Microsoft

Azure SQL & SQL Server 2022 Intelligent Database Futures

Pedro Lopes
Principal Architect
@SQLPedro

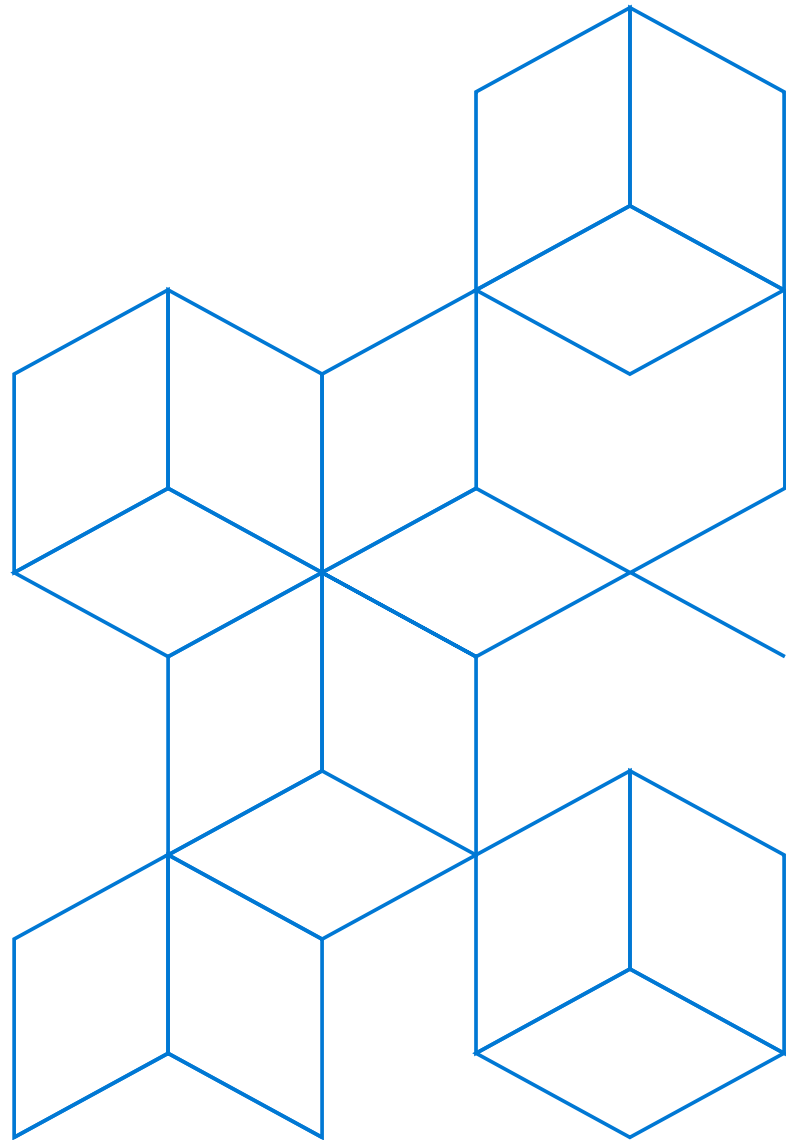
At the end, please rate this session

https://sqlb.it/?7187

Agenda

- Query Store
  - ON by default
  - Query Store hints
  - Query Store for Secondary Replicas
- IQP
  - Parameter-sensitive Plan (PSP) Optimization
  - Improvements to Memory Grant Feedback
  - CE Feedback
  - DOP Feedback
  - Optimized Plan Forcing using compilation replay
  - Approximate Percentile

Query Store ON by default



Query Store ON by default

For new databases only

- SQL never changes database defaults when restoring/attaching to higher version engine.

