

From Zero to Hero: SQL Server Performance Made Easier

Pedro Lopes



Pedro Lopes

@sqlpto

pedro.lopes@microsoft.com

Senior Program Manager

Focused on SQL Server
Relational Engine and
Security

8+ years at Microsoft

15+ years with SQL
Server



Session Objectives And Takeaways



SQL Server Tiger Team



Show new diagnostics improvements for SQL Server engine

Show new diagnostic tools built into SQL Server for performance scenarios



Learn how to use the new diagnostics to troubleshoot common performance issues

Show of hands



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



Context for Slow-Running Query Analysis



- Is the performance problem related to a component other than queries?
 - For example, is the problem slow network performance?
 - Are there any other components that might be causing or contributing to performance degradation?
- If the performance issue is related to queries, which query or set of queries is involved?
- Was the query optimized with useful statistics?
- Are suitable indexes available?
- Are there any data or index hot spots?
- If you have a large volume of data, do you need to partition it?
- Is the Query Optimizer provided with the best opportunity to optimize a complex query?

Performance Dashboard in SSMS



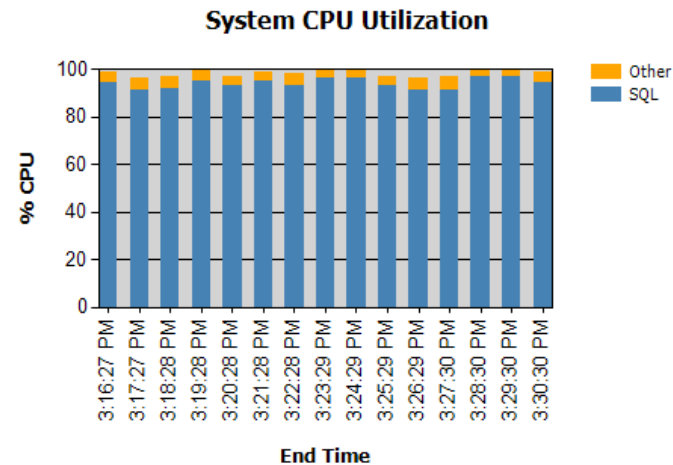
Microsoft SQL Server Performance Dashboard

Report Local Time: 5/31/2017 3:31:04 PM

(13.0.4422.0 - Enterprise Edition (64-bit))

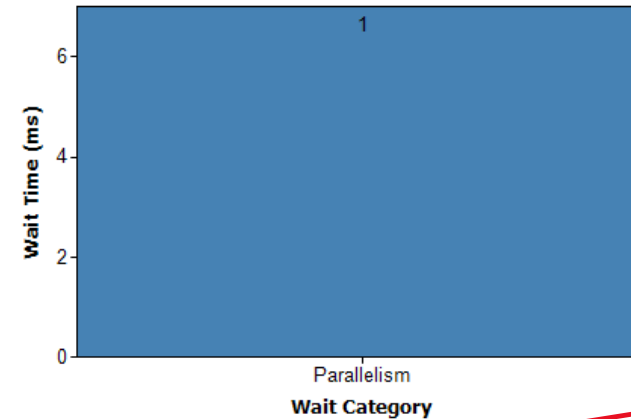


Overall performance may be degraded because the system shows signs of being CPU-bound. This SQL Server instance is consuming the majority of the CPU. Click on any of the SQL data points in the chart below to investigate further.



Current Activity		
	User Requests	User Sessions
Count	27	32
Elapsed Time (ms)	4573004	741818
CPU Time (ms)	2043203(44.68%)	101108(13.63%)
Wait Time (ms)	2529801(55.32%)	640710(86.37%)
Cache Hit Ratio	100.000%	98.313%

Current Waiting Requests



No extra downloads!
No new schema to deploy!
Long standing request by
CSS and customers

