# Gems to help you troubleshoot query performance

Microsoft

SQL Server Tiger Team

# Pedro Lopes

@sqlpto
pedro.lopes@microsoft.com

Senior Program Manager

Focused on SQL Server Relational Engine

7+ years at Microsoft

14+ years with SQL Server

SQL Server Tiger Team

When was the last time you dealt with a **query** performance issue?

SQL Server Tiger Team

# Query Performance Troubleshooting Fundamentals

## Why does a query slow down?

- Excessive resource consumption
- Poor indexing strategy
- Lack of useful statistics
- Lack of useful partitioning
- Consequence of blocked queries
- Incorrect server configurations

SQL Server Tiger Team

## How do I analyze the performance of a slow-running query?

## Some tools we will use today

- SHOWPLAN XML (a.k.a. Actual Execution Plan)
- Query Store
- Plan Comparison Tool
- Live Query Stats
- xEvents
- … and more ☺

# A map to the execution context

- Query plans include:
  - How data is accessed
  - How data is joined
  - Sequence of operations
  - Use of temporary worktables and sorts
  - Estimated rowcounts, iterations, and costs from each step
  - Actual rowcounts and iterations
  - How data is aggregated
  - Use of parallelism
  - Query execution warnings
  - Query execution stats
  - Hardware/Resource stats

SQL Server Tiger Team

## Diagnostic and Troubleshooting Enhancements

SQL Server Tiger Team

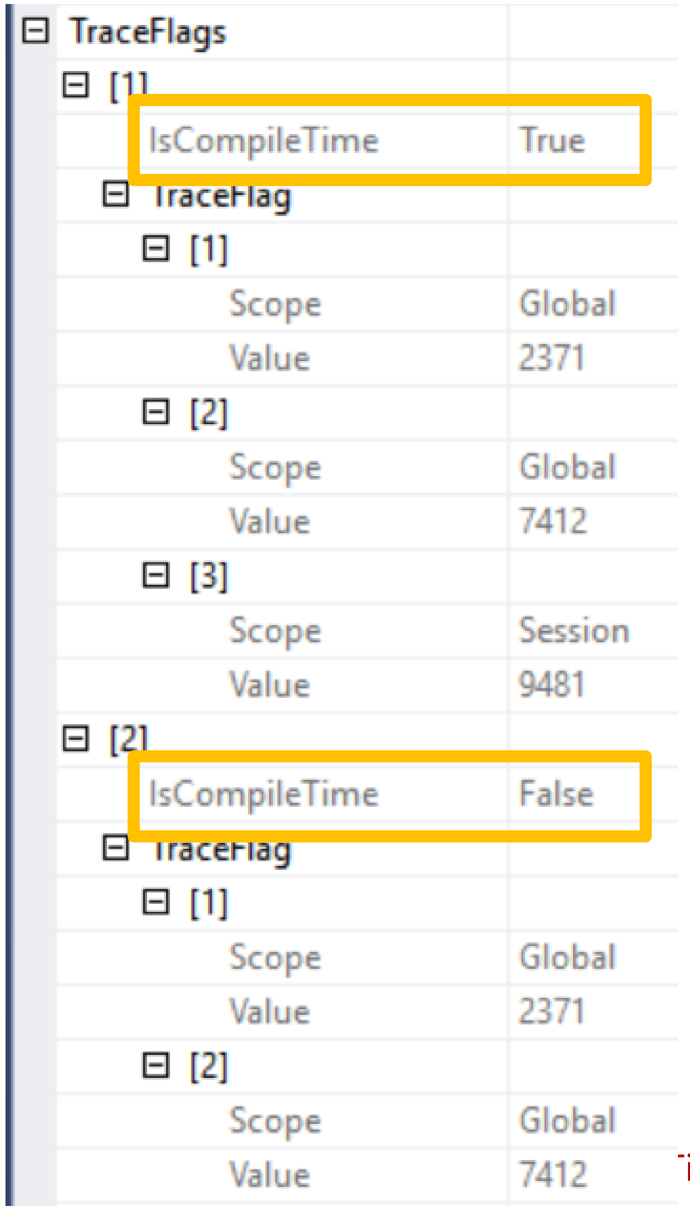
It’s all about avoiding roundtrips to collect additional information.

# Getting all context info in Showplan: Trace Flags

Shows list of active trace flags:

- Query
- Session
- Global

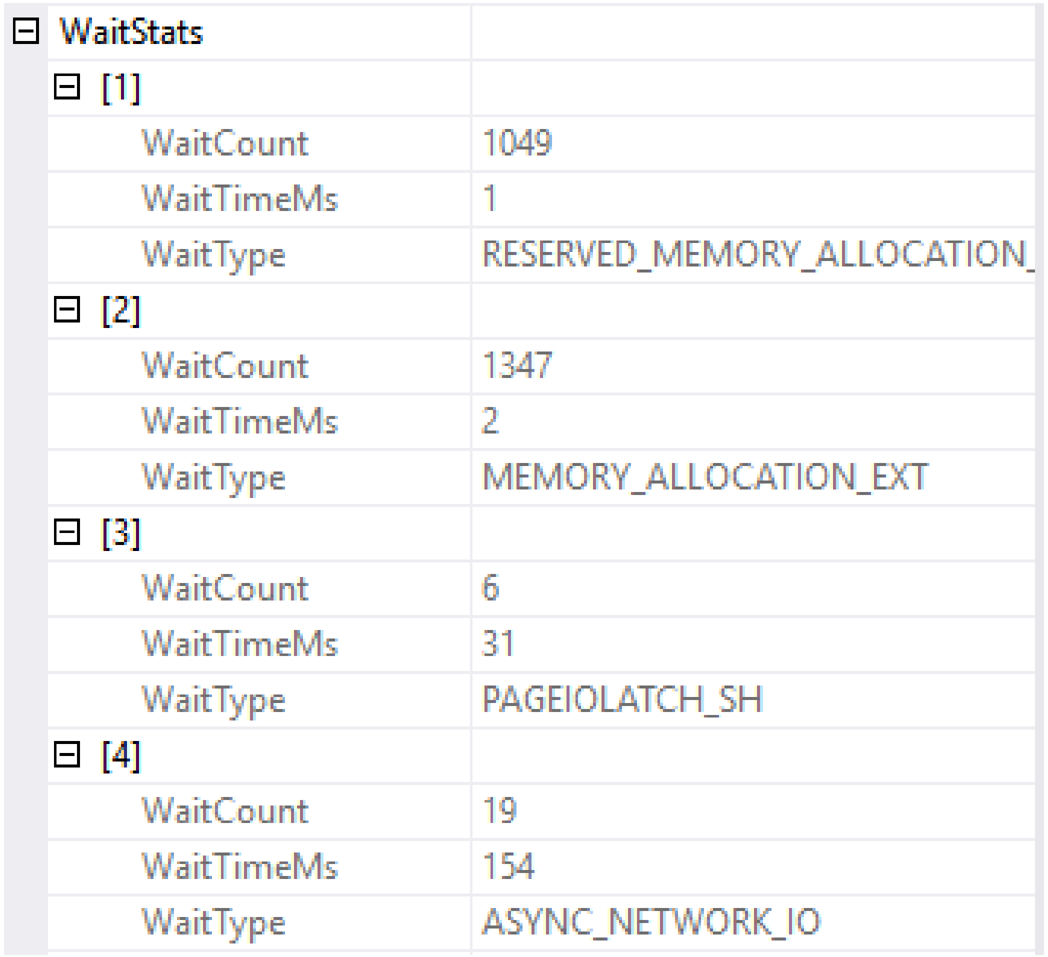
Useful to understand if active Trace Flags influence execution context



| | |
|---|---|
| ⊟ TraceFlags | |
| ⊟ [1] | |
| IsCompileTime | True |
| ⊟ TraceFlag | |
| ⊟ [1] | |
| Scope | Global |
| Value | 2371 |
| ⊟ [2] | |
| Scope | Global |
| Value | 7412 |
| ⊟ [3] | |
| Scope | Session |
| Value | 9481 |
| ⊟ [2] | |
| IsCompileTime | False |
| ⊟ TraceFlag | |
| ⊟ [1] | |
| Scope | Global |
| Value | 2371 |
| ⊟ [2] | |
| Scope | Global |
| Value | 7412 |

