SQL Server 2019 Diagnostic Information

Query Scripts

Asfaw Gedamu

July 28,2023

First Step: Check the major product version to see if it is SQL Server 2019 CTP 2 or greater

```
IF NOT EXISTS (SELECT * WHERE CONVERT(varchar(128),
SERVERPROPERTY('ProductVersion')) LIKE '15%')
        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
```

1. **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 2019 Builds

-- Build                        Description
        Release Date    URL to KB Article
-- 15.0.1000.34                 CTP 2.0
        9/24/2018
-- 15.0.1100.94                 CTP 2.1
        11/7/2018
```

-- 15.0.1200.24 CTP 2.2
12/6/2018

-- 15.0.1300.359 CTP 2.3
3/1/2019

-- 15.0.1400.75 CTP 2.4
3/26/2019

-- 15.0.1500.28 CTP 2.5
4/23/2019

-- 15.0.1600.8 CTP 3.0
5/22/2019

-- 15.0.1700.37 CTP 3.1
6/26/2019

-- 15.0.1800.32 CTP 3.2
7/24/2019

-- 15.0.4053.23 CU6

-- Performance and Stability Fixes in SQL Server 2019 CU Builds

2. Get socket, physical core and logical core count from the SQL Server Error log (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

3. 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('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];
------

-- 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('IsTempdbMetadataMemoryOptimized') is a new option for SQL Server
2019
-- SERVERPROPERTY (Transact-SQL)
-- https://bit.ly/2eeaXeI
```

4. 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 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://bit.ly/2HsyDZI

5. Returns a list of all global trace flags that are enabled (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 - Supresses logging of successful database backup messages to the SQL Server Error Log

-- https://bit.ly/38zDNAK

-- TF 6534 - Enables use of native code to improve performance with spatial data

-- https://bit.ly/2HrQUpU

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

-- https://bit.ly/2GU69Km

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

-- https://bit.ly/2FuSvPg

5. SQL Server Process Address space info (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://bit.ly/3iUgQgC
-- How to enable the "locked pages" feature in SQL Server 2012
-- https://bit.ly/2F5UjOA
-- Memory Management Architecture Guide
-- https://bit.ly/2JKkadC
```

7. 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://bit.ly/2oKa1Un

8. 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],
   MAX(CASE WHEN [type] = 'D' THEN bs.backup_finish_date ELSE NULL END) AS [Last Full Backup],
         MAX(CASE WHEN [type] = 'D' THEN bmf.physical_device_name ELSE NULL END) AS [Last Full Backup Location],
   MAX(CASE WHEN [type] = 'I' THEN bs.backup_finish_date ELSE NULL END) AS [Last Differential Backup],
         MAX(CASE WHEN [type] = 'I' THEN bmf.physical_device_name ELSE NULL END) AS [Last Differential Backup Location],
   MAX(CASE WHEN [type] = 'L' THEN bs.backup_finish_date ELSE NULL END) AS [Last Log Backup],
         MAX(CASE WHEN [type] = 'L' THEN bmf.physical_device_name ELSE NULL END) AS [Last Log Backup Location]
FROM sys.databases AS d WITH (NOLOCK)
LEFT OUTER JOIN msdb.dbo.backupset AS bs WITH (NOLOCK)
ON bs.[database_name] = d.[name]
LEFT OUTER JOIN msdb.dbo.backupmediafamily AS bmf WITH (NOLOCK)
ON bs.media_set_id = bmf.media_set_id
AND bs.backup_finish_date > GETDATE()- 30
WHERE d.name <> N'tempdb'
```

```
GROUP BY ISNULL(d.[name], bs.[database_name]), d.recovery_model_desc,
d.log_reuse_wait_desc, d.[name]

ORDER BY d.recovery_model_desc, d.[name] OPTION (RECOMPILE);
```

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

9. Get SQL Server Agent jobs and Category information (Query 9) (SQL Server Agent Jobs)

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

s.[enabled] AS [Sched Enabled], js.next_run_date, js.next_run_time

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

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

ON sj.category_id = sc.category_id

LEFT OUTER JOIN msdb.dbo.sysjobschedules AS js WITH (NOLOCK)

ON sj.job_id = js.job_id

LEFT OUTER JOIN msdb.dbo.sysschedules AS s WITH (NOLOCK)

