Query Performance Insights: What’s new?

Slides in http://aka.ms/tigertoolbox

Under “Sessions” folder

sqlbits

Microsoft

PEDRO LOPES

Sr. Program Manager, SQL Server Engineering

in /pedroazevedolopes

@SQLPedro

Focus Areas

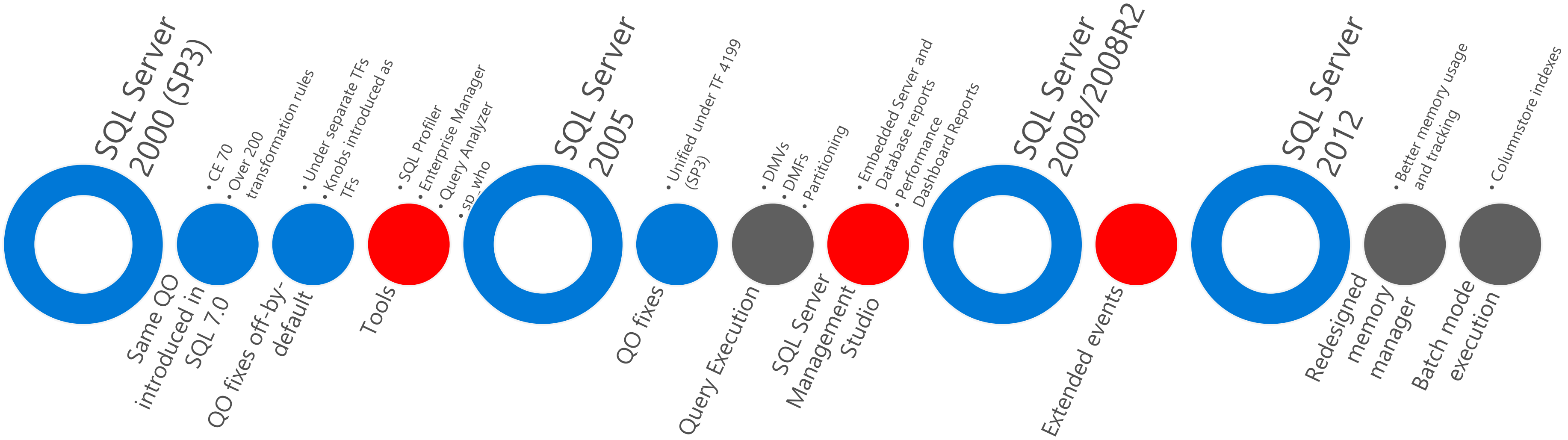
Relational Engine: Query Processor, Programmability, Performance

Agenda

- A brief history of SQL Server query performance
- Diagnostics improvements
- Adaptive Query Processing in SQL Server 2017
- Intelligent Query Processing in SQL Server 2019

A brief history of SQL Server Query Performance

Query Performance Journey



SQL Server 2000 (SP3)

Same QO introduced in SQL 7.0
- CE 70
- Over 200 transformation rules

QO fixes off-by-default
- Under separate TFs
- Knobs introduced as TFs

Tools
- SQL Profiler
- Enterprise Manager
- Query Analyzer
- sp_who

SQL Server 2005

QO fixes
- Unified under TF 4199 (SP3)

Query Execution
- DMVs
- DMFs
- Partitioning

SQL Server Management Studio
- Embedded Server and Database reports
- Performance Dashboard Reports

SQL Server 2008/2008R2

Extended events

SQL Server 2012

Redesigned memory manager
- Better memory usage and tracking

Batch mode execution
- Columnstore indexes

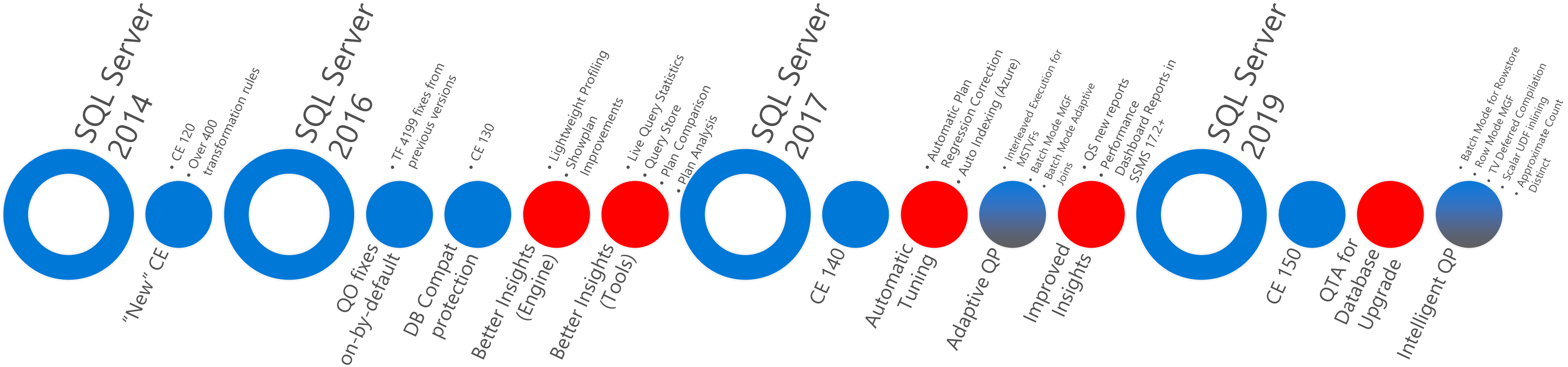
🟦 Query Optimizer

🟥 Diagnostics and Tools

⬛ Query Execution

Retired SQL 2000 docs available in PDF at https://download.microsoft.com/download/5/4/A/54AFD350-6477-4910-9DF2-4C472C906684/SQL2000_release.pdf

Query Performance Journey



SQL Server 2014

"New" CE
- CE 120
- Over 400 transformation rules

SQL Server 2016

QO fixes on-by-default
- TF 4199 fixes from previous versions

DB Compat protection
- CE 130

Better Insights (Engine)
- Lightweight Profiling
- Showplan Improvements

Better Insights (Tools)
- Live Query Statistics
- Query Store
- Plan Comparison
- Plan Analysis

SQL Server 2017

CE 140

Automatic Tuning
- Automatic Plan Regression Correction
- Auto Indexing (Azure)

Adaptive QP
- Interleaved Execution for MSTVFs
- Batch Mode MGF
- Batch Mode Adaptive Joins

Improved Insights
- QS new reports
- Performance Dashboard Reports in SSMS 17.2+

SQL Server 2019

CE 150

QTA for Database Upgrade

Intelligent QP
- Batch Mode for Rowstore
- Row Mode MGF
- TV Deferred Compilation
- Scalar UDF inlining
- Approximate Count Distinct