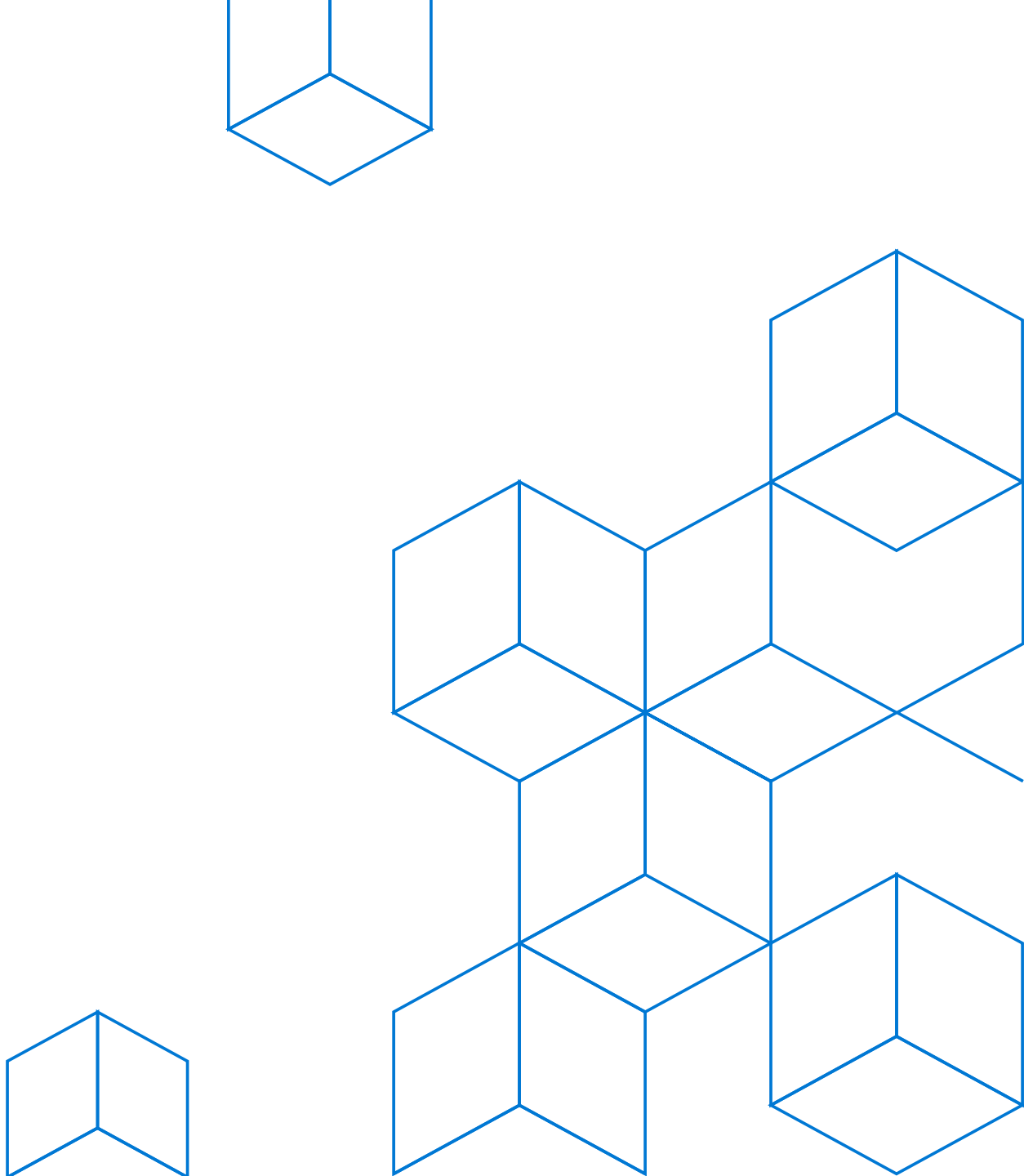
Why now?

- We've added numerous scalability improvements over the years in Azure and in SQL Server
- Better defaults starting with SQL Server 2019 = Azure
- Handles heavy ad-hoc workloads due to internal memory limits and throttling
- Custom capture policies available for fine tuning

Query Store hints

Now in public preview in Azure SQL Database

SQL Server 2022 CTP 1.0



What are Query hints?

