI\_CALL', N'HADR\_FILESTREAM\_IOMGR\_IOCOMPLETION', N'HADR\_LOGCAPTURE\_WAIT',

N'HADR\_NOTIFICATION\_DEQUEUE', N'HADR\_TIMER\_TASK', N'HADR\_WORK\_QUEUE',

N'KSOURCE\_WAKEUP', N'LAZYWRITER\_SLEEP', N'LOGMGR\_QUEUE',

N'MEMORY\_ALLOCATION\_EXT', N'ONDEMAND\_TASK\_QUEUE',

N'PARALLEL\_REDO\_DRAIN\_WORKER', N'PARALLEL\_REDO\_LOG\_CACHE', N'PARALLEL\_REDO\_TRAN\_LIST',

N'PARALLEL\_REDO\_WORKER\_SYNC', N'PARALLEL\_REDO\_WORKER\_WAIT\_WORK',

N'PREEMPTIVE\_HADR\_LEASE\_MECHANISM', N'PREEMPTIVE\_SP\_SERVER\_DIAGNOSTICS',

N'PREEMPTIVE\_OS\_LIBRARYOPS', N'PREEMPTIVE\_OS\_COMOPS', N'PREEMPTIVE\_OS\_CRYPTOPS',

N'PREEMPTIVE\_OS\_PIPEOPS', N'PREEMPTIVE\_OS\_AUTHENTICATIONOPS',

N'PREEMPTIVE\_OS\_GENERICOPS', N'PREEMPTIVE\_OS\_VERIFYTRUST',

N'PREEMPTIVE\_OS\_DELETESECURITYCONTEXT', N'PREEMPTIVE\_OS\_REPORTEVENT',

N'PREEMPTIVE\_OS\_FILEOPS', N'PREEMPTIVE\_OS\_DEVICEOPS', N'PREEMPTIVE\_OS\_QUERYREGISTRY',

N'PREEMPTIVE\_OS\_WRITEFILE', N'PREEMPTIVE\_OS\_WRITEFILEGATHER',

N'PREEMPTIVE\_XE\_CALLBACKEXECUTE', N'PREEMPTIVE\_XE\_DISPATCHER',

N'PREEMPTIVE\_XE\_GETTARGETSTATE', N'PREEMPTIVE\_XE\_SESSIONCOMMIT',

N'PREEMPTIVE\_XE\_TARGETINIT', N'PREEMPTIVE\_XE\_TARGETFINALIZE',

N'POPULATE\_LOCK\_ORDINALS',

N'PWAIT\_ALL\_COMPONENTS\_INITIALIZED', N'PWAIT\_DIRECTLOGCONSUMER\_GETNEXT',

N'PWAIT\_EXTENSIBILITY\_CLEANUP\_TASK',

N'QDS\_PERSIST\_TASK\_MAIN\_LOOP\_SLEEP', N'QDS\_ASYNC\_QUEUE',

N'QDS\_CLEANUP\_STALE\_QUERIES\_TASK\_MAIN\_LOOP\_SLEEP', N'REQUEST\_FOR\_DEADLOCK\_SEARCH',

N'RESOURCE\_QUEUE', N'SERVER\_IDLE\_CHECK', N'SLEEP\_BPOOL\_FLUSH', N'SLEEP\_DBSTARTUP',

N'SLEEP\_DCOMSTARTUP', N'SLEEP\_MASTERDBREADY', N'SLEEP\_MASTERMDREADY',

N'SLEEP\_MASTERUPGRADED', N'SLEEP\_MSDBSTARTUP', N'SLEEP\_SYSTEMTASK', N'SLEEP\_TASK',

N'SLEEP\_TEMPDBSTARTUP', N'SNI\_HTTP\_ACCEPT', N'SOS\_WORK\_DISPATCHER',

N'SP\_SERVER\_DIAGNOSTICS\_SLEEP', N'SOS\_WORKER\_MIGRATION', N'VDI\_CLIENT\_OTHER',

N'SQLTRACE\_BUFFER\_FLUSH', N'SQLTRACE\_INCREMENTAL\_FLUSH\_SLEEP', N'SQLTRACE\_WAIT\_ENTRIES',

N'STARTUP\_DEPENDENCY\_MANAGER',

N'WAIT\_FOR\_RESULTS', N'WAITFOR', N'WAITFOR\_TASKSHUTDOWN', N'WAIT\_XTP\_HOST\_WAIT',

N'WAIT\_XTP\_OFFLINE\_CKPT\_NEW\_LOG', N'WAIT\_XTP\_CKPT\_CLOSE', N'WAIT\_XTP\_RECOVERY',

N'XE\_BUFFERMGR\_ALLPROCESSED\_EVENT', N'XE\_DISPATCHER\_JOIN',

N'XE\_DISPATCHER\_WAIT', N'XE\_LIVE\_TARGET\_TVF', N'XE\_TIMER\_EVENT')

AND waiting\_tasks\_count > 0)

SELECT

MAX (W1.wait\_type) AS [WaitType],

CAST (MAX (W1.Percentage) AS DECIMAL (5,2)) AS [Wait Percentage],

CAST ((MAX (W1.WaitS) / MAX (W1.WaitCount)) AS DECIMAL (16,4)) AS [AvgWait\_Sec],

CAST ((MAX (W1.ResourceS) / MAX (W1.WaitCount)) AS DECIMAL (16,4)) AS [AvgRes\_Sec],

CAST ((MAX (W1.SignalS) / MAX (W1.WaitCount)) AS DECIMAL (16,4)) AS [AvgSig\_Sec],

CAST (MAX (W1.WaitS) AS DECIMAL (16,2)) AS [Wait\_Sec],

CAST (MAX (W1.ResourceS) AS DECIMAL (16,2)) AS [Resource\_Sec],

CAST (MAX (W1.SignalS) AS DECIMAL (16,2)) AS [Signal\_Sec],

MAX (W1.WaitCount) AS [Wait Count],

CAST (N'https://urldefense.com/v3/\_\_https://www.sqlskills.com/help/waits/\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQT1wS0A\_Q$ ' + W1.wait\_type AS XML) AS [Help/Info URL]

FROM Waits AS W1

INNER JOIN Waits AS W2

ON W2.RowNum <= W1.RowNum

GROUP BY W1.RowNum, W1.wait\_type

HAVING SUM (W2.Percentage) - MAX (W1.Percentage) < 99 -- percentage threshold

OPTION (RECOMPILE);

------

-- Cumulative wait stats are not as useful on an idle instance that is not under load or performance pressure

-- SQL Server Wait Types Library

-- https://urldefense.com/v3/\_\_https://bit.ly/2ePzYO2\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQRJgdFOlQ$

-- The SQL Server Wait Type Repository

-- https://urldefense.com/v3/\_\_https://bit.ly/1afzfjC\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQQ\_7\_8LBg$

-- Wait statistics, or please tell me where it hurts

-- https://urldefense.com/v3/\_\_https://bit.ly/2wsQHQE\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQT1koXp0A$

-- SQL Server 2005 Performance Tuning using the Waits and Queues

-- https://urldefense.com/v3/\_\_https://bit.ly/1o2NFoF\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQRf4TFtWg$

-- sys.dm\_os\_wait\_stats (Transact-SQL)

-- https://urldefense.com/v3/\_\_https://bit.ly/2Hjq9Yl\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQSNVrISeA$

-- Get a count of SQL connections by IP address (Query 42) (Connection Counts by IP Address)

SELECT ec.client\_net\_address, es.[program\_name], es.[host\_name], es.login\_name,

COUNT(ec.session\_id) AS [connection count]

FROM sys.dm\_exec\_sessions AS es WITH (NOLOCK)

INNER JOIN sys.dm\_exec\_connections AS ec WITH (NOLOCK)

ON es.session\_id = ec.session\_id

GROUP BY ec.client\_net\_address, es.[program\_name], es.[host\_name], es.login\_name

ORDER BY ec.client\_net\_address, es.[program\_name] OPTION (RECOMPILE);

------

-- This helps you figure where your database load is coming from

-- and verifies connectivity from other machines

-- Solving Connectivity errors to SQL Server

-- https://urldefense.com/v3/\_\_https://bit.ly/2EgzoD0\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQQS2lAAZg$

-- Get Average Task Counts (run multiple times) (Query 43) (Avg Task Counts)