🟦 Query Optimizer

🟥 Diagnostics and Tools

⬛ Query Execution

Diagnostics Enhancements

- Server and Database

Performance Dashboard in SSMS

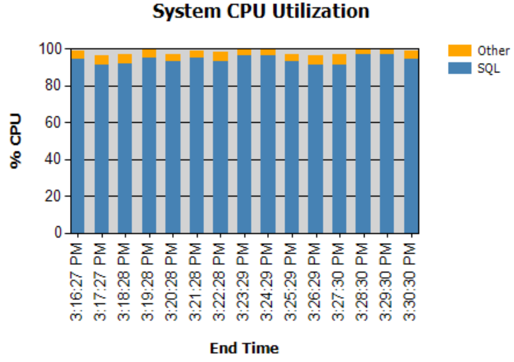
Starting with SSMS v17.2
No extra downloads!
No new schema to deploy!
Long standing request by
CSS and customers

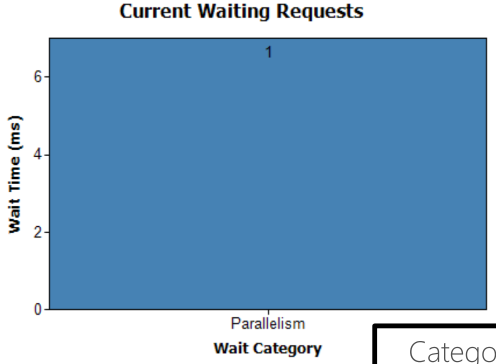
Microsoft SQL Server Performance Dashboard

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

[redacted](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.





Categorized Wait stats page

New categorized Latches page

Scoring added to Missing Index Report

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%

Historical Information

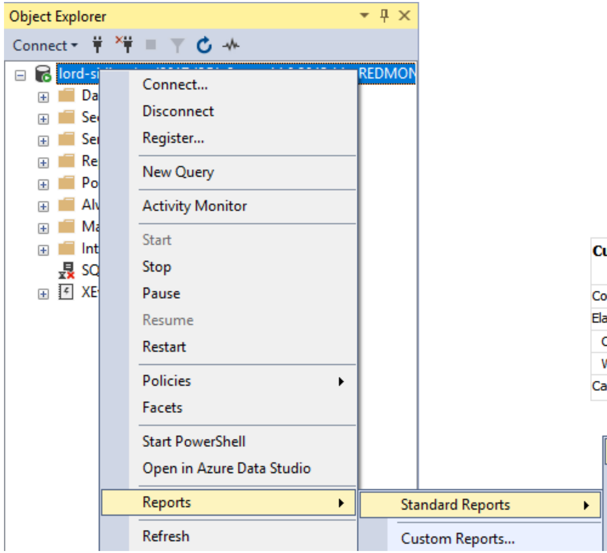
Waits
IO Statistics
Latches

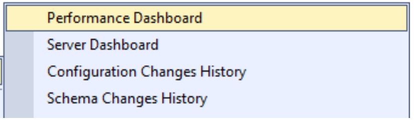
Expensive Queries

By CPU
By Duration
By Logical Reads
By Physical Reads
By Logical Writes
By CLR Time

Miscellaneous Information

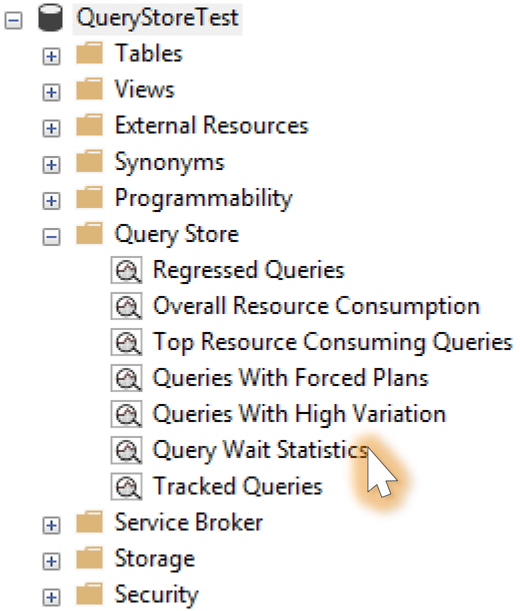
Active Traces	1
Active Xevent Sessions	4
Databases	16
Missing Indexes	11





Query Store – new reports

Comprehensive query-performance information when you need it most!



- QueryStoreTest
  - Tables
  - Views
  - External Resources
  - Synonyms
  - Programmability
  - Query Store
    - Regressed Queries
    - Overall Resource Consumption
    - Top Resource Consuming Queries
    - Queries With Forced Plans
    - Queries With High Variation
    - Query Wait Statistics
    - Tracked Queries
  - Service Broker
  - Storage
  - Security

Diagnostics Enhancements

- Query Analysis

Query plans: fundamental query perf diagnostics

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

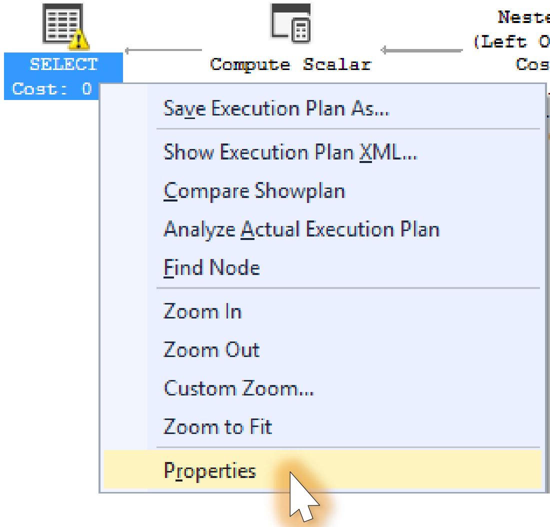
How data is aggregated

Use of parallelism

Query execution warnings

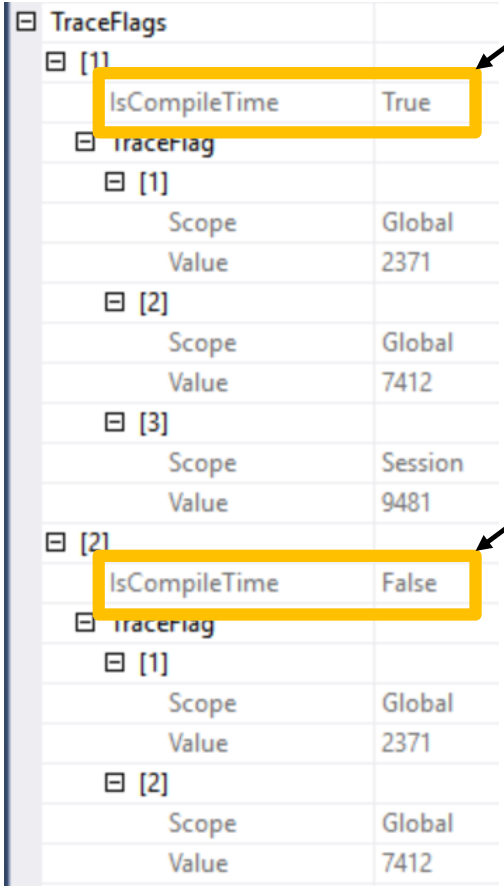
Query execution stats

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 Flags influence execution context



