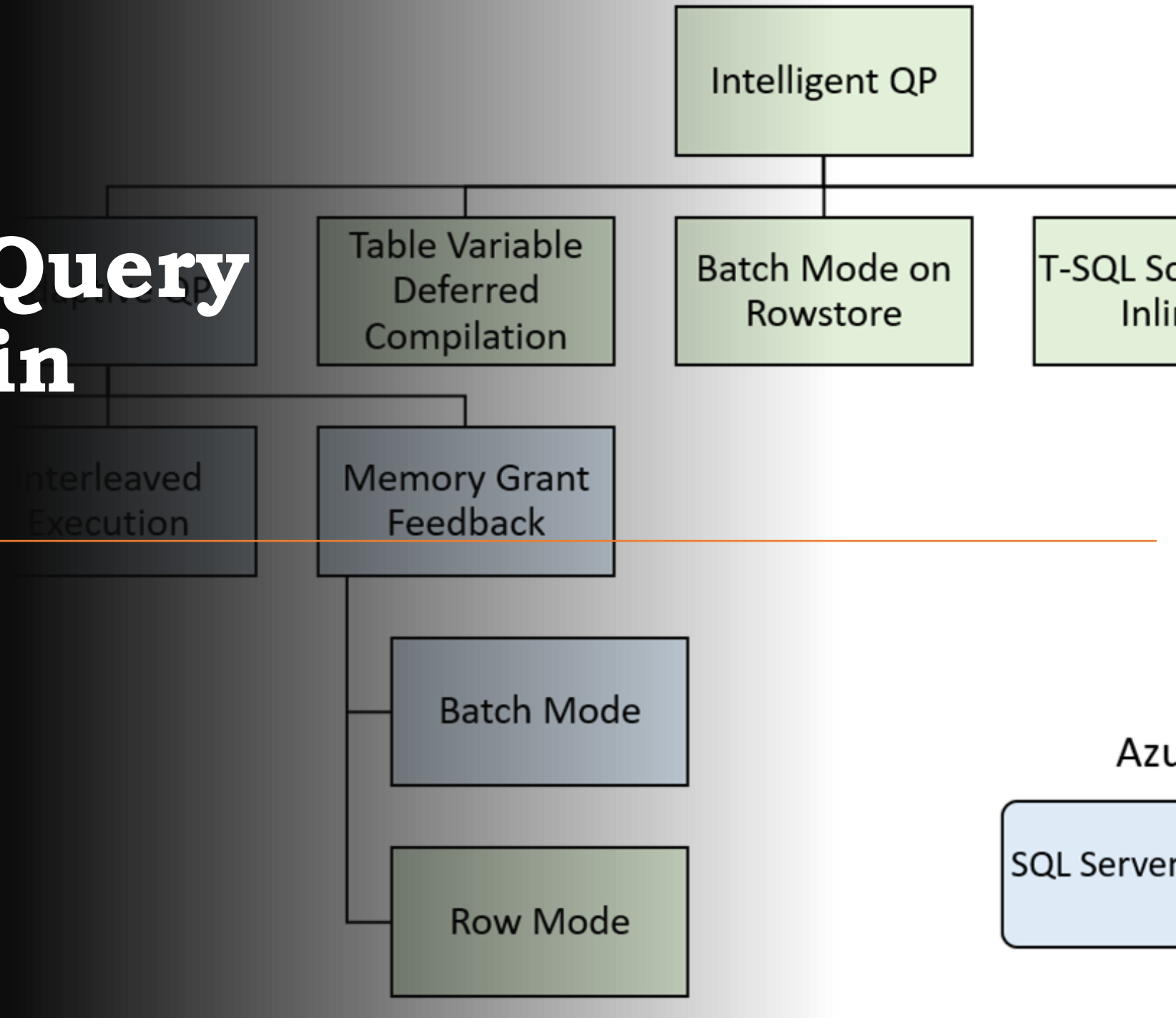
Intelligent Query Processing in SQL Server