ON js.schedule_id = s.schedule_id

ORDER BY sj.name OPTION (RECOMPILE);
```

-- 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://bit.ly/2paDEOP

-- SQL Server Maintenance Solution (Ola Hallengren)

-- https://bit.ly/1pgchQu

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

-- https://bit.ly/3ane0gN

10. Get SQL Server Agent Alert Information (Query 10) (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://bit.ly/2v5YR37

11. Host information (Query 11) (Host Info)

```
SELECT host_platform, host_distribution, host_release,
    host_service_pack_level, host_sku, os_language_version
FROM sys.dm_os_host_info WITH (NOLOCK) OPTION (RECOMPILE);
```

-- host_release codes (only valid for Windows)

-- 10.0 is either Windows 10 or Windows Server 2016

-- 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 2019 requires Windows Server 2016 or newer

-- Hardware and Software Requirements for Installing SQL Server

-- https://bit.ly/2y3ka5L

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

-- https://bit.ly/2F7Ax0P

12. SQL Server NUMA Node information (Query 12) (SQL Server NUMA Info)

```
SELECT node_id, node_state_desc, memory_node_id, processor_group, cpu_count,
online_scheduler_count,
    idle_scheduler_count, active_worker_count, avg_load_balance, resource_monitor_state
FROM sys.dm_os_nodes WITH (NOLOCK)
WHERE 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 2019 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://bit.ly/2pn5Mw8

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

-- https://bit.ly/3i4TyVR

13. Good basic information about OS memory amounts and state (Query 13) (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://bit.ly/2pcV0xq
-- You can skip the next two queries if you know you don't have a clustered instance
```

14. Get information about your cluster nodes and their status (Query 14) (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

-- Recommended hotfixes and updates for Windows Server 2012 R2-based failover clusters

-- https://bit.ly/1z5BfCw

15. Get information about any AlwaysOn AG cluster this instance is a part of (Query 15) (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

16. Good overview of AG health and status (Query 16) (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 ﷺ It Just Runs Faster: Always On Availability Groups Turbocharged
-- https://bit.ly/2dn1H6r
```

17. Hardware information from SQL Server 2019 (Query 17) (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 -- New in SQL Server 2019
```

```
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://bit.ly/2pczOYs

-- 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://bit.ly/2HTpKJt

-- sql_memory_model_desc values (Added in SQL Server 2016 SP1)

-- CONVENTIONAL

-- LOCK_PAGES

-- LARGE_PAGES
```

18. Get System Manufacturer and model number from SQL Server Error log (Query 18) (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

19. Get BIOS date from Windows Registry (Query 19) (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

20. Get processor description from Windows Registry (Query 20) (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://bit.ly/QhR6xF

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

-- https://bit.ly/2F3aVlP

21. Get information on location, time and size of any memory dumps from SQL Server (Query 21) (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://bit.ly/2elwWll

22. Look at Suspect Pages table (Query 22) (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://bit.ly/2Fvr1c9

23. Read most recent entries from all SQL Server Error Logs (Query 23) (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
		SET @sql = 'INSERT INTO #SQLErrorLog_AllLogs
				(LogDate, ProcessInfo, LogText)
				EXEC master.sys.sp_readerrorlog ' + CAST(@i AS
NVARCHAR(2)) + N';'
		EXEC master.sys.sp_executesql @sql;
		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
```

24. Get number of data files in tempdb database (Query 24) (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
```

25. File names and paths for all user and system databases on instance (Query 25) (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)?

26. Drive information for all fixed drives visible to the operating system (Query 26) (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://bit.ly/2EZoHLj
```

27. Volume info for all LUNS that have database files on the current instance (Query 27) (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://bit.ly/2oBPNNr

28. Drive level latency information (Query 28) (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://bit.ly/3bRWUc0


-- sys.dm_os_volume_stats (Transact-SQL)
-- https://bit.ly/33thz2j
```

29. Calculates average latency per read, per write, and per total input/output for each database file (Query 29) (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://bit.ly/3bRWUc0
```