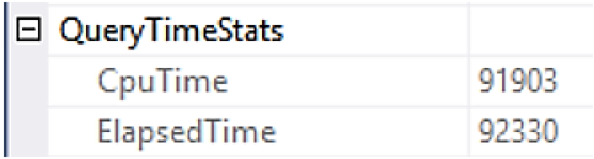
```
⊟ TraceFlags
  ⊟ [1]
      IsCompileTime        True
    ⊟ TraceFlag
      ⊟ [1]
          Scope            Global
          Value            2371
      ⊟ [2]
          Scope            Global
          Value            7412
      ⊟ [3]
          Scope            Session
          Value         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

TFs enabled at query compile time

TFs active at query execution time

Getting all context info in Showplan: Times

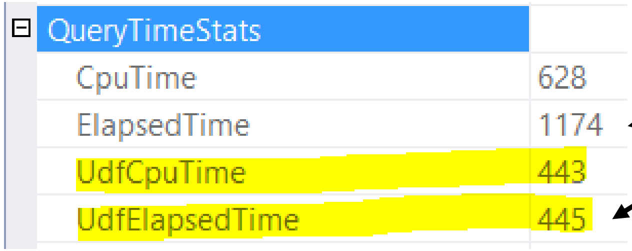
Persisting information on elapsed and CPU times



⊟ 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

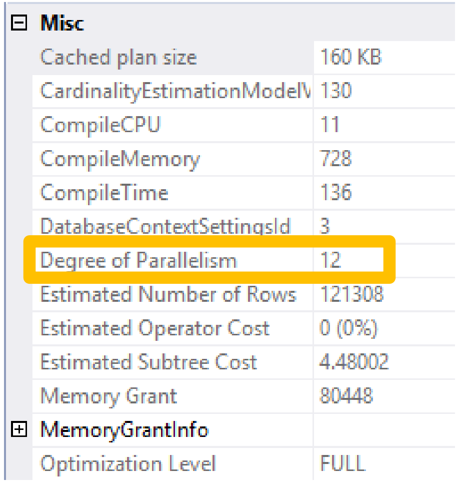
How much elapsed time is spent on UDF?

More on Scalar UDFs later

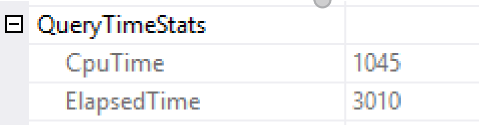
Getting all context info in Showplan: Waits

Shows top 10 waits from sys.dm_exec_session_wait_stats

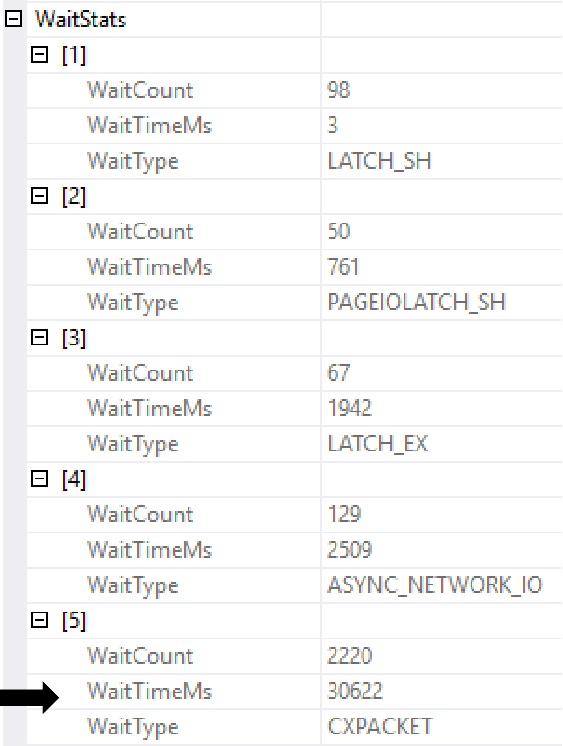
Correlate waits with overall query times



⊟ **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



⊟ QueryTimeStats	
CpuTime	1045
ElapsedTime	3010

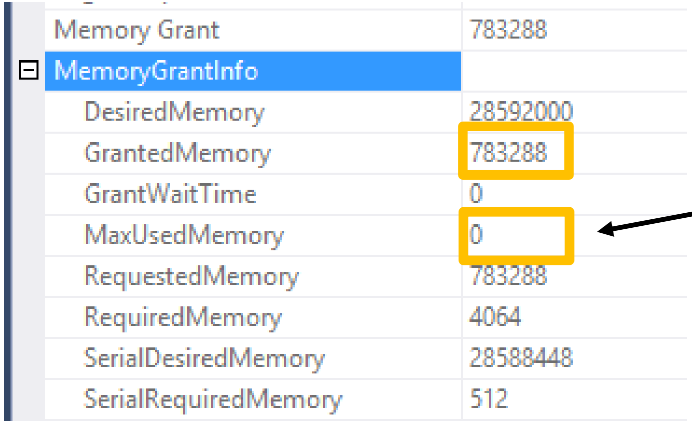


⊟ WaitStats	
⊟ [1]	
WaitCount	98
WaitTimeMs	3
WaitType	LATCH_SH
⊟ [2]	
WaitCount	50
WaitTimeMs	761
WaitType	PAGEIOLATCH_SH
⊟ [3]	
WaitCount	67
WaitTimeMs	1942
WaitType	LATCH_EX
⊟ [4]	
WaitCount	129
WaitTimeMs	2509
WaitType	ASYNC_NETWORK_IO
⊟ [5]	
WaitCount	2220
WaitTimeMs	30622
WaitType	CXPACKET

Note: Parallelism waits available in SQL Server 2016 SP2, 2017 CU3 and 2019

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? Great!

Is the used memory **much below** granted?
Hurting concurrency…

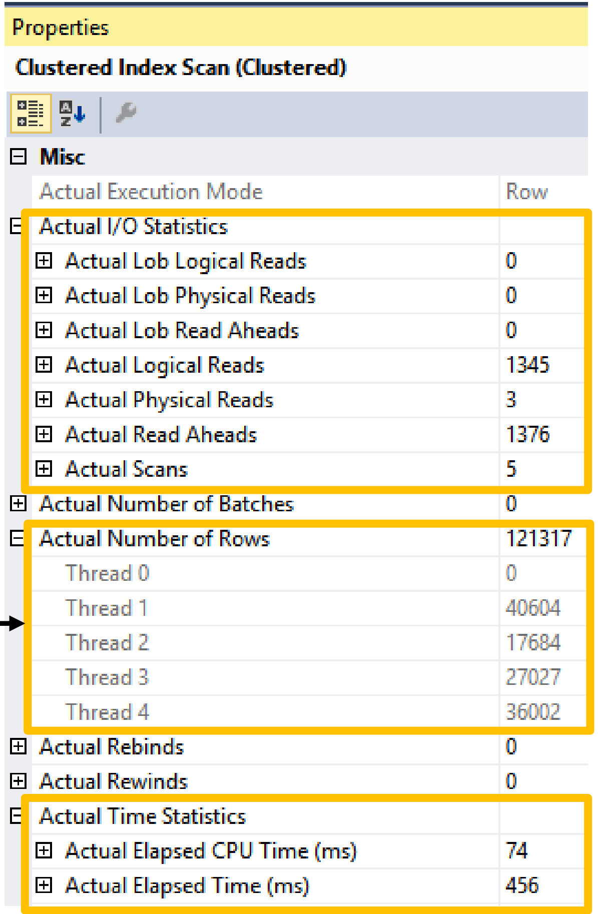
Is the used memory **above** granted?
Possibility of OOM…

Look for **memory warnings**

Also found in *sys.dm_exec_query_stats*

More on
this later

Insights into every query plan node



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

Widely unbalanced parallelism?
Start by looking at statistics

SET STATISTICS IO not needed

SET STATISTICS TIME not needed

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.

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

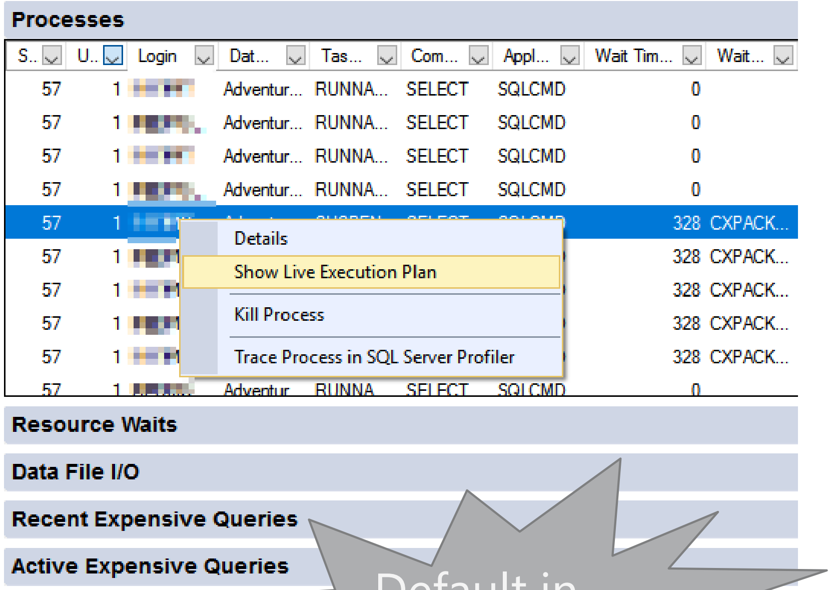
So… actual query plans aren't suitable for troubleshooting complex performance issues:

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

What we need – Query progress: anytime, anywhere