## Getting all context info in Showplan: Waits

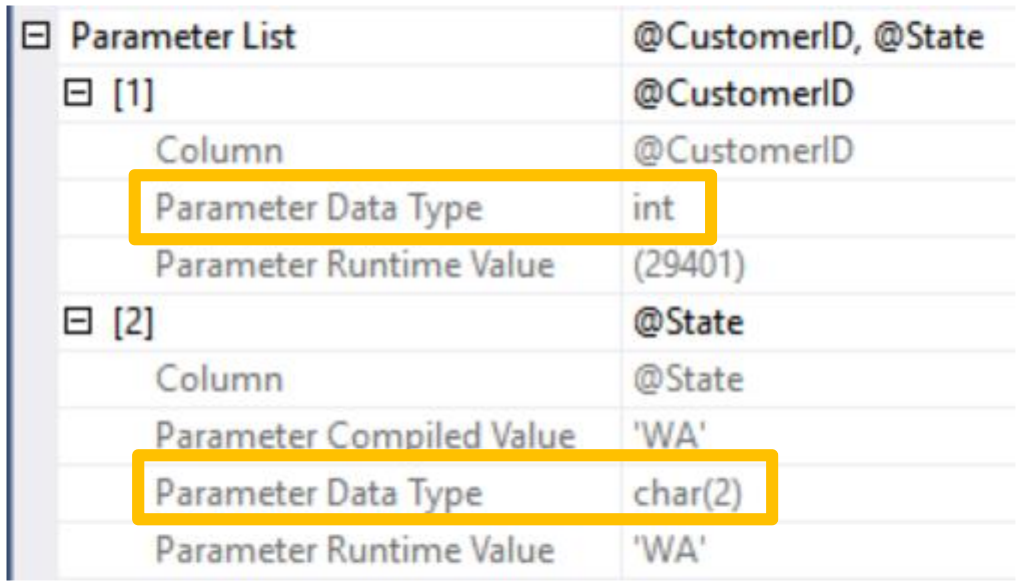
Shows top 10 waits from sys.dm_exec_session_wait_stats



| | |
|---|---|
| ⊟ WaitStats | |
| ⊟ [1] | |
| WaitCount | 1049 |
| WaitTimeMs | 1 |
| WaitType | RESERVED_MEMORY_ALLOCATION_ |
| ⊟ [2] | |
| WaitCount | 1347 |
| WaitTimeMs | 2 |
| WaitType | MEMORY_ALLOCATION_EXT |
| ⊟ [3] | |
| WaitCount | 6 |
| WaitTimeMs | 31 |
| WaitType | PAGEIOLATCH_SH |
| ⊟ [4] | |
| WaitCount | 19 |
| WaitTimeMs | 154 |
| WaitType | ASYNC_NETWORK_IO |

# Getting all context info in Showplan: Parameter Data Types

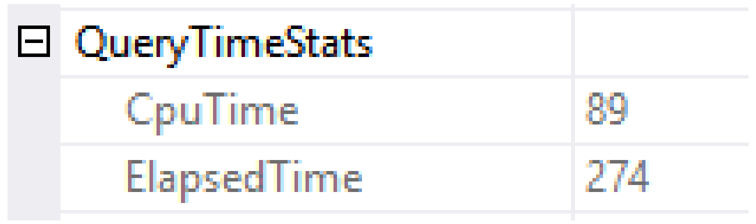
Easier detection of type conversion issues



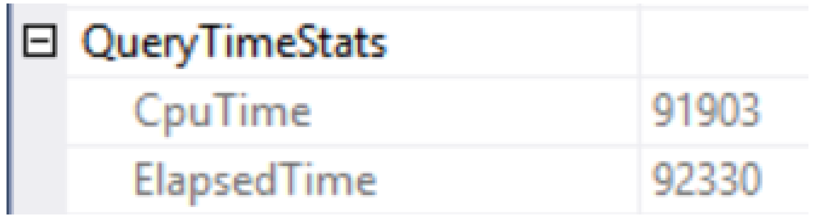
|  |  |
|---|---|
| ⊟ Parameter List | @CustomerID, @State |
| ⊟ [1] | @CustomerID |
| Column | @CustomerID |
| Parameter Data Type | int |
| Parameter Runtime Value | (29401) |
| ⊟ [2] | @State |
| Column | @State |
| Parameter Compiled Value | 'WA' |
| Parameter Data Type | char(2) |
| Parameter Runtime Value | 'WA' |

# Getting all context info in Showplan: Times

## Persisting information on elapsed and CPU times



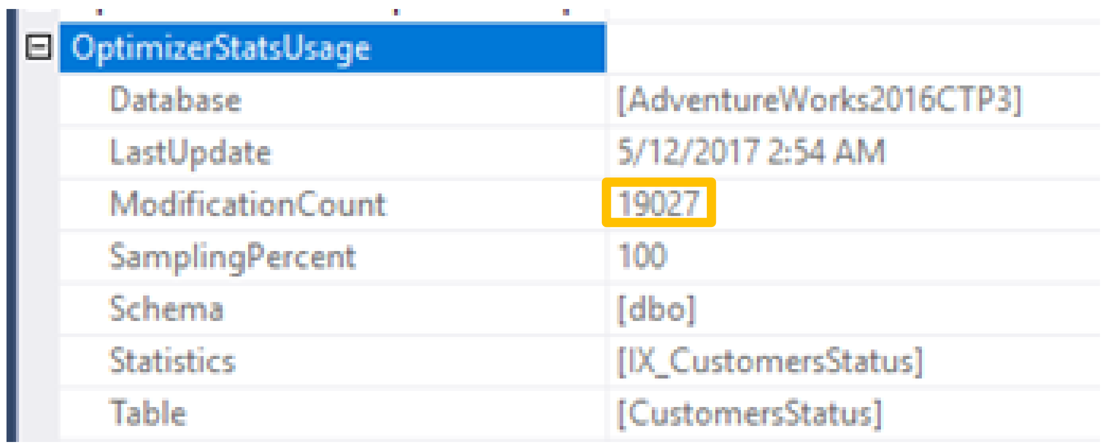
| QueryTimeStats | |
|---|---|
| CpuTime | 89 |
| ElapsedTime | 274 |



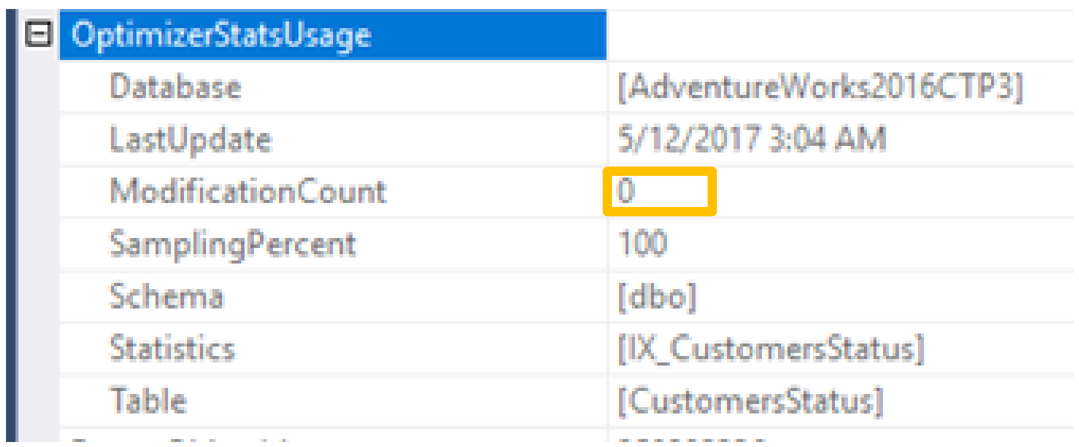
| QueryTimeStats | |
|---|---|
| CpuTime | 91903 |
| ElapsedTime | 92330 |

# Statistics information in Showplan

- Identify which statistics were used by the Query Optimizer for a given compilation.
- Gain actionable insight to where estimations came from.



| OptimizerStatsUsage | |
|---|---|
| Database | [AdventureWorks2016CTP3] |
| LastUpdate | 5/12/2017 2:54 AM |
| ModificationCount | 19027 |
| SamplingPercent | 100 |
| Schema | [dbo] |
| Statistics | [IX_CustomersStatus] |
| Table | [CustomersStatus] |



| OptimizerStatsUsage | |
|---|---|
| Database | [AdventureWorks2016CTP3] |
| LastUpdate | 5/12/2017 3:04 AM |
| ModificationCount | 0 |
| SamplingPercent | 100 |
| Schema | [dbo] |
| Statistics | [IX_CustomersStatus] |
| Table | [CustomersStatus] |