SELECT AVG(current\_tasks\_count) AS [Avg Task Count],

AVG(work\_queue\_count) AS [Avg Work Queue Count],

AVG(runnable\_tasks\_count) AS [Avg Runnable Task Count],

AVG(pending\_disk\_io\_count) AS [Avg Pending DiskIO Count]

FROM sys.dm\_os\_schedulers WITH (NOLOCK)

WHERE scheduler\_id < 255 OPTION (RECOMPILE);

------

-- Sustained values above 10 suggest further investigation in that area

-- High Avg Task Counts are often caused by blocking/deadlocking or other resource contention

-- Sustained values above 1 suggest further investigation in that area

-- High Avg Runnable Task Counts are a good sign of CPU pressure

-- High Avg Pending DiskIO Counts are a sign of disk pressure

-- How to Do Some Very Basic SQL Server Monitoring

-- https://urldefense.com/v3/\_\_https://bit.ly/30IRla0\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQSmPzS7nA$

-- Detect blocking (run multiple times) (Query 44) (Detect Blocking)

SELECT t1.resource\_type AS [lock type], DB\_NAME(resource\_database\_id) AS [database],

t1.resource\_associated\_entity\_id AS [blk object],t1.request\_mode AS [lock req], -- lock requested

t1.request\_session\_id AS [waiter sid], t2.wait\_duration\_ms AS [wait time], -- spid of waiter

(SELECT [text] FROM sys.dm\_exec\_requests AS r WITH (NOLOCK) -- get sql for waiter

CROSS APPLY sys.dm\_exec\_sql\_text(r.[sql\_handle])

WHERE r.session\_id = t1.request\_session\_id) AS [waiter\_batch],

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

(CASE WHEN r.statement\_end\_offset = -1

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

ELSE r.statement\_end\_offset END - r.statement\_start\_offset)/2)

FROM sys.dm\_exec\_requests AS r WITH (NOLOCK)

CROSS APPLY sys.dm\_exec\_sql\_text(r.[sql\_handle]) AS qt

WHERE r.session\_id = t1.request\_session\_id) AS [waiter\_stmt], -- statement blocked

t2.blocking\_session\_id AS [blocker sid], -- spid of blocker

(SELECT [text] FROM sys.sysprocesses AS p -- get sql for blocker

CROSS APPLY sys.dm\_exec\_sql\_text(p.[sql\_handle])

WHERE p.spid = t2.blocking\_session\_id) AS [blocker\_batch]

FROM sys.dm\_tran\_locks AS t1 WITH (NOLOCK)

INNER JOIN sys.dm\_os\_waiting\_tasks AS t2 WITH (NOLOCK)

ON t1.lock\_owner\_address = t2.resource\_address OPTION (RECOMPILE);

------

-- Helps troubleshoot blocking and deadlocking issues

-- The results will change from second to second on a busy system

-- You should run this query multiple times when you see signs of blocking

-- Show page level contention (Query 45) (Page Contention)

SELECT er.session\_id, er.wait\_type, er.wait\_resource,

OBJECT\_NAME(pinfo.[object\_id], pinfo.database\_id) AS [object\_name],

er.blocking\_session\_id, er.command,

SUBSTRING(st.text, (er.statement\_start\_offset/2)+1,

((CASE er.statement\_end\_offset

WHEN -1 THEN DATALENGTH(st.text)

ELSE er.statement\_end\_offset

END - er.statement\_start\_offset)/2) + 1) AS statement\_text,

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

pinfo.[file\_id], pinfo.page\_id, pinfo.[object\_id], pinfo.index\_id, pinfo.page\_type\_desc

FROM sys.dm\_exec\_requests AS er WITH (NOLOCK)

CROSS APPLY sys.dm\_exec\_sql\_text(er.sql\_handle) AS st

CROSS APPLY sys.fn\_PageResCracker(er.page\_resource) AS r

CROSS APPLY sys.dm\_db\_page\_info(r.[db\_id], r.[file\_id], r.page\_id, N'DETAILED') AS pinfo

WHERE er.wait\_type LIKE N'%page%' OPTION (RECOMPILE);

------

-- sys.fn\_PageResCracker (Transact-SQL)

-- https://urldefense.com/v3/\_\_https://bit.ly/3sgwp9B\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQRwb\_DjjQ$

-- Get CPU Utilization History for last 256 minutes (in one minute intervals) (Query 46) (CPU Utilization History)

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

SELECT TOP(256) 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 WITH (NOLOCK)

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

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

ORDER BY record\_id DESC OPTION (RECOMPILE);

------

-- Look at the trend over the entire period

-- Also look at high sustained 'Other Process' CPU Utilization values

-- Note: This query sometimes gives inaccurate results (negative values)

-- on high core count (> 64 cores) systems

-- Get top total worker time queries for entire instance (Query 47) (Top Worker Time Queries)

SELECT TOP(50) DB\_NAME(t.[dbid]) AS [Database Name],

REPLACE(REPLACE(LEFT(t.[text], 255), CHAR(10),''), CHAR(13),'') AS [Short Query Text],

qs.total\_worker\_time AS [Total Worker Time], qs.min\_worker\_time AS [Min Worker Time],

qs.total\_worker\_time/qs.execution\_count AS [Avg Worker Time],

qs.max\_worker\_time AS [Max Worker Time],

qs.min\_elapsed\_time AS [Min Elapsed Time],

qs.total\_elapsed\_time/qs.execution\_count AS [Avg Elapsed Time],

qs.max\_elapsed\_time AS [Max Elapsed Time],

qs.min\_logical\_reads AS [Min Logical Reads],

qs.total\_logical\_reads/qs.execution\_count AS [Avg Logical Reads],

qs.max\_logical\_reads AS [Max Logical Reads],

qs.execution\_count AS [Execution Count],

CASE WHEN CONVERT(nvarchar(max), qp.query\_plan) COLLATE Latin1\_General\_BIN2 LIKE N'%<MissingIndexes>%' THEN 1 ELSE 0 END AS [Has Missing Index],

qs.creation\_time AS [Creation Time]

--,t.[text] AS [Query Text], qp.query\_plan AS [Query Plan] -- uncomment out these columns if not copying results to Excel

FROM sys.dm\_exec\_query\_stats AS qs WITH (NOLOCK)

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

CROSS APPLY sys.dm\_exec\_query\_plan(plan\_handle) AS qp

ORDER BY qs.total\_worker\_time DESC OPTION (RECOMPILE);

------

-- Helps you find the most expensive queries from a CPU perspective across the entire instance

-- Can also help track down parameter sniffing issues

-- Page Life Expectancy (PLE) value for each NUMA node in current instance (Query 48) (PLE by NUMA Node)

SELECT @@SERVERNAME AS [Server Name], RTRIM([object\_name]) AS [Object Name],

instance\_name, cntr\_value AS [Page Life Expectancy], GETDATE() AS [System Time]

FROM sys.dm\_os\_performance\_counters WITH (NOLOCK)

WHERE [object\_name] LIKE N'%Buffer Node%' -- Handles named instances

AND counter\_name = N'Page life expectancy' OPTION (RECOMPILE);

------

-- PLE is a good measurement of internal memory pressure

-- Higher PLE is better. Watch the trend over time, not the absolute value

-- This will only return one row for non-NUMA systems

-- Page Life Expectancy isn’t what you think…

-- https://urldefense.com/v3/\_\_https://bit.ly/2EgynLa\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQSWsrDs0Q$

-- Memory Grants Pending value for current instance (Query 49) (Memory Grants Pending)

SELECT @@SERVERNAME AS [Server Name], RTRIM([object\_name]) AS [Object Name], cntr\_value AS [Memory Grants Pending]

FROM sys.dm\_os\_performance\_counters WITH (NOLOCK)

WHERE [object\_name] LIKE N'%Memory Manager%' -- Handles named instances

AND counter\_name = N'Memory Grants Pending' OPTION (RECOMPILE);

------

-- Run multiple times, and run periodically if you suspect you are under memory pressure

-- Memory Grants Pending above zero for a sustained period is a very strong indicator of internal memory pressure

-- Memory Clerk Usage for instance (Query 50) (Memory Clerk Usage)

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

SELECT TOP(10) mc.[type] AS [Memory Clerk Type],

CAST((SUM(mc.pages\_kb)/1024.0) AS DECIMAL (15,2)) AS [Memory Usage (MB)]

FROM sys.dm\_os\_memory\_clerks AS mc WITH (NOLOCK)

GROUP BY mc.[type]

ORDER BY SUM(mc.pages\_kb) DESC OPTION (RECOMPILE);