Starting with SQL Server 2016 SP1 and 2017, the new *lightweight query execution statistics profile infrastructure* allows continuous collection of per-operator query execution statistics. How?

- Using global TF 7412
- Enabling *query_thread_profile*, *query_plan_profile**, and *query_post_execution_plan_profile*** extended events
- Using query hint USE HINT('query_plan_profile')***



Processes

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

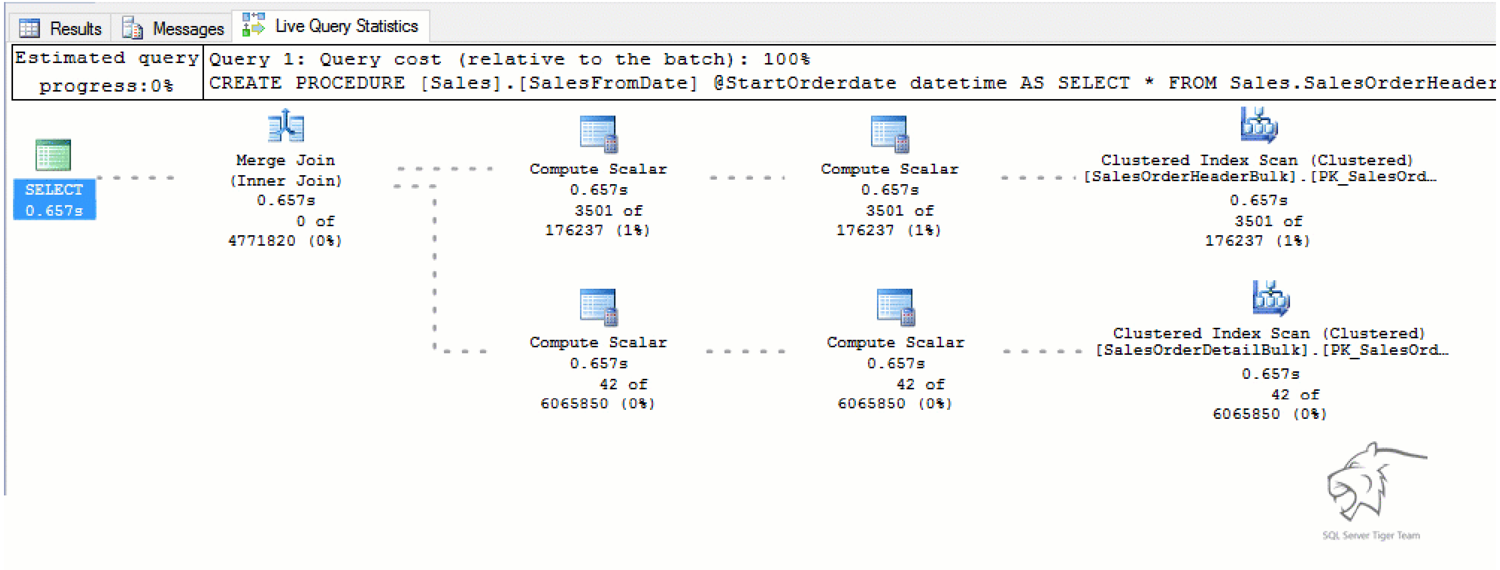
Default in SQL Server 2019

* SQL Server 2016 SP2 CU3, 2017 CU11 and 2019

** SQL Server 2017 CU14 and 2019

*** SQL Server 2017 CU11, 2016 SP2 CU3 and 2019

We get Live Query Statistics for all sessions!



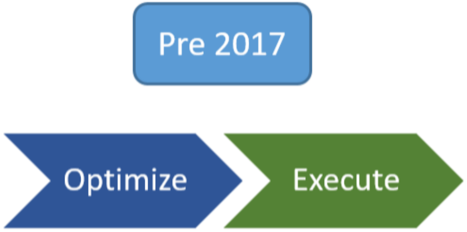
```
Estimated query
 progress:0%
Query 1: Query cost (relative to the batch): 100%
CREATE PROCEDURE [Sales].[SalesFromDate] @StartOrderdate datetime AS SELECT * FROM Sales.SalesOrderHeader
```

Adaptive Query Processing

A brief retrospective

Interleaved Execution for MSTVFs

Problem: Multi-statement table valued functions (MSTVFs) are treated as a black box by QP and we use a fixed optimization guess.



100 rows guessed for MSTVFs

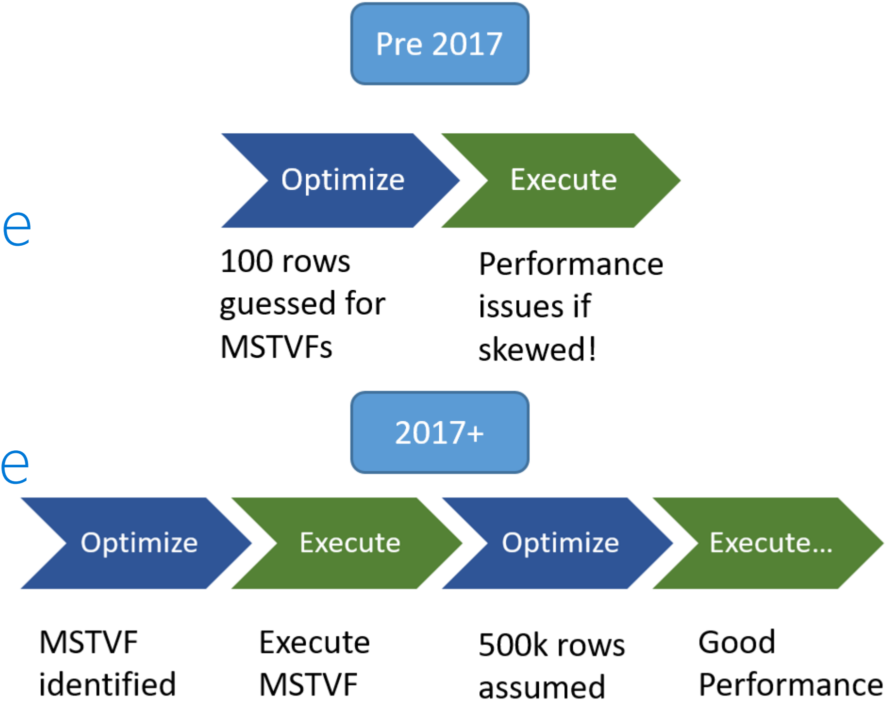
Performance issues if skewed!

Interleaved Execution for MSTVFs

Problem: Multi-statement table valued functions (MSTVFs) are treated as a black box by QP and we use a fixed optimization guess

Interleaved Execution will materialize and use row counts for MSTVFs.

Downstream operations will benefit from the corrected MSTVF cardinality estimate.



Pre 2017

Optimize

Execute

100 rows guessed for MSTVFs

Performance issues if skewed!

2017+

Optimize

Execute

Optimize

Execute…

MSTVF identified

Execute MSTVF

500k rows assumed

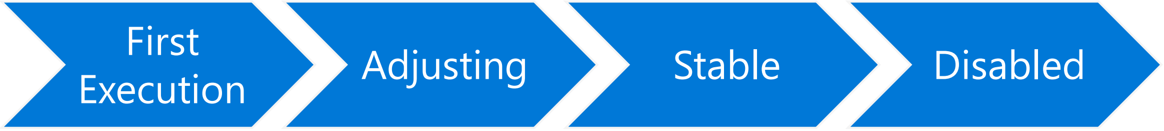
Good Performance!

140 database compatibility level

Batch Mode Memory Grant Feedback (MGF)

Problem: Queries may spill to disk or take too much memory based on poor cardinality estimates. Memory misestimations result in spills, and overestimations hurt concurrency.

MGF will adjust memory grants based on execution feedback.



MGF will remove spills and improve concurrency for repeating queries.

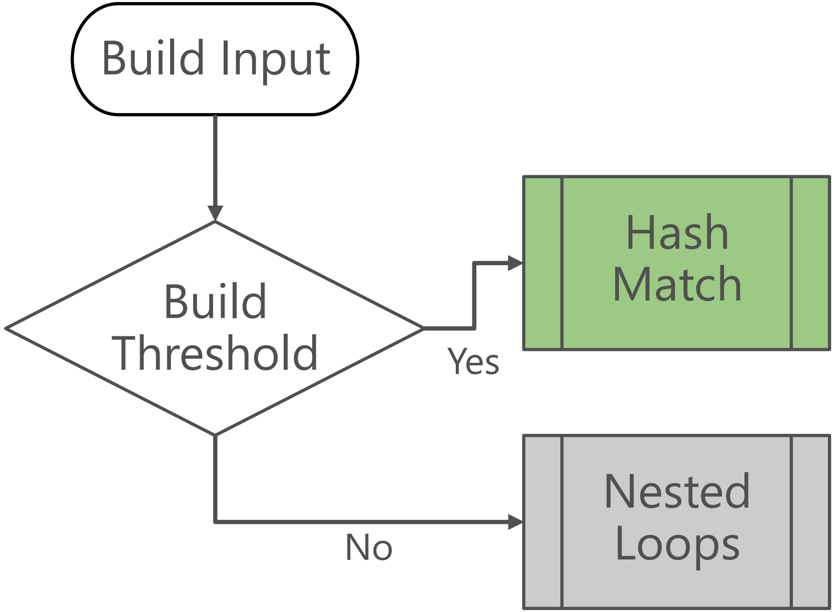
140 database compatibility level

Batch Mode Adaptive Joins (AJ)

Problem: If cardinality estimates are skewed, we may choose an inappropriate join algorithm.

AJ will defer the choice of Hash Match or Nested Loops join until after the first join input has been scanned.