- Ideally the Query Optimizer selects an optimal execution plan for a query.
- When this doesn't happen, a developer or DBA may wish to manually optimize for specific conditions.
- Specified via the OPTION clause, query hints can be used to influence the behavior of operators in a statement.

```
SELECT COUNT(DISTINCT [WWI Order ID])
FROM [Fact].[OrderHistoryExtended]
OPTION (USE HINT('DISALLOW_BATCH_MODE'), RECOMPILE);
```

Applying Query hints Today

- Query hints help provide localized solutions to various performance-related issues – but they do require a rewrite of the original query text

- DBAs often cannot make changes directly to T-SQL code
  - T-SQL hard-coded into application
  - T-SQL automatically generated by the application

- DBA may have to rely on plan guides
  - Common feedback: “plan guides are complex to use and manage”

Query Store hint Scope and Behavior

Query Store hints support statement (query)-level hints in the first version

Query Store hints are **persisted**, surviving restarts

They override hard-coded statement level hints and plan guide hints

If hints contradict what is possible, *we will not block* query execution and Query Store hint will *not* be applied

How to use Query Store hints - Step 1

Find the Query Store query_id of the query you wish to modify:

```
SELECT query_sql_text, q.query_id
FROM sys.query_store_query_text qt
INNER JOIN sys.query_store_query q ON
      qt.query_text_id = q.query_text_id
WHERE query_sql_text like N'%April Minerd%';
```

query_sql_text	query_id
SELECT T_S_SYMB, AVG(T_TRADE_PRICE) AS AVG_TRADE_...	46006

How to use Query Store hints – Step 2

Execute **sp_query_store_set_hints** with the query_id and query hint string you wish to apply to the query:

```
-- Setting a single query hint
EXEC sp_query_store_set_hints 46006, N'OPTION(MAXDOP 1)';


-- Setting multiple query hints
EXEC sp_query_store_set_hints 46006, N'OPTION(MAXDOP 1, USE
HINT(''QUERY_OPTIMIZER_COMPATIBILITY_LEVEL_120''))';
```

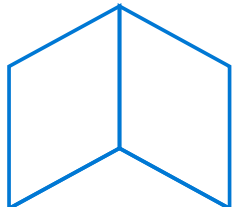
To remove a hint:

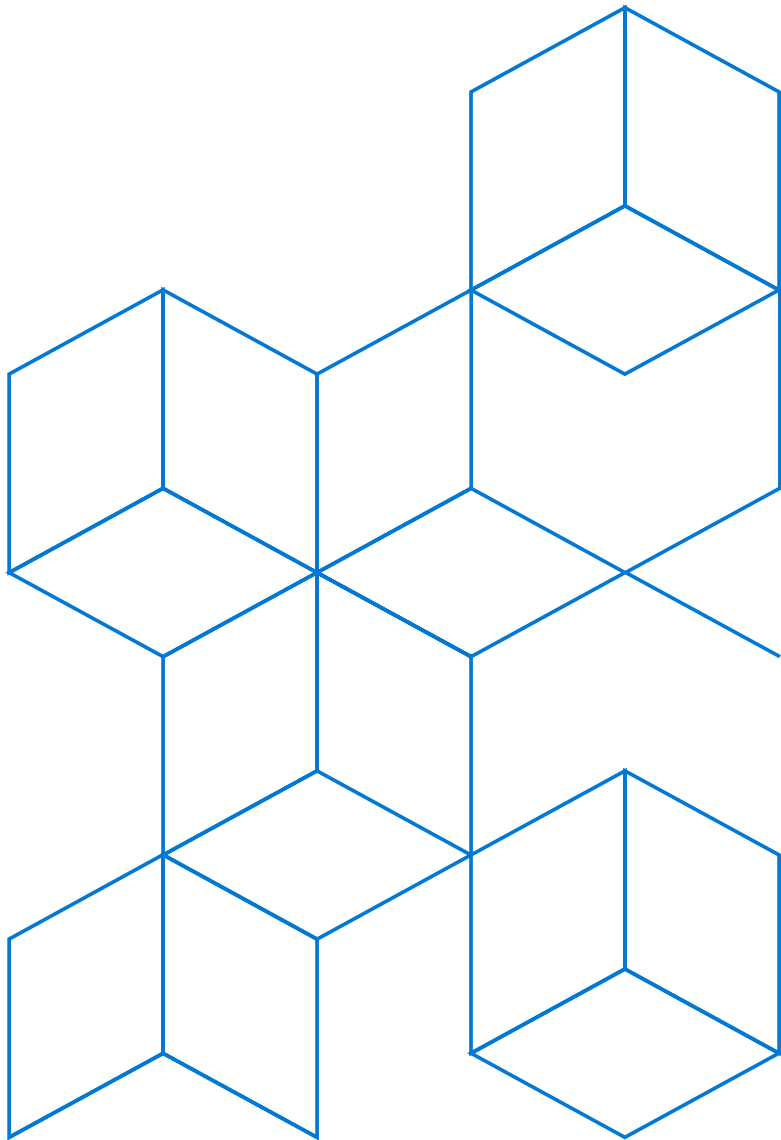
```
EXEC sp_query_store_clear_hints 46006;
```