Taiob Ali



Taiob Ali

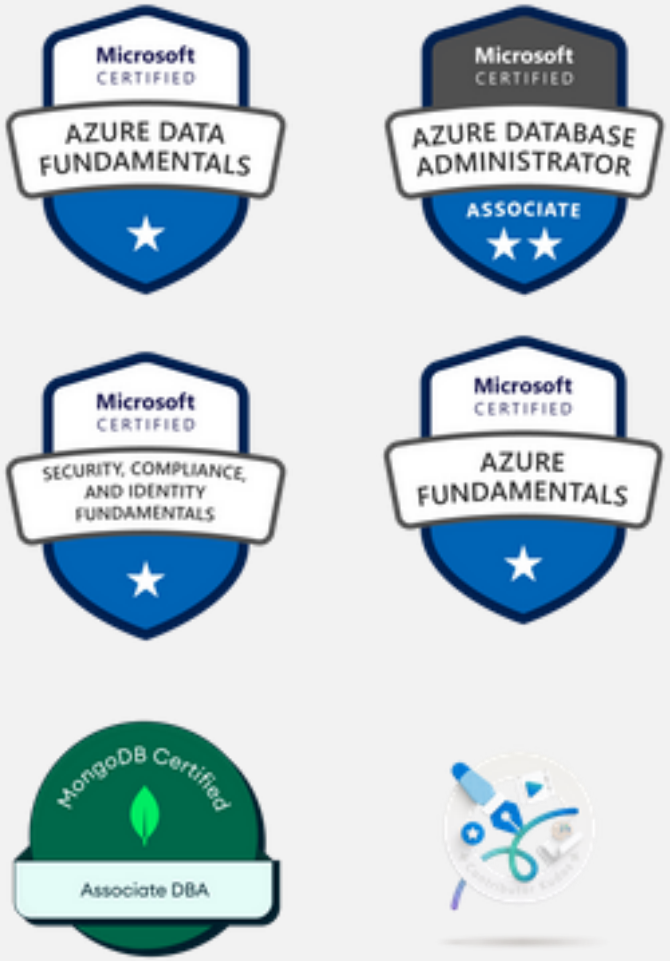
Database Solutions Manager, GMO LLC

http://sqlworldwide.com/

/sqlworldwide

@sqlworldwide

taiob@sqlworldwide.com



Parsing

Transformation

Compare Cost

Plan Cache

Query Execution

Cost

- Parallel
- Serial

Memory Grant

- In Memory
- Spill to Disk

Access Method

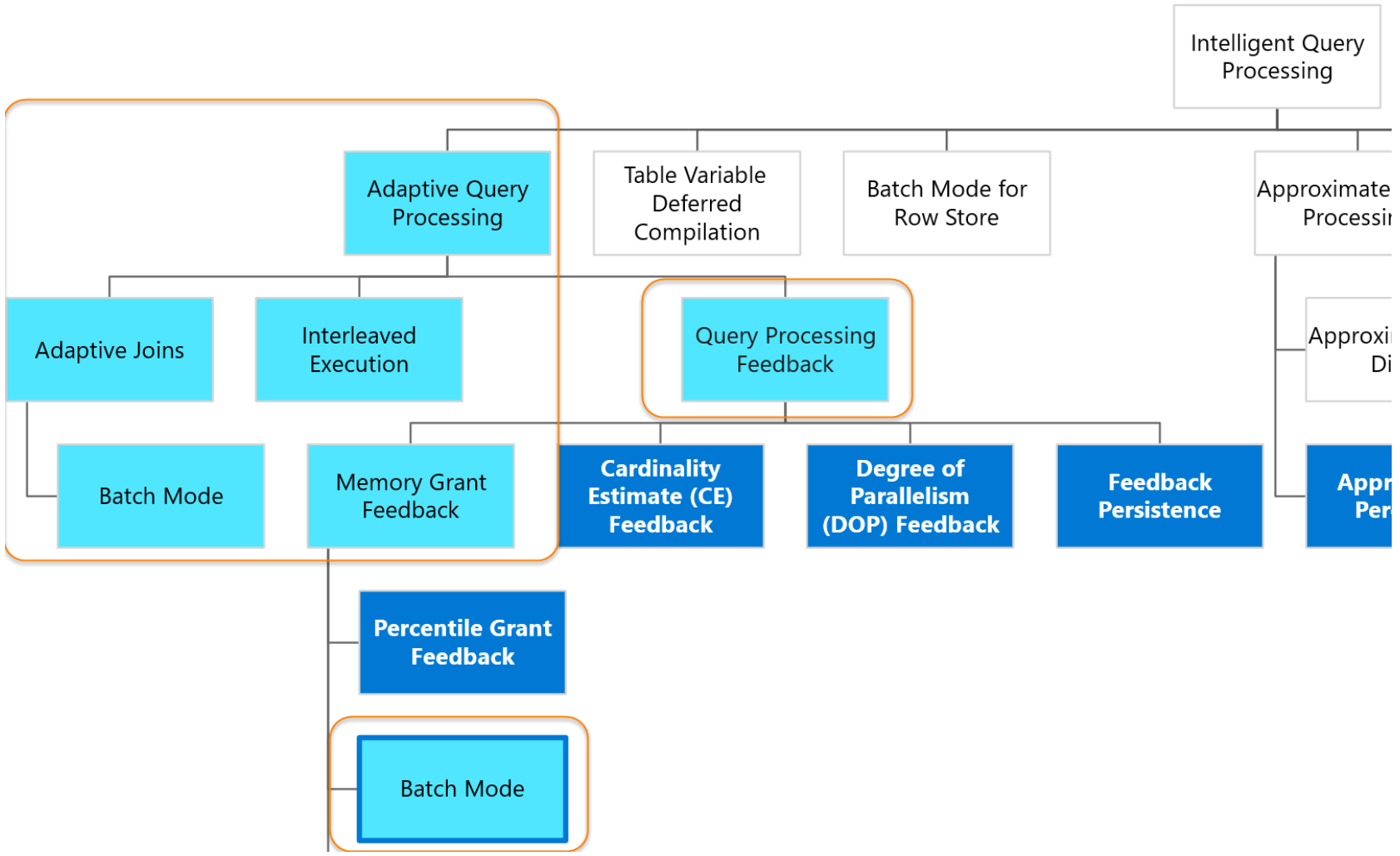
- Seek
- Scan
- Seek + Scan

Algorithm

- Join
- Aggregate
- Sort