30. Look for I/O requests taking longer than 15 seconds in the six most recent SQL Server Error Logs (Query 30) (IO Warnings)

```
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 #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://bit.ly/2qtaw73

31. Resource Governor Resource Pool information (Query 31) (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://bit.ly/2MVU0Vy

32. Recovery model, log reuse wait description, log file size, log usage size (Query 32) (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.recovery_model_desc AS [Recovery Model],
db.state_desc, db.containment_desc, db.log_reuse_wait_desc AS [Log Reuse Wait Description],
CONVERT(DECIMAL(18,2), ls.cntr_value/1024.0) AS [Log Size (MB)],
CONVERT(DECIMAL(18,2), lu.cntr_value/1024.0) AS [Log Used (MB)],
```

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

db.[compatibility_level] AS [DB Compatibility Level],

db.is_mixed_page_allocation_on, db.page_verify_option_desc AS [Page Verify Option],

db.is_auto_create_stats_on, db.is_auto_update_stats_on, db.is_auto_update_stats_async_on,
db.is_parameterization_forced,

db.snapshot_isolation_state_desc, db.is_read_committed_snapshot_on, 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.group_database_id, db.replica_id,db.is_memory_optimized_elevate_to_snapshot_on,

db.delayed_durability_desc, db.is_query_store_on, db.is_sync_with_backup,

db.is_temporal_history_retention_enabled, db.is_remote_data_archive_enabled,

db.is_encrypted, de.encryption_state, de.percent_complete, de.key_algorithm, de.key_length,

db.is_accelerated_database_recovery_on

FROM sys.databases AS db WITH (NOLOCK)

INNER JOIN sys.dm_os_performance_counters AS lu WITH (NOLOCK)

ON db.name = lu.instance_name

INNER JOIN sys.dm_os_performance_counters AS ls WITH (NOLOCK)

ON db.name = ls.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 ls.cntr_value > 0

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

------


-- sys.databases (Transact-SQL)

-- https://bit.ly/2G5wqaX
```

-- sys.dm_os_performance_counters (Transact-SQL)

-- https://bit.ly/3kEO2JR

-- sys.dm_database_encryption_keys (Transact-SQL)

-- https://bit.ly/3mE7kkx

-- 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?

-- 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://bit.ly/2evRZSR

-- 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://bit.ly/2pdggk3

33. Missing Indexes for all databases by Index Advantage (Query 33) (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],

FORMAT(migs.last_user_seek, 'yyyy-MM-dd HH:mm:ss') 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!!!

-- H?kan Winther has given me some great suggestions for this query

-- SQL Server Index Design Guide

-- https://bit.ly/2qtZr4N
```

34. Get VLF Counts for all databases on the instance (Query 34) (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://bit.ly/3jpmqsd
-- sys.databases (Transact-SQL)
-- https://bit.ly/2G5wqaX
-- SQL Server Transaction Log Architecture and Management Guide
-- https://bit.ly/2JjmQRZ
-- VLF Growth Formula (SQL Server 2014 and newer)
-- If the log growth increment is less than 1/8th the current size of the log
--              Then:          1 new VLF
-- Otherwise:
--              Up to 64MB:     4 new VLFs
--              64MB to 1GB:    8 new VLFs
--              More than 1GB:  16 new VLFs
```

35. Get CPU utilization by database (Query 35) (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://bit.ly/32tHCGH
-- sys.dm_exec_plan_attributes (Transact-SQL)
-- https://bit.ly/35iP2hV
```

36. Get I/O utilization by database (Query 36) (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://bit.ly/3bRWUc0
```

37. Get total buffer usage by database for current instance (Query 37) (Total Buffer Usage by Database)

```
-- This make take some time to run on a busy instance
WITH AggregateBufferPoolUsage
AS
(SELECT DB_NAME(database_id) AS [Database Name],
CAST(COUNT(*) * 8/1024.0 AS DECIMAL (10,2))  AS [CachedSize]
```

```
FROM sys.dm_os_buffer_descriptors WITH (NOLOCK)
WHERE database_id <> 32767 -- ResourceDB
GROUP BY DB_NAME(database_id))
SELECT ROW_NUMBER() OVER(ORDER BY CachedSize DESC) AS [Buffer Pool Rank],
[Database Name], CachedSize AS [Cached Size (MB)],
    CAST(CachedSize / SUM(CachedSize) OVER() * 100.0 AS DECIMAL(5,2)) AS [Buffer
Pool Percent]
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://bit.ly/36s7aFo
```

38. Get tempdb version store space usage by database (Query 38) (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://bit.ly/2vh3Bmk
-- Clear Wait Stats with this command
-- DBCC SQLPERF('sys.dm_os_wait_stats', CLEAR);
```