Query Store for Secondary Replicas

2022 public preview in Azure SQL Database

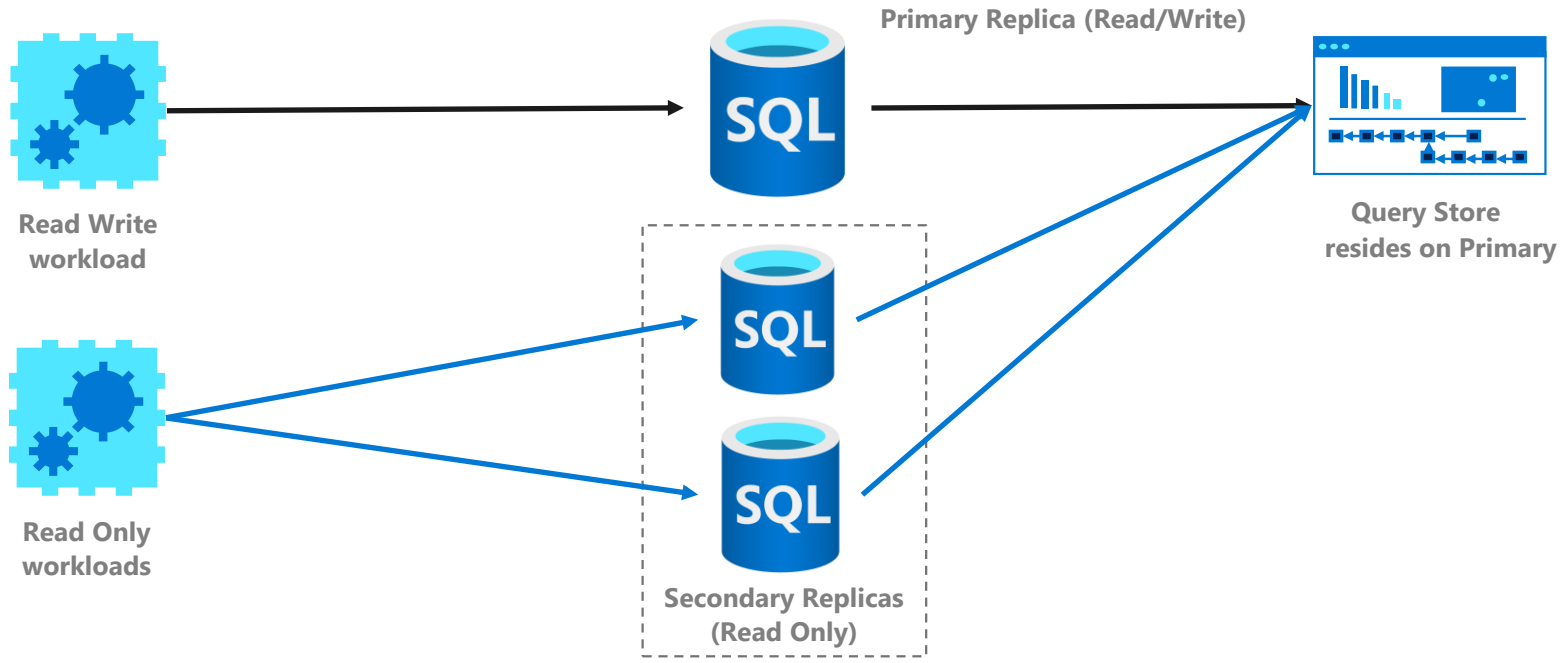
SQL Server 2022 CTP 1.2





Query Store for Secondary Replicas

- Get the same support for the secondary replicas as you already do on the primary