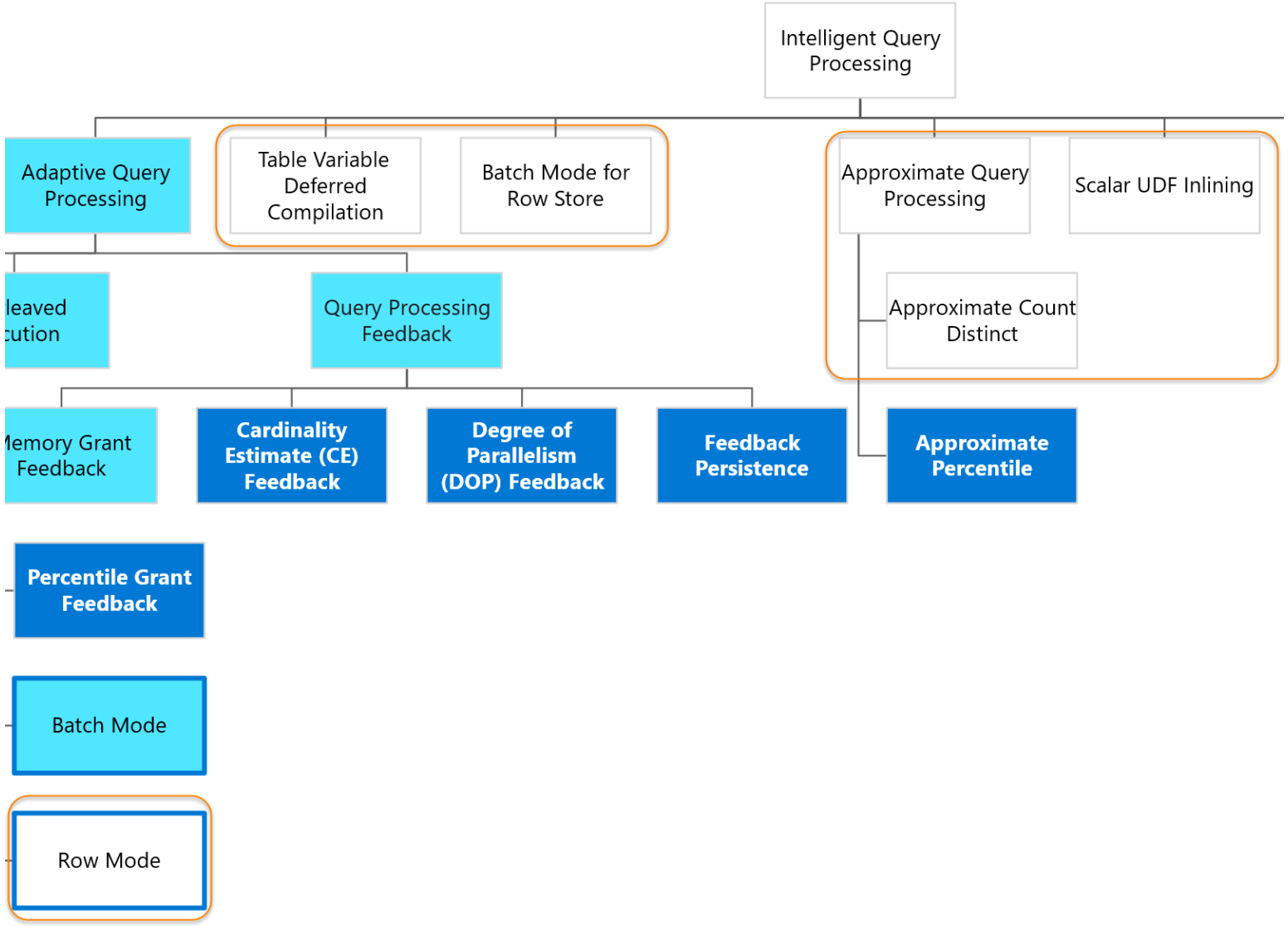

SQL

2017


Intelligent Query Processing
Adaptive Query Processing
Table Variable Deferred Compilation
Batch Mode for Row Store
Approximate Processi
Adaptive Joins
Interleaved Execution
Query Processing Feedback
Approxi Di
Batch Mode
Memory Grant Feedback
Cardinality Estimate (CE) Feedback
Degree of Parallelism (DOP) Feedback
Feedback Persistence
Appr Per
Percentile Grant Feedback
Batch Mode