- Available in SQL Server 2017

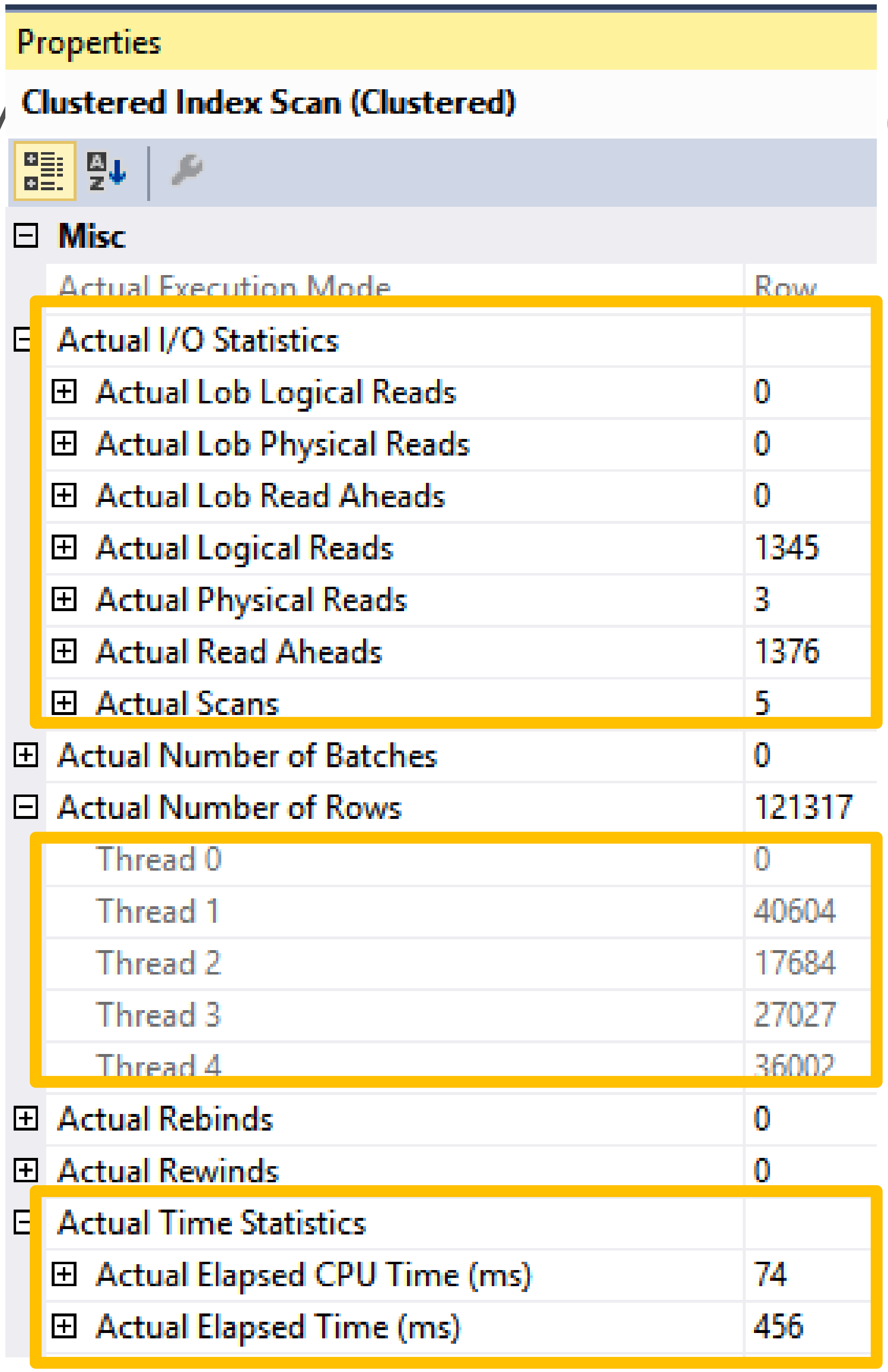
# Missing perf insights on query plan nodes

- Per operator performance statistics for node and threads
- Showplan extended to include *RunTimeCountersPerThread*
- Node costs for parent and children:
  - Cumulative values for Row mode operators
  - Singleton values for Batch mode operators

| Runtime Info | Up to SQL 2016 | SQL 2016 / SQL 2014 SP2 | SQL 2016 SP1 |
|---|---|---|---|
| **ActualRows** | X | X | X |
| ActualRowsRead | | X | X |
| Batches | | X | X |
| **ActualEndOfScans** | X | X | X |
| **ActualExecutions** | X | X | X |
| ActualExecutionMode | | X | X |
| ActualElapsedms | | X | X |
| ActualCPUms | | X | X |
| ActualScans | | X | X |
| ActualLogicalReads | | X | X |
| ActualPhysicalReads | | X | X |
| ActualReadAheads | | X | X |
| ActualLobLogicalReads | | X | X |
| ActualLobPhysicalReads | | X | X |
| ActualLobReadAheads | | X | X |
| InputMemoryGrant | | | X |
| OutputMemoryGrant | | | X |
| UsedMemoryGrant | | | X |

# M on query plan nodes

SQL Server Tiger Team



Properties

**Clustered Index Scan (Clustered)**

| | |
|---|---|
| ⊟ **Misc** | |
| Actual Execution Mode | Row |
| ⊟ Actual I/O Statistics | |
| ⊞ Actual Lob Logical Reads | 0 |
| ⊞ Actual Lob Physical Reads | 0 |
| ⊞ Actual Lob Read Aheads | 0 |
| ⊞ Actual Logical Reads | 1345 |
| ⊞ Actual Physical Reads | 3 |
| ⊞ Actual Read Aheads | 1376 |
| ⊞ Actual Scans | 5 |
| ⊞ Actual Number of Batches | 0 |
| ⊟ Actual Number of Rows | 121317 |
| Thread 0 | 0 |
| Thread 1 | 40604 |
| Thread 2 | 17684 |
| Thread 3 | 27027 |
| Thread 4 | 36002 |
| ⊞ Actual Rebinds | 0 |
| ⊞ Actual Rewinds | 0 |
| ⊟ Actual Time Statistics | |
| ⊞ Actual Elapsed CPU Time (ms) | 74 |
| ⊞ Actual Elapsed Time (ms) | 456 |

```
<RunTimeInformation>
<RunTimeCountersPerThread Thread="0" ActualRows="121317"
ActualRowsRead="10000000" Batches="0" ActualEndOfScans="3"
ActualExecutions="1" ActualExecutionMode="Row"
ActualElapsedms="456" ActualCPUms="74" ActualScans="3"
ActualLogicalReads="1345" ActualPhysicalReads="3"
ActualReadAheads="1376" ActualLobLogicalReads="0"
 ctualLobPhysicalReads="0" ActualLobReadAheads="0" />
<     TimeInformation>
```

SET STATISTICS IO not needed

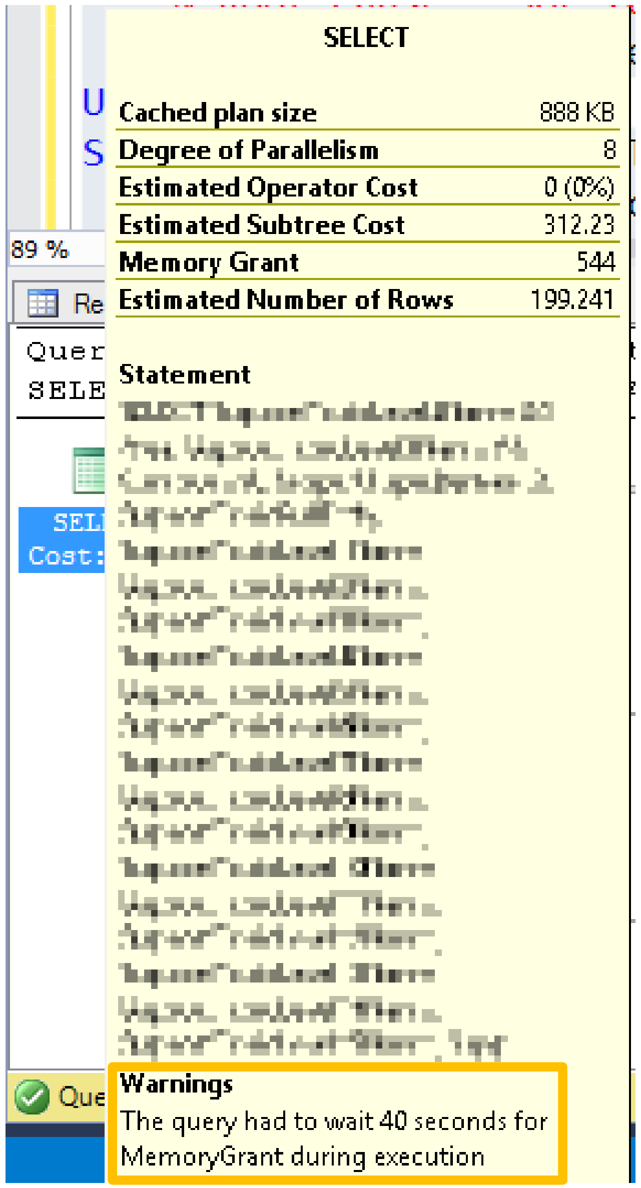
SET STATISTICS TIME not needed

SQL Server Tiger Team

# Demo

Per-operator level performance stats

# Memory Grant Wait Showplan Warning



SELECT

| | |
|---|---|
| Cached plan size | 888 KB |
| Degree of Parallelism | 8 |
| Estimated Operator Cost | 0 (0%) |
| Estimated Subtree Cost | 312.23 |
| Memory Grant | 544 |
| Estimated Number of Rows | 199.241 |

**Statement**

[redacted]

**Warnings**

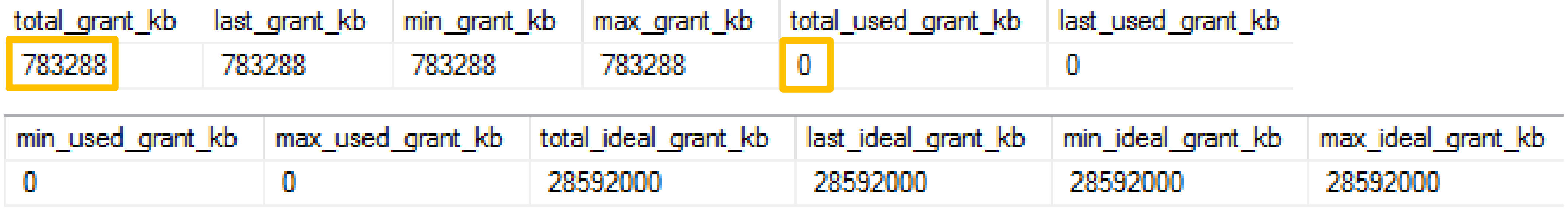
The query had to wait 40 seconds for MemoryGrant during execution

