


# QDS Hitting Memory Limits Statement HashMap 131072

Last updated by | Francisco O | Jan 12, 2023 at 1:35 AM PST

## Contents

- [Issue](#)
- [Investigation/Analysis](#)
- [Mitigation](#)
- [Internal References](#)

## Issue

QDS going into read-only mode is an expected behavior from QDS operations, when certain limits are reached, which prevent it from growing. This is represented by the `readonly_reason` value in [sys.database\\_query\\_store\\_options](#) .

For the depicted scenario, the value was **131072: "The number of different statements in Query Store has reached the internal memory limit. Consider removing queries that you do not need or upgrading to a higher service tier to enable transferring Query Store to read-write mode."**

The options that affect this are entirely on the customer side - changing capture policy and other parameters, as well as parameterizing the workload. This affects how quickly QDS fills up.

QDS has mechanisms of emptying, but if it fills up at a faster rate than it is emptied, limits will eventually be reached. These mechanisms of emptying are size based cleanup and time based cleanup. Time based cleanup removes queries from QDS that are older than the stale query threshold, which the customer can set. The size based cleanup is triggered when 90% of the memory has been filled, and allocates 10% of its memory (roughly) in queries to be deleted (this takes time).

Given that the limit hit in this depicted scenario is the statement count rather than the memory limit, size based cleanup is not triggered, so queries are only removed by way of time based cleanup, which is usually not enough to offset the customer workload.

## Investigation/Analysis

Parameterizing the workload is relevant because QDS stores queries based on their statement text. Two queries that differ in something that could be a parameter, but is instead part of the query text, will be stored differently, taking up more space than is necessary. This can be seen when the number of different statement hashes is much larger than the number of different query hashes (statement hash is based on text, query hash is based on shape) like below:

```
MonWiQdsExecStats
| where LogicalServerName == 'LogicalServerNameHere'
| where database_name =~ 'DBNameHere'
| summarize count(), dcount(statement_sql_hash), dcount(query_hash), dcount(query_plan_hash)
```

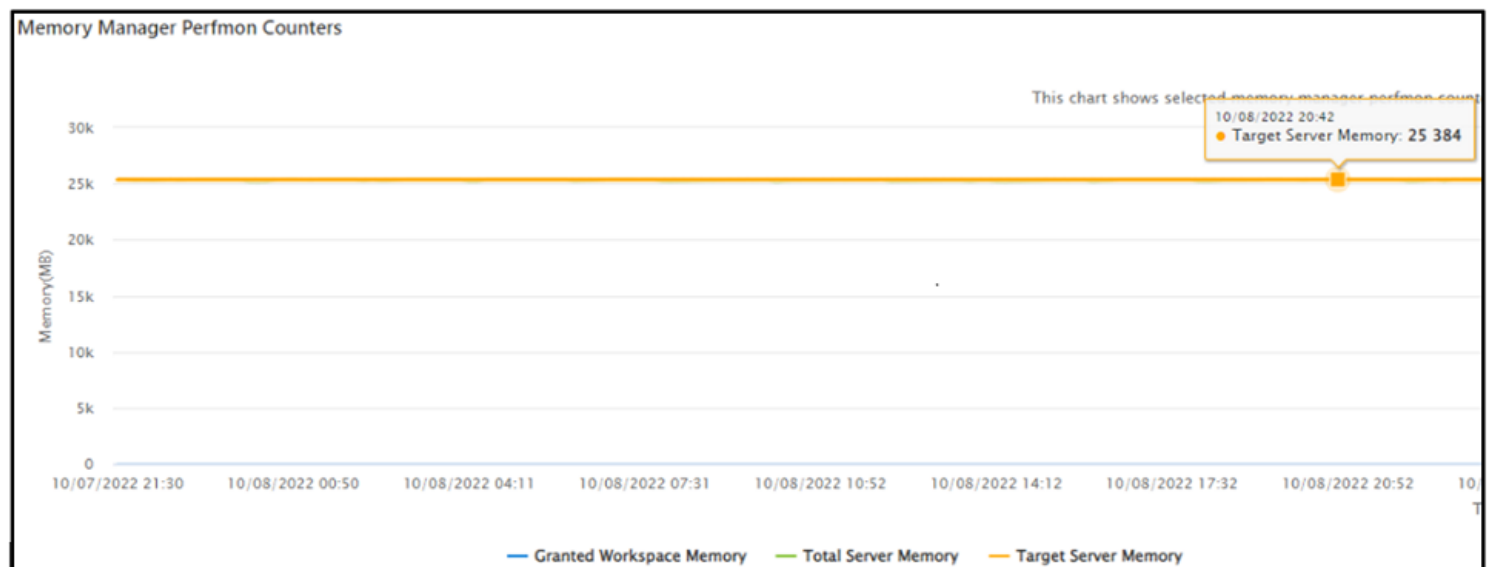
Or, alternatively, if you want to check it at an **Elastic Pool** level:

```
let startTime = ago(30d);
let endTime = now();
let serverName = "serverNameHere";
let databaseName = "dbNameHere";
let appName = "c7face8d41b5";
let nodeName = "DB_HS1.16";
let clusterName = "tr7299.eastus1-a.worker.database.windows.net";
MonWiQdsExecStats
| where originalEventTimestamp > startTime
| where originalEventTimestamp < endTime
| where LogicalServerName =~ serverName
| where AppName == appName
//| where database_name =~ databaseName
| summarize count(), dcount(statement_sql_hash), dcount(query_hash), dcount(query_plan_hash) by database_name
| summarize sum(dcount_statement_sql_hash), sum(dcount_query_hash)
```

It's relevant to mention that parameterized workloads fill QDS up significantly slower, allowing cleanup to more easily handle the load and prevent going into read-only mode. As an added benefit, they reduce the need for query recompilations (first column in the results above), increasing performance.

