DMV Diagnostic Queries Detailed Archives


sqlskills.com/blogs/glenn/category/dmv-diagnostic-queries-detailed

By: Glenn Berry

Over the course of the month of January 2016, I wrote a series of daily blog posts that went through each of the 70 queries in the January 2016 version of my SQL Server 2016 Diagnostic Information Queries, with documentation of the various views and functions behind each query, along with what I hope is some useful information about how to interpret the results of each query.

Below are links to each of the daily blog posts for this series, along with a list of the queries that were covered that day:

SQL Server Diagnostic Information Queries Detailed, Day 1

> Version Info, Core Counts

SQL Server Diagnostic Information Queries Detailed, Day 2

> Server Properties, Configuration Values

SQL Server Diagnostic Information Queries Detailed, Day 3

> Global Trace Flags, Process Memory

SQL Server Diagnostic Information Queries Detailed, Day 4

> SQL Server Services Info, SQL Server Agent Jobs

SQL Server Diagnostic Information Queries Detailed, Day 5

> SQL Server Agent Alerts, Windows Info

SQL Server Diagnostic Information Queries Detailed, Day 6

> SQL Server NUMA Info, System Memory

SQL Server Diagnostic Information Queries Detailed, Day 7

> SQL Server Error Log, Cluster Node Properties, AlwaysOn AG Cluster

SQL Server Diagnostic Information Queries Detailed, Day 8

> Hardware Info, System Manufacturer, Processor Description

SQL Server Diagnostic Information Queries Detailed, Day 9

> BPW Configuration, BPE Usage

SQL Server Diagnostic Information Queries Detailed, Day 10

> Memory Dump Info, Database Filenames and Paths

SQL Server Diagnostic Information Queries Detailed, Day 11

> Volume Info, Drive-Level Latency, IO Stalls by File, IO Warnings

SQL Server Diagnostic Information Queries Detailed, Day 12

> Database Properties, Missing Indexes All Databases, VLF Counts

SQL Server Diagnostic Information Queries Detailed, Day 13

> CPU Usage by Database, IO Usage by Database, Total Buffer Usage by Database

SQL Server Diagnostic Information Queries Detailed, Day 14

> Top Waits, Connection Counts by IP Address

SQL Server Diagnostic Information Queries Detailed, Day 15

> Avg Task Counts, Detect Blocking

SQL Server Diagnostic Information Queries Detailed, Day 16

> CPU Utilization History, Top Worker Time Queries

SQL Server Diagnostic Information Queries Detailed, Day 17

> PLE by NUMA Node, Memory Grants Pending,

SQL Server Diagnostic Information Queries Detailed, Day 18

> Memory Clerk Usage, Ad hoc Queries, Top Logical Reads Queries

SQL Server Diagnostic Information Queries Detailed, Day 19

> File Sizes and Space, IO Stats by File

SQL Server Diagnostic Information Queries Detailed, Day 20

> Query Execution Counts, SP Execution Counts

SQL Server Diagnostic Information Queries Detailed, Day 21

> SP Avg Elapsed Time, SP Worker Time

SQL Server Diagnostic Information Queries Detailed, Day 22

> SP Logical Reads, SP Physical Reads

SQL Server Diagnostic Information Queries Detailed, Day 23

> SP Logical Writes, Top IO Statements

SQL Server Diagnostic Information Queries Detailed, Day 24

> Bad NC Indexes, Missing Indexes, Missing Index Warnings

SQL Server Diagnostic Information Queries Detailed, Day 25

> Buffer Usage, Table Sizes

SQL Server Diagnostic Information Queries Detailed, Day 26

> Table Properties, Statistics Update

SQL Server Diagnostic Information Queries Detailed, Day 27

> Volatile Indexes, Index Fragmentation

SQL Server Diagnostic Information Queries Detailed, Day 28

> Overall Index Usage – Reads, Overall Index Usage – Writes

SQL Server Diagnostic Information Queries Detailed, Day 29

> XTP Index Usage, Lock Waits

SQL Server Diagnostic Information Queries Detailed, Day 30

> UDF Statistics, QueryStore Options

SQL Server Diagnostic Information Queries Detailed, Day 31

> High Aggregate Duration Queries, Recent Full Backups

These three Pluralsight Courses go into even more detail about how to run these queries and interpret the results.