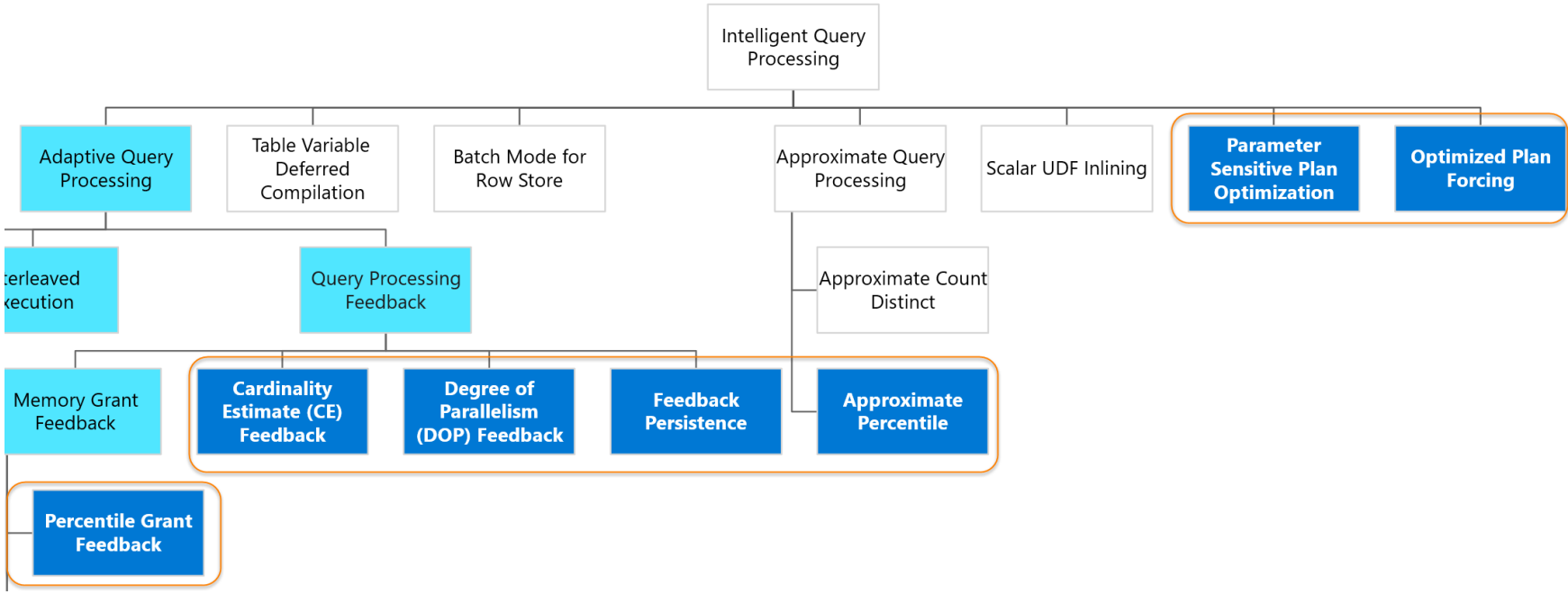

SQL

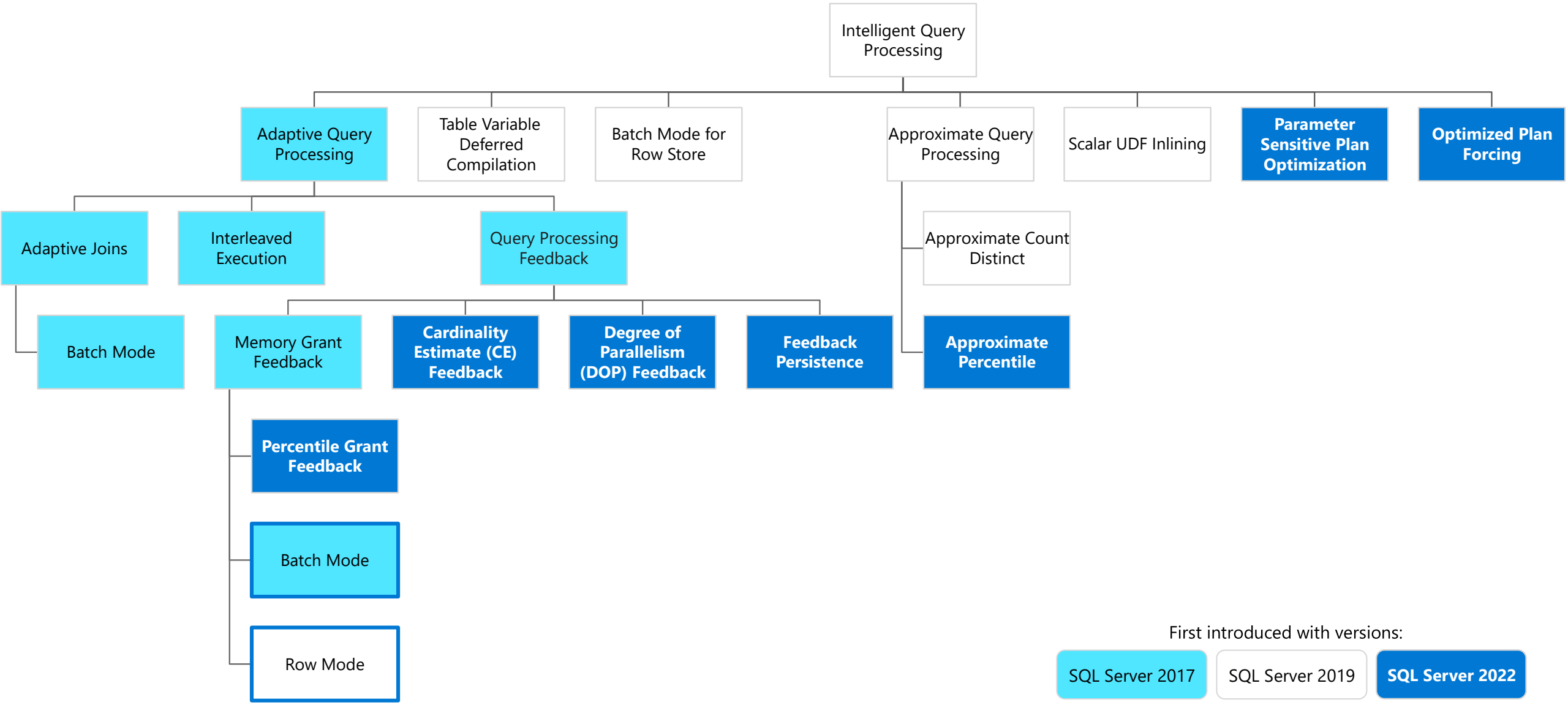
2019


Intelligent Query Processing
Adaptive Query Processing
Table Variable Deferred Compilation
Batch Mode for Row Store
Approximate Query Processing
Scalar UDF Inlining
leaved cution
Query Processing Feedback
Approximate Count Distinct
Memory Grant Feedback
Cardinality Estimate (CE) Feedback
Degree of Parallelism (DOP) Feedback
Feedback Persistence
Approximate Percentile
Percentile Grant Feedback
Batch Mode
Row Mode