- Occurs when a T-SQL statement or stored procedure waits more than one second for a memory grant or when the initial attempt to get memory fails.
- Since SQL Server 2012

SQL Server Tiger Team

# Understanding memory usage per execution

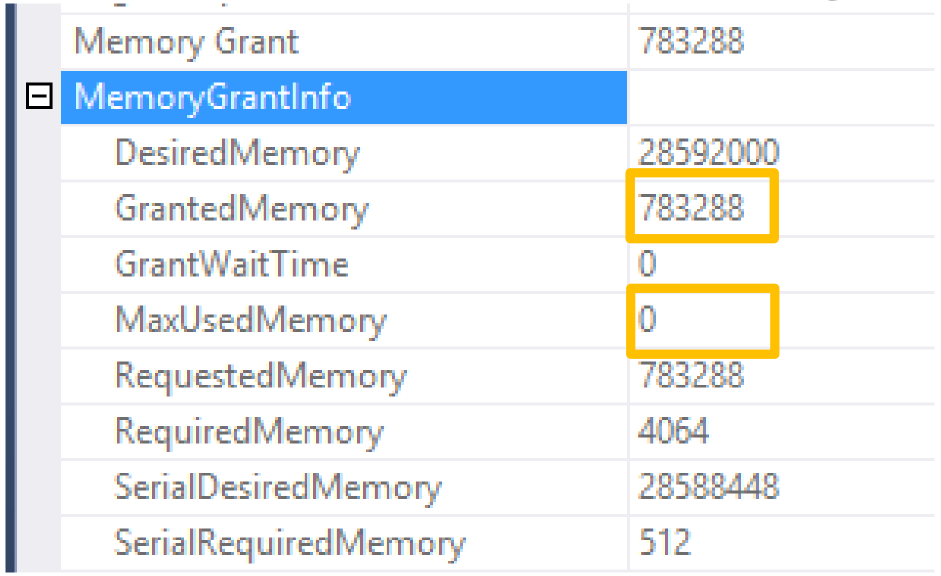
## New columns in *sys.dm_exec_query_stats*



| total_grant_kb | last_grant_kb | min_grant_kb | max_grant_kb | total_used_grant_kb | last_used_grant_kb |
|---|---|---|---|---|---|
| 783288 | 783288 | 783288 | 783288 | 0 | 0 |

| min_used_grant_kb | max_used_grant_kb | total_ideal_grant_kb | last_ideal_grant_kb | min_ideal_grant_kb | max_ideal_grant_kb |
|---|---|---|---|---|---|
| 0 | 0 | 28592000 | 28592000 | 28592000 | 28592000 |

## Showplan extended to include grant usage per thread and iterator



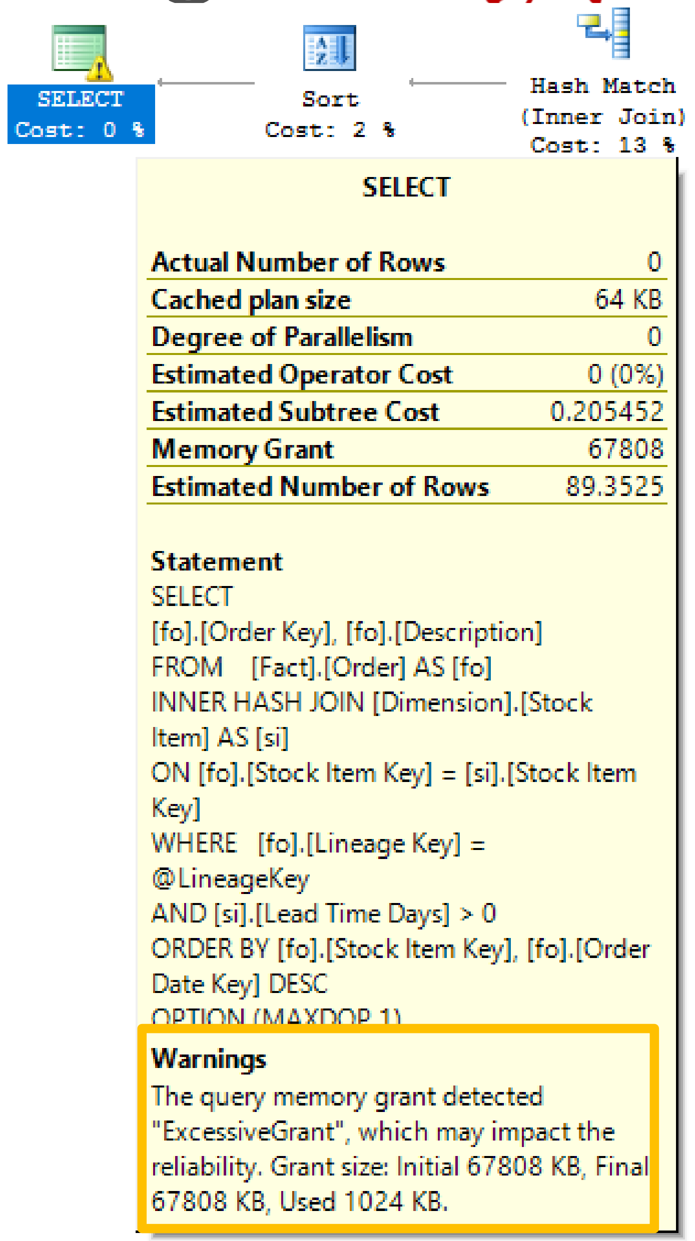
| | |
|---|---|
| Memory Grant | 783288 |
| MemoryGrantInfo | |
| DesiredMemory | 28592000 |
| GrantedMemory | 783288 |
| GrantWaitTime | 0 |
| MaxUsedMemory | 0 |
| RequestedMemory | 783288 |
| RequiredMemory | 4064 |
| SerialDesiredMemory | 28588448 |
| SerialRequiredMemory | 512 |

# New Memory Grant Showplan Warning

## SQL Server 2014 SP2 and SQL Server 2016 SP1

- 3 conditions:
  - **Excessive Grant**: when max used memory is too small compared to the granted memory. This scenario can cause blocking and less efficient usage when large grants exist and a fraction of that memory was used.

KB
3172997



SELECT

| | |
|---|---|
| Actual Number of Rows | 0 |
| Cached plan size | 64 KB |
| Degree of Parallelism | 0 |
| Estimated Operator Cost | 0 (0%) |
| Estimated Subtree Cost | 0.205452 |
| Memory Grant | 67808 |
| Estimated Number of Rows | 89.3525 |

**Statement**

```
SELECT
[fo].[Order Key], [fo].[Description]
FROM    [Fact].[Order] AS [fo]
INNER HASH JOIN [Dimension].[Stock
Item] AS [si]
ON [fo].[Stock Item Key] = [si].[Stock Item
Key]
WHERE   [fo].[Lineage Key] =
@LineageKey
AND [si].[Lead Time Days] > 0
ORDER BY [fo].[Stock Item Key], [fo].[Order
Date Key] DESC
OPTION (MAXDOP 1)
```

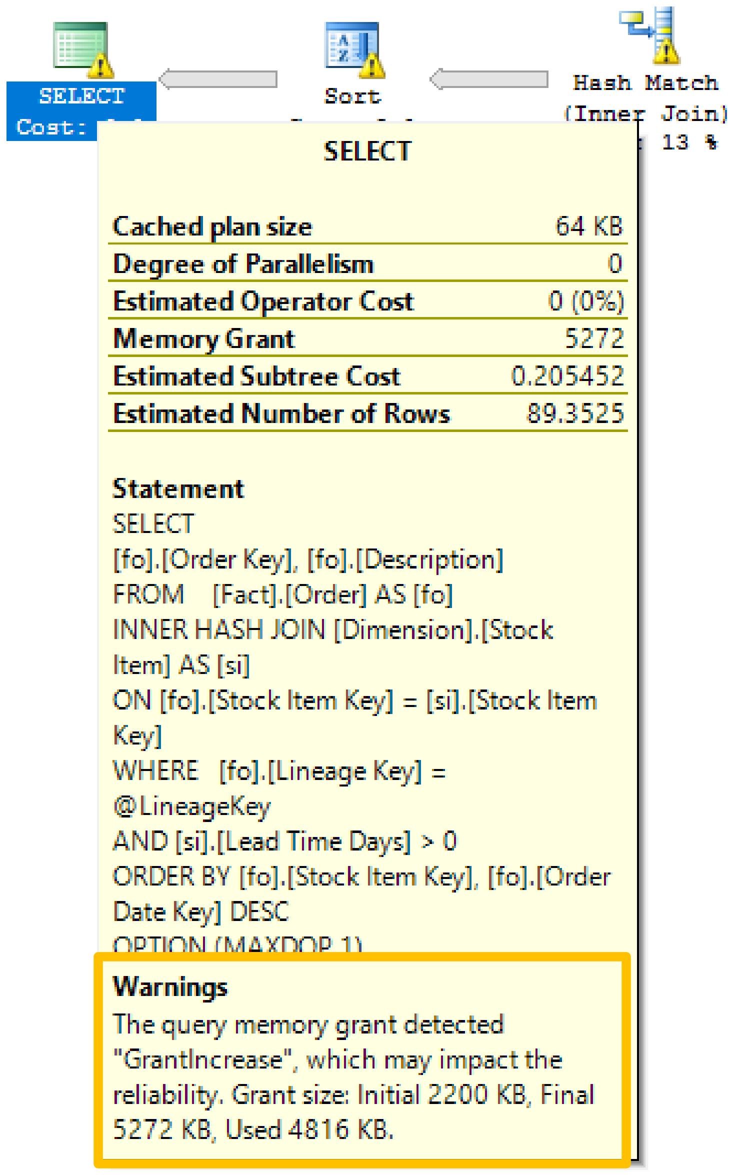
**Warnings**

The query memory grant detected "ExcessiveGrant", which may impact the reliability. Grant size: Initial 67808 KB, Final 67808 KB, Used 1024 KB.