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	16
Missing Indexes	11

Categorized Wait
stats page

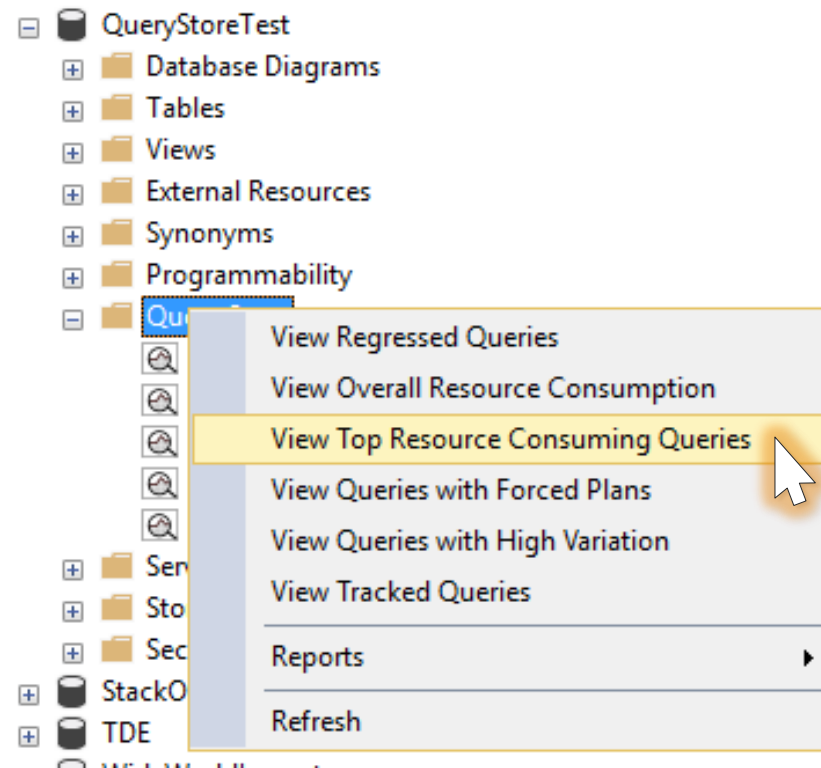
New categorized
Latches page

Scoring added to
Missing Index Report

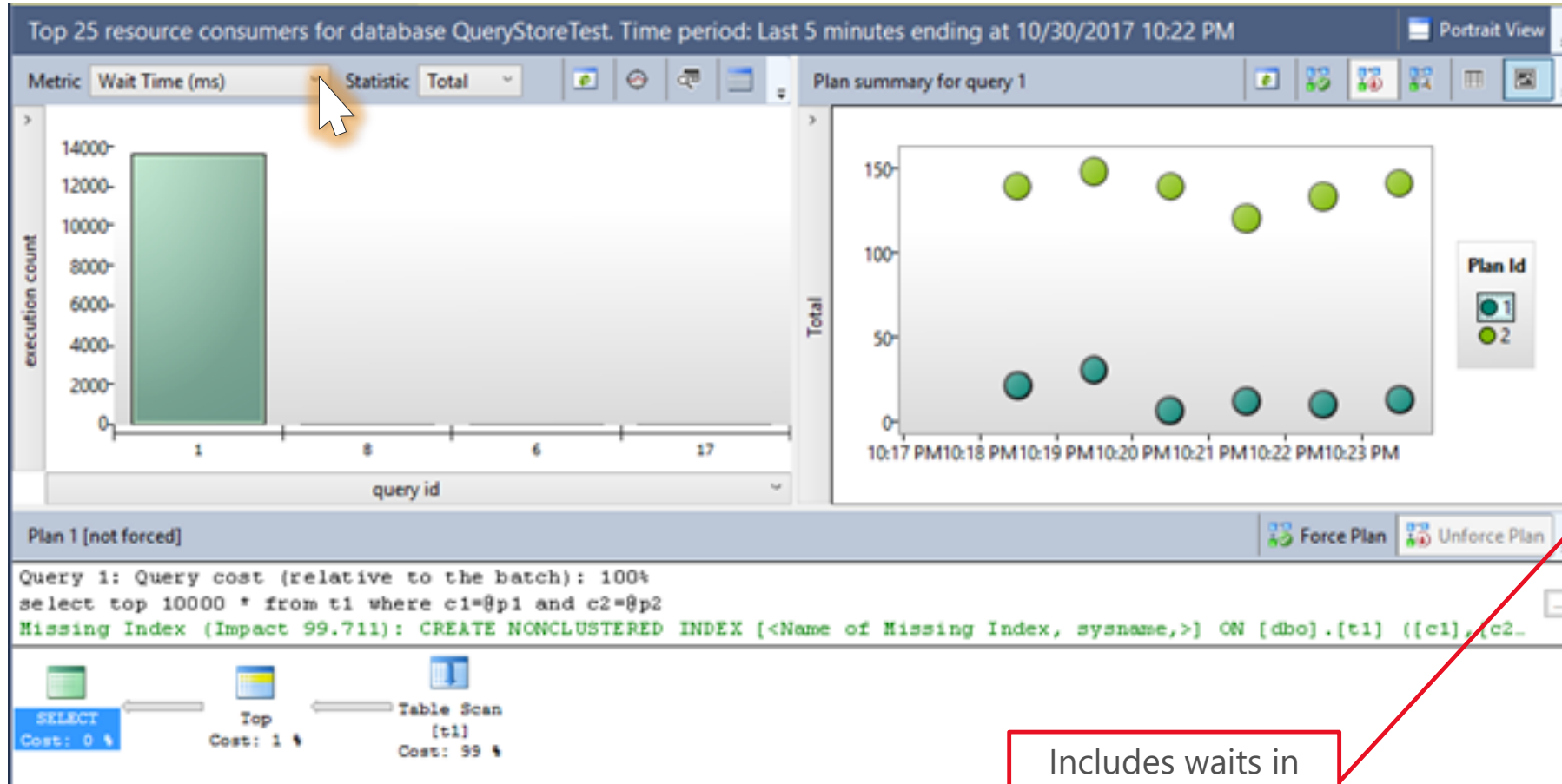
Query Store



Comprehensive query-performance information when you need it most!