SQL

2022


Intelligent Query Processing
Adaptive Query Processing
Table Variable Deferred Compilation
Batch Mode for Row Store
Approximate Query Processing
Scalar UDF Inlining
Parameter Sensitive Plan Optimization
Optimized Plan Forcing
erleaved xecution
Query Processing Feedback
Approximate Count Distinct
Memory Grant Feedback
Cardinality Estimate (CE) Feedback
Degree of Parallelism (DOP) Feedback
Feedback Persistence
Approximate Percentile
Percentile Grant Feedback


Intelligent Query Processing
Adaptive Query Processing
Table Variable Deferred Compilation
Batch Mode for Row Store
Approximate Query Processing
Scalar UDF Inlining
Parameter Sensitive Plan Optimization
Optimized Plan Forcing
Adaptive Joins
Interleaved Execution
Query Processing Feedback
Approximate Count Distinct
Batch Mode
Memory Grant Feedback
Cardinality Estimate (CE) Feedback
Degree of Parallelism (DOP) Feedback
Feedback Persistence
Approximate Percentile
Percentile Grant Feedback
Batch Mode
Row Mode
First introduced with versions:
SQL Server 2017
SQL Server 2019
SQL Server 2022

SYS.DATABASE_SCOPED_CONFIGURATIONS

SYS.DM_EXEC_VALID_USE_HINTS

Problem

Solution

Caution

Adaptive Joins Batch Mode

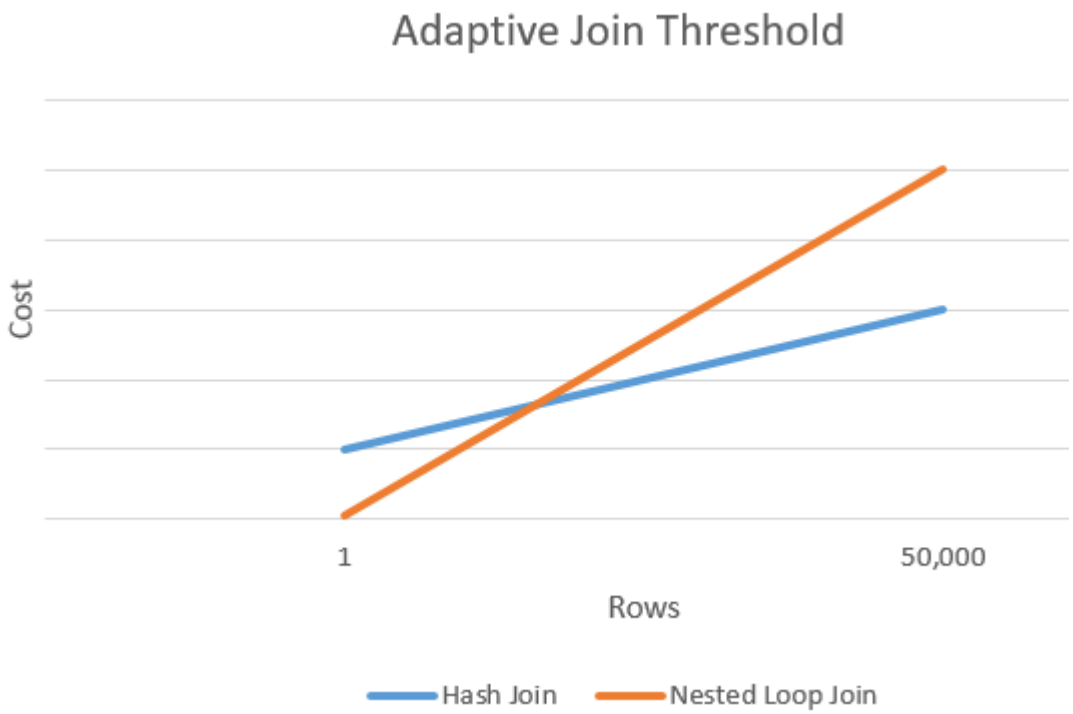
140+

- Join Hint
- Parameter sensitive query

Adaptive Joins Batch Mode

140+

- Dynamically switch to a better join
- Nested loop or Hash Join
- Based on threshold



https://docs.microsoft.com/en-us/sql/relational-databases/performance/media/6_aqpjointhreshold.png?view=sql-server-ver15

Adaptive Joins Batch Mode

140+

- The query is a SELECT statement
- Join needs to be eligible with both Hash and Nested Loop join
- Hash join uses batch mode
- Both joins should have same outer reference
- Introduce a higher memory requirement

Interleaved Execution MSTVFs

140+

- MSTVFs have a fixed cardinality guess of
  - 100 in SQL Server 2014 (12.x)
  - 1 in earlier versions

Interleaved Execution MSTVFs

140+

- Actual row counts are used to make better-informed decision
- Greater performance impact with higher skew

Interleaved Execution MSTVFs

140+

- Must be read-only and NOT part of a data modification
- Must use runtime constant
- Once an interleaved execution plan is cached, that revised estimate is used for consecutive executions without re-instantiating interleaved execution

Table Variable Deferred Compilation

150+

- Works ok with low number of rows but not as rows increase
- Table variables do not have statistics
- Table variables do not have ‘Automatic stats creation’
- Only inline index definitions
- Does not trigger recompile
- Fixed cardinality guess of 1

Table Variable Deferred Compilation

150+

- Optimizer delays the compilation
  - Same as what temporary table does today
- Accurate cardinality – better execution plan
  - Example Hash join instead of Nested loop join

Table Variable Deferred Compilation

150+

- Does not change any other characteristics
- Does not increase recompilation frequency
- Does not fix Parameter Sniffing issues