> SQL Server 2014 DMV Diagnostic Queries – Part 1

> SQL Server 2014 DMV Diagnostic Queries – Part 2

> SQL Server 2014 DMV Diagnostic Queries – Part 3

For Day 31 of this series, we start out with **Query #69**, which is High Aggregate Duration Queries. This query retrieves information from the sys.query_store_query_text query store catalog view, the sys.query_store_query query store catalog view, the sys.query_store_plan query store catalog view, the sys.query_store_runtime_stats query store catalog view, and the sys.query_store_runtime_stats_interval query store catalog view about the highest aggregate duration queries in the current database over the past hour. Query #69 is shown in Figure 1.

```
   1: -- Get highest aggregate duration queries over last hour (Query 69) (High
Aggregate Duration Queries)
```

```
 2: WITH AggregatedDurationLastHour
 3: AS
 4: (SELECT q.query_id, SUM(count_executions * avg_duration) AS total_duration,
 5:    COUNT (distinct p.plan_id) AS number_of_plans
 6:    FROM sys.query_store_query_text AS qt WITH (NOLOCK)
 7:    INNER JOIN sys.query_store_query AS q WITH (NOLOCK)
 8:    ON qt.query_text_id = q.query_text_id
 9:    INNER JOIN sys.query_store_plan AS p WITH (NOLOCK)
10:    ON q.query_id = p.query_id
11:    INNER JOIN sys.query_store_runtime_stats AS rs WITH (NOLOCK)
12:    ON rs.plan_id = p.plan_id
13:    INNER JOIN sys.query_store_runtime_stats_interval AS rsi WITH (NOLOCK)
14:    ON rsi.runtime_stats_interval_id = rs.runtime_stats_interval_id
15:    WHERE rsi.start_time >= DATEADD(hour, -1, GETUTCDATE())
16:    AND rs.execution_type_desc = N'Regular'
17:    GROUP BY q.query_id),
18: OrderedDuration AS
19: (SELECT query_id, total_duration, number_of_plans,
20:  ROW_NUMBER () OVER (ORDER BY total_duration DESC, query_id) AS RN
21:  FROM AggregatedDurationLastHour)
22: SELECT OBJECT_NAME(q.object_id) AS [Containing Object], qt.query_sql_text,
23: od.total_duration AS [Total Duration (microsecs)],
24: od.number_of_plans AS [Plan Count],
25: p.is_forced_plan, p.is_parallel_plan, p.is_trivial_plan,
26: q.query_parameterization_type_desc, p.[compatibility_level],
27: p.last_compile_start_time, q.last_execution_time,
28: CONVERT(xml, p.query_plan) AS query_plan_xml
29: FROM OrderedDuration AS od
30: INNER JOIN sys.query_store_query AS q WITH (NOLOCK)
31: ON q.query_id  = od.query_id
32: INNER JOIN sys.query_store_query_text AS qt WITH (NOLOCK)
```

```
33: ON q.query_text_id = qt.query_text_id

34: INNER JOIN sys.query_store_plan AS p WITH (NOLOCK)

35: ON q.query_id = p.query_id

36: WHERE od.RN <= 50

37: ORDER BY total_duration DESC OPTION (RECOMPILE);

38:

39: -- New for SQL Server 2016

40: -- Requires that QueryStore is enabled for this database
```

Figure 1: Query #69 High Aggregate Duration Queries

If you are using the QueryStore feature in SQL Server 2016, (meaning that you have enabled it for the current database), then you can either use the built in functionality in SSMS, or use queries like this to examine the data that it collects and exposes. Personally, I like to be able to write custom queries like this to analyze the information.

This query lets you identify which queries in the current database have the highest aggregate duration over the past hour. This lets you find queries that might benefit from your query and index tuning efforts, especially ones that may show a noticeable benefit from any improvements.

Query #70 is Recent Full Backups. This query retrieves information from the dbo.backupset table in the msdb system database about the most recent Full database backups for the current database. Query #70 is shown in Figure 2.

```
   1: -- Look at recent Full backups for the current database (Query 70) (Recent
Full Backups)

   2: SELECT TOP (30) bs.machine_name, bs.server_name, bs.database_name AS
[Database Name], bs.recovery_model,

   3: CONVERT (BIGINT, bs.backup_size / 1048576 ) AS [Uncompressed Backup Size
(MB)],

   4: CONVERT (BIGINT, bs.compressed_backup_size / 1048576 ) AS [Compressed Backup
Size (MB)],

   5: CONVERT (NUMERIC (20,2), (CONVERT (FLOAT, bs.backup_size) /

   6: CONVERT (FLOAT, bs.compressed_backup_size))) AS [Compression Ratio],
bs.has_backup_checksums, bs.is_copy_only, bs.encryptor_type,

   7: DATEDIFF (SECOND, bs.backup_start_date, bs.backup_finish_date) AS [Backup
Elapsed Time (sec)],

   8: bs.backup_finish_date AS [Backup Finish Date]
```

```
 9: FROM msdb.dbo.backupset AS bs WITH (NOLOCK)

10: WHERE bs.database_name = DB_NAME(DB_ID())

11: AND bs.[type] = 'D' -- Change to L if you want Log backups

12: ORDER BY bs.backup_finish_date DESC OPTION (RECOMPILE);

13:

14: -- Are your backup sizes and times changing over time?

15: -- Are you using backup compression?

16: -- Have you done any backup tuning with striped backups, or changing the
parameters of the backup command?
```