------

-- MEMORYCLERK\_SQLBUFFERPOOL was new for SQL Server 2012. It should be your highest consumer of memory

-- CACHESTORE\_SQLCP - SQL Plans

-- These are cached SQL statements or batches that aren't in stored procedures, functions and triggers

-- Watch out for high values for CACHESTORE\_SQLCP

-- Enabling 'optimize for ad hoc workloads' at the instance level can help reduce this

-- Running DBCC FREESYSTEMCACHE ('SQL Plans'); periodically may be required to better control this

-- CACHESTORE\_OBJCP - Object Plans

-- These are compiled plans for stored procedures, functions and triggers

-- If you see very high usage by MEMORYCLERK\_SQLLOGPOOL

-- SQL Server 2019 CU9 added a new command, DBCC FREESYSTEMCACHE ('LogPool');

-- sys.dm\_os\_memory\_clerks (Transact-SQL)

-- https://urldefense.com/v3/\_\_https://bit.ly/2H31xDR\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQTokcH9cg$

-- Find single-use, ad-hoc and prepared queries that are bloating the plan cache (Query 51) (Ad hoc Queries)

SELECT TOP(50) DB\_NAME(t.[dbid]) AS [Database Name],

REPLACE(REPLACE(LEFT(t.[text], 255), CHAR(10),''), CHAR(13),'') AS [Short Query Text],

cp.objtype AS [Object Type], cp.cacheobjtype AS [Cache Object Type],

cp.size\_in\_bytes/1024 AS [Plan Size in KB],

CASE WHEN CONVERT(nvarchar(max), qp.query\_plan) COLLATE Latin1\_General\_BIN2 LIKE N'%<MissingIndexes>%' THEN 1 ELSE 0 END AS [Has Missing Index]

--,t.[text] AS [Query Text], qp.query\_plan AS [Query Plan] -- uncomment out these columns if not copying results to Excel

FROM sys.dm\_exec\_cached\_plans AS cp WITH (NOLOCK)

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

CROSS APPLY sys.dm\_exec\_query\_plan(plan\_handle) AS qp

WHERE cp.cacheobjtype = N'Compiled Plan'

AND cp.objtype IN (N'Adhoc', N'Prepared')

AND cp.usecounts = 1

ORDER BY cp.size\_in\_bytes DESC, DB\_NAME(t.[dbid]) OPTION (RECOMPILE);

------

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

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

-- Running DBCC FREESYSTEMCACHE ('SQL Plans') periodically may be required to better control this

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

-- Plan cache, adhoc workloads and clearing the single-use plan cache bloat

-- https://urldefense.com/v3/\_\_https://bit.ly/2EfYOkl\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQQ7suNS\_A$

-- Get top total logical reads queries for entire instance (Query 52) (Top Logical Reads Queries)

SELECT TOP(50) DB\_NAME(t.[dbid]) AS [Database Name],

REPLACE(REPLACE(LEFT(t.[text], 255), CHAR(10),''), CHAR(13),'') AS [Short Query Text],

qs.total\_logical\_reads AS [Total Logical Reads],

qs.min\_logical\_reads AS [Min Logical Reads],

qs.total\_logical\_reads/qs.execution\_count AS [Avg Logical Reads],

qs.max\_logical\_reads AS [Max Logical Reads],

qs.min\_worker\_time AS [Min Worker Time],

qs.total\_worker\_time/qs.execution\_count AS [Avg Worker Time],

qs.max\_worker\_time AS [Max Worker Time],

qs.min\_elapsed\_time AS [Min Elapsed Time],

qs.total\_elapsed\_time/qs.execution\_count AS [Avg Elapsed Time],

qs.max\_elapsed\_time AS [Max Elapsed Time],

qs.execution\_count AS [Execution Count],

CASE WHEN CONVERT(nvarchar(max), qp.query\_plan) COLLATE Latin1\_General\_BIN2 LIKE N'%<MissingIndexes>%' THEN 1 ELSE 0 END AS [Has Missing Index],

qs.creation\_time AS [Creation Time]

--,t.[text] AS [Complete Query Text], qp.query\_plan AS [Query Plan] -- uncomment out these columns if not copying results to Excel

FROM sys.dm\_exec\_query\_stats AS qs WITH (NOLOCK)

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

CROSS APPLY sys.dm\_exec\_query\_plan(plan\_handle) AS qp

ORDER BY qs.total\_logical\_reads DESC OPTION (RECOMPILE);

------

-- Helps you find the most expensive queries from a memory perspective across the entire instance

-- Can also help track down parameter sniffing issues

-- Get top average elapsed time queries for entire instance (Query 53) (Top Avg Elapsed Time Queries)

SELECT TOP(50) DB\_NAME(t.[dbid]) AS [Database Name],

REPLACE(REPLACE(LEFT(t.[text], 255), CHAR(10),''), CHAR(13),'') AS [Short Query Text],

qs.total\_elapsed\_time/qs.execution\_count AS [Avg Elapsed Time],

qs.min\_elapsed\_time, qs.max\_elapsed\_time, qs.last\_elapsed\_time,

qs.execution\_count AS [Execution Count],

qs.total\_logical\_reads/qs.execution\_count AS [Avg Logical Reads],

qs.total\_physical\_reads/qs.execution\_count AS [Avg Physical Reads],

qs.total\_worker\_time/qs.execution\_count AS [Avg Worker Time],

CASE WHEN CONVERT(nvarchar(max), qp.query\_plan) COLLATE Latin1\_General\_BIN2 LIKE N'%<MissingIndexes>%' THEN 1 ELSE 0 END AS [Has Missing Index],

qs.creation\_time AS [Creation Time]

--,t.[text] AS [Complete Query Text], qp.query\_plan AS [Query Plan] -- uncomment out these columns if not copying results to Excel

FROM sys.dm\_exec\_query\_stats AS qs WITH (NOLOCK)

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

CROSS APPLY sys.dm\_exec\_query\_plan(plan\_handle) AS qp

ORDER BY qs.total\_elapsed\_time/qs.execution\_count DESC OPTION (RECOMPILE);

------

-- Helps you find the highest average elapsed time queries across the entire instance

-- Can also help track down parameter sniffing issues

-- Look at UDF execution statistics (Query 54) (UDF Stats by DB)

SELECT TOP (25) DB\_NAME(database\_id) AS [Database Name],

OBJECT\_NAME(object\_id, database\_id) AS [Function Name],

total\_worker\_time, execution\_count, total\_elapsed\_time,

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

last\_elapsed\_time, last\_execution\_time, cached\_time, [type\_desc]

FROM sys.dm\_exec\_function\_stats WITH (NOLOCK)

ORDER BY total\_worker\_time DESC OPTION (RECOMPILE);

------

-- sys.dm\_exec\_function\_stats (Transact-SQL)

-- https://urldefense.com/v3/\_\_https://bit.ly/2q1Q6BM\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQQwAdjf3g$

-- Showplan Enhancements for UDFs

-- https://urldefense.com/v3/\_\_https://bit.ly/2LVqiQ1\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQSvriMXDg$

-- Look for long duration buffer pool scans (Query 55) (Long Buffer Pool Scans)

EXEC sys.xp\_readerrorlog 0, 1, N'Buffer pool scan took';

------

-- Finds buffer pool scans that took more than 10 seconds in the current SQL Server Error log

-- This should happen much less often in SQL Server 2022

-- Operations that trigger buffer pool scan may run slowly on large-memory computers - SQL Server | Microsoft Docs

-- https://urldefense.com/v3/\_\_https://bit.ly/3QrFC81\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQTAITR07w$

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

-- \*\*\*\* Please switch to a user database that you are interested in! \*\*\*\*\*

--USE YourDatabaseName; -- make sure to change to an actual database on your instance, not the master system database

--GO

-- Individual File Sizes and space available for current database (Query 56) (File Sizes and Space)

SELECT f.[name] AS [File Name] , f.physical\_name AS [Physical Name],

CAST((f.size/128.0) AS DECIMAL(15,2)) AS [Total Size in MB],

CAST((f.size/128.0) AS DECIMAL(15,2)) -

CAST(f.size/128.0 - CAST(FILEPROPERTY(f.name, 'SpaceUsed') AS int)/128.0 AS DECIMAL(15,2))

AS [Used Space in MB],

CAST(f.size/128.0 - CAST(FILEPROPERTY(f.name, 'SpaceUsed') AS int)/128.0 AS DECIMAL(15,2))