Memory Grant Feedback Batch Mode

140+

- Performance suffers from incorrect Memory Grant
- Insufficient grant
  - Spill to disk
- Excessive grants
  - Wasted memory
  - Reduced concurrency

Memory Grant Feedback Batch Mode

140+

- Trigger recalculate
  - Result in a spill to disk
  - Granted memory > 2 x size of the actual used memory
- New SSMS property ‘IsMemoryGrantFeedbackAdjusted’ to track feedback

Memory Grant Feedback Batch Mode

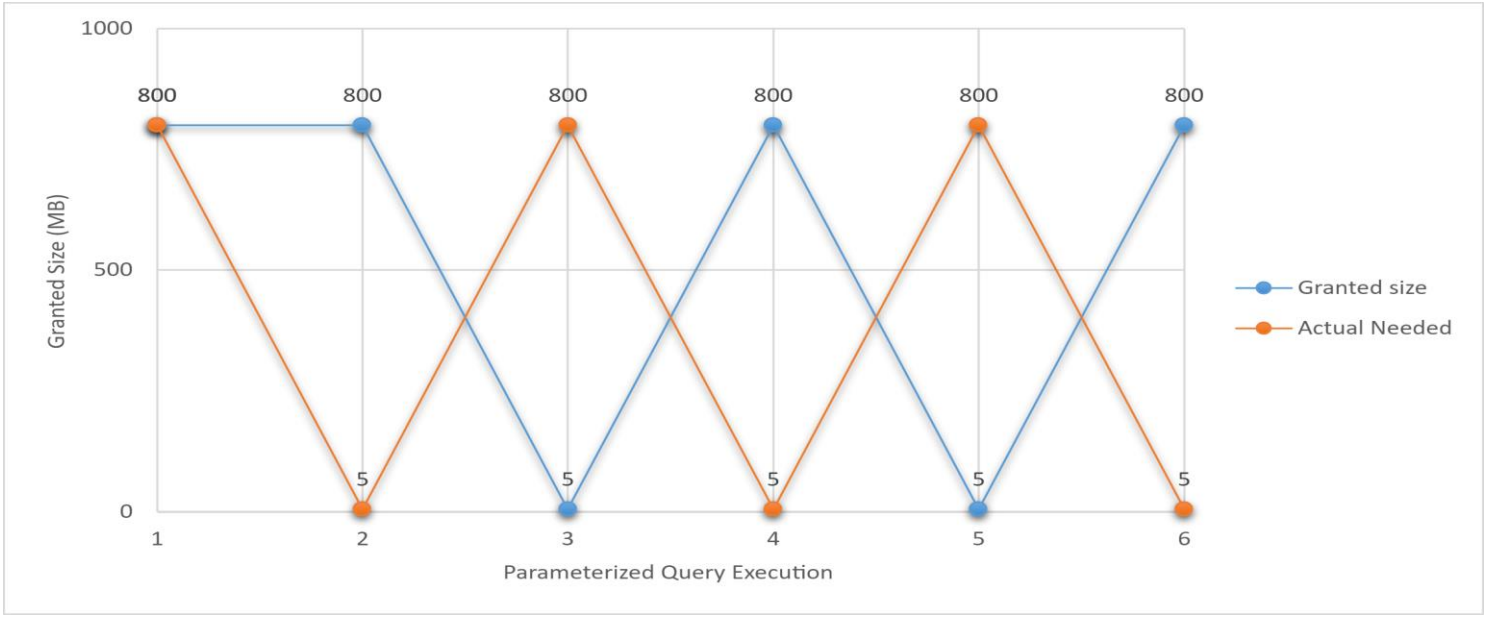
140+

- Will disable itself for parameter-sensitive queries
- Grants under 1 MB will not be recalculated
- Changes are not captured in the Query Store with compatibility level 140
- Memory Granted honors limitation by the resource governor or query hint

Percentile grant Feedback

140+

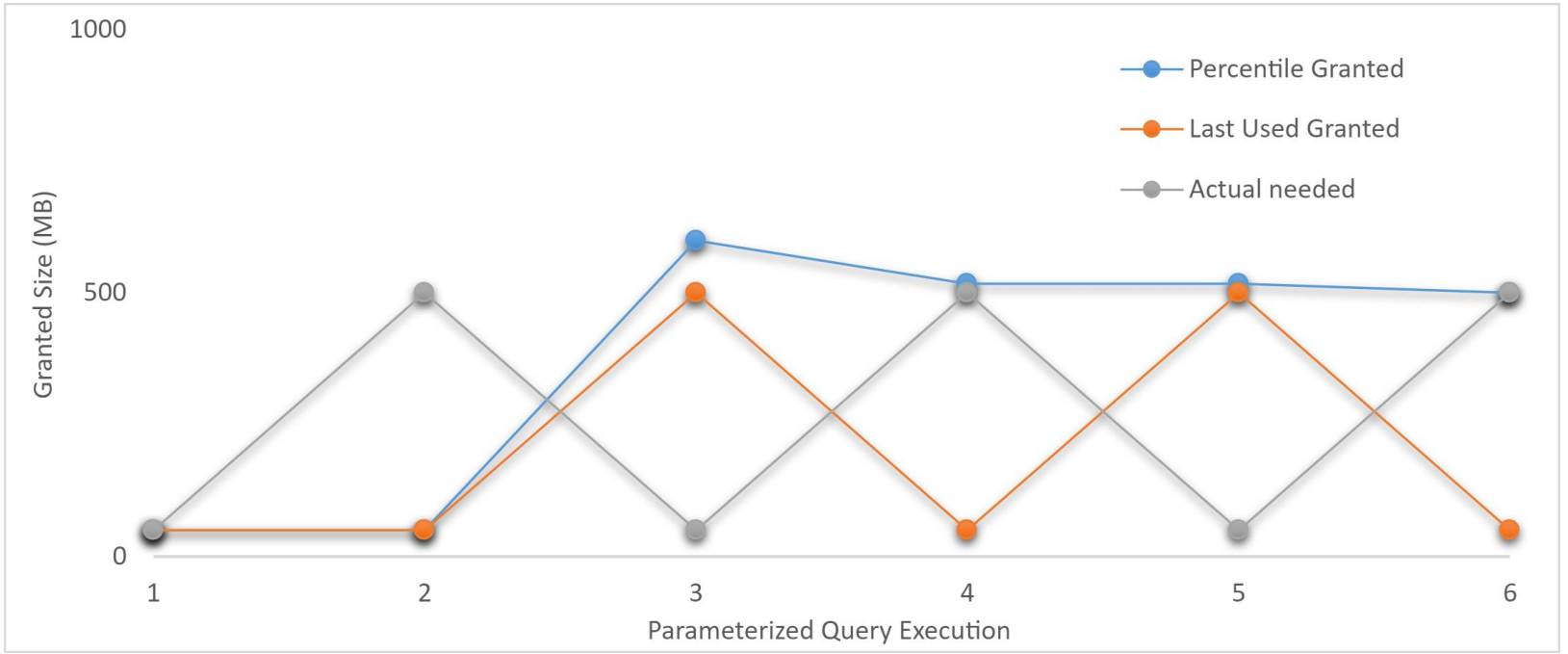
- Grant size adjustments only accounted for the most recently used grant