AJ uses Nested Loops for small inputs, Hash Match for large inputs.



140 database compatibility level

Intelligent Query Processing

Broadening the scope…

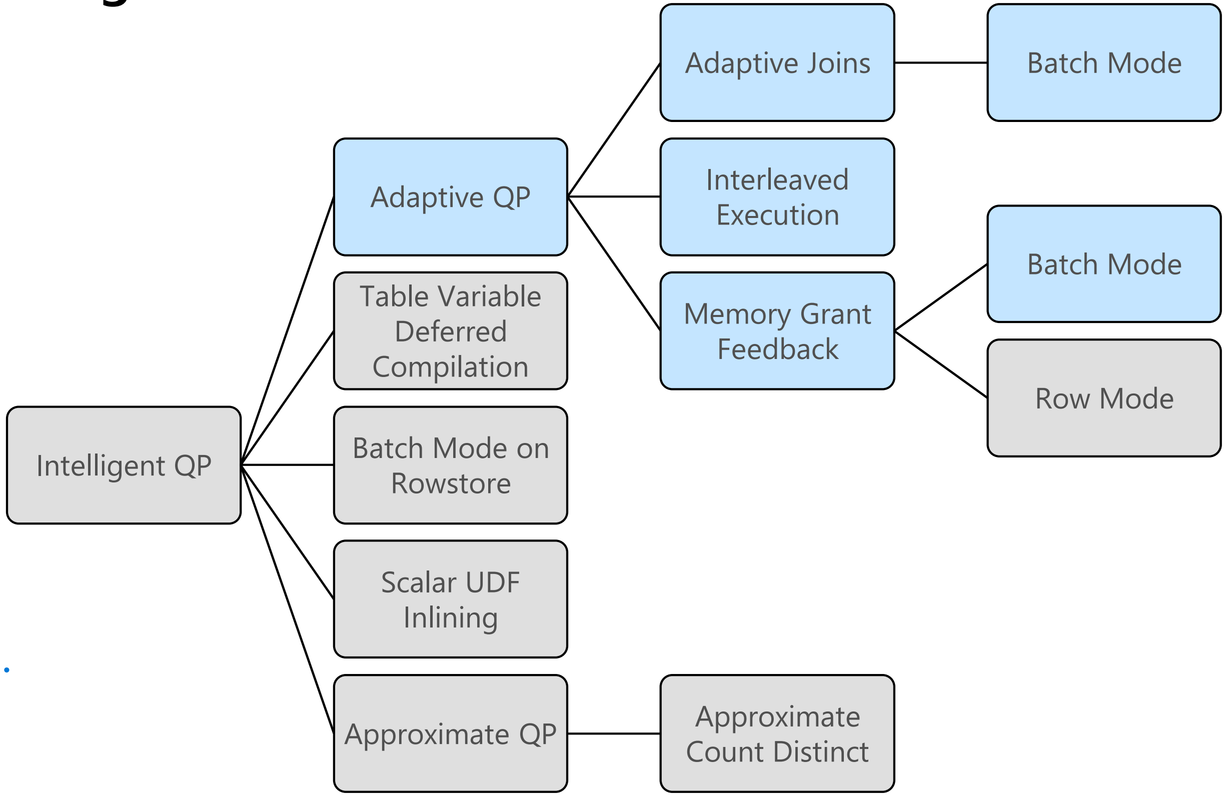
Intelligent Query Processing

The **intelligent query processing** feature family includes features with broad impact that improve the performance of existing workloads with minimal implementation effort.

Azure SQL Database

SQL Server 2017

SQL Server 2019

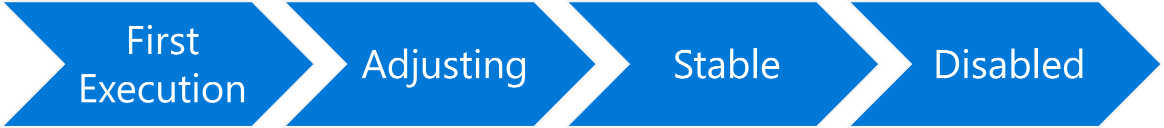


- Intelligent QP
  - Adaptive QP
    - Adaptive Joins
      - Batch Mode
    - Interleaved Execution
    - Memory Grant Feedback
      - Batch Mode
      - Row Mode
  - Table Variable Deferred Compilation
  - Batch Mode on Rowstore
  - Scalar UDF Inlining
  - Approximate QP
    - Approximate Count Distinct

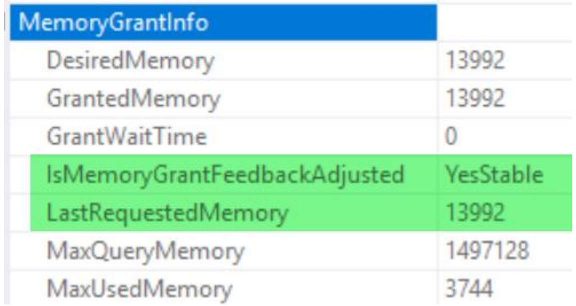
Row Mode Memory Grant Feedback (MGF)

Same as batch-mode MGF, updating the cached plan for:

- Row-mode spills to disk → MGF corrects grant misestimations
- Row-mode excessive memory grant → MGF corrects wasted memory, improves concurrency



New query execution plan attributes to understand the state of memory grant feedback:



MemoryGrantInfo	
DesiredMemory	13992
GrantedMemory	13992
GrantWaitTime	0
IsMemoryGrantFeedbackAdjusted	YesStable
LastRequestedMemory	13992
MaxQueryMemory	1497128
MaxUsedMemory	3744

150 database compatibility level

Table Variable Deferred Compilation

Legacy behavior

Area	Temporary Tables	Table Variables
Manual stats creation and update	Yes	No
Indexes	Yes	Only inline index definitions allowed.
Constraints	Yes	Only PK, uniqueness and check constraints.
Automatic stats creation	Yes	No
Creating and using a temporary object in a single batch	Compilation of a statement that references a temp table that doesn’t exist is deferred until the first execution of the statement	A statement that references a table variable is compiled along with all other statements before any statement that populates the TV is executed, so compilation sees it as “1”.

Table Variable Deferred Compilation

150 database compatibility level

Azure SQL Database and SQL Server 2019 behavior

Area	Temporary Tables	Table Variables
Manual stats creation and update	Yes	No
Indexes	Yes	Only inline index definitions allowed.
Constraints	Yes	Only PK, uniqueness and check constraints.
Automatic stats creation	Yes	No