Query Store



Includes waits in
SQL Server 2017

Configure Top Resource Consumption

Resource Consumption Criteria

Check for top consumers of:

- ☐ Execution Count
- ☒ Duration (ms)
- ☐ CPU Time (ms)
- ☐ Logical Reads (KB)
- ☐ Logical Writes (KB)
- ☐ Physical Reads (KB)
- ☐ CLR Time (ms)
- ☐ DOP
- ☐ Memory Consumption (KB)
- ☐ Row Count
- ☐ Log Memory Used (KB)
- ☐ Temp DB Memory Used (KB)
- ☐ Wait Time (ms)

Based on:

- ☐ Avg
- ☐ Max
- ☐ Min
- ☐ Std Dev
- ☒ Total

Time Interval

Last 5 minutes From To

Time Format: ☒ Local ☐ UTC

Return

- ☐ All
- ☒ Top 25

Filters

Minimum number of query plans: 1

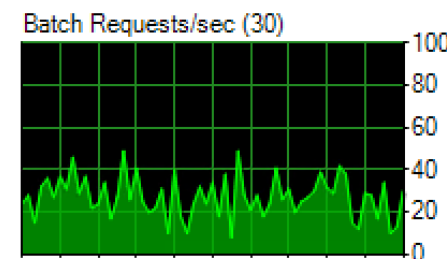
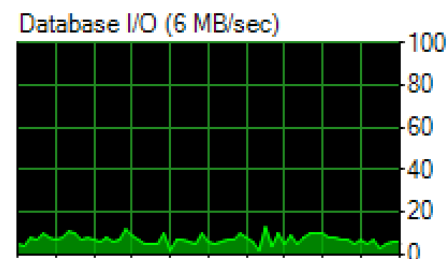
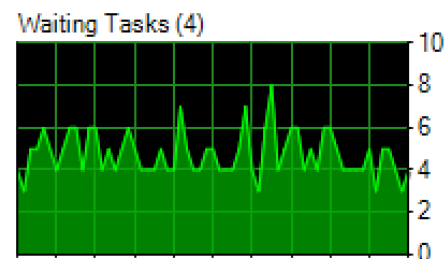
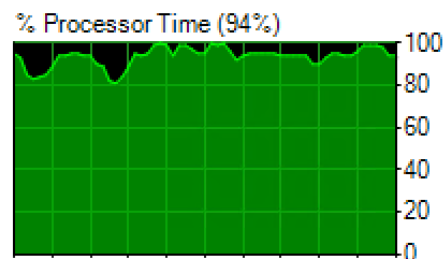
OK Cancel Apply

And yes. Activity Monitor



SQL Server Tiger Team

Overview



Processes

Resource Waits

Data File I/O

Recent Expensive Queries

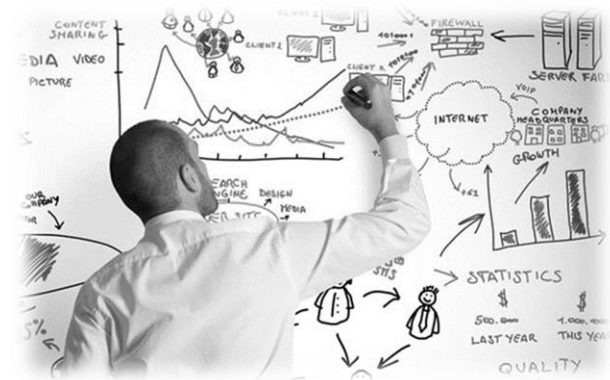
Active Expensive Queries

Query	Ses...	CPU (ms...	Database	Elapsed ...	Physical ...	Writes	Logical ...	Row Co...	Allocate...
SELECT e.[BusinessEntityID], p.[Title], ...	53	67440	AdventureW...	139490	0	250	3540276	0	115520
SELECT e.[BusinessEntityID], p.[Title], ...	54	19952	AdventureW...	21127	0	245	1160914	0	115520
SELECT e.[BusinessEntityID], p.[Title], ...	55	59409	AdventureW...	173721	0	250	3128131	0	115520
SELECT e.[BusinessEntityID], p.[Title], ...	56	43750	AdventureW...	133006	0	246	2348542	0	115520
SELECT ProductID, Total = SUM(LineTotal)FR...	57	3	AdventureW...	3	0	0	75	0	1072
SELECT [SalesOrderDetailID], [OrderQty] ...	58	1104	AdventureW...	11836	0	0	10	0	264
SELECT e.[BusinessEntityID], p.[Title], ...	59	87290	AdventureW...	173110	0	251	4556573	0	115520
SELECT e.[BusinessEntityID], p.[Title], ...	63	54900	AdventureW...	59061	0	248	3066894	0	115520
SELECT [SalesOrderDetailID], [OrderQty] ...	64	992	AdventureW...	11615	0	0	10	0	264
SELECT e.[BusinessEntityID], p.[Title], ...	65	55181	AdventureW...	169077	0	249	2913449	0	115520



SQL Server Tiger Team

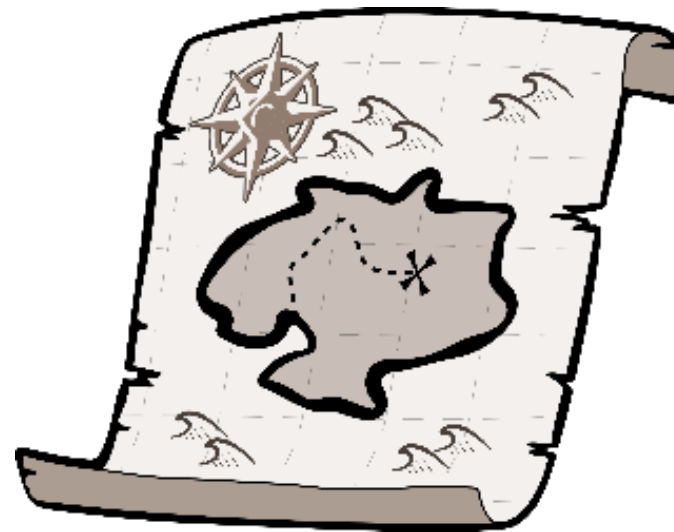
Once I find the slow-running query, how do I analyze it?