AS [Available Space In MB],

f.[file\_id], fg.name AS [Filegroup Name],

f.is\_percent\_growth, f.growth, fg.is\_default, fg.is\_read\_only, fg.is\_autogrow\_all\_files

FROM sys.database\_files AS f WITH (NOLOCK)

LEFT OUTER JOIN sys.filegroups AS fg WITH (NOLOCK)

ON f.data\_space\_id = fg.data\_space\_id

ORDER BY f.[type], f.[file\_id] OPTION (RECOMPILE);

------

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

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

-- In order to get IFI on log files, the auto growth increment should be 64MB or less

-- is\_autogrow\_all\_files was new for SQL Server 2016. Equivalent to TF 1117 for user databases

-- SQL Server 2016: Changes in default behavior for autogrow and allocations for tempdb and user databases

-- https://urldefense.com/v3/\_\_https://bit.ly/2evRZSR\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQT-jaf-dQ$

-- Log space usage for current database (Query 57) (Log Space Usage)

SELECT DB\_NAME(lsu.database\_id) AS [Database Name], db.recovery\_model\_desc AS [Recovery Model],

CAST(lsu.total\_log\_size\_in\_bytes/1048576.0 AS DECIMAL(10, 2)) AS [Total Log Space (MB)],

CAST(lsu.used\_log\_space\_in\_bytes/1048576.0 AS DECIMAL(10, 2)) AS [Used Log Space (MB)],

CAST(lsu.used\_log\_space\_in\_percent AS DECIMAL(10, 2)) AS [Used Log Space %],

CAST(lsu.log\_space\_in\_bytes\_since\_last\_backup/1048576.0 AS DECIMAL(10, 2)) AS [Used Log Space Since Last Backup (MB)],

db.log\_reuse\_wait\_desc

FROM sys.dm\_db\_log\_space\_usage AS lsu WITH (NOLOCK)

INNER JOIN sys.databases AS db WITH (NOLOCK)

ON lsu.database\_id = db.database\_id

OPTION (RECOMPILE);

------

-- Look at log file size and usage, along with the log reuse wait description for the current database

-- sys.dm\_db\_log\_space\_usage (Transact-SQL)

-- https://urldefense.com/v3/\_\_https://bit.ly/2H4MQw9\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQTah5LPdA$

-- Status of last VLF for current database (Query 58) (Last VLF Status)

SELECT TOP(1) DB\_NAME(li.database\_id) AS [Database Name], li.[file\_id],

li.vlf\_size\_mb, li.vlf\_sequence\_number, li.vlf\_active, li.vlf\_status

FROM sys.dm\_db\_log\_info(DB\_ID()) AS li

ORDER BY vlf\_sequence\_number DESC OPTION (RECOMPILE);

------

-- Determine whether you will be able to shrink the transaction log file

-- vlf\_status Values

-- 0 is inactive

-- 1 is initialized but unused

-- 2 is active

-- sys.dm\_db\_log\_info (Transact-SQL)

-- https://urldefense.com/v3/\_\_https://bit.ly/2EQUU1v\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQRjFwNF8g$

-- Get database scoped configuration values for current database (Query 59) (Database-scoped Configurations)

SELECT configuration\_id, name, [value] AS [value\_for\_primary], value\_for\_secondary, is\_value\_default

FROM sys.database\_scoped\_configurations WITH (NOLOCK) OPTION (RECOMPILE);

------

-- This lets you see the value of these new properties for the current database

-- Clear plan cache for current database

-- ALTER DATABASE SCOPED CONFIGURATION CLEAR PROCEDURE\_CACHE;

-- ALTER DATABASE SCOPED CONFIGURATION (Transact-SQL)

-- https://urldefense.com/v3/\_\_https://bit.ly/2sOH7nb\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQQPfGgsBw$

-- New in SQL Server 2022

-- PAUSED\_RESUMABLE\_INDEX\_ABORT\_DURATION\_MINUTES

-- DW\_COMPATIBILITY\_LEVEL

-- EXEC\_QUERY\_STATS\_FOR\_SCALAR\_FUNCTIONS

-- PARAMETER\_SENSITIVE\_PLAN\_OPTIMIZATION

-- ASYNC\_STATS\_UPDATE\_WAIT\_AT\_LOW\_PRIORITY

-- CE\_FEEDBACK

-- MEMORY\_GRANT\_FEEDBACK\_PERSISTENCE

-- MEMORY\_GRANT\_FEEDBACK\_PERCENTILE\_GRANT

-- OPTIMIZED\_PLAN\_FORCING

-- DOP\_FEEDBACK

-- LEDGER\_DIGEST\_STORAGE\_ENDPOINT

-- FORCE\_SHOWPLAN\_RUNTIME\_PARAMETER\_COLLECTION

-- I/O Statistics by file for the current database (Query 60) (IO Stats By File)

SELECT DB\_NAME(DB\_ID()) AS [Database Name], df.name AS [Logical Name], vfs.[file\_id], df.type\_desc,

df.physical\_name AS [Physical Name], CAST(vfs.size\_on\_disk\_bytes/1048576.0 AS DECIMAL(15, 2)) AS [Size on Disk (MB)],

vfs.num\_of\_reads, vfs.num\_of\_writes, vfs.io\_stall\_read\_ms, vfs.io\_stall\_write\_ms,

CAST(100. \* vfs.io\_stall\_read\_ms/(vfs.io\_stall\_read\_ms + vfs.io\_stall\_write\_ms) AS DECIMAL(10,1)) AS [IO Stall Reads Pct],

CAST(100. \* vfs.io\_stall\_write\_ms/(vfs.io\_stall\_write\_ms + vfs.io\_stall\_read\_ms) AS DECIMAL(10,1)) AS [IO Stall Writes Pct],

(vfs.num\_of\_reads + vfs.num\_of\_writes) AS [Writes + Reads],

CAST(vfs.num\_of\_bytes\_read/1048576.0 AS DECIMAL(15, 2)) AS [MB Read],

CAST(vfs.num\_of\_bytes\_written/1048576.0 AS DECIMAL(15, 2)) AS [MB Written],

CAST(100. \* vfs.num\_of\_reads/(vfs.num\_of\_reads + vfs.num\_of\_writes) AS DECIMAL(15,1)) AS [# Reads Pct],

CAST(100. \* vfs.num\_of\_writes/(vfs.num\_of\_reads + vfs.num\_of\_writes) AS DECIMAL(15,1)) AS [# Write Pct],

CAST(100. \* vfs.num\_of\_bytes\_read/(vfs.num\_of\_bytes\_read + vfs.num\_of\_bytes\_written) AS DECIMAL(15,1)) AS [Read Bytes Pct],

CAST(100. \* vfs.num\_of\_bytes\_written/(vfs.num\_of\_bytes\_read + vfs.num\_of\_bytes\_written) AS DECIMAL(15,1)) AS [Written Bytes Pct]

FROM sys.dm\_io\_virtual\_file\_stats(DB\_ID(), NULL) AS vfs

INNER JOIN sys.database\_files AS df WITH (NOLOCK)

ON vfs.[file\_id]= df.[file\_id] OPTION (RECOMPILE);

------

-- This helps you characterize your workload better from an I/O perspective for this database

-- It helps you determine whether you have an OLTP or DW/DSS type of workload

-- Get most frequently executed queries for this database (Query 61) (Query Execution Counts)

SELECT TOP(50) LEFT(t.[text], 50) AS [Short Query Text], qs.execution\_count AS [Execution Count],

qs.total\_logical\_reads AS [Total Logical Reads],

qs.total\_logical\_reads/qs.execution\_count AS [Avg Logical Reads],

qs.total\_worker\_time AS [Total Worker Time],

qs.total\_worker\_time/qs.execution\_count AS [Avg Worker Time],

qs.total\_elapsed\_time AS [Total Elapsed Time],

qs.total\_elapsed\_time/qs.execution\_count AS [Avg Elapsed Time],

CASE WHEN CONVERT(nvarchar(max), qp.query\_plan) COLLATE Latin1\_General\_BIN2 LIKE N'%<MissingIndexes>%' THEN 1 ELSE 0 END AS [Has Missing Index],

CONVERT(nvarchar(25), qs.last\_execution\_time, 20) AS [Last Execution Time],

CONVERT(nvarchar(25), qs.creation\_time, 20) AS [Plan Cached Time]