39. Isolate top waits for server instance since last restart or wait statistics clear (Query 39) (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_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'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'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://www.sqlskills.com/help/waits/' + 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://bit.ly/2ePzYO2

-- The SQL Server Wait Type Repository

-- https://bit.ly/1afzfjC

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

-- https://bit.ly/2wsQHQE

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

-- https://bit.ly/1o2NFoF

-- sys.dm_os_wait_stats (Transact-SQL)

-- https://bit.ly/2Hjq9Yl

40. Get a count of SQL connections by IP address (Query 40) (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://bit.ly/2EgzoD0

41. Get Average Task Counts (run multiple times) (Query 41) (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://bit.ly/30IRla0

42. Detect blocking (run multiple times) (Query 42) (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
```

43. Get CPU Utilization History for last 256 minutes (in one minute intervals) (Query 43) (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
```

44. Get top total worker time queries for entire instance (Query 44) (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 45) (PLE by NUMA Node)

```
SELECT @@SERVERNAME AS [Server Name], RTRIM([object_name]) AS [Object Name],
    instance_name, cntr_value AS [Page Life Expectancy]
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://bit.ly/2EgynLa

46. Memory Grants Pending value for current instance (Query 46) (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

47. Memory Clerk Usage for instance (Query 47) (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
-- sys.dm_os_memory_clerks (Transact-SQL)
-- https://bit.ly/2H31xDR
```

48. Find single-use, ad-hoc and prepared queries that are bloating the plan cache (Query 48) (Ad hoc Queries)

```
SELECT TOP(50) DB_NAME(t.[dbid]) AS [Database Name], t.[text] AS [Query Text],
cp.objtype AS [Object Type], cp.cacheobjtype AS [Cache Object Type],
cp.size_in_bytes/1024 AS [Plan Size in KB]
FROM sys.dm_exec_cached_plans AS cp WITH (NOLOCK)
CROSS APPLY sys.dm_exec_sql_text(plan_handle) AS t
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://bit.ly/2EfYOkl

49. Get top total logical reads queries for entire instance (Query 49) (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

50. Get top average elapsed time queries for entire instance (Query 50) (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
```

51. Look at UDF execution statistics (Query 51) (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://bit.ly/2q1Q6BM
-- Showplan Enhancements for UDFs
-- https://bit.ly/2LVqiQ1
```

-- 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

52. Individual File Sizes and space available for current database (Query 52) (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!!

-- 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://bit.ly/2evRZSR

53. Log space usage for current database (Query 53) (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://bit.ly/2H4MQw9

54. Status of last VLF for current database (Query 54) (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://bit.ly/2EQUU1v
```

55. Get database scoped configuration values for current database (Query 55) (Database-scoped Configurations)

```
SELECT configuration_id, name, [value] AS [value_for_primary], value_for_secondary
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://bit.ly/2sOH7nb
```

56. I/O Statistics by file for the current database (Query 56) (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(10, 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(10, 2)) AS [MB Read],

CAST(vfs.num_of_bytes_written/1048576.0 AS DECIMAL(10, 2)) AS [MB Written],

CAST(100. * vfs.num_of_reads/(vfs.num_of_reads + vfs.num_of_writes) AS DECIMAL(10,1))
AS [# Reads Pct],

CAST(100. * vfs.num_of_writes/(vfs.num_of_reads + vfs.num_of_writes) AS DECIMAL(10,1))
AS [# Write Pct],

CAST(100. * vfs.num_of_bytes_read/(vfs.num_of_bytes_read + vfs.num_of_bytes_written) AS
DECIMAL(10,1)) AS [Read Bytes Pct],

CAST(100. * vfs.num_of_bytes_written/(vfs.num_of_bytes_read + vfs.num_of_bytes_written)
AS DECIMAL(10,1)) 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
```

57. Get most frequently executed queries for this database (Query 57) (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],

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

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

-- Queries 58 through 64 are the "Bad Man List" for stored procedures

58. Top Cached SPs By Execution Count (Query 58) (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],

FORMAT(qs.last_execution_time, 'yyyy-MM-dd HH:mm:ss', 'en-US') AS [Last Execution
Time],

FORMAT(qs.cached_time, 'yyyy-MM-dd HH:mm:ss', 'en-US') 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
```