Query plans: a map to the execution context



- 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



Getting all context info in Showplan: Trace Flags



Shows list of active trace flags:

- Query
- Session
- Global

Useful to understand if active Trace 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: Param 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'

Look for
conversion
warnings

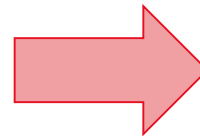


Getting all context info in Showplan: Statistics

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]

Getting all context info in Showplan: Times



Persisting information on elapsed and CPU times

[-] QueryTimeStats	
CpuTime	89
ElapsedTime	274

[-] QueryTimeStats	
CpuTime	91903
ElapsedTime	92330

Is all the elapsed
time spent on CPU?
Look for waits

And Scalar UDF elapsed and CPU times

[-] QueryTimeStats	
CpuTime	628
ElapsedTime	1174
UdfCpuTime	443
UdfElapsedTime	445

How much elapsed
time is spent on
UDF?

Getting all context info in Showplan: Waits



Shows top 10 waits from
sys.dm_exec_session_wait_stats

Correlate
waits with
overall query
times

QueryTimeStats	
CpuTime	1045
ElapsedTime	3010

Misc	
Cached plan size	160 KB
CardinalityEstimationModelV	130
CompileCPU	11
CompileMemory	728
CompileTime	136
DatabaseContextSettingsId	3
Degree of Parallelism	12
Estimated Number of Rows	121308
Estimated Operator Cost	0 (0%)
Estimated Subtree Cost	4.48002
Memory Grant	80448
MemoryGrantInfo	
Optimization Level	FULL

WaitStats	
[1]	
WaitCount	98
WaitTimeMs	3
WaitType	LATCH_SH
[2]	
WaitCount	50
WaitTimeMs	761
WaitType	PAGEIOLATCH_SH
[3]	
WaitCount	67
WaitTimeMs	1942
WaitType	LATCH_EX
[4]	
WaitCount	129
WaitTimeMs	2509
WaitType	ASYNC_NETWORK_IO
[5]	
WaitCount	2220
WaitTimeMs	30622
WaitType	CXPACKET



Getting all context info in Showplan: memory

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

Is the used memory close to granted?
Is the memory above granted?
Look for warnings

Also 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



Getting all context info in Showplan: RG info

List attributes of Resource Governor Settings

- MaxCompileMemory for maximum query optimizer memory in KB during compile under RG.
- MaxQueryMemory for maximum query memory grant under RG
MAX_MEMORY_PERCENT hint.

MemoryGrantInfo	
DesiredMemory	63136
GrantedMemory	63136
GrantWaitTime	0
MaxQueryMemory	1492408
MaxUsedMemory	56024
RequestedMemory	63136
RequiredMemory	7104
SerialDesiredMemory	57544
SerialRequiredMemory	1536
Optimization Level	FULL
OptimizerHardwareDependent	
EstimatedAvailableDegreeOfParallelism	2
EstimatedAvailableMemory	417483
EstimatedPagesCached	104370
MaxCompileMemory	653072

Is this running with memory limitations?



SQL Server Tiger Team

Analyzing query plan properties

Notice the warnings



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.

Been there since SQL Server 2012

SQL Server Tiger Team

Warning: Memory Grant

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.	

Warning: Memory Grant



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
Cost: 13 %

Sort
Hash Match (Inner Join)

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.

Warning: Spills



SQL Server Tiger Team

Sort	Cost: 32 %
Hash Match (Inner Join)	Cost: 6 %
Parall (Repartitic)	Cost: %
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	

Is this spill relevant to go after? Does it consume too many resources?

And this one? What if this executes dozens of times per minute?

Hash Match (Inner Join)	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	
[AdventureWorks2014].[Sales].[Customer].CustomerID	



SQL Server Tiger Team

Analyzing query plan properties

Looking at per-operator runtime stats

Insights into every query plan node



Properties	
Clustered Index Scan (Clustered)	
<div><div></div><div>A Z</div><div></div></div>	
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

SET STATISTICS IO not needed

SET STATISTICS TIME not needed



Predicate Pushdown

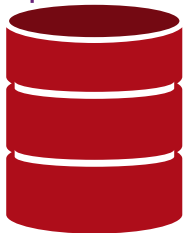
- Query specifies a predicate that can be used to filter rows.
- That filter can be evaluated on top of a table or **index scan/range** operation – predicate is pushed down to Storage Engine.
 - Almost always SQL Server will do this, even if your predicate doesn't filter all rows.
- Any remaining parts of the predicate are known as "residual" and must be evaluated for each row output by the scan or range operation – usually using a **Filter** operator.