--,t.[text] AS [Complete Query Text], qp.query\_plan AS [Query Plan] -- uncomment out these columns if not copying results to Excel

FROM sys.dm\_exec\_query\_stats AS qs WITH (NOLOCK)

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

CROSS APPLY sys.dm\_exec\_query\_plan(plan\_handle) AS qp

WHERE t.dbid = DB\_ID()

ORDER BY qs.execution\_count DESC OPTION (RECOMPILE);

------

-- Tells you which cached queries are called the most often

-- This helps you characterize and baseline your workload

-- It also helps you find possible caching opportunities

-- CREATE PROCEDURE (Transact-SQL)

-- https://urldefense.com/v3/\_\_https://bit.ly/3gxcuxG\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQQYh8S0Og$

-- Queries 62 through 68 are the "Bad Man List" for stored procedures

-- Top Cached SPs By Execution Count (Query 62) (SP Execution Counts)

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

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

qs.total\_elapsed\_time/qs.execution\_count AS [Avg Elapsed Time],

qs.total\_worker\_time/qs.execution\_count AS [Avg Worker Time],

qs.total\_logical\_reads/qs.execution\_count AS [Avg Logical Reads],

CASE WHEN CONVERT(nvarchar(max), qp.query\_plan) COLLATE Latin1\_General\_BIN2 LIKE N'%<MissingIndexes>%' THEN 1 ELSE 0 END AS [Has Missing Index],

CONVERT(nvarchar(25), qs.last\_execution\_time, 20) AS [Last Execution Time],

CONVERT(nvarchar(25), qs.cached\_time, 20) AS [Plan Cached Time]

-- ,qp.query\_plan AS [Query Plan] -- Uncomment if you want the Query Plan

FROM sys.procedures AS p WITH (NOLOCK)

INNER JOIN sys.dm\_exec\_procedure\_stats AS qs WITH (NOLOCK)

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

CROSS APPLY sys.dm\_exec\_query\_plan(qs.plan\_handle) AS qp

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

AND DATEDIFF(Minute, qs.cached\_time, GETDATE()) > 0

ORDER BY qs.execution\_count DESC OPTION (RECOMPILE);

------

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

-- This helps you characterize and baseline your workload

-- It also helps you find possible caching opportunities

-- Top Cached SPs By Avg Elapsed Time (Query 63) (SP Avg Elapsed Time)

SELECT TOP(25) p.name AS [SP Name], qs.min\_elapsed\_time, qs.total\_elapsed\_time/qs.execution\_count AS [avg\_elapsed\_time],

qs.max\_elapsed\_time, qs.last\_elapsed\_time, qs.total\_elapsed\_time, qs.execution\_count,

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

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

qs.total\_worker\_time AS [TotalWorkerTime],

CASE WHEN CONVERT(nvarchar(max), qp.query\_plan) COLLATE Latin1\_General\_BIN2 LIKE N'%<MissingIndexes>%' THEN 1 ELSE 0 END AS [Has Missing Index],

CONVERT(nvarchar(25), qs.last\_execution\_time, 20) AS [Last Execution Time],

CONVERT(nvarchar(25), qs.cached\_time, 20) AS [Plan Cached Time]

-- ,qp.query\_plan AS [Query Plan] -- Uncomment if you want the Query Plan

FROM sys.procedures AS p WITH (NOLOCK)

INNER JOIN sys.dm\_exec\_procedure\_stats AS qs WITH (NOLOCK)

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

CROSS APPLY sys.dm\_exec\_query\_plan(qs.plan\_handle) AS qp

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

AND DATEDIFF(Minute, qs.cached\_time, GETDATE()) > 0

ORDER BY avg\_elapsed\_time DESC OPTION (RECOMPILE);

------

-- This helps you find high average elapsed time cached stored procedures that

-- may be easy to optimize with standard query tuning techniques

-- Top Cached SPs By Total Worker time. Worker time relates to CPU cost (Query 64) (SP Worker Time)

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(Minute, qs.cached\_time, GETDATE()), 0) AS [Calls/Minute],

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

CASE WHEN CONVERT(nvarchar(max), qp.query\_plan) COLLATE Latin1\_General\_BIN2 LIKE N'%<MissingIndexes>%' THEN 1 ELSE 0 END AS [Has Missing Index],

CONVERT(nvarchar(25), qs.last\_execution\_time, 20) AS [Last Execution Time],

CONVERT(nvarchar(25), qs.cached\_time, 20) AS [Plan Cached Time]

-- ,qp.query\_plan AS [Query Plan] -- Uncomment if you want the Query Plan

FROM sys.procedures AS p WITH (NOLOCK)

INNER JOIN sys.dm\_exec\_procedure\_stats AS qs WITH (NOLOCK)

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

CROSS APPLY sys.dm\_exec\_query\_plan(qs.plan\_handle) AS qp

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

AND DATEDIFF(Minute, qs.cached\_time, GETDATE()) > 0

ORDER BY qs.total\_worker\_time DESC OPTION (RECOMPILE);

------

-- 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. Logical reads relate to memory pressure (Query 65) (SP Logical Reads)

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(Minute, qs.cached\_time, GETDATE()), 0) AS [Calls/Minute],

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

CASE WHEN CONVERT(nvarchar(max), qp.query\_plan) COLLATE Latin1\_General\_BIN2 LIKE N'%<MissingIndexes>%' THEN 1 ELSE 0 END AS [Has Missing Index],

CONVERT(nvarchar(25), qs.last\_execution\_time, 20) AS [Last Execution Time],

CONVERT(nvarchar(25), qs.cached\_time, 20) AS [Plan Cached Time]

-- ,qp.query\_plan AS [Query Plan] -- Uncomment if you want the Query Plan

FROM sys.procedures AS p WITH (NOLOCK)

INNER JOIN sys.dm\_exec\_procedure\_stats AS qs WITH (NOLOCK)

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

CROSS APPLY sys.dm\_exec\_query\_plan(qs.plan\_handle) AS qp

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

AND DATEDIFF(Minute, qs.cached\_time, GETDATE()) > 0

ORDER BY qs.total\_logical\_reads DESC OPTION (RECOMPILE);

------

-- 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. Physical reads relate to disk read I/O pressure (Query 66) (SP Physical Reads)

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],

CASE WHEN CONVERT(nvarchar(max), qp.query\_plan) COLLATE Latin1\_General\_BIN2 LIKE N'%<MissingIndexes>%' THEN 1 ELSE 0 END AS [Has Missing Index],

CONVERT(nvarchar(25), qs.last\_execution\_time, 20) AS [Last Execution Time],

CONVERT(nvarchar(25), qs.cached\_time, 20) AS [Plan Cached Time]

-- ,qp.query\_plan AS [Query Plan] -- Uncomment if you want the Query Plan

FROM sys.procedures AS p WITH (NOLOCK)

INNER JOIN sys.dm\_exec\_procedure\_stats AS qs WITH (NOLOCK)

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

CROSS APPLY sys.dm\_exec\_query\_plan(qs.plan\_handle) AS qp

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

AND qs.total\_physical\_reads > 0

ORDER BY qs.total\_physical\_reads DESC, qs.total\_logical\_reads DESC OPTION (RECOMPILE);

------

-- 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 (Query 67) (SP Logical Writes)

-- 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(Minute, qs.cached\_time, GETDATE()), 0) AS [Calls/Minute],

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

CASE WHEN CONVERT(nvarchar(max), qp.query\_plan) COLLATE Latin1\_General\_BIN2 LIKE N'%<MissingIndexes>%' THEN 1 ELSE 0 END AS [Has Missing Index],

CONVERT(nvarchar(25), qs.last\_execution\_time, 20) AS [Last Execution Time],

CONVERT(nvarchar(25), qs.cached\_time, 20) AS [Plan Cached Time]

-- ,qp.query\_plan AS [Query Plan] -- Uncomment if you want the Query Plan

FROM sys.procedures AS p WITH (NOLOCK)

INNER JOIN sys.dm\_exec\_procedure\_stats AS qs WITH (NOLOCK)

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

CROSS APPLY sys.dm\_exec\_query\_plan(qs.plan\_handle) AS qp

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

AND qs.total\_logical\_writes > 0

AND DATEDIFF(Minute, qs.cached\_time, GETDATE()) > 0

ORDER BY qs.total\_logical\_writes DESC OPTION (RECOMPILE);

------

-- 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

-- Cached SPs Missing Indexes by Execution Count (Query 68) (SP Missing Index)