Creating and using a temporary object in a single batch	Compilation of a statement that references a temp table that doesn't exist is deferred until the first execution of the statement	Compilation of a statement that references a table variable that doesn't exist is deferred until the first execution of the statement

T-SQL Scalar User-Defined Functions (UDFs)

User-Defined Functions that are implemented in Transact-SQL and return a single data value are referred to as **T-SQL Scalar User-Defined Functions**

T-SQL UDFs are an elegant way to achieve code reuse and modularity across SQL queries

Some computations (such as complex business rules) are easier to express in imperative UDF form

UDFs help in building up complex logic without requiring expertise in writing complex SQL queries

T-SQL Scalar UDF performance issues!

Iterative invocation: Invoked once per qualifying row. Repeated context switching – and even worse for UDFs that have T-SQL queries that access data

Lack of costing: Scalar operators are not costed (realistically)

Interpreted execution: Each statement itself is compiled, and the compiled plan is cached. Although this caching strategy saves some time as it avoids recompilations, each statement executes in isolation. No cross-statement optimizations are carried out.

Serial execution: SQL Server does not allow intra-query parallelism in queries that invoke Scalar UDFs. In other words, Scalar UDFs are parallelism inhibitors.

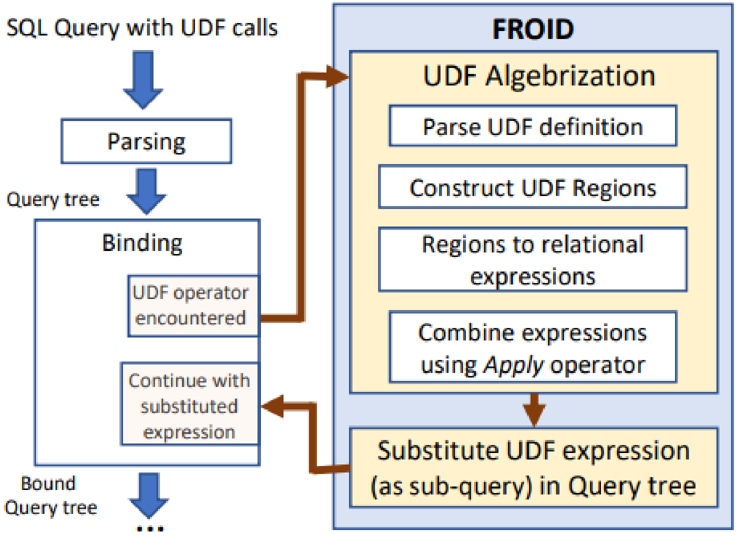
T-SQL Scalar UDF Inlining

Enable the benefits of UDFs without the performance penalty!

- Goal of the *Scalar UDF Inlining* feature is to improve performance for queries that invoke scalar UDFs where UDF execution is the main bottleneck

Before SQL 2019/DB Compat 150:

- Using query rewriting techniques, UDFs are transformed into equivalent relational expressions that are “inlined” into the calling query



SQL Query with UDF calls
Parsing
Query tree
Binding
UDF operator encountered
Continue with substituted expression
Bound Query tree
...
FROID
UDF Algebrization
Parse UDF definition
Construct UDF Regions
Regions to relational expressions
Combine expressions using *Apply* operator
Substitute UDF expression (as sub-query) in Query tree

150 database compatibility level

Source: “Froid: Optimization of Imperative Programs in a Relational Database”

T-SQL Scalar UDF Inlining

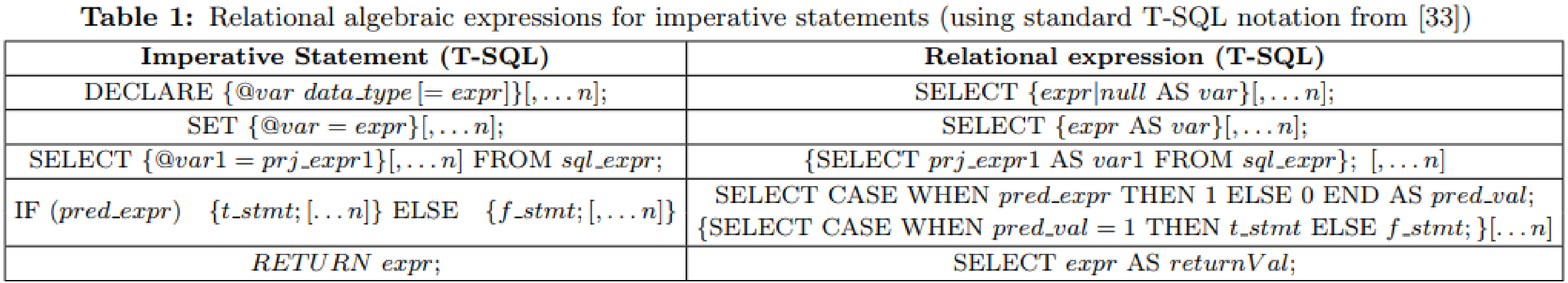


Table 1: Relational algebraic expressions for imperative statements (using standard T-SQL notation from [33])

Imperative Statement (T-SQL)	**Relational expression (T-SQL)**