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	StoreID	OrderQty	SalesAmount
2010-12-31	106	01	1	30
2011-01-07	103	04	1	17
2011-01-07	109	04	7	25
2011-02-12	103	03	5	16
2011-03-08	106	05	7	40
2011-04-16	106	02	10	50
2011-07-20	102	02	12	55
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

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

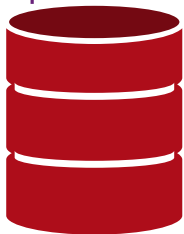
Range Scan

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			1	17
2011-01-07			7	20
2011-02-12	106	03	5	40
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

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

Predicate Pushdown as seen in Showplan



SQL Server Tiger Team

```
SELECT * FROM [Production].[TransactionHistory]
WHERE [ProductID] = 870 AND [Quantity] > 10
```

Key Stats

- Actual Rows in result set = 39
- Actual Rows Read = 113443

Clustered Index Scan (Clustered)	
Scanning a clustered index, entirely or only a range.	
Physical Operation	Clustered Index Scan
Logical Operation	Clustered Index Scan
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Actual Number of Rows	39
Actual Number of Batches	0
Estimated Operator Cost	0.651523 (88%)
Estimated I/O Cost	0.589051
Estimated CPU Cost	0.0624722
Estimated Subtree Cost	0.651523
Number of Executions	4
Estimated Number of Executions	1
Estimated Number of Rows	1500.73
Estimated Row Size	54 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	False
Node ID	1

SQL 2014

Clustered Index Scan (Clustered)	
Scanning a clustered index, entirely or only a range.	
Physical Operation	Clustered Index Scan
Logical Operation	Clustered Index Scan
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Number of Rows Read	113443
Actual Number of Rows	39
Actual Number of Batches	0
Estimated I/O Cost	0.589051
Estimated Operator Cost	0.620287 (92%)
Estimated Subtree Cost	0.620287
Estimated CPU Cost	0.0312361
Number of Executions	8
Estimated Number of Executions	1
Estimated Number of Rows	1500.73
Estimated Row Size	54 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	False
Node ID	1

SQL 2016

Clustered Index Scan (Clustered)	
Scanning a clustered index, entirely or only a range.	
Physical Operation	Clustered Index Scan
Logical Operation	Clustered Index Scan
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Number of Rows Read	113443
Actual Number of Rows	39
Actual Number of Batches	0
Estimated I/O Cost	0.589792
Estimated Operator Cost	0.621028 (92%)
Estimated CPU Cost	0.0312361
Estimated Subtree Cost	0.621028
Number of Executions	8
Estimated Number of Executions	1
Estimated Number of Rows	1500.73
Estimated Number of Rows to be Read	113443
Estimated Row Size	54 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	False
Node ID	1

SQL 2016 SP1



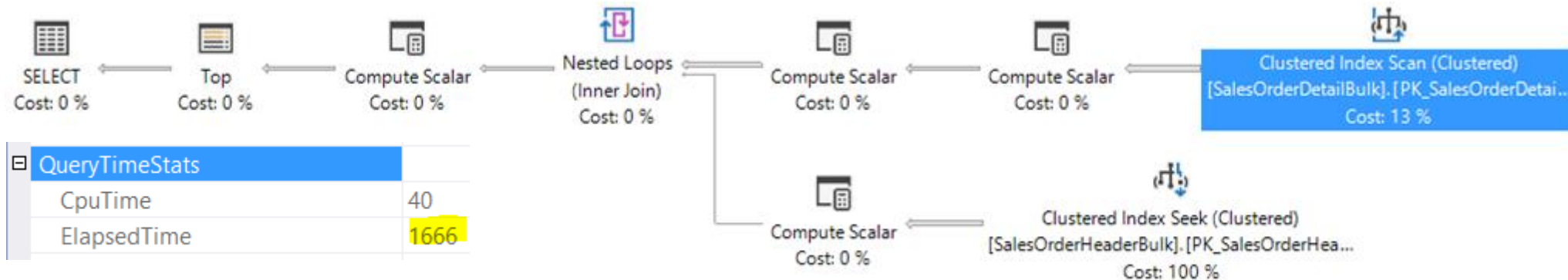
Optimizer Row Goal

- Query specifies a target number of rows (a.k.a. row goal) that may be expected at runtime.
- How? When query uses a **TOP**, **IN** or **EXISTS** clause, the **FAST** query hint, or a **SET ROWCOUNT** statement – that row goal is used as part of the query optimization process.
- If the row goal plan is applied, the *estimated number of rows* is reduced.
 - Optimizer assumes that a smaller number of rows will have to be processed, in order to reach the row goal.