Percentile grant Feedback

140+

- Using a percentile-based calculation over the recent history of the query



Percentile grant Feedback

140+

- This feature was introduced in SQL Server 2022 (16.x), but is available with CE 140+

Memory Grant Feedback Row Mode

150+

- Row mode memory grant feedback expands on the batch mode

Feedback Persistence

140+

- Existing feature does not work with plan eviction
- Poor performance the first few times a query is executed after an eviction
- Provides new functionality to persist memory grant feedback (an existing feature)

Feedback Persistence

140+

- Using Query Store
- The Query Store must be enabled for every database where the persistence portion of this feature is used.

Feedback Persistence

140+

- Query Store to be enabled for the database and in a “read-write" state
- No impact if Query Store is not enabled
- During failover, the memory grant feedback from the old primary replica is applied to the new primary replica

Cardinality Estimate (CE) Feedback

160+

- No single set of CE models and assumptions can accommodate the vast array of customer workloads and data distributions
- The scenarios include Correlation, Join Containment, and Optimizer row goal

Cardinality Estimate (CE) Feedback

160+

- CE feedback **identifies** model-related assumptions and evaluates whether they're accurate for repeating queries
- If an assumption looks incorrect, a subsequent execution of the same query is tested with a query plan that adjusts the impactful CE model assumption and **verifies** if it helps
- If it improves plan quality, the old query plan is **replaced** with a query plan that uses the appropriate USE HINT query hint that adjusts the estimation model, implemented through the Query Store hint mechanism.

Cardinality Estimate (CE) Feedback

160+

- CE feedback currently only benefits primary replicas, even though Query store for Secondary replica is enabled in SQL 2022
- If a query uses hard-coded query hints or uses Query Store hints set by the user, CE feedback won't be used for that query

Degree of Parallelism (DOP) Feedback

160+

- Instead of incurring the pains of an all-encompassing default or manual adjustments to each query, DOP feedback self-adjusts DOP
- OLTP-centric queries that are executed in parallel could experience performance issues when the time spent coordinating all threads outweighs the advantages of using a parallel plan

Degree of Parallelism (DOP) Feedback

160+

- Parallelism inefficiencies for repeating queries based on elapsed time and waits
- If parallelism usage is deemed inefficient, DOP feedback will lower the DOP for the next execution of the query
- Minimum DOP for any query adjusted with DOP feedback is 2

Degree of Parallelism (DOP) Feedback

160+

- To enable DOP feedback, enable the DOP_FEEDBACK database scoped configuration in a database
- The Query Store must be enabled for every database where DOP feedback is used, and in the "Read write" state
- Stable feedback is reverified upon plan recompilation and may readjust up or down, but never above MAXDOP setting (including a MAXDOP hint)
- DOP feedback is Replica aware

Batch Mode on Rowstore

150+

- Row by Row processing is slow and cpu intensive
- Columnstore indexes may not be appropriate for some applications
- Features might restrict use of Columnstore index

Batch Mode on Rowstore

150+

- Uses heuristics – during estimation phase
  - Table sizes
  - Operators used
  - Estimated cardinalities
- Additional checkpoints, to evaluate plans with batch mode
- Support for all existing batch mode-enabled operator
- Workload consists of analytics queries especially with joins or aggregates
- Workload that is CPU bound

Batch Mode on Rowstore

150+

- Batch mode restriction always applicable
  - Example-Queries involving cursors
- Not applicable for in-memory OLTP tables
- Not applicable for any index other than on-disk heaps and B-trees
- Won’t kick in for
  - Large Object (LOB) column
  - XML column
  - Sparse column sets
- Two features are independent

Approximate Count Distinct

150+

- Responsiveness is important than absolute precision
- Example
  - Dashboard scenarios
  - Data science trying to understand data distributions

Approximate Count Distinct

150+

- Access of data sets that are millions of rows or higher
- Aggregation of a column or columns that have many distinct values
- Use less memory compared to exhaustive COUNT DISTINCT
- Based on HyperLogLog algorithm