59. Top Cached SPs By Avg Elapsed Time (Query 59) (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],
```

```
FORMAT(qs.last_execution_time, 'yyyy-MM-dd HH:mm:ss', 'en-US') AS [Last Execution Time],

FORMAT(qs.cached_time, 'yyyy-MM-dd HH:mm:ss', 'en-US') 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
```

60. Top Cached SPs By Total Worker time. Worker time relates to CPU cost (Query 60) (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],

FORMAT(qs.last_execution_time, 'yyyy-MM-dd HH:mm:ss', 'en-US') AS [Last Execution Time],

FORMAT(qs.cached_time, 'yyyy-MM-dd HH:mm:ss', 'en-US') 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
```

61. Top Cached SPs By Total Logical Reads. Logical reads relate to memory pressure (Query 61) (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],
FORMAT(qs.last_execution_time, 'yyyy-MM-dd HH:mm:ss', 'en-US') AS [Last Execution
Time],
FORMAT(qs.cached_time, 'yyyy-MM-dd HH:mm:ss', 'en-US') 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

62.Top Cached SPs By Total Physical Reads. Physical reads relate to disk read I/O pressure (Query 62) (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],
FORMAT(qs.last_execution_time, 'yyyy-MM-dd HH:mm:ss', 'en-US') AS [Last Execution
Time],
FORMAT(qs.cached_time, 'yyyy-MM-dd HH:mm:ss', 'en-US') 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

63.Top Cached SPs By Total Logical Writes (Query 63) (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],
FORMAT(qs.last_execution_time, 'yyyy-MM-dd HH:mm:ss', 'en-US') AS [Last Execution
Time],
FORMAT(qs.cached_time, 'yyyy-MM-dd HH:mm:ss', 'en-US') 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

64. Cached SPs Missing Indexes by Execution Count (Query 64) (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],
FORMAT(qs.last_execution_time, 'yyyy-MM-dd HH:mm:ss', 'en-US') AS [Last Execution
Time],
FORMAT(qs.cached_time, 'yyyy-MM-dd HH:mm:ss', 'en-US') 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

65. Lists the top statements by average input/output usage for the current database (Query 65) (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
```

66. Possible Bad NC Indexes (writes > reads) (Query 66) (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!
```

67. Missing Indexes for current database by Index Advantage (Missing Indexes)

```
SELECT CONVERT(decimal(18,2), migs.user_seeks * migs.avg_total_user_cost * (migs.avg_user_impact * 0.01)) AS [index_advantage],
FORMAT(migs.last_user_seek, 'yyyy-MM-dd HH:mm:ss') 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!!!
-- H?kan Winther has given me some great suggestions for this query
```

68. Find missing index warnings for cached plans in the current database (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(query_plan AS NVARCHAR(MAX)) LIKE N'%MissingIndex%'
AND 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

69. Breaks down buffers used by current database by object (table, index) in the buffer cache (Query 69) (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
```

70. Get Table names, row counts, and compression status for clustered index or heap (Query 70) (Table Sizes)

```
SELECT SCHEMA_NAME(o.Schema_ID) AS [Schema Name], OBJECT_NAME(p.object_id)
AS [ObjectName],
SUM(p.Rows) AS [RowCount], p.data_compression_desc AS [Compression Type]
FROM sys.partitions AS p WITH (NOLOCK)
INNER JOIN sys.objects AS o WITH (NOLOCK)
ON p.object_id = o.object_id
WHERE index_id < 2 --ignore the partitions from the non-clustered index if any
AND OBJECT_NAME(p.object_id) NOT LIKE N'sys%'
AND OBJECT_NAME(p.object_id) NOT LIKE N'spt_%'
AND OBJECT_NAME(p.object_id) NOT LIKE N'queue_%'
AND OBJECT_NAME(p.object_id) NOT LIKE N'filestream_tombstone%'
AND OBJECT_NAME(p.object_id) NOT LIKE N'fulltext%'
AND OBJECT_NAME(p.object_id) NOT LIKE N'ifts_comp_fragment%'
AND OBJECT_NAME(p.object_id) NOT LIKE N'filetable_updates%'
AND OBJECT_NAME(p.object_id) NOT LIKE N'xml_index_nodes%'
AND OBJECT_NAME(p.object_id) NOT LIKE N'sqlagent_job%'
AND OBJECT_NAME(p.object_id) NOT LIKE N'plan_persist%'
```