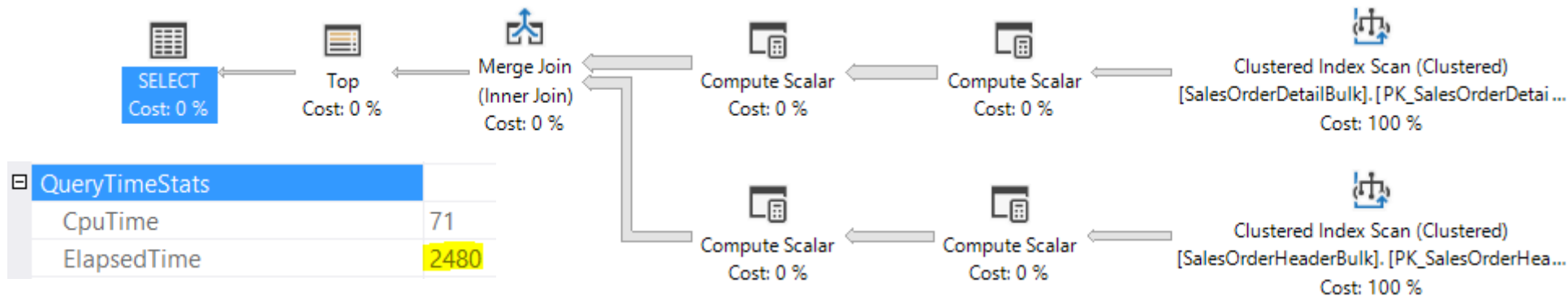
Optimizer row goal – the good case



```
SELECT TOP (100) * FROM Sales.SalesOrderHeaderBulk AS s
INNER JOIN Sales.SalesOrderDetailBulk AS d ON s.SalesOrderID = d.SalesOrderID
WHERE s.TotalDue > 1000 OPTION (RECOMPILE);
```



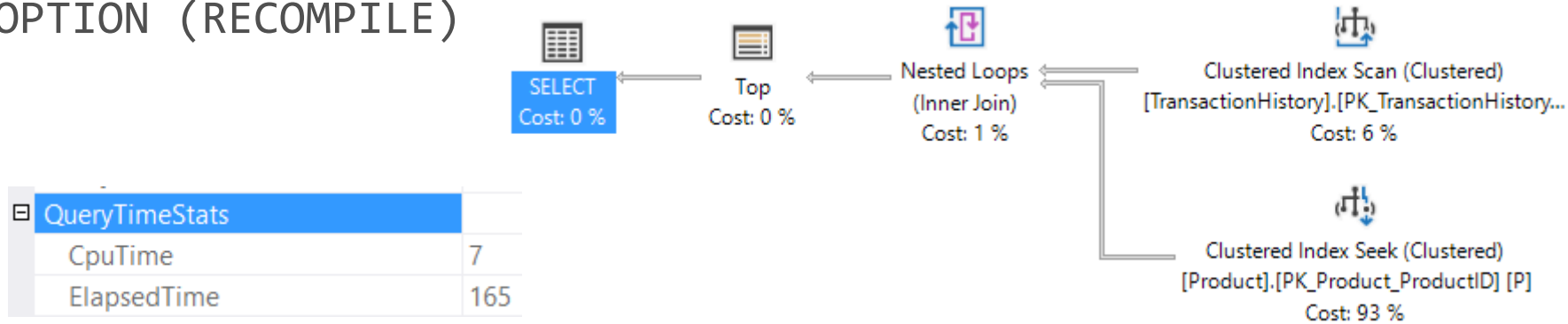
```
SELECT TOP (100) * FROM Sales.SalesOrderHeaderBulk AS s
INNER JOIN Sales.SalesOrderDetailBulk AS d ON s.SalesOrderID = d.SalesOrderID
WHERE s.TotalDue > 1000 OPTION (RECOMPILE, USE HINT('DISABLE_OPTIMIZER_ROWGOAL'));
```



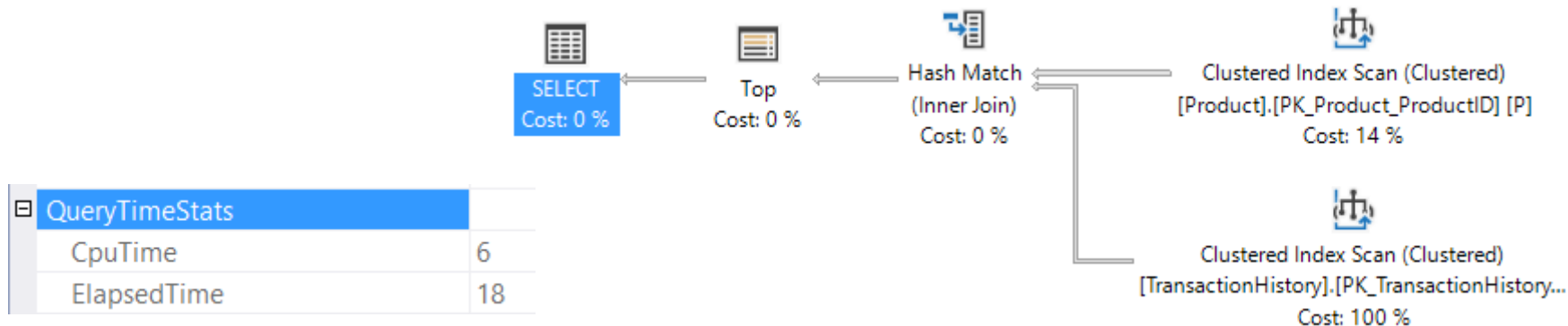


Optimizer row goal – the “bad” case

```
SELECT TOP 250 * FROM Production.TransactionHistory H
INNER JOIN Production.Product P ON H.ProductID = P.ProductID
OPTION (RECOMPILE)
```



```
SELECT TOP 250 * FROM Production.TransactionHistory H
INNER JOIN Production.Product P ON H.ProductID = P.ProductID
OPTION (RECOMPILE, USE HINT('DISABLE_OPTIMIZER_ROWGOAL'));
```