- Query data will be visible by role type or replica name

Enable Query Store for Secondary Replicas

Connect to the Primary Replica and enable Query Store:

```
ALTER DATABASE [Database_Name] SET QUERY_STORE = ON;
```

Turn on the query store for the secondary (execute on Primary):

```
ALTER DATABASE [Database_Name]
FOR SECONDARY SET QUERY_STORE = ON (OPERATION_MODE = READ_WRITE);
```

Force and Unforce plan

New optional plan scope argument added to the force and unforce plan procedures:

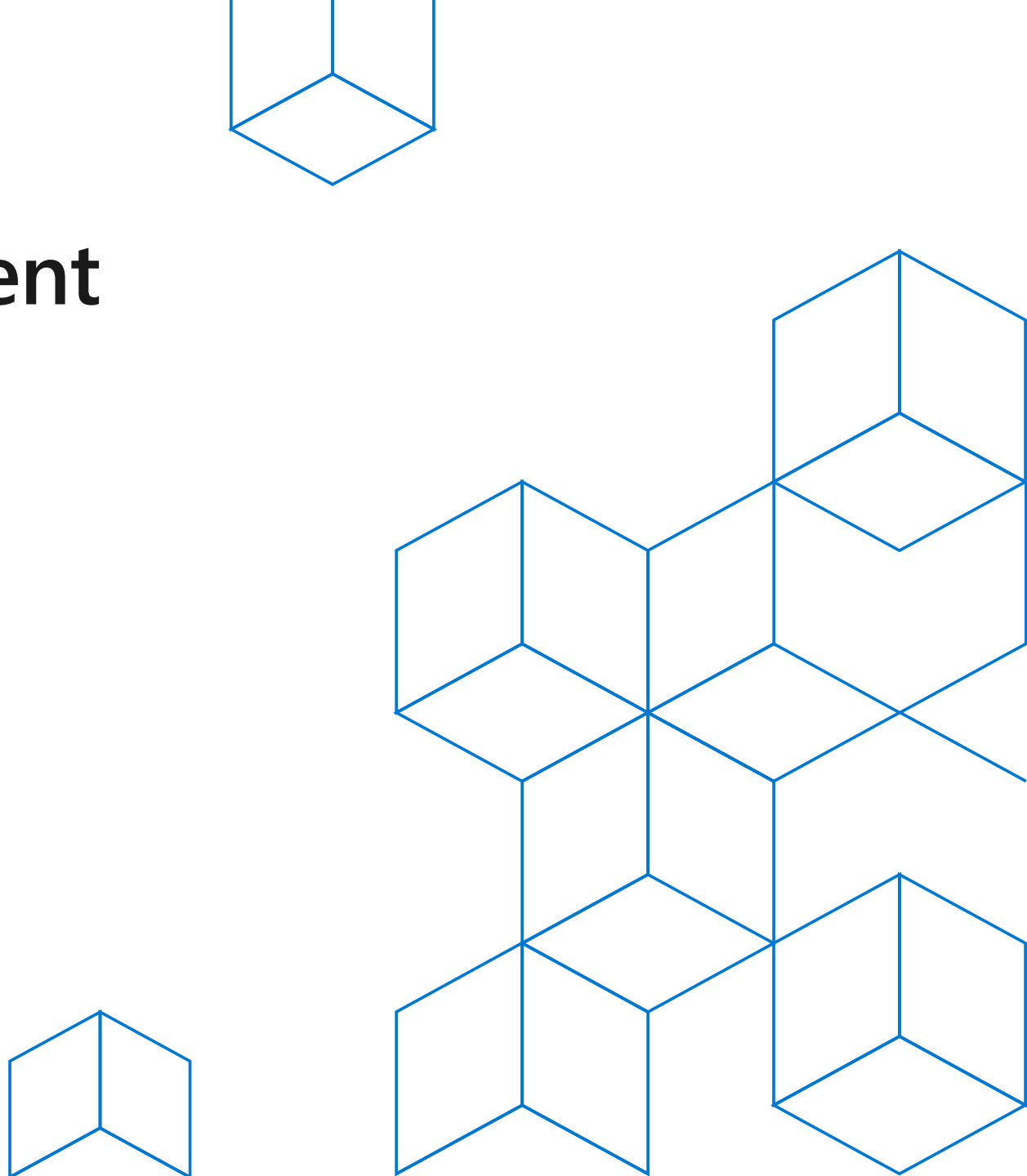
```
EXEC sp_query_store_force_plan 46006, 2, 1
EXEC sp_query_store_unforce_plan 46006, 2, 1
```