Approximate Count Distinct

150+

- The function implementation guarantees up to a 2% error rate within a 97% probability

Approximate Percentile

110+

- Large datasets where negligible error with a faster response is acceptable as compared to accurate percentile value with a slow response

Approximate Percentile

110+

- The algorithm used for these functions is KLL sketch which is a randomized algorithm
- Every time the sketch is built, random values are picked
- These functions provide rank-based error guarantees, not value-based

Approximate Percentile

110+

- The output of the function may not be the same in all executions
- The function implementation guarantees up to a 1.33% error bounds within a 99% confidence

TSQL Scalar UDF Inlining

150+

- Iterative invocation
- Lack of costing
- Serial Execution
- Interpreted execution
- Imperative code does not scale

TSQL Scalar UDF Inlining

150+

- UDFs are automatically transformed into
  - Scalar Expressions
  - Scalar subqueries
- Further optimization followed by transformation
- Refactors the Imperative code into Relational Algebric Expression – Froid Framework
- Resulting execution plan
  - Efficient
  - Set-Oriented
  - Parallel
- New SSMS property ‘ContainsInlineScalarTsqlUdfs’ to track inlining

TSQL Scalar UDF Inlining

150+

- Requirements to be eligible
- Scalar UDF Inlining issues in SQL Server 2019
- sys.sql_modules – Can this UDF be inlined?
- Can disable within function definition

```
-- Transact-SQL Function Clauses
<function_option>::=
{
    [ ENCRYPTION ]
  | [ SCHEMABINDING ]
  | [ RETURNS NULL ON NULL INPUT | CALLED ON NULL INPUT ]
  | [ EXECUTE_AS_Clause ]
  | [ INLINE = { ON | OFF }]
}
```

Parameter Sensitive Plan Optimization

160+

- Single cached plan for a parameterized query isn't optimal for all possible incoming parameter values

Parameter Sensitive Plan Optimization

160+

- During the initial compilation, column statistics histograms identify non-uniform distributions and evaluate the most *at-risk* parameterized predicates, up to three out of all available predicates
- PSP feature limits the number of predicates that are evaluated to avoid bloating the plan cache and the Query Store (if Query Store is enabled) with too many plans
- Initial compilation produces a *dispatcher plan* that contains the PSP optimization logic called a *dispatcher expression*
- A dispatcher plan maps to *query variants* based on the cardinality range boundary values predicates

Parameter Sensitive Plan Optimization

160+

- The PSP optimization feature currently only works with equality predicates
- Query variant plans will recompile independently as needed, as with any other query plan type
- When multiple predicates are part of the same table, PSP optimization will select the predicate that has the most data skew based on the underlying statistics histogram

Optimized Plan Forcing

160+

- Optimized plan forcing reduces compilation overhead for repeating *forced* queries

Optimized Plan Forcing

160+

- When a query first goes through the compilation process, a threshold based on estimating the time spent in optimization (based on the query optimizer input tree) will determine whether an optimization replay script is created
- These runtime metrics include the number of objects accessed, the number of joins, the number of optimization tasks executed during optimization, and the actual optimization time.

Optimized Plan Forcing

160+

- Only query plans that go through full optimization are eligible
- Statements with RECOMPILE hint and distributed queries are not eligible
- Even if an optimization replay script was generated, it might not be persisted in the Query Store if the Query Store configured capture policies criteria aren't met, notably the number of executions of that statement and its cumulated compile and execution times

Query Store Hints

160+

- Need immediate behavior change
- No access to source code
- Plan guide – never easy to use
- Example
  - Recompile a query on each execution.
  - Cap the memory grant size for a bulk insert operation.
  - Limit the maximum degree of parallelism for a statistics update operation.
  - Use a Hash join instead of a Nested Loops join.
  - Use compatibility level 110 for a specific query while keeping everything else in the database at compatibility level 150.
  - Disable row goal optimization for a SELECT TOP query.

Query Store Hints

160+

Query Executed

Query captured in Query Store

DBA creates a Query Store hint on a query

Query executes using Query Store hint

Query Store Hints

160+

- Query store hints override statement level hints (hard-coded) and plan guide hints
- If hints contradict, query execution will not be blocked
- Query Store hints are persisted and survive restarts and failovers

```
Disabling any of these features without changing the compatibility
level
-- SQL Server 2017
ALTER DATABASE SCOPED CONFIGURATION SET
DISABLE_BATCH_MODE_MEMORY_GRANT_FEEDBACK = ON;

-- Starting with SQL Server 2019, and in Azure SQL Database
ALTER DATABASE SCOPED CONFIGURATION SET
BATCH_MODE_MEMORY_GRANT_FEEDBACK = OFF;

You can also disable any of these features for a specific query by
using ‘USE HINT’ query hint
OPTION (USE HINT ('DISABLE_BATCH_MODE_MEMORY_GRANT_FEEDBACK'));
```

Resource

- Intelligent Query Processing in SQL databases
- Enterprise Only Features?
- Intelligent Query Processing Demos – I
- Intelligent Query Processing Demos - II
- Compatibility Certification
- Get Your Scalar UDFs to Run Faster Without Code Changes
- Batch Mode Bitmaps in SQL Server by Paul White
- Query Store hints

SQL2022 CU4
SSMS 19.1

https://www.invotide.com/demo

@sqlworldwide

linkedin.com/in/sqlworldwide

sqlworldwide.com

taiob@sqlworldwide.com


THANK
YOU