Discoverability of Optimizer row goal



SQL Server Tiger Team

Clustered Index Scan (Clustered)

Actual Execution Mode	Row
Misc	
Actual I/O Statistics	
Actual Number of Batches	0
Actual Number of Rows	250
Actual Rebinds	0
Actual Rewinds	0
Actual Time Statistics	
Defined Values	[AdventureWorks2016
Description	Scanning a clustered i
Estimated CPU Cost	0.124944
Estimated Execution Mode	Row
Estimated I/O Cost	0.589792
Estimated Number of Executions	1
Estimated Number of Rows	250
Estimated Number of Rows to be R	113443
Estimated Operator Cost	0.0048447 (6%)
Estimated Rebinds	0
Estimated Rewinds	0
Estimated Row Size	54 B
Estimated Subtree Cost	0.0048447
EstimateRowsWithoutRowGoal	113443
Forced Index	False
ForceScan	False
Logical Operation	Clustered Index Scan
Node ID	2
NoExpandHint	False
Number of Executions	1
Number of Rows Read	250

Diff *Estimated rows* vs
Estimated rows
without Row Goal?

Diff *Estimated*
number of rows to be
Read and Number of
rows read?

Clustered Index Seek (Clustered)

Actual Execution Mode	Row
Misc	
Actual I/O Statistics	
Actual Number of Batches	0
Actual Number of Rows	250
Actual Rebinds	0
Actual Rewinds	0
Actual Time Statistics	
Defined Values	[AdventureWo
Description	Scanning a pa
Estimated CPU Cost	0.0001581
Estimated Execution Mode	Row
Estimated I/O Cost	0.003125
Estimated Number of Executions	251.00025
Estimated Number of Rows	1
Estimated Number of Rows to be R	1
Estimated Operator Cost	0.0803081 (93'
Estimated Rebinds	247.856
Estimated Rewinds	2.14425
Estimated Row Size	229 B
Estimated Subtree Cost	0.0803081
Forced Index	False
ForceScan	False
ForceSeek	False
Logical Operation	Clustered Inde
Node ID	3
NoExpandHint	False
Number of Executions	250
Number of Rows Read	250

See now, and no
attribute *Estimated*
rows without Row
Goal



SQL Server Tiger Team

Faster insights

The middle-of-the-night call

You're on call for supporting the data tier of a mission-critical SQL Server instance

Key business processes are being delayed when ETL is running.

You get a call asking to **mitigate** the issue and then determine the **root cause**.





Defining the problem

Reasonable hypothesis: a long running query.

Query completion is a prerequisite for the availability of an actual query plan.

Actual query plans unsuitable for troubleshooting complex performance issues:

- Long running queries
- Queries that run indefinitely and never finish execution.

What if I could do live query troubleshooting?



- What we need is live execution plan!
- Now as default behavior:
- To have in-flight query execution statistics, the query execution statistics profile infrastructure must be enabled **on demand**.
- But cost overhead goes up to 75% with TPC-C like workload.
- It makes bad things worse if running all the time.

Processes									
S...	U...	Login	Dat...	Tas...	Com...	Appl...	Wait Tim...	Wait...	
51	1		master			Microsoft...	0		
53	1		tempdb	RUNNING	SELECT	Microsoft...	0		
54	1		Adventur...	SUSPEN...	SELECT	SQLCMD	25	CXPACK...	
54	1		Adventur...	SUSPEN...	SELECT	SQLCMD	25	CXPACK...	
54	1		Adventur...	SUSPEN...	SELECT	SQLCMD	25	CXPACK...	
54	1		Adventur...	SUSPEN...	SELECT	SQLCMD	25	CXPACK...	
54	1		Adventur...	SUSPEN...	SELECT	SQLCMD	25	CXPACK...	
54	1		Adventur...	SUSPEN...	SELECT	SQLCMD	25	CXPACK...	
54	1		Adventur...	SUSPEN...	SELECT	SQLCMD	25	CXPACK...	
54	1		Adventur...	SUSPEN...	SELECT	SQLCMD	25	CXPACK...	
54	1		Adventur...	RUNNING	SELECT	SQLCMD	0		