SELECT TOP(25) p.name AS [SP Name], qs.execution\_count AS [Execution Count],

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

qs.total\_elapsed\_time/qs.execution\_count AS [Avg Elapsed Time],

qs.total\_worker\_time/qs.execution\_count AS [Avg Worker Time],

qs.total\_logical\_reads/qs.execution\_count AS [Avg Logical Reads],

CONVERT(nvarchar(25), qs.last\_execution\_time, 20) AS [Last Execution Time],

CONVERT(nvarchar(25), qs.cached\_time, 20) AS [Plan Cached Time]

-- ,qp.query\_plan AS [Query Plan] -- Uncomment if you want the Query Plan

FROM sys.procedures AS p WITH (NOLOCK)

INNER JOIN sys.dm\_exec\_procedure\_stats AS qs WITH (NOLOCK)

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

CROSS APPLY sys.dm\_exec\_query\_plan(qs.plan\_handle) AS qp

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

AND DATEDIFF(Minute, qs.cached\_time, GETDATE()) > 0

AND CONVERT(nvarchar(max), qp.query\_plan) COLLATE Latin1\_General\_BIN2 LIKE N'%<MissingIndexes>%'

ORDER BY qs.execution\_count DESC OPTION (RECOMPILE);

------

-- This helps you find the most frequently executed cached stored procedures that have missing index warnings

-- This can often help you find index tuning candidates

-- Lists the top statements by average input/output usage for the current database (Query 69) (Top IO Statements)

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

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

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 WITH (NOLOCK)

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

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

ORDER BY [Avg IO] DESC OPTION (RECOMPILE);

------

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

-- Possible Bad NC Indexes (writes > reads) (Query 70) (Bad NC Indexes)

SELECT SCHEMA\_NAME(o.[schema\_id]) AS [Schema Name],

OBJECT\_NAME(s.[object\_id]) AS [Table Name],

i.name AS [Index Name], i.index\_id,

i.is\_disabled, i.is\_hypothetical, i.has\_filter, i.fill\_factor,

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

s.user\_updates - (s.user\_seeks + s.user\_scans + s.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

INNER JOIN sys.objects AS o WITH (NOLOCK)

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

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

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

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

AND i.index\_id > 1 AND i.[type\_desc] = N'NONCLUSTERED'

AND i.is\_primary\_key = 0 AND i.is\_unique\_constraint = 0 AND i.is\_unique = 0

ORDER BY [Difference] DESC, [Total Writes] DESC, [Total Reads] ASC OPTION (RECOMPILE);

------

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

-- Consider your complete workload, and how long your instance has been running

-- Investigate further before dropping an index!

-- Missing Indexes for current database by Index Advantage (Query 71) (Missing Indexes)

SELECT CONVERT(decimal(18,2), migs.user\_seeks \* migs.avg\_total\_user\_cost \* (migs.avg\_user\_impact \* 0.01)) AS [index\_advantage],

CONVERT(nvarchar(25), migs.last\_user\_seek, 20) AS [last\_user\_seek],

mid.[statement] AS [Database.Schema.Table],

COUNT(1) OVER(PARTITION BY mid.[statement]) AS [missing\_indexes\_for\_table],

COUNT(1) OVER(PARTITION BY mid.[statement], mid.equality\_columns) AS [similar\_missing\_indexes\_for\_table],

mid.equality\_columns, mid.inequality\_columns, mid.included\_columns, migs.user\_seeks,

CONVERT(decimal(18,2), migs.avg\_total\_user\_cost) AS [avg\_total\_user\_,cost], migs.avg\_user\_impact,

REPLACE(REPLACE(LEFT(st.[text], 255), CHAR(10),''), CHAR(13),'') AS [Short Query Text],

OBJECT\_NAME(mid.[object\_id]) AS [Table Name], p.rows AS [Table Rows]

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

INNER JOIN sys.dm\_db\_missing\_index\_group\_stats\_query AS migs WITH(NOLOCK)

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

CROSS APPLY sys.dm\_exec\_sql\_text(migs.last\_sql\_handle) AS st

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

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

INNER JOIN sys.partitions AS p WITH (NOLOCK)

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

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

AND p.index\_id < 2

ORDER BY index\_advantage DESC OPTION (RECOMPILE);

------

-- Look at index advantage, 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!!!

-- Håkan Winther has given me some great suggestions for this query

-- Find missing index warnings for cached plans in the current database (Query 72) (Missing Index Warnings)

-- Note: This query could take some time on a busy instance

SELECT TOP(25) OBJECT\_NAME(objectid) AS [ObjectName],

cp.objtype, cp.usecounts, cp.size\_in\_bytes, qp.query\_plan

FROM sys.dm\_exec\_cached\_plans AS cp WITH (NOLOCK)

CROSS APPLY sys.dm\_exec\_query\_plan(cp.plan\_handle) AS qp

WHERE CAST(qp.query\_plan AS NVARCHAR(MAX)) LIKE N'%MissingIndex%'

AND qp.dbid = DB\_ID()

ORDER BY cp.usecounts DESC OPTION (RECOMPILE);

------

-- Helps you connect missing indexes to specific stored procedures or queries

-- This can help you decide whether to add them or not

-- Breaks down buffers used by current database by object (table, index) in the buffer cache (Query 73) (Buffer Usage)

-- Note: This query could take some time on a busy instance

SELECT fg.name AS [Filegroup Name], SCHEMA\_NAME(o.Schema\_ID) AS [Schema Name],

OBJECT\_NAME(p.[object\_id]) AS [Object Name], p.index\_id,

CAST(COUNT(\*)/128.0 AS DECIMAL(10, 2)) AS [Buffer size(MB)],

COUNT(\*) AS [BufferCount], p.[Rows] AS [Row Count],

p.data\_compression\_desc AS [Compression Type]

FROM sys.allocation\_units AS a WITH (NOLOCK)

INNER JOIN sys.dm\_os\_buffer\_descriptors AS b WITH (NOLOCK)

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

INNER JOIN sys.partitions AS p WITH (NOLOCK)

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

INNER JOIN sys.objects AS o WITH (NOLOCK)

ON p.object\_id = o.object\_id

INNER JOIN sys.database\_files AS f WITH (NOLOCK)

ON b.file\_id = f.file\_id

INNER JOIN sys.filegroups AS fg WITH (NOLOCK)

ON f.data\_space\_id = fg.data\_space\_id

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

AND p.[object\_id] > 100

AND OBJECT\_NAME(p.[object\_id]) NOT LIKE N'plan\_%'

AND OBJECT\_NAME(p.[object\_id]) NOT LIKE N'sys%'

AND OBJECT\_NAME(p.[object\_id]) NOT LIKE N'xml\_index\_nodes%'

GROUP BY fg.name, o.Schema\_ID, p.[object\_id], p.index\_id,

p.data\_compression\_desc, p.[Rows]

ORDER BY [BufferCount] DESC OPTION (RECOMPILE);

------

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

-- It can help identify possible candidates for data compression

-- Get Schema names, Table names, object size, row counts, and compression status for clustered index or heap (Query 74) (Table Sizes)

SELECT DB\_NAME(DB\_ID()) AS [Database Name], SCHEMA\_NAME(o.Schema\_ID) AS [Schema Name],

OBJECT\_NAME(p.object\_id) AS [Table Name],

CAST(SUM(ps.reserved\_page\_count) \* 8.0 / 1024 AS DECIMAL(19,2)) AS [Object Size (MB)],

SUM(p.Rows) AS [Row Count],

p.data\_compression\_desc AS [Compression Type]

FROM sys.objects AS o WITH (NOLOCK)

INNER JOIN sys.partitions AS p WITH (NOLOCK)

ON p.object\_id = o.object\_id

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

ON p.object\_id = ps.object\_id

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

AND p.index\_id < 2 -- ignore the partitions from the non-clustered indexes if any

AND o.type\_desc = N'USER\_TABLE'

GROUP BY SCHEMA\_NAME(o.Schema\_ID), p.object\_id, ps.reserved\_page\_count, p.data\_compression\_desc

ORDER BY SUM(ps.reserved\_page\_count) DESC, SUM(p.Rows) DESC OPTION (RECOMPILE);

------

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

-- Get some key table properties (Query 75) (Table Properties)