```
GROUP BY  SCHEMA_NAME(o.Schema_ID), p.object_id, data_compression_desc
ORDER BY SUM(p.Rows) DESC OPTION (RECOMPILE);
```

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

71. Get some key table properties (Query 71) (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://bit.ly/2Gk7998

72. When were Statistics last updated on all indexes? (Query 72) (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,
        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]
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://bit.ly/2GyAxrn
-- UPDATEs to Statistics (Erin Stellato)
-- https://bit.ly/2vhrYQy
```

73. Look at most frequently modified indexes and statistics (Query 73) (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
```

74. Get fragmentation info for all indexes above a certain size in the current database (Query 74) (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, ps.avg_fragmentation_in_percent,
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

75. Index Read/Write stats (all tables in current DB) ordered by Reads (Query 75) (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,
    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.[type_desc] AS [Index Type], i.fill_factor AS [Fill Factor], i.has_filter,
i.filter_definition,
        s.last_user_scan, s.last_user_lookup, s.last_user_seek
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

76. Index Read/Write stats (all tables in current DB) ordered by Writes (Query 76) (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,
        s.user_updates AS [Writes], s.user_seeks + s.user_scans + s.user_lookups AS [Total
Reads],
        i.[type_desc] AS [Index Type], i.fill_factor AS [Fill Factor], i.has_filter,
i.filter_definition,
        s.last_system_update, s.last_user_update
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

77. Get lock waits for current database (Query 77) (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.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

78. Look at UDF execution statistics (Query 78) (UDF Statistics)

```
SELECT OBJECT_NAME(object_id) AS [Function Name], execution_count,
        total_worker_time, total_logical_reads, total_physical_reads, total_elapsed_time,
        total_elapsed_time/execution_count AS [avg_elapsed_time],
        FORMAT(cached_time, 'yyyy-MM-dd HH:mm:ss', 'en-US') 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://bit.ly/2q1Q6BM
```

79. Determine which scalar UDFs are in-lineable (Query 79) (Inlineable UDFs)

```
SELECT OBJECT_NAME(m.object_id) AS [Function Name], is_inlineable, inline_type
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'
OPTION (RECOMPILE);
------
-- Scalar UDF Inlining
-- https://bit.ly/2JU971M
-- sys.sql_modules (Transact-SQL)
-- https://bit.ly/2Qt216S
```

80. Get QueryStore Options for this database (Query 80) (QueryStore 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
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://bit.ly/1kHSl7w
```

81. Get input buffer information for the current database (Query 81) (Input Buffer)

```
SELECT es.session_id, DB_NAME(es.database_id) AS [Database 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://bit.ly/2uHKMbz

-- sys.dm_exec_input_buffer (Transact-SQL)

-- https://bit.ly/2J5Hf9q

82. Get any resumable index rebuild operation information (Query 82) (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://bit.ly/2pYSWqq

83. Get database automatic tuning options (Query 83) (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://bit.ly/2FHhLkL

84. Look at recent Full backups for the current database (Query 84) (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.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
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?

-- In SQL Server 2016, native SQL Server backup compression actually works

-- much better with databases that are using TDE than in previous versions

-- https://bit.ly/28Rpb2x

-- Microsoft Visual Studio Dev Essentials

-- https://bit.ly/2qjNRxi

-- Microsoft Azure Learn

-- https://bit.ly/2O0Hacc

Source: sqlserver-kit/Scripts/SQL Server 2019 Diagnostic Information Queries.sql at master · ktaranov/sqlserver-kit · GitHub

Credit:

-- **Glenn Berry**

-- Last Modified: December 3, 2020

-- https://glennsqlperformance.com/

-- https://sqlserverperformance.wordpress.com/

-- YouTube: https://bit.ly/2PkoAM1

-- Twitter: GlennAlanBerry

-- Diagnostic Queries are available here

-- https://glennsqlperformance.com/resources/

-- 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://dbatools.io/

-- Invoke-DbaDiagnosticQuery

-- https://dbatools.io/functions/invoke-dbadiagnosticquery/