DECLARE {@var $data\_type$ [= $expr$]}[, ... n];	SELECT {$expr \mid null$ AS var}[, ... n];
SET {@var = $expr$}[, ... n];	SELECT {$expr$ AS var}[, ... n];
SELECT {@$var1$ = prj_expr1}[, ... n] FROM sql_expr;	{SELECT prj_expr1 AS $var1$ FROM sql_expr}; [, ... n]
IF ($pred\_expr$) {t_stmt; [... n]} ELSE {f_stmt; [, ... n]}	SELECT CASE WHEN $pred\_expr$ THEN 1 ELSE 0 END AS $pred\_val$; {SELECT CASE WHEN $pred\_val = 1$ THEN t_stmt ELSE f_stmt; }[... n]
$RETURN$ $expr$;	SELECT $expr$ AS $returnVal$;

Source: “Froid: Optimization of Imperative Programs in a Relational Database”

Scalar UDF Inlining candidates

Existing UDFs will be inlined during compilation with **no need** to make changes.

First version candidates:

- DECLARE, SET: Variable declaration and assignments.
- SELECT: SQL query with multiple variable assignments.
- IF/ELSE: Branching with arbitrary levels of nesting.
- RETURN: Single or multiple return statements.
- UDF: Nested/recursive function calls.
- Others: Relational operations such as EXISTS, ISNULL.

To inline, or not to inline

See *sys.sql_modules* catalog view includes a property called is_inlineable:

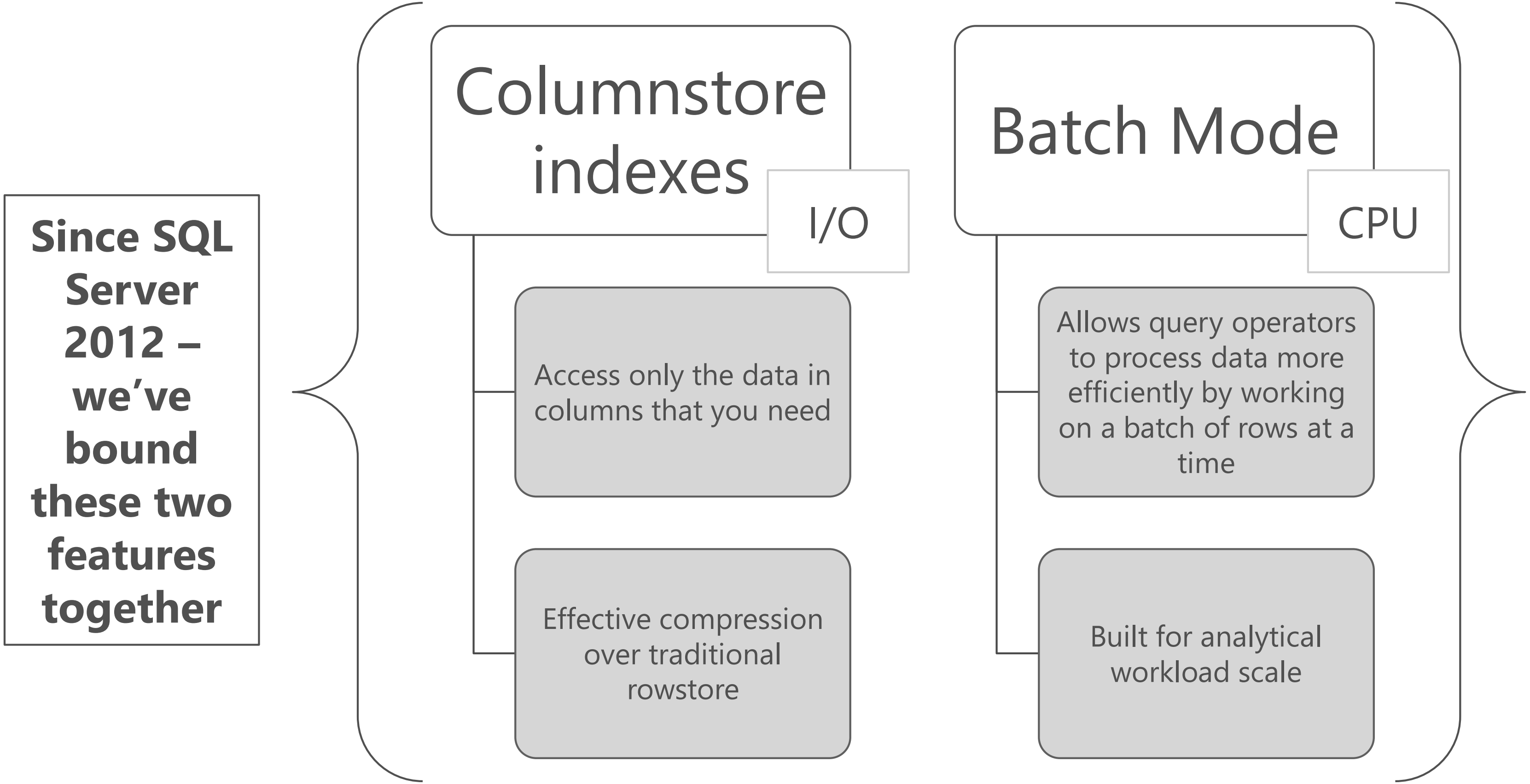
- 1 indicates that it is inlineable, and 0 indicates otherwise
- Value of 1 for all inline TVFs as well

If a scalar UDF is inlineable, it doesn't imply that it will always be inlined. SQL Server will decide (on a per-query, per-UDF basis) whether to inline a UDF or not if:

- UDF definition has thousands of lines of code (itself or by using nesting)
- UDF used in a GROUP BY clause

Decision is made when the query referencing a scalar UDF is compiled.

Batch Mode and Columnstore



Since SQL Server 2012 – we've bound these two features together

Columnstore indexes

I/O

Access only the data in columns that you need

Effective compression over traditional rowstore

Batch Mode

CPU

Allows query operators to process data more efficiently by working on a batch of rows at a time

Built for analytical workload scale

Batch Mode on Rowstore

150 database compatibility level

Sometimes Columnstore isn't an option:

- OLTP-sensitive workloads
- Vendor support
- Columnstore interoperability limitations

Now get analytical processing CPU-benefits **without Columnstore indexes.**

Batch mode on rowstore supports:

- On-disk heaps and B-tree indexes and existing batch-capable operators (**new scan operator** can evaluate batch mode bitmap filters)