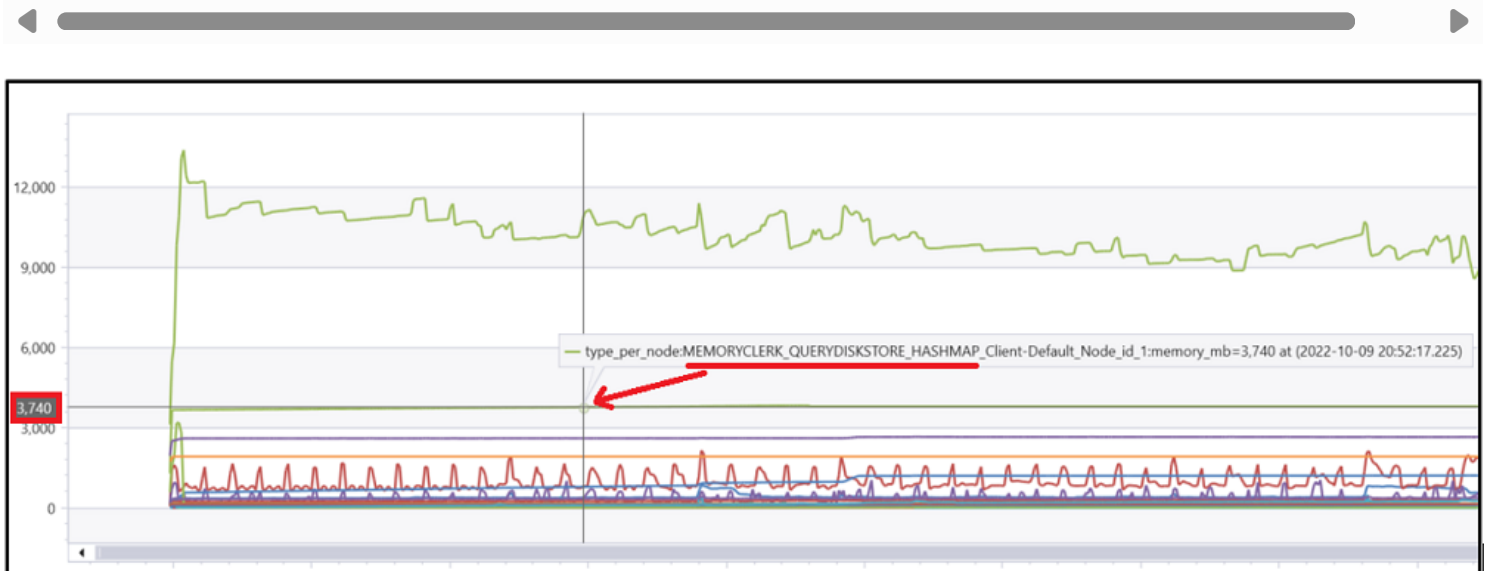
In order to mitigate the immediate issue, it can be considered to reduce the stale query threshold so that time based cleanup removes some number of queries as soon as possible. This puts QDS into read-write mode and gives the customer time to implement the other changes in order to prevent going into read-only mode again.

The memory limit is a instance wide limit set to 15% of available server memory. To have a clearer understanding of the read-only change due to this, consider the depicted case where the customer had a 6vCore GP elastic pool, with target server memory floating around 25k MB:



This makes QDS hash map size limit being set around ~3800MB (25300 \* 0.15). Looking at clerk stats we can see that QDS memory clerk is around the limit which suggests that QDS entered read only state because of this (as reported by QDS telemetry):

```
let startTime = ago(20d);
let endTime = now();
let appName = "c7face8d41b5";
let nodeName = "DB_HS1.16";
let clusterName = "tr7299.eastus1-a.worker.database.windows.net";
MonSqlMemoryClerkStats
| where originalEventTimestamp > startTime
| where originalEventTimestamp < endTime
| where AppName == appName
| where NodeName == nodeName
| extend type_per_node = strcat(memory_clerk_type, "_", memory_clerk_name, "_Node_id_", toString(memory_node_id))
| extend memory_kb = peak_pages_kb + vm_committed_kb
| extend memory_mb = memory_kb / 1024
| project originalEventTimestamp, type_per_node, memory_mb
| render timechart
```