# New Memory Grant Showplan Warning

SQL Server 2014 SP2 and SQL Server 2016 SP1

- 3 conditions:
  - **Excessive Grant**: when max used memory is too small compared to the granted memory. This scenario can cause blocking and less efficient usage when large grants exist and a fraction of that memory was used.
  - **Grant Increase**: when the dynamic grant starts to increase too much, based on the ratio between the max used memory and initial request memory. This scenario can cause server instability and unpredictable workload performance.
  - **Used More Than Granted**: when the max used memory exceeds the granted memory. This scenario can cause OOM conditions on the server.



SELECT

| | |
|---|---|
| **Cached plan size** | 64 KB |
| **Degree of Parallelism** | 0 |
| **Estimated Operator Cost** | 0 (0%) |
| **Memory Grant** | 5272 |
| **Estimated Subtree Cost** | 0.205452 |
| **Estimated Number of Rows** | 89.3525 |

**Statement**

```
SELECT
[fo].[Order Key], [fo].[Description]
FROM    [Fact].[Order] AS [fo]
INNER HASH JOIN [Dimension].[Stock
Item] AS [si]
ON [fo].[Stock Item Key] = [si].[Stock Item
Key]
WHERE   [fo].[Lineage Key] =
@LineageKey
AND [si].[Lead Time Days] > 0
ORDER BY [fo].[Stock Item Key], [fo].[Order
Date Key] DESC
OPTION (MAXDOP 1)
```

**Warnings**

The query memory grant detected "GrantIncrease", which may impact the reliability. Grant size: Initial 2200 KB, Final 5272 KB, Used 4816 KB.

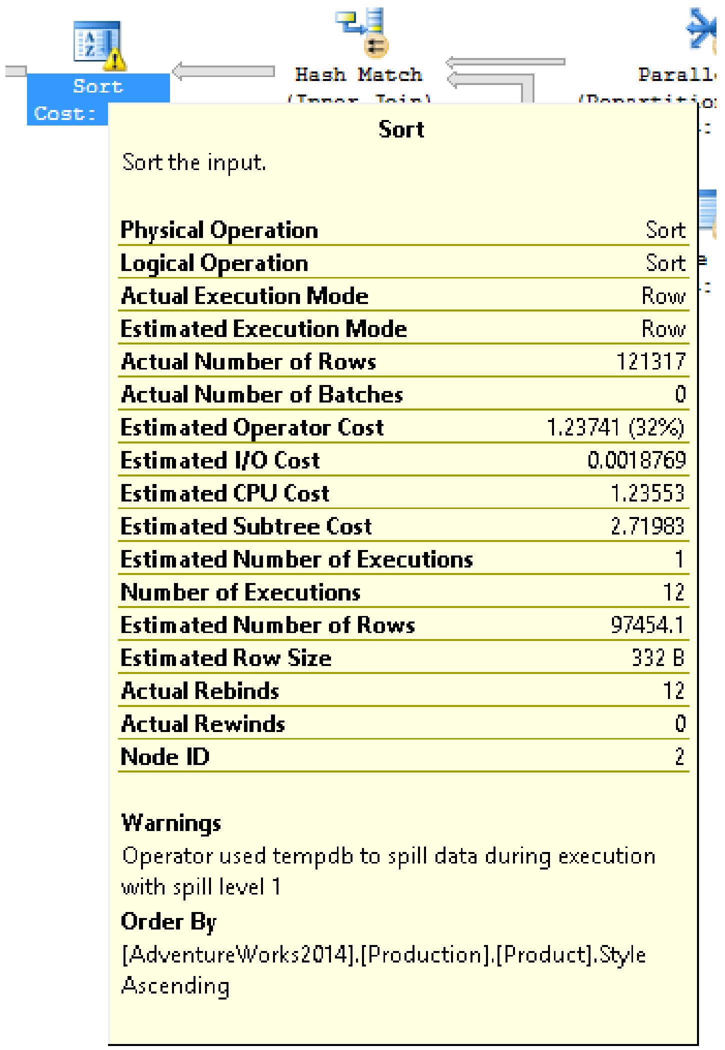
# min and max query memory grant option

## SQL Server 2016 and SQL Server 2014 SP2

- User control over min and max grant size in percentages
  - OPTION (MAX_GRANT_PERCENT=0.1), meaning 0.1% of max allowed query memory under Resource Governor
  - The valid value is between 0 and 100%
  - MAX_GRANT_PERCENT >= MIN_GRANT_PERCENT
- Why use a floating point value?
  - On a high end machine with 1 TB of memory, 1% can be already 10GB

# New Spills Warnings - Sort

## Up to SQL Server 2016



**Sort**

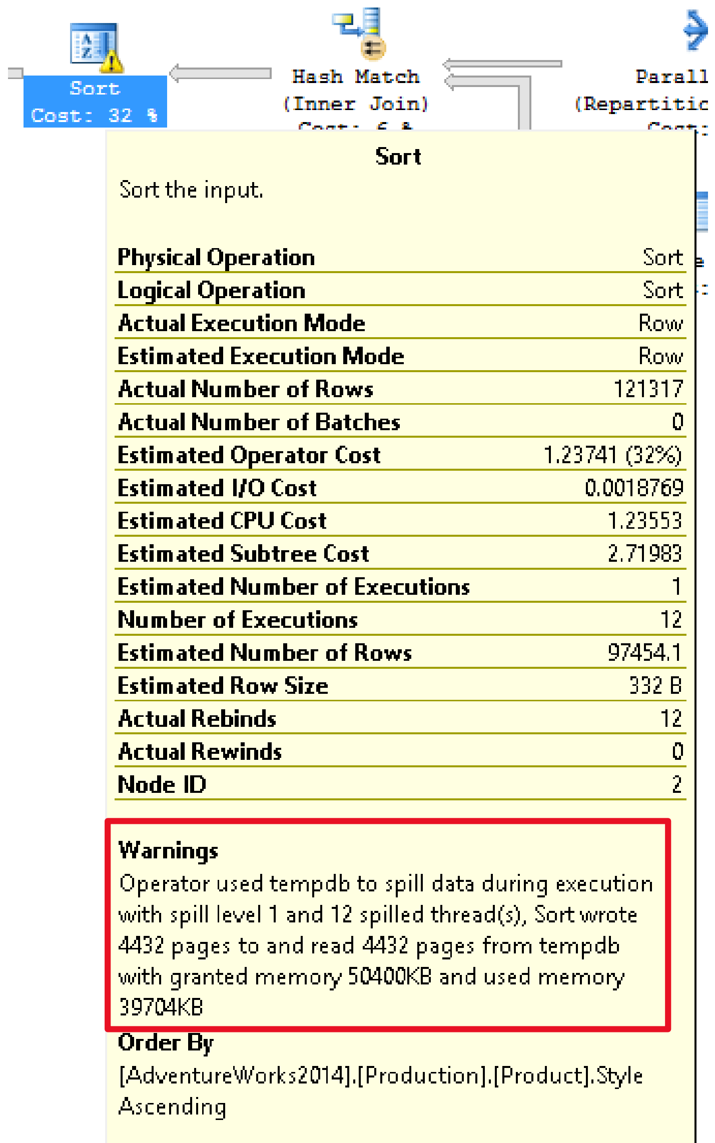
Sort the input.

| | |
|---|---|
| **Physical Operation** | Sort |
| **Logical Operation** | Sort |
| **Actual Execution Mode** | Row |
| **Estimated Execution Mode** | Row |
| **Actual Number of Rows** | 121317 |
| **Actual Number of Batches** | 0 |
| **Estimated Operator Cost** | 1.23741 (32%) |
| **Estimated I/O Cost** | 0.0018769 |
| **Estimated CPU Cost** | 1.23553 |
| **Estimated Subtree Cost** | 2.71983 |
| **Estimated Number of Executions** | 1 |
| **Number of Executions** | 12 |
| **Estimated Number of Rows** | 97454.1 |
| **Estimated Row Size** | 332 B |
| **Actual Rebinds** | 12 |
| **Actual Rewinds** | 0 |
| **Node ID** | 2 |

**Warnings**
Operator used tempdb to spill data during execution with spill level 1

**Order By**
[AdventureWorks2014].[Production].[Product].Style Ascending

## SQL Server 2016 and 2014 SP2



**Sort**

Sort the input.

