Performance Monitoring Using Extended Events, DMVs & Query Store

Ola Hallengren, Principal Database Architect, Saxo Bank

Ola Hallengren

Principal Database Architect, Saxo Bank

E-mail: ola@hallengren.com

Microsoft MVP – Data Platform

Author of the popular maintenance scripts at https://ola.hallengren.com

Database Consultant

Agenda

- Extended Events, DMVs and Query Store
- How are they working in different scenarios?
- How can you use them together?
- Comparing performance data
- Experiences from the bank and when working with customers
- Troubleshooting performance problems
- Collecting data into a monitoring database

Extended Events – What data do we have?

Event	Level	Reads	Writes	CPU time	Dura-tion	Result
sql_batch_completed	Outer	Yes	Yes	Yes	Yes	OK/Abort/Error
rpc_completed	Outer	Yes	Yes	Yes	Yes	OK/Abort/Error
module_end	Module *	-	-	-	Yes	-
sp_statement_completed	Statement	Yes	Yes	Yes	Yes	Only fires if completed successfully
sql_statement_completed	Statement	Yes	Yes	Yes	Yes	Only fires if completed successfully

* A module could be a stored procedure or a function

A stored procedure was timing out – What statement was executing?

- A stored procedure was timing out. You are looking at a rpc_completed event with result = Abort, and a module_end event. What statement was in process executing when the timeout happened?

- module_end tsql_frame
  Successful execution: <frame … line="0" offsetStart="-1" offsetEnd="-1“ />
  Aborted execution: <frame … line="6" offsetStart="74" offsetEnd="202”/>

- You can pass the sql_handle to sys.dm_exec_sql_text to retrieve the sql text, and then use the offsets to calculate the statement

- Demo

Query Store Intro

- Added in SQL Server 2016
- The typical use case for Query Store is to look at changes to query plans

- Automatically stores queries, query plans, run_time statistics and wait statistics
- Run_time statistics and wait statistics aggregated for one hour intervals
- Stored by execution type: Regular, Aborted, and Exception

- Query Store data stored in the user database

- Here is how to enable Query Store:
  ALTER DATABASE WideWorldImporters SET QUERY_STORE = ON

Query Store – How to find the query_id?

Real-world case: A critical report timed out last night. You have the application name, the host name, and the timestamp. Unfortunately the error message in the application was not very clear, and you are not sure what the query was.

- Limitation: Query Store does not have any information about application name, host name or login name. How can I find the query in Query Store? You need some additional data to find the query:

- sys.dm_exec_requests: statement_sql_handle and statement_context_id

- Extended Events: sql_handle, start_offset, and end_offset in tsql_frame

- Demo

Query Store - The important context_setting_id

Real-world case: Some days later another critical report times out. This time you know the name of the stored procedure. You are thinking that it should be easy to find the query in Query Store and maybe force a good plan.

- To your surprise there are several query_id's for the same object_id and offsets in sys.query_store_query. Only the context_setting_id is different.
- It turns out that some developers have been executing the stored procedure in SSMS to try o find out about the problem. SSMS has different set_options than the application.
- Again you need statement_context_id in sys.dm_exec_requests.
- Demo

How Query Store works with killed sessions

Real-world case: A query has been running for a long time. There is no command timeout in the application, so eventually you have to kill it. You look in the Query Store to compare the plans, but you can't find any Aborted executions.

- Limitation: Executions are not captured in the Query Store sys.query_store_runtime_stats and sys.query_store_wait_stats, if a session is killed or if the client application or host is restarted or crash.
- The plan is still captured in sys.query_store_plan.
- Demo

Query Store vs sys.dm_exec_query_stats vs sys.dm_exec_procedure_stats - Part 1

- What happens with the statistics in Query Store, sys.dm_exec_query_stats, and sys.dm_exec_procedure_stats in different situations?

- A recompile triggered by a statistics update? sp_recompile? WITH RECOMPILE? OPTION (RECOMPILE)?

- DBCC FREEPROCCACHE?

Query Store vs sys.dm_exec_query_stats vs sys.dm_exec_procedure_stats - Part 2

	Level	Includes	Stored	Interval
Query Store	Statement	All executions, except killed	In memory → User database	1 hour (default)
sys.dm_exec_query_stats	Statement	Successful executions only	In memory	No interval
sys.dm_exec_procedure_stats	Stored procedure	All executions	In memory	No interval

Query Store vs sys.dm_exec_query_stats vs sys.dm_exec_procedure_stats - Part 3

	Recompile triggered by statistics update or index rebuild	sp_recompile	DBCC FREEPROC CACHE
Query Store	Not affected	Not affected	Not affected
sys.dm_exec_query_stats	Stats reset plan_generation_ num updated	Entry removed	All entries removed
sys.dm_exec_procedure_stats	Not affected	Entry removed	All entries removed

Query Store vs sys.dm_exec_query_stats vs sys.dm_exec_procedure_stats - Part 4

	WITH RECOMPILE	OPTION (RECOMPILE)
Query Store	Included	Included
sys.dm_exec_query_stats	Not included	Only last execution included
sys.dm_exec_procedure_stats	Not included	Included

Log writes and Always On synchronizations

Real-world case: You are troubleshooting an issue with slow transactions. You are collecting data from Extended Events, DMVs, and also looking in Query Store. The durations just does not add up.