- Existing batch mode operators

Batch Mode on Rowstore candidate workloads

A significant part of the workload consists of analytical queries **AND**

The workload is CPU bound **AND**

- Creating a columnstore index adds too much overhead to the transactional part of your workload **OR**
- Creating a columnstore index is not feasible because your application depends on a feature that is not yet supported with columnstore indexes **OR**
- You depend on a feature not supported with columnstore (for example, triggers)

Batch Mode on Rowstore considerations

There is no guarantee that query plans will use batch mode.

No guarantee that if you get a row mode plan, it will be the same as the plan you get in a lower compatibility level.

No guarantee that if you get a batch mode plan, it will be the same as the plan you'd get with a columnstore index.

Plans may also change in subtle ways for queries that mix columnstore and rowstore indexes, because of the new batch mode rowstore scan.

APPROX_COUNT_DISTINCT - When approximate is good enough…

Provides approximate COUNT DISTINCT for big data scenarios with the benefit of high performance and a **(very) low memory** footprint.

Dashboard scenarios and trend analysis against big data sets with many distinct values (for example, distinct orders counts over a time period) – and many concurrent users where exact values are not necessary.

Data science big data set exploration. Need to understand data distributions quickly and exact values are not paramount.

Not banking applications or anywhere an exact value is required!

DEMO

Intelligent Query Processing

Intelligent QP next steps…

- Regressions due to a feature?
- Situations where something didn't kick off and you think it should have?

These features are in public preview – and we want your feedback!

Please email IntelligentQP@Microsoft.com

Bookmarks

SQL Server Tiger Team

SQL Server Team (Tiger) Blog	http://aka.ms/sqlserverteam
Tiger Toolbox GitHub	http://aka.ms/tigertoolbox
SQL Server Release Blog	http://aka.ms/sqlreleases
Best Practices and Perf Checks	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
SQL Server Guides	http://aka.ms/sqlserverguides
SQL Feedback (New “Connect”)	http://aka.ms/sqlfeedback
T-SQL Syntax Conventions	http://aka.ms/sqlconventions
SQL Server Errors	http://aka.ms/sqlerrors
SQL Performance Center	http://aka.ms/sqlperfcenter
Twitter	@mssqltiger

sqlbits

We’d love your feedback

Aka.ms/SQLBits19

Microsoft

Microsoft