Details

Show Live Execution Plan

Kill Process

Trace Process in SQL Server Profiler

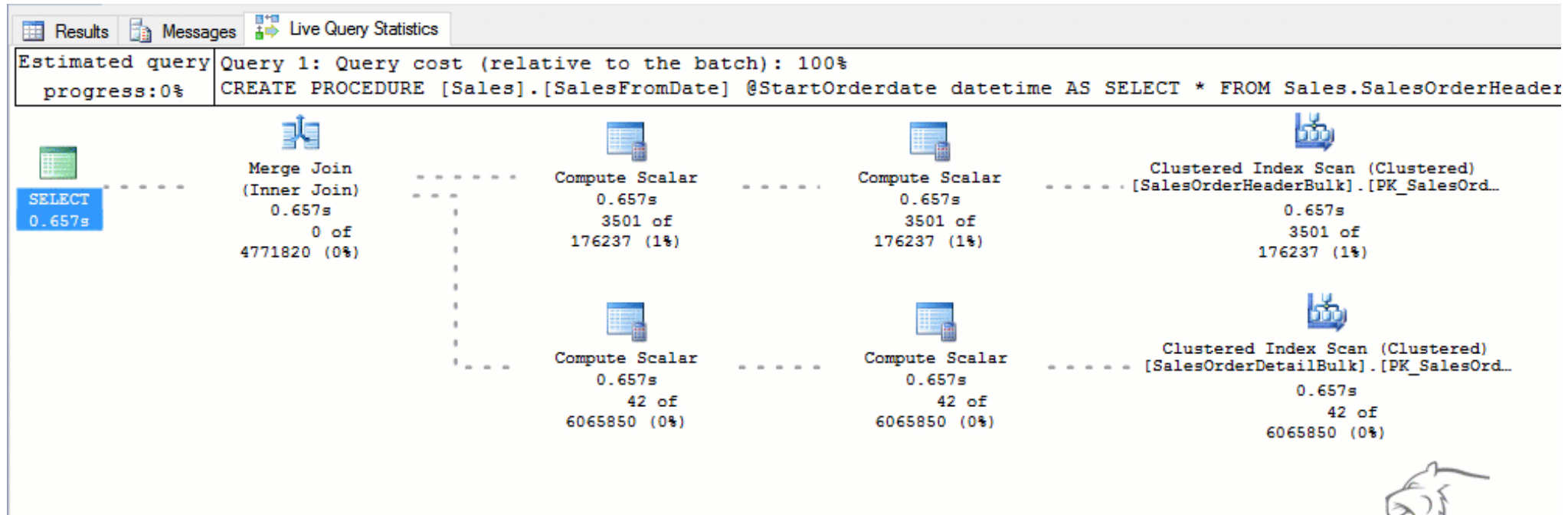
Resource Waits									
Data File I/O									
Recent Expensive Queries									
Active Expensive Queries									

What if I could do live query troubleshooting?



Can be enabled for a target session:

- Specifying *Include Live Query Statistics* in SSMS.
- SET STATISTICS XML ON
- SET STATISTICS PROFILE ON



What if I could do live query troubleshooting?



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.

Processes									
S...	U...	Login	Dat...	Tas...	Com...	Appl...	Wait Tim...	Wait...	
57	1		Adventur...	RUNNA...	SELECT	SQLCMD	0		
57	1		Adventur...	RUNNA...	SELECT	SQLCMD	0		
57	1		Adventur...	RUNNA...	SELECT	SQLCMD	0		
57	1		Adventur...	RUNNA...	SELECT	SQLCMD	0		
57	1		Adventur...	RUNNA...	SELECT	SQLCMD	328	CXPACK...	
57	1		Adventur...	RUNNA...	SELECT	SQLCMD	328	CXPACK...	
57	1		Adventur...	RUNNA...	SELECT	SQLCMD	328	CXPACK...	
57	1		Adventur...	RUNNA...	SELECT	SQLCMD	328	CXPACK...	
57	1		Adventur...	RUNNA...	SELECT	SQLCMD	328	CXPACK...	
57	1		Adventur...	RUNNA...	SELECT	SQLCMD	0		

Details

Show Live Execution Plan

Kill Process

Trace Process in SQL Server Profiler

Resource Waits									
Data File I/O									
Recent Expensive Queries									
Active Expensive Queries									

Query progress – anytime, anywhere



Starting with SQL Server 2016 SP1* we have enabled *lightweight query execution statistics profile infrastructure* to continuously collecting per-operator query execution statistics.

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.

What is the impact of live query troubleshooting?



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

Infra Type	Overhead percent (up to)	
	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 and above	2	14.3



SQL Server Tiger Team

Demo

Bringing it all together with live troubleshooting

-



SQL Server Tiger Team

Project TunA

Bob did mention I'd be talking about fish 😊

Query Tuning Assistant



SQL Server Tiger Team

QueryTuningAssistant

File Edit View Window Help

Discovery Setup Collection Analysis Findings Validations

Discovery

Please login to the database you want to tune with:

Server name

Authentication type

Database name

User name

Password

Session Management Finish Session Next Close

Query Tuning Assistant



SQL Server Tiger Team

QueryTuningAssistant

File Edit View Window Help

✓

Discovery

Setup

Collection

Analysis

Findings

Validations

Setup

Please set estimated number of days to execute workload to tune:

Please choose target compatibility level:

Your Query Store settings:

Operation Mode

Data Flush Interval (Secs)

Statistics Collection Interval (Mins)

Max Size (MB)

Query Store Capture Mode

Size Based Cleanup Mode

Stale Query Threshold (Days)

Recommended Query Store settings:

Note: Stale Query Threshold should be at least twice the estimated number of days to executed workload to tune.

Operation Mode

Data Flush Interval (Secs)

Statistics Collection Interval (Mins)

Max Size (MB)

Query Store Capture Mode

Size Based Cleanup Mode

Stale Query Threshold (Days)

Use Recommended QDS Setting

Finish Session

Back

Next

Close

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>

BP 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/sqlifecycle>

SQL Server Updates <http://aka.ms/sqlupdates>

SQL Server Guides <http://aka.ms/sqlserverguides>

Twitter [@mssqltiger](https://twitter.com/mssqltiger)