| | |
|---|---|
| **Physical Operation** | Sort |
| **Logical Operation** | Sort |
| **Actual Execution Mode** | Row |
| **Estimated Execution Mode** | Row |
| **Actual Number of Rows** | 121317 |
| **Actual Number of Batches** | 0 |
| **Estimated Operator Cost** | 1.23741 (32%) |
| **Estimated I/O Cost** | 0.0018769 |
| **Estimated CPU Cost** | 1.23553 |
| **Estimated Subtree Cost** | 2.71983 |
| **Estimated Number of Executions** | 1 |
| **Number of Executions** | 12 |
| **Estimated Number of Rows** | 97454.1 |
| **Estimated Row Size** | 332 B |
| **Actual Rebinds** | 12 |
| **Actual Rewinds** | 0 |
| **Node ID** | 2 |

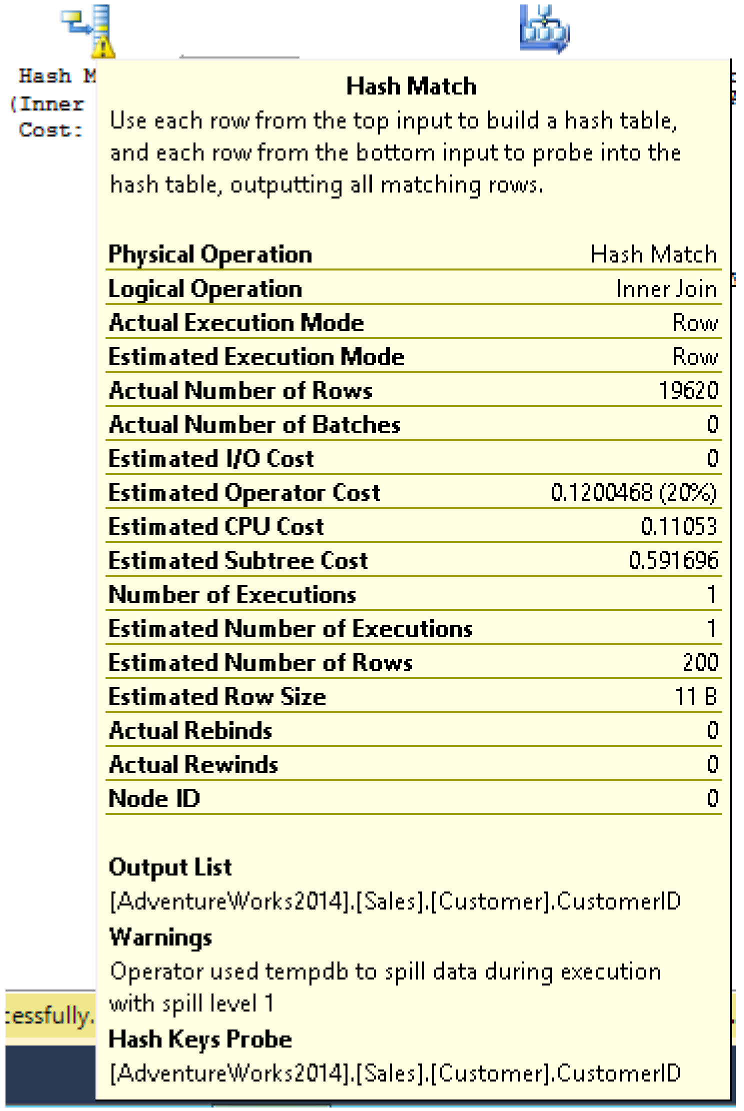
**Warnings**
Operator used tempdb to spill data during execution with spill level 1 and 12 spilled thread(s), Sort wrote 4432 pages to and read 4432 pages from tempdb with granted memory 50400KB and used memory 39704KB

**Order By**
[AdventureWorks2014].[Production].[Product].Style Ascending

SQL Server Tiger Team

# New Spills Warnings - Hash

## Up to SQL Server 2016



Hash M
(Inner
Cost:

**Hash Match**

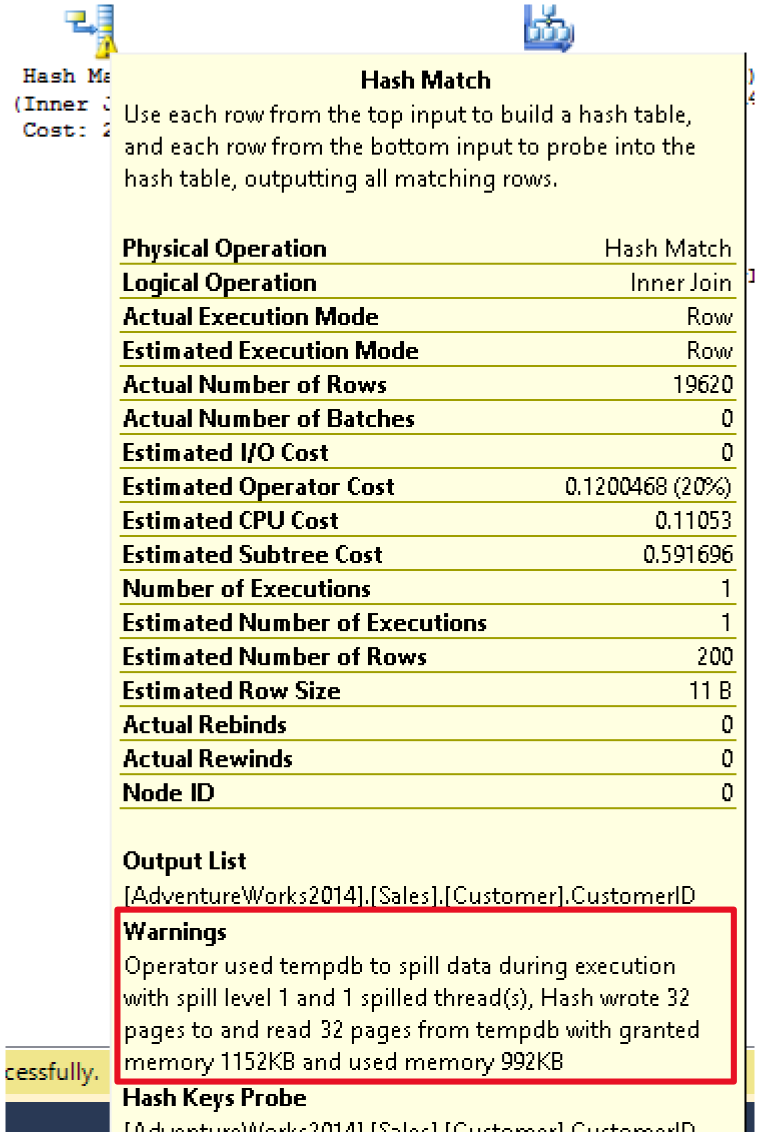
Use each row from the top input to build a hash table, and each row from the bottom input to probe into the hash table, outputting all matching rows.

| | |
|---|---|
| **Physical Operation** | Hash Match |
| **Logical Operation** | Inner Join |
| **Actual Execution Mode** | Row |
| **Estimated Execution Mode** | Row |
| **Actual Number of Rows** | 19620 |
| **Actual Number of Batches** | 0 |
| **Estimated I/O Cost** | 0 |
| **Estimated Operator Cost** | 0.1200468 (20%) |
| **Estimated CPU Cost** | 0.11053 |
| **Estimated Subtree Cost** | 0.591696 |
| **Number of Executions** | 1 |
| **Estimated Number of Executions** | 1 |
| **Estimated Number of Rows** | 200 |
| **Estimated Row Size** | 11 B |
| **Actual Rebinds** | 0 |
| **Actual Rewinds** | 0 |
| **Node ID** | 0 |

**Output List**
[AdventureWorks2014].[Sales].[Customer].CustomerID
**Warnings**
Operator used tempdb to spill data during execution with spill level 1
**Hash Keys Probe**
[AdventureWorks2014].[Sales].[Customer].CustomerID

:essfully.

## SQL Server 2016 and 2014 SP2



Hash Ma
(Inner J
Cost: 2

**Hash Match**

Use each row from the top input to build a hash table, and each row from the bottom input to probe into the hash table, outputting all matching rows.

| | |
|---|---|
| **Physical Operation** | Hash Match |
| **Logical Operation** | Inner Join |
| **Actual Execution Mode** | Row |
| **Estimated Execution Mode** | Row |
| **Actual Number of Rows** | 19620 |
| **Actual Number of Batches** | 0 |
| **Estimated I/O Cost** | 0 |
| **Estimated Operator Cost** | 0.1200468 (20%) |
| **Estimated CPU Cost** | 0.11053 |
| **Estimated Subtree Cost** | 0.591696 |
| **Number of Executions** | 1 |
| **Estimated Number of Executions** | 1 |
| **Estimated Number of Rows** | 200 |
| **Estimated Row Size** | 11 B |
| **Actual Rebinds** | 0 |
| **Actual Rewinds** | 0 |
| **Node ID** | 0 |

**Output List**
[AdventureWorks2014].[Sales].[Customer].CustomerID
**Warnings**
Operator used tempdb to spill data during execution with spill level 1 and 1 spilled thread(s), Hash wrote 32 pages to and read 32 pages from tempdb with granted memory 1152KB and used memory 992KB
**Hash Keys Probe**

cessfully.