SELECT OBJECT\_NAME(t.[object\_id]) AS [ObjectName], p.[rows] AS [Table Rows], p.index\_id,

p.data\_compression\_desc AS [Index Data Compression],

t.create\_date, t.lock\_on\_bulk\_load, t.is\_replicated, t.has\_replication\_filter,

t.is\_tracked\_by\_cdc, t.lock\_escalation\_desc, t.is\_filetable,

t.is\_memory\_optimized, t.durability\_desc,

t.temporal\_type\_desc, t.is\_remote\_data\_archive\_enabled, t.is\_external -- new for SQL Server 2016

FROM sys.tables AS t WITH (NOLOCK)

INNER JOIN sys.partitions AS p WITH (NOLOCK)

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

WHERE OBJECT\_NAME(t.[object\_id]) NOT LIKE N'sys%'

ORDER BY OBJECT\_NAME(t.[object\_id]), p.index\_id OPTION (RECOMPILE);

------

-- Gives you some good information about your tables

-- is\_memory\_optimized and durability\_desc were new in SQL Server 2014

-- temporal\_type\_desc, is\_remote\_data\_archive\_enabled, is\_external were new in SQL Server 2016

-- sys.tables (Transact-SQL)

-- https://urldefense.com/v3/\_\_https://bit.ly/2Gk7998\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQQC-IhVOA$

-- When were Statistics last updated on all indexes? (Query 76) (Statistics Update)

SELECT SCHEMA\_NAME(o.Schema\_ID) + N'.' + o.[NAME] AS [Object Name], o.[type\_desc] AS [Object Type],

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, s.is\_incremental, s.is\_temporary,

s.has\_persisted\_sample, sp.persisted\_sample\_percent,

(sp.rows\_sampled \* 100)/sp.rows AS [Actual Sample Percent], sp.modification\_counter,

st.row\_count, st.used\_page\_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]

CROSS APPLY sys.dm\_db\_stats\_properties(s.object\_id, s.stats\_id) AS sp

WHERE o.[type] IN ('U', 'V')

AND st.row\_count > 0

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

------

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

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

-- sys.stats (Transact-SQL)

-- https://urldefense.com/v3/\_\_https://bit.ly/2GyAxrn\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQQCgLxZPA$

-- UPDATEs to Statistics (Erin Stellato)

-- https://urldefense.com/v3/\_\_https://bit.ly/2vhrYQy\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQQhGkE9Kg$

-- Look at most frequently modified indexes and statistics (Query 77) (Volatile Indexes)

SELECT o.[name] AS [Object Name], o.[object\_id], o.[type\_desc], s.[name] AS [Statistics Name],

s.stats\_id, s.no\_recompute, s.auto\_created, s.is\_incremental, s.is\_temporary,

sp.modification\_counter, sp.[rows], sp.rows\_sampled, sp.last\_updated

FROM sys.objects AS o WITH (NOLOCK)

INNER JOIN sys.stats AS s WITH (NOLOCK)

ON s.object\_id = o.object\_id

CROSS APPLY sys.dm\_db\_stats\_properties(s.object\_id, s.stats\_id) AS sp

WHERE o.[type\_desc] NOT IN (N'SYSTEM\_TABLE', N'INTERNAL\_TABLE')

AND sp.modification\_counter > 0

ORDER BY sp.modification\_counter DESC, o.name OPTION (RECOMPILE);

------

-- This helps you understand your workload and make better decisions about

-- things like data compression and adding new indexes to a table

-- Get fragmentation info for all indexes above a certain size in the current database (Query 78) (Index Fragmentation)

-- Note: This query could take some time on a very large database

SELECT DB\_NAME(ps.database\_id) AS [Database Name], SCHEMA\_NAME(o.[schema\_id]) AS [Schema Name],

OBJECT\_NAME(ps.OBJECT\_ID) AS [Object Name], i.[name] AS [Index Name], ps.index\_id, ps.index\_type\_desc,

CAST(ps.avg\_fragmentation\_in\_percent AS DECIMAL (15,3)) AS [Avg Fragmentation in Pct],

ps.fragment\_count, ps.page\_count, i.fill\_factor, i.has\_filter, i.filter\_definition, i.[allow\_page\_locks]

FROM sys.dm\_db\_index\_physical\_stats(DB\_ID(),NULL, NULL, NULL , N'LIMITED') AS ps

INNER JOIN sys.indexes AS i WITH (NOLOCK)

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

AND ps.index\_id = i.index\_id

INNER JOIN sys.objects AS o WITH (NOLOCK)

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

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

AND ps.page\_count > 2500

ORDER BY ps.avg\_fragmentation\_in\_percent DESC OPTION (RECOMPILE);

------

-- Helps determine whether you have framentation in your relational indexes

-- and how effective your index maintenance strategy is

--- Index Read/Write stats (all tables in current DB) ordered by Reads (Query 79) (Overall Index Usage - Reads)

SELECT SCHEMA\_NAME(t.[schema\_id]) AS [SchemaName], OBJECT\_NAME(i.[object\_id]) AS [ObjectName],

i.[name] AS [IndexName], i.index\_id, i.[type\_desc] AS [Index Type],

s.user\_seeks, s.user\_scans, s.user\_lookups,

s.user\_seeks + s.user\_scans + s.user\_lookups AS [Total Reads],

s.user\_updates AS [Writes],

i.fill\_factor AS [Fill Factor], i.has\_filter, i.filter\_definition,

s.last\_user\_scan, s.last\_user\_lookup, s.last\_user\_seek, i.[allow\_page\_locks]

FROM sys.indexes AS i WITH (NOLOCK)

LEFT OUTER JOIN sys.dm\_db\_index\_usage\_stats AS s WITH (NOLOCK)

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

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

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

LEFT OUTER JOIN sys.tables AS t WITH (NOLOCK)

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

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

ORDER BY s.user\_seeks + s.user\_scans + s.user\_lookups DESC OPTION (RECOMPILE); -- Order by reads

------

-- Show which indexes in the current database are most active for Reads

--- Index Read/Write stats (all tables in current DB) ordered by Writes (Query 80) (Overall Index Usage - Writes)

SELECT SCHEMA\_NAME(t.[schema\_id]) AS [SchemaName],OBJECT\_NAME(i.[object\_id]) AS [ObjectName],

i.[name] AS [IndexName], i.index\_id, i.[type\_desc] AS [Index Type],

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

i.fill\_factor AS [Fill Factor], i.has\_filter, i.filter\_definition,

s.last\_system\_update, s.last\_user\_update, i.[allow\_page\_locks]

FROM sys.indexes AS i WITH (NOLOCK)

LEFT OUTER JOIN sys.dm\_db\_index\_usage\_stats AS s WITH (NOLOCK)

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

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

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

LEFT OUTER JOIN sys.tables AS t WITH (NOLOCK)

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

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

ORDER BY s.user\_updates DESC OPTION (RECOMPILE); -- Order by writes

------

-- Show which indexes in the current database are most active for Writes

-- Get lock waits for current database (Query 81) (Lock Waits)

SELECT o.name AS [table\_name], i.name AS [index\_name], ios.index\_id, ios.partition\_number,

SUM(ios.row\_lock\_wait\_count) AS [total\_row\_lock\_waits],

SUM(ios.row\_lock\_wait\_in\_ms) AS [total\_row\_lock\_wait\_in\_ms],

SUM(ios.index\_lock\_promotion\_attempt\_count) AS [total index\_lock\_promotion\_attempt\_count],

SUM(ios.index\_lock\_promotion\_count) AS [ios.index\_lock\_promotion\_count],

SUM(ios.page\_lock\_wait\_count) AS [total\_page\_lock\_waits],

SUM(ios.page\_lock\_wait\_in\_ms) AS [total\_page\_lock\_wait\_in\_ms],

SUM(ios.page\_lock\_wait\_in\_ms)+ SUM(row\_lock\_wait\_in\_ms) AS [total\_lock\_wait\_in\_ms]

FROM sys.dm\_db\_index\_operational\_stats(DB\_ID(), NULL, NULL, NULL) AS ios

INNER JOIN sys.objects AS o WITH (NOLOCK)

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

INNER JOIN sys.indexes AS i WITH (NOLOCK)

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

AND ios.index\_id = i.index\_id

WHERE o.[object\_id] > 100

GROUP BY o.name, i.name, ios.index\_id, ios.partition\_number

HAVING SUM(ios.page\_lock\_wait\_in\_ms)+ SUM(row\_lock\_wait\_in\_ms) > 0

ORDER BY total\_lock\_wait\_in\_ms DESC OPTION (RECOMPILE);