- Limitation: Query Store, sys.dm_exec_query_stats, and the statement - level Extended Events do not include hardening of the log write to disk, or synchronization with the secondary replica in Always On.

- Demo

When is Log writes and Always On included?

	Includes transaction log write to disk	Includes Always On synchronization *
sql_batch_completed	Yes	Yes
rpc_completed	Yes	Yes
module_end	Yes	Yes
sp_statement_completed	No	No
sql_statement_completed	Only if outside a module_end	Only if outside a module_end
Query Store	No	No
sys.dm_exec_query_stats	No	No
sys.dm_exec_procedure_stats	Yes	Yes

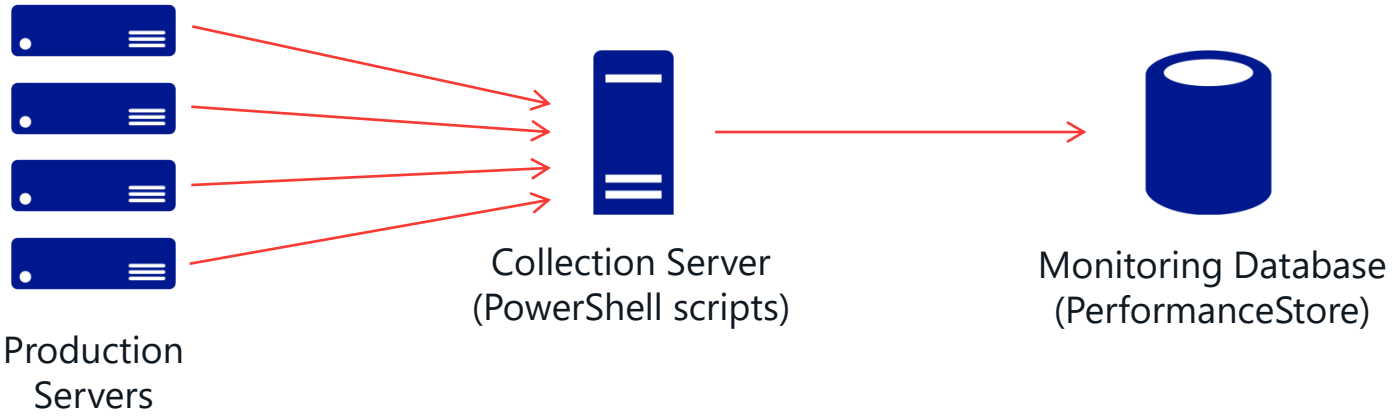
* Always On with synchronous-commit

Looking at problems back in time

- How do you answer questions like "Why did this stored procedure time out for the last few days." When you are investigating problems back in time, the key thing is to have the data.
- For Query Store the data collection happens automatically in the user database.
- We have seen that in many cases the data collected in Query Store is not enough. Often you also need data from Extended Events and DMVs.
- For Extended Events and DMVs, you need to do the collection of data.

Collecting data into a monitoring database

- Collecting data from execution related DMVs and Extended Events into a monitoring database for easy querying
- The raw data in the tables
- Views to transform and combine the data



Quick overview of the monitoring database

Data	Table	Views
Extended Events	ExtendedEvents.Events	ExtendedEvents.AbortedExecutions ExtendedEvents.SlowExecutions ExtendedEvents.BlockedProcesses
DMVs	dbo.dm_exec_sessions_extended	dbo.Sessions
SQL Texts	dbo. dm_exec_sql_text	

A query timeout from different angles

Real-world case : An important stored procedure was timing out early in the morning. The business wants to know what happened and you start investigating it.

- Troubleshooting using data in your monitoring database and in Query Store
- Demo

Putting it all together

- If you have a completed/end event in Extended Events, you have the timestamp, and the duration. The timestamp is the end time. Using the duration you can calculate the start time. Then you can find session/request snapshots for the same session_id and in this interval.

- You can use statement_sql_handle and statement_context_id in sys.dm_exec_requests to find the query_id in Query Store

- You can use transaction_id to find blocked_process_report events, and then change the filter to monitor_loop_id to look at a snapshot of the blocking

blocked_process_report – What is the monitor_loop_id?

- The blocked_process_report event is very useful when investigating blocking problems
- Handled by the same thread in SQL Server that is searching for deadlocks
- Every time the monitor thread wakes up and is looking for blocking it has a new monitor_loop_id
- <blocked-process-report monitorLoop="1369">
- If you want to get a “snapshot” of all blocking that was going on at a time, filter on monitor_loop_id
- Demo

Questions?

- The presentation and demo code will be available at http://www.sqlbits.com
- The code for the monitoring database is available at https://ola.hallengren.com/scripts/PerformanceStore.zip
- You can contact me at ola@hallengren.com

- Thank You

- Please complete the session feedback forms

sqlbits

SQLBits - It's all about the community…

Please visit Community Corner, we are trying this year to get more people to learn about the SQL Community, equally if you would be happy to visit the community corner we'd really appreciate it.