SQL Server Tiger Team

# Detecting predicate search inefficiencies?

*Actual number of rows* returned are rows <u>after</u> the predicate is applied.

- Not the actual number of rows that are scanned from a table or index.

Scenario hidden from an actual execution plan:

- SCAN or SEEK returns only 10 rows, why is it taking so long?
- You see high CPU or many logical reads, but the query plan doesn't reflect that.

Now what?? ☹

# Searching without pushdown

SQL Server Tiger Team

```
SELECT [ProductID]
FROM [Sales].[SalesOrderDetail]
WHERE [ModifiedDate] BETWEEN '2011-01-01' AND '2012-01-01'
AND [OrderQty] >= 10
```

Sales.SalesOrderDetail

| ModifiedDate | ProductID | Store | | lesAmount |
|---|---|---|---|---|
| 2010-12-31 | 106 | 01 | | 30 |
| 2011-01-07 | 103 | 04 | 1 | 17 |
| 2011-01-07 | 109 | 04 | 7 | |
| 2011-02-12 | 103 | 03 | 5 | |
| 2011-03-08 | 106 | 05 | 7 | 25 |
| 2011-04-16 | **106** | 02 | 10 | 40 |
| 2011-07-20 | **102** | 02 | 12 | 50 |
| 2011-10-21 | **106** | 03 | 16 | 55 |
| 2011-12-15 | **103** | 03 | 20 | 55 |
| (...) | **(...)** | (...) | (...) | (...) |
| 2012-01-01 | **109** | 01 | 11 | 16 |
| 2012-01-11 | 102 | 05 | 5 | 10 |

Actual Rows

Filter

Range Scan

Result Set

| ModifiedDate | ProductID | StoreID | OrderQty | SalesAmount |
|---|---|---|---|---|
| 2011-04-16 | 106 | 02 | 10 | 40 |
| 2011-07-20 | 102 | 02 | 12 | |
| 2011-10-21 | 106 | 03 | 16 | |
| 2011-12-15 | 103 | 03 | 20 | 55 |
| (...) | (...) | (...) | (...) | (...) |
| 2012-01-01 | 109 | 01 | 11 | 16 |

# Searching with pushdown

SQL Server Tiger Team

```
SELECT [ProductID]
FROM [Sales].[SalesOrderDetail]
WHERE [ModifiedDate] BETWEEN '2011-01-01' AND '2012-01-01'
AND [OrderQty] >= 10
```

Sales.SalesOrderDetail

| ModifiedDate | ProductID | StoreID | OrderQty | SalesAmount |
|---|---|---|---|---|
| 2010-12-31 | 106 | 01 | 12 | 30 |
| 2011-01-07 | [illegible] | [illegible] | 1 | 17 |
| 2011-01-07 | [illegible] | [illegible] | 7 | 20 |
| 2011-02-12 | [illegible] | 03 | 5 | 40 |
| 2011-03-08 | [illegible] | 05 | 7 | 25 |
| 2011-04-16 | **106** | 02 | 10 | 40 |
| 2011-07-20 | **102** | 02 | 12 | 50 |
| 2011-10-21 | **106** | 03 | 16 | 55 |
| 2011-12-15 | **103** | 03 | 20 | 55 |
| (...) | **(...)** | (...) | (...) | (...) |
| 2012-01-01 | **109** | 01 | 11 | 16 |
| 2012-01-11 | 102 | 05 | 5 | 10 |

Range Scan

Actual Rows

Result Set

| ModifiedDate | ProductID | StoreID | OrderQty | SalesAmount |
|---|---|---|---|---|
| 2011-04-16 | 106 | 02 | 10 | 40 |
| 2011-07-20 | 102 | 02 | 12 | 50 |
| 2011-10-21 | 106 | 03 | 16 | 55 |
| 2011-12-15 | 103 | 03 | 20 | 55 |
| (...) | (...) | (...) | (...) | (...) |
| 2012-01-01 | 109 | 01 | 11 | 16 |

SQL Server Tiger Team

# Demo

Predicate Pushdown in Showplan

## Production Alert: Application is slow!

SQL Server Tiger Team

Run data collection tools:

PSSDiag

xEvents

Profiler

…

All after the fact

Take the data:

Repro

Analyze

Deploy mitigation

## Production Alert: Query Perf Issues!

What if I could do live query troubleshooting?

- To have in-flight query execution statistics, the *query execution statistics profile infrastructure* must be enabled **on demand**.
- But cost overhead makes bad things worse if running all the time.
- This is why you are still reverting back to the pattern *collect data + post-collection analysis*.

# Tracking query progress (estimated)

- To have in-flight query execution statistics, the *query execution statistics profile infrastructure* must be enabled on demand.
- Can be enabled for a target session:
  - Specifying Include Live Query Statistics in SSMS.
  - SET STATISTICS XML ON
  - SET STATISTICS PROFILE ON
- Or globally to view the LQS from other sessions (such as from Activity Monitor):
  - Enabling *query_post_execution_showplan* extended event.
- High overhead (75% with TPC-C like workload)

KB 3170113

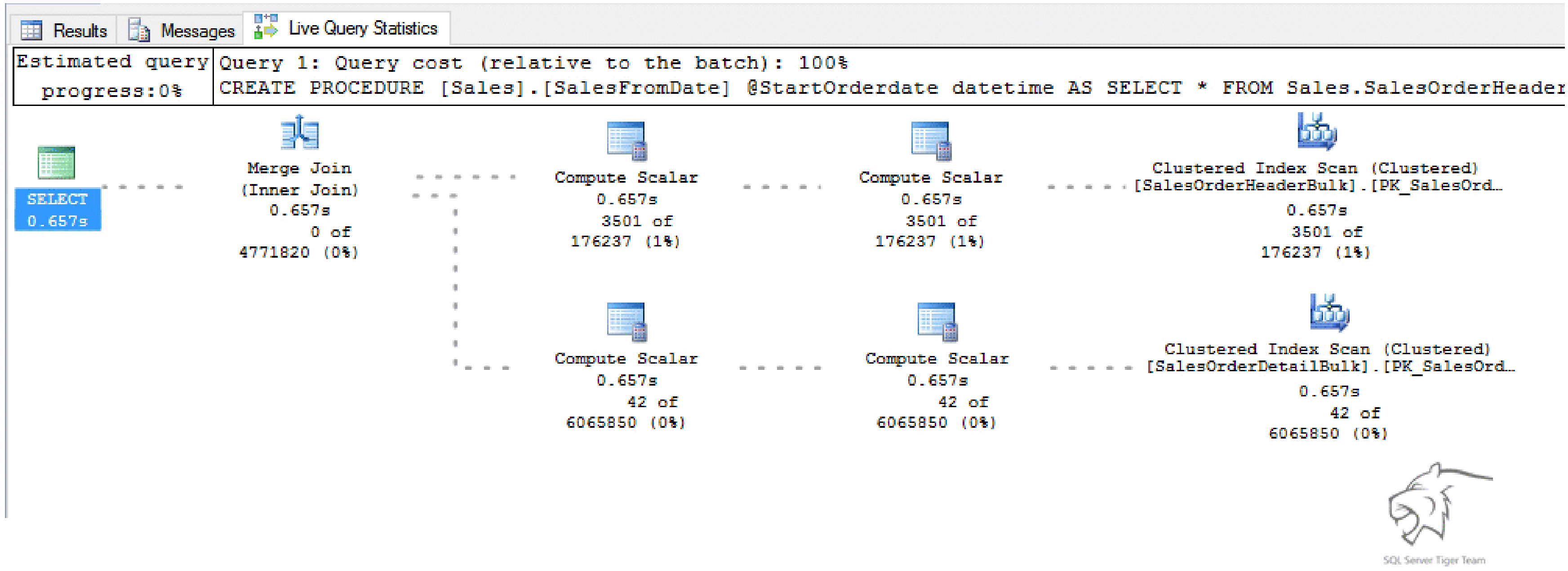
# Unleash Lightweight Profiling

SQL Server Tiger Team

Tap to in-flight execution

Find the hotspot

Deploy mitigation



```
Estimated query progress:0%
Query 1: Query cost (relative to the batch): 100%
CREATE PROCEDURE [Sales].[SalesFromDate] @StartOrderdate datetime AS SELECT * FROM Sales.SalesOrderHeader
```

```
SELECT
0.657s

Merge Join
(Inner Join)
0.657s
0 of
4771820 (0%)

Compute Scalar
0.657s
3501 of
176237 (1%)

Compute Scalar
0.657s
3501 of
176237 (1%)

Clustered Index Scan (Clustered)
[SalesOrderHeaderBulk].[PK_SalesOrd…
0.657s
3501 of
176237 (1%)

Compute Scalar
0.657s
42 of
6065850 (0%)

Compute Scalar
0.657s
42 of
6065850 (0%)

Clustered Index Scan (Clustered)
[SalesOrderDetailBulk].[PK_SalesOrd…
0.657s
42 of
6065850 (0%)
```

SQL Server Tiger Team

## Lightweight Tracking query progress (estimated)

- Lightweight query execution profiling **dramatically reduces performance overhead** of continuously collecting per-operator query execution statistics (such as actual number of rows)
- Can be enabled by:
  - Using global TF 7412.
  - Enabling *query_thread_profile* extended event.
  - When lightweight profiling is on, *sys.dm_exec_query_profiles* is also populated for all sessions.
- This enables usage of LQS feature in SSMS (including Activity Monitor) and of the new DMF **sys.dm_exec_query_statistics_xml**.
- The following still use regular profiling infra:
  - SET STATISTICS XML (or Include Actual Plan).
  - *query_post_execution_showplan* extended event.