------

-- This query is helpful for troubleshooting blocking and deadlocking issues

-- sys.dm\_db\_index\_operational\_stats (Transact-SQL)

-- https://urldefense.com/v3/\_\_https://bit.ly/3l5rGEw\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQTQRfY7sg$

-- Look at UDF execution statistics (Query 82) (UDF Statistics)

SELECT OBJECT\_NAME(object\_id) AS [Function Name], execution\_count,

total\_worker\_time, total\_worker\_time/execution\_count AS [avg\_worker\_time],

total\_logical\_reads, total\_physical\_reads, total\_elapsed\_time,

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

CONVERT(nvarchar(25), last\_execution\_time, 20) AS [Last Execution Time],

CONVERT(nvarchar(25), cached\_time, 20) AS [Plan Cached Time]

FROM sys.dm\_exec\_function\_stats WITH (NOLOCK)

WHERE database\_id = DB\_ID()

ORDER BY total\_worker\_time DESC OPTION (RECOMPILE);

------

-- New for SQL Server 2016

-- Helps you investigate scalar UDF performance issues

-- Does not return information for table valued functions

-- sys.dm\_exec\_function\_stats (Transact-SQL)

-- https://urldefense.com/v3/\_\_https://bit.ly/2q1Q6BM\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQQwAdjf3g$

-- Determine which scalar UDFs are in-lineable (Query 83) (Inlineable UDFs)

SELECT OBJECT\_NAME(m.object\_id) AS [Function Name], is\_inlineable, inline\_type,

efs.total\_worker\_time

FROM sys.sql\_modules AS m WITH (NOLOCK)

LEFT OUTER JOIN sys.dm\_exec\_function\_stats AS efs WITH (NOLOCK)

ON m.object\_id = efs.object\_id

WHERE efs.type\_desc = N'SQL\_SCALAR\_FUNCTION'

ORDER BY efs.total\_worker\_time DESC

OPTION (RECOMPILE);

------

-- Scalar UDF Inlining

-- https://urldefense.com/v3/\_\_https://bit.ly/2JU971M\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQRpAUwAtA$

-- sys.sql\_modules (Transact-SQL)

-- https://urldefense.com/v3/\_\_https://bit.ly/2Qt216S\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQQdR3NuJw$

-- Get Query Store Options for this database (Query 84) (Query Store Options)

SELECT actual\_state\_desc, desired\_state\_desc, [interval\_length\_minutes],

current\_storage\_size\_mb, [max\_storage\_size\_mb],

query\_capture\_mode\_desc, size\_based\_cleanup\_mode\_desc, wait\_stats\_capture\_mode\_desc

FROM sys.database\_query\_store\_options WITH (NOLOCK) OPTION (RECOMPILE);

------

-- New for SQL Server 2016

-- Requires that Query Store is enabled for this database

-- Make sure that the actual\_state\_desc is the same as desired\_state\_desc

-- Make sure that the current\_storage\_size\_mb is less than the max\_storage\_size\_mb

-- Tuning Workload Performance with Query Store

-- https://urldefense.com/v3/\_\_https://bit.ly/1kHSl7w\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQRKvqbRWg$

-- Emergency shutoff for Query Store (SQL Server 2019 CU6 or newer)

-- ALTER DATABASE [DatabaseName] SET QUERY\_STORE = OFF(FORCED);

-- Query Store is enabled by default in SQL Server 2022

-- https://urldefense.com/v3/\_\_https://bit.ly/3A2JXsp\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQQd7buhIw$

-- Get input buffer information for the current database (Query 85) (Input Buffer)

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

es.[program\_name], es.[host\_name], es.login\_name,

es.login\_time, es.cpu\_time, es.logical\_reads, es.memory\_usage,

es.[status], ib.event\_info AS [Input Buffer]

FROM sys.dm\_exec\_sessions AS es WITH (NOLOCK)

CROSS APPLY sys.dm\_exec\_input\_buffer(es.session\_id, NULL) AS ib

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

AND es.session\_id > 50

AND es.session\_id <> @@SPID OPTION (RECOMPILE);

------

-- Gives you input buffer information from all non-system sessions for the current database

-- Replaces DBCC INPUTBUFFER

-- New DMF for retrieving input buffer in SQL Server

-- https://urldefense.com/v3/\_\_https://bit.ly/2uHKMbz\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQTwEswcdg$

-- sys.dm\_exec\_input\_buffer (Transact-SQL)

-- https://urldefense.com/v3/\_\_https://bit.ly/2J5Hf9q\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQQaKTXI8Q$

-- Get any resumable index rebuild operation information (Query 86) (Resumable Index Rebuild)

SELECT OBJECT\_NAME(iro.object\_id) AS [Object Name], iro.index\_id, iro.name AS [Index Name],

iro.sql\_text, iro.last\_max\_dop\_used, iro.partition\_number, iro.state\_desc,

iro.start\_time, iro.percent\_complete

FROM sys.index\_resumable\_operations AS iro WITH (NOLOCK)

OPTION (RECOMPILE);

------

-- index\_resumable\_operations (Transact-SQL)

-- https://urldefense.com/v3/\_\_https://bit.ly/2pYSWqq\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQTTRQfQxA$

-- Get database automatic tuning options (Query 87) (Automatic Tuning Options)

SELECT [name], desired\_state\_desc, actual\_state\_desc, reason\_desc

FROM sys.database\_automatic\_tuning\_options WITH (NOLOCK)

OPTION (RECOMPILE);

------

-- sys.database\_automatic\_tuning\_options (Transact-SQL)

-- https://urldefense.com/v3/\_\_https://bit.ly/2FHhLkL\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQTZTHS0KA$

-- Look at recent Full backups for the current database (Query 88) (Recent Full Backups)

SELECT TOP (30) bs.machine\_name, bs.server\_name, bs.database\_name AS [Database Name], bs.recovery\_model,

CONVERT (BIGINT, bs.backup\_size / 1048576 ) AS [Uncompressed Backup Size (MB)],

CONVERT (BIGINT, bs.compressed\_backup\_size / 1048576 ) AS [Compressed Backup Size (MB)],

CONVERT (NUMERIC (20,2), (CONVERT (FLOAT, bs.backup\_size) /

CONVERT (FLOAT, bs.compressed\_backup\_size))) AS [Compression Ratio], bs.compression\_algorithm,

bs.has\_backup\_checksums, bs.is\_copy\_only, bs.encryptor\_type,

DATEDIFF (SECOND, bs.backup\_start\_date, bs.backup\_finish\_date) AS [Backup Elapsed Time (sec)],

bs.backup\_finish\_date AS [Backup Finish Date], bmf.physical\_device\_name AS [Backup Location],

bmf.physical\_block\_size, bs.last\_valid\_restore\_time

FROM msdb.dbo.backupset AS bs WITH (NOLOCK)

INNER JOIN msdb.dbo.backupmediafamily AS bmf WITH (NOLOCK)

ON bs.media\_set\_id = bmf.media\_set\_id

WHERE bs.database\_name = DB\_NAME(DB\_ID())

AND bs.[type] = 'D' -- Change to L if you want Log backups

ORDER BY bs.backup\_finish\_date DESC OPTION (RECOMPILE);

------

-- Things to look at:

-- Are your backup sizes and times changing over time?

-- Are you using backup compression?

-- Are you using backup checksums?

-- Are you doing copy\_only backups?

-- Are you doing encrypted backups?

-- Have you done any backup tuning with striped backups, or changing the parameters of the backup command?

-- Where are the backups going to?

-- New columns in SQL Server 2022

-- compression\_algorithm and last\_valid\_restore\_time

-- last\_valid\_restore\_time is only valid for transaction log backups

-- In SQL Server 2016 and newer, native SQL Server backup compression actually works

-- much better with databases that are using TDE than in previous versions

-- https://urldefense.com/v3/\_\_https://bit.ly/28Rpb2x\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQQCuP2Y4A$

-- Microsoft Visual Studio Dev Essentials

-- https://urldefense.com/v3/\_\_https://bit.ly/2qjNRxi\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQSjr6YtxQ$

-- Microsoft Azure Learn

-- https://urldefense.com/v3/\_\_https://bit.ly/2O0Hacc\_\_;!!PyqGXjU!I7gZVLcRmph2IdAZK3HuTjoSQ0-wuORsGEyuHjbc4Y4RaASoSXs7iZ3QA\_W3RvS4XVcF5D1SUQRTf6t4SQ$