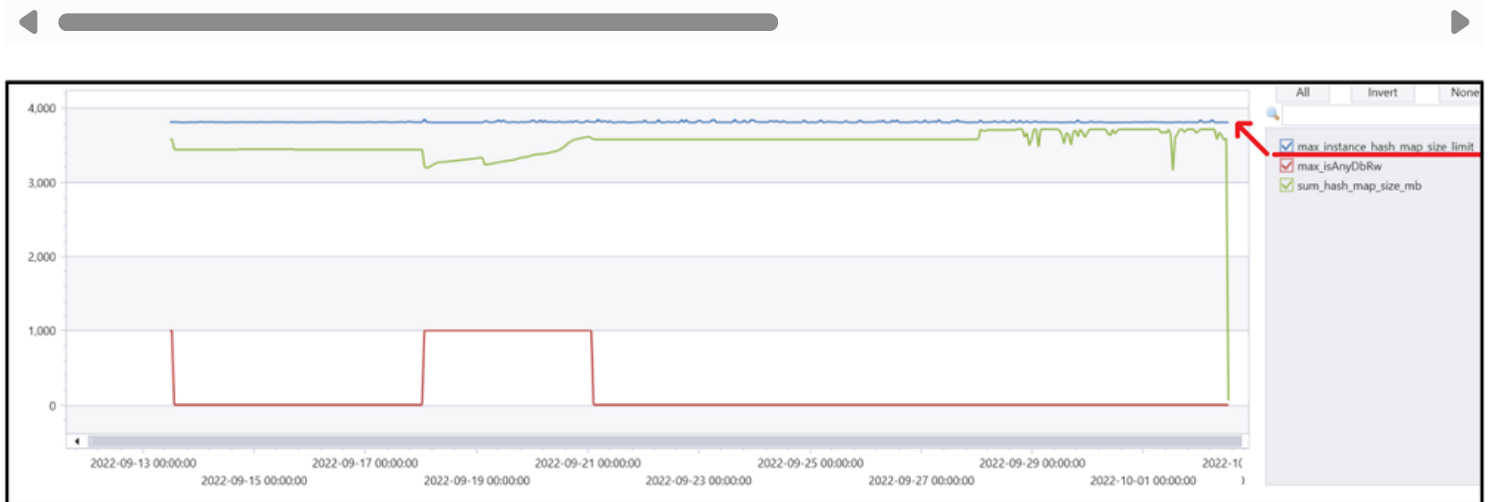


From QDS telemetry we can also check the progression of the hash map memory limit, check if it's hitting the respective limit (causing the read-only state), or if QDS managed to cleanup the items and stay below 85% of the limit (which would cause QDS to be in RW mode again). Note that the query below can be used to check for an entire elastic pool, but you can use the database parameter to check for a particular database.

```

let startTime = ago(30d);
let endTime = ago(0d);
let serverName = "servernameHere";
let databaseName = "dbNameHere";
let appName = "c7face8d41b5";
let nodeName = "DB_HS1.16";
let clusterName = "tr7299.eastus1-a.worker.database.windows.net";
MonQueryStoreInfo
| where originalEventTimestamp > startTime
| where originalEventTimestamp < endTime
| where LogicalServerName == serverName
| where AppName == appName
//| where logical_database_name == databaseName
| where event == "query_store_db_diagnostics"
| where query_store_read_only_reason != 8
| summarize max(query_count), max(query_text_count), max(plan_count), max(current_stmt_hash_map_size_kb), max(
| extend query_count_k = max_query_count/1000
| extend query_text_count_k = max_query_text_count/1000
| extend max_plan_count_k = max_plan_count/1000
| extend hash_map_size_mb = max_current_stmt_hash_map_size_kb/1024
| extend instance_hash_map_size_limit = max_max_stmt_hash_map_size_kb*3/1024
| extend isAnyDbRw = (max_db_state_actual-1) * 1000
//| project originalEventTimestamp, toString(database_id), hash_map_size_mb
| summarize sum(hash_map_size_mb), max(instance_hash_map_size_limit), max(isAnyDbRw) by originalEventTimestamp

```



## Mitigation

Query Store is entering read-only state because "The number of different statements in Query Store has reached the internal memory limit" as documented at: <https://learn.microsoft.com/en-us/sql/relational-databases/system-catalog-views/sys-database-query-store-options-transact-sql?view=sql-server-ver16>

Limit is set to 15% of elastic pool memory. There are multiple corrective actions customer can take to avoid hitting the limit:

1. Remove some of the databases from the pool
2. Increase SLO if 1) is not possible - total max hash map size depends on SLO
3. Set custom Query Store capture policy to have less queries captured by QDS reducing overall memory footprint - <https://learn.microsoft.com/en-us/sql/relational-databases/performance/best-practice-with-the-query-store?view=sql-server-ver16>
4. Parametrize workload if there are no parameter sensitive queries to reduce number of entries in QDS
5. Remove some of the entries manually (this might be tedious as there are many databases/many entries)

6. Clear QDS on some of the databases (this has a risk of deleting forced plans if any which may lead to performance degradation) - "ALTER DATABASE [DbName] SET QUERY\_STORE CLEAR"

## Internal References

- [lcM 339135557](#) 

**How good have you found this content?**