KB 3190871

# What is the impact of live query troubleshooting?

## Query Execution Statistics Profiling Infrastructure tests with TPC-C like workloads

**Overhead percent (up to)**

| **Infra Type** | **no active xEvents** | **Active xEvent query_post_execution_showplan** |
|---|---|---|
| Regular | **75.5** | 93.17 |
| Lightweight in SQL Server 2014 SP2/2016 | 3.5 | 62.02 |
| Lightweight in SQL Server 2016 SP1 | **2** | 14.3 |

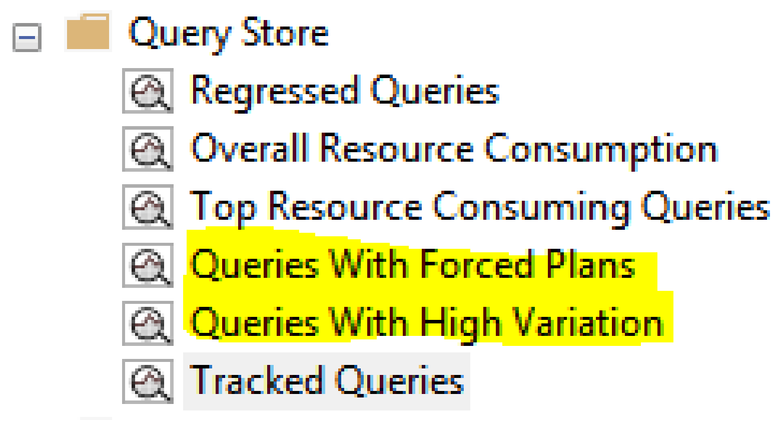
SQL Server Tiger Team

## Demo

Live Query Troubleshooting

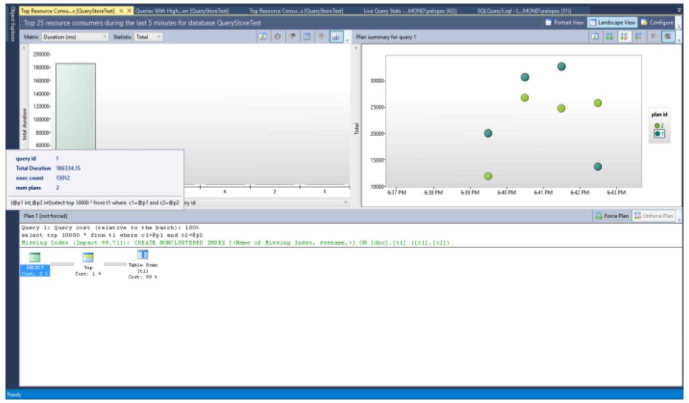
# Query performance insights in SSMS

- Still in last v16:
  - Support for multi-statement showplan comparison
  - Per-operator level performance stats in showplan Properties window
  - Query Store
    - Filter by number of different plans
- New with v17:
  - Query Store: new reports
  - Query analysis scenarios: Introduces CE diff search

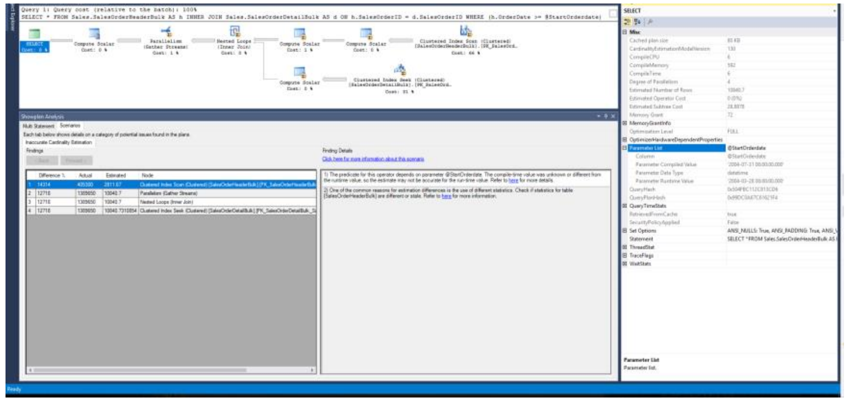

Query Store
Regressed Queries
Overall Resource Consumption
Top Resource Consuming Queries
Queries With Forced Plans
Queries With High Variation
Tracked Queries

# How can you leverage SSMS experience?

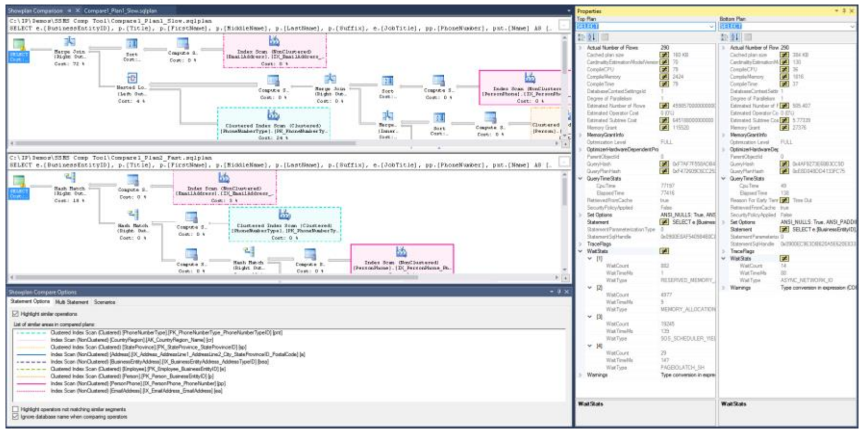
## Query Store UI



## SSMS Plan Analysis

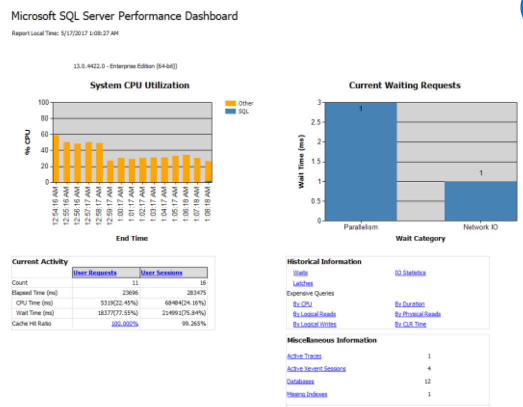


## SSMS Plan Comparison



## SSMS Performance Dashboard

Coming Soon!



Microsoft SQL Server Performance Dashboard

Report Local Time: 5/17/2017 1:08:27 AM

13.0.4422.0 - Enterprise Edition (64-bit)

Current Activity

| | User Requests | User Sessions |
|---|---|---|
| Count | 11 | 16 |
| Elapsed Time (ms) | 23696 | 283475 |
| CPU Time (ms) | 5319(22.45%) | 68484(24.16%) |
| Wait Time (ms) | 18377(77.55%) | 214991(75.84%) |
| Cache Hit Ratio | 100.000% | 99.265% |

Historical Information

Waits — IO Statistics

Latches

Expensive Queries

By CPU — By Duration

By Logical Reads — By Physical Reads

By Logical Writes — By CLR Time

Miscellaneous Information

| | |
|---|---|
| Active Traces | 1 |
| Active Xevent Sessions | 4 |
| Databases | 12 |
| Missing Indexes | 1 |

SQL Server Tiger Team

# Demo

Query Store UI and Plan Comparison

Plan Analysis

SQL Server Tiger Team

## In closing…

## …remember that getting to SQL Server 2016 or Azure SQL DB is easy!

SQL Server Tiger Team

### **Assess** with DMA (Data Migration Assistant)

- Detecting code compatibility issues between SQL versions
- Recommends performance and reliability improvements for your target environment (Columnstore, In-memory, etc)

### **Migrate** with DMA, SSMA or DMS (Azure Data Migration Service)

- Move your schema, data, and uncontained objects from your source server to your target server
- Move from SQL, Oracle, MySQL, Sybase and DB2
- Sign-up for DMS private previews at aka.ms/sqldatabase-migrationpreview

### **Validate** with DEA (Data Experimentation Assistant)

- A/B testing solution for changes in SQL Server environments (e.g. upgrade, new PDS design, etc.).
- Do it before migration to catch performance regressions while testing your good known workload

# Bookmarks

| | |
|---|---|
| SQL Server Tiger Team Blog | http://aka.ms/sqlserverteam |
| Tiger Toolbox GitHub | http://aka.ms/tigertoolbox |
| SQL Server Release Blog | http://aka.ms/sqlreleases |
| **B**est **P**ractices Check | http://aka.ms/bpcheck |
| SQL Server Standards Support | http://aka.ms/sqlstandards |
| Trace Flags | http://aka.ms/traceflags |
| SQL Server Support lifecycle | http://aka.ms/sqllifecycle |
| SQL Server Updates | http://aka.ms/sqlupdates |
| Twitter | @mssqltiger |

Microsoft