Figure 2: Query #70 Recent Full Backups

This query gives you some useful statistics and properties about your most recent Full database backups for the current database. It shows you the uncompressed size of the backup, along with the compressed size and the compression ratio, if any. It also lets you know if the backup is using checksums, whether it is a copy-only backup, and whether it is using native backup encryption, which was a new feature in SQL Server 2014. Finally, it shows you when each backup finished and how long it took to complete.

Keeping an eye on the size and elapsed time for your database backups is always a good idea. As your database gets larger, you may have to make changes to how and when you do your backups or to the underlying resources at location to where they are going to make sure that they are finishing in a reliable and timely manner.

These three Pluralsight Courses go into even more detail about how to run these queries and interpret the results.

> SQL Server 2014 DMV Diagnostic Queries – Part 1

> SQL Server 2014 DMV Diagnostic Queries – Part 2

> SQL Server 2014 DMV Diagnostic Queries – Part 3

We have finally made it to the end of this series! I'll be putting up a recap post for the entire series, with links to each post.

For Day 30 of this series, we start out with **Query #67**, which is UDF Statistics. This query retrieves information from the sys.dm_exec_function_stats dynamic management view about aggregate runtime metrics for user-defined functions in the current database, ordered by object name. Query #67 is shown in Figure 1.

```
1: -- Look at UDF execution statistics (Query 67) (UDF Statistics)

2: SELECT OBJECT_NAME(object_id) AS [Function Name], execution_count,

3:    total_elapsed_time/1000 AS [time_milliseconds], fs.[type_desc]
```

```
4: FROM sys.dm_exec_function_stats AS fs WITH (NOLOCK)

5: WHERE database_id = DB_ID()

6: ORDER BY OBJECT_NAME(object_id) OPTION (RECOMPILE);

7:

8: -- New for SQL Server 2016

9: -- Helps you investigate UDF performance issues
```

Figure 1: Query #67 UDF Statistics

One fairly well-known, long-running issue with SQL Server is how often you see performance problems with scalar, user-defined functions (UDFs). As a DBA, I always tried to avoid having my developers ever find out that scalar UDFs even existed, but they sometimes discovered them on their own, unfortunately. One big problem with scalar UDFs is that their actual cost does not show up when you look at the execution plan or statistics IO output for the query or stored procedure that called them.

In the past, you needed to use tools like SQL Profiler or Extended Events to see what was going on when you had scalar UDF usage. In SQL Server 2016, you will be able to use this new DMV and this query to have some visibility about what your UDFs are doing. As far as mitigation goes, doing things like converting a scalar UDF to a table UDF that just returns just one column and one row, or converting it to a stored procedure are often pretty effective and easy to do.

Query #68 is QueryStore Options. This query retrieves information from the sys.database_query_store_options dynamic management view about the current QueryStore options for the current database. Query #68 is shown in Figure 2.

```
1: -- Get QueryStore Options for this database (Query 68) (QueryStore Options)

2: SELECT actual_state, actual_state_desc, readonly_reason,

3:        current_storage_size_mb, max_storage_size_mb

4: FROM sys.database_query_store_options WITH (NOLOCK)

5: OPTION (RECOMPILE);

6:

7: -- New for SQL Server 2016

8: -- Requires that QueryStore is enabled for this database
```

Figure 2: Query #68 QueryStore Options

QueryStore is one of the more exciting new features in SQL Server 2016. It gives you a lot of visibility about what is happening with your query plans in a particular database over time. You also get the ability to force the query optimizer to use a particular “good” query plan. This should make it much easier to troubleshoot and correct plan regression issues.

So far, Microsoft has not announced whether this is an Enterprise Edition–only feature or not. Before you can use QueryStore, you have to enable it for the database that you are concerned with.

These three Pluralsight Courses go into even more detail about how to run these queries and interpret the results.

> SQL Server 2014 DMV Diagnostic Queries – Part 1

> SQL Server 2014 DMV Diagnostic Queries – Part 2

> SQL Server 2014 DMV Diagnostic Queries – Part 3