Plan forcing scope parameter:

- 0 = force on read-write replica (default if omitted)
- 1 = force/unforce on all read-only replicas
- 2 = force/unforce on all replicas

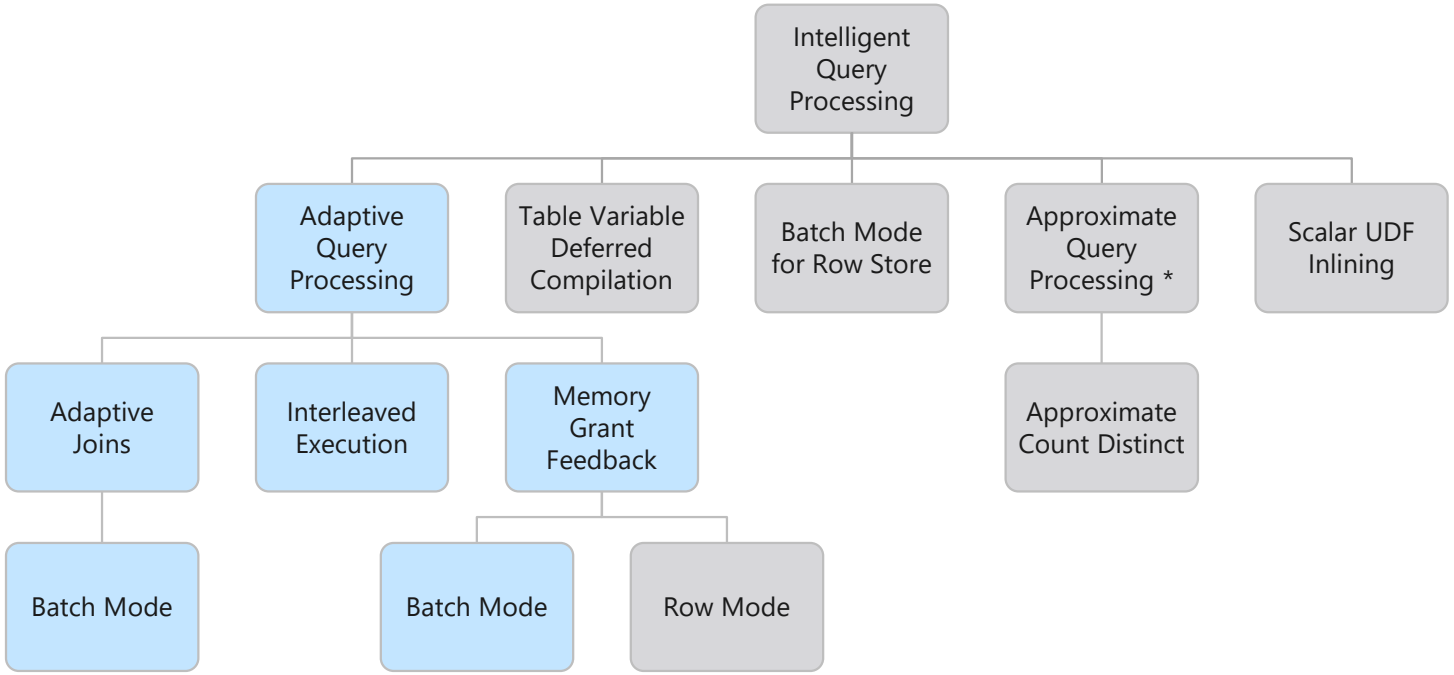
A brief history of Intelligent Database

…and how to get started



Intelligent Query Processing

The Intelligent Query Processing feature family



Azure SQL DB

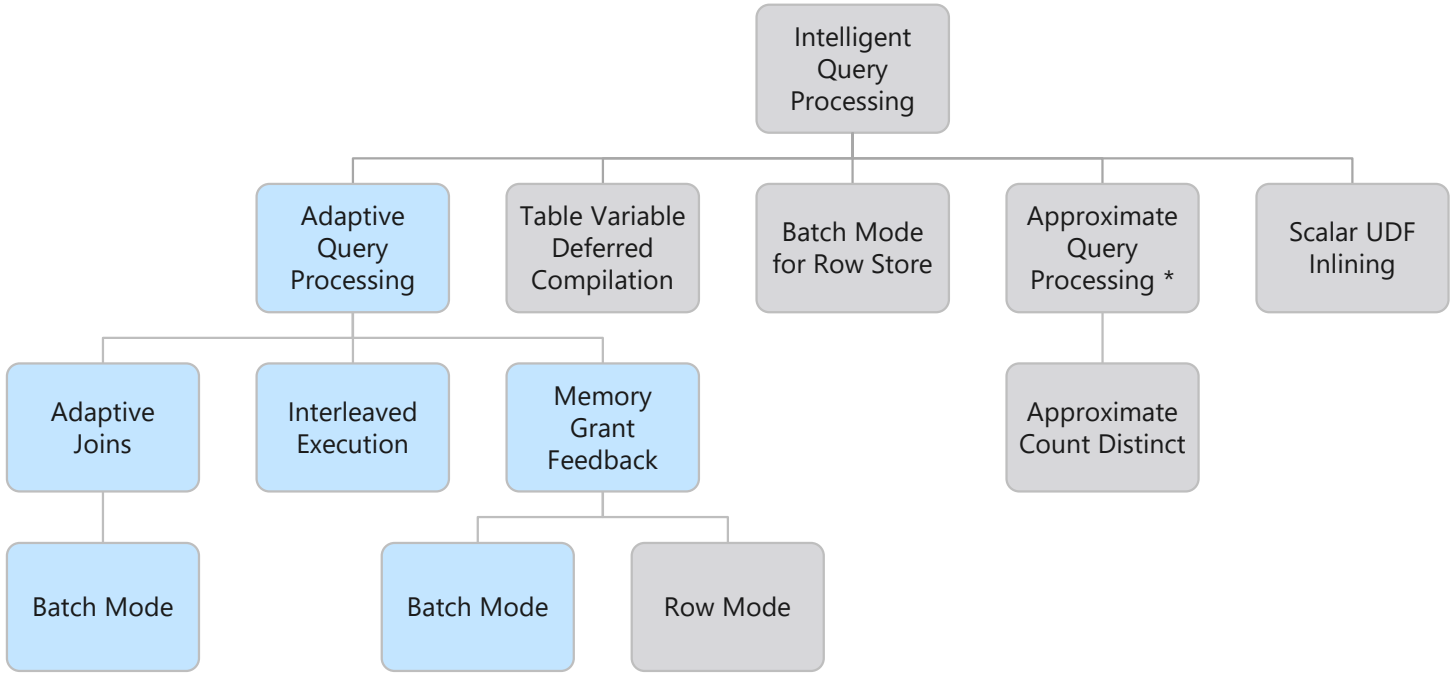
SQL Server 2017

SQL Server 2019

Intelligent Query Processing

- Available by default on the latest database compatibility level setting
- Delivering broad impact that improves the performance of existing workloads with zero implementation effort
- Critical parallel workloads improve when running at scale, while remaining adaptive to changes in data

The Intelligent Query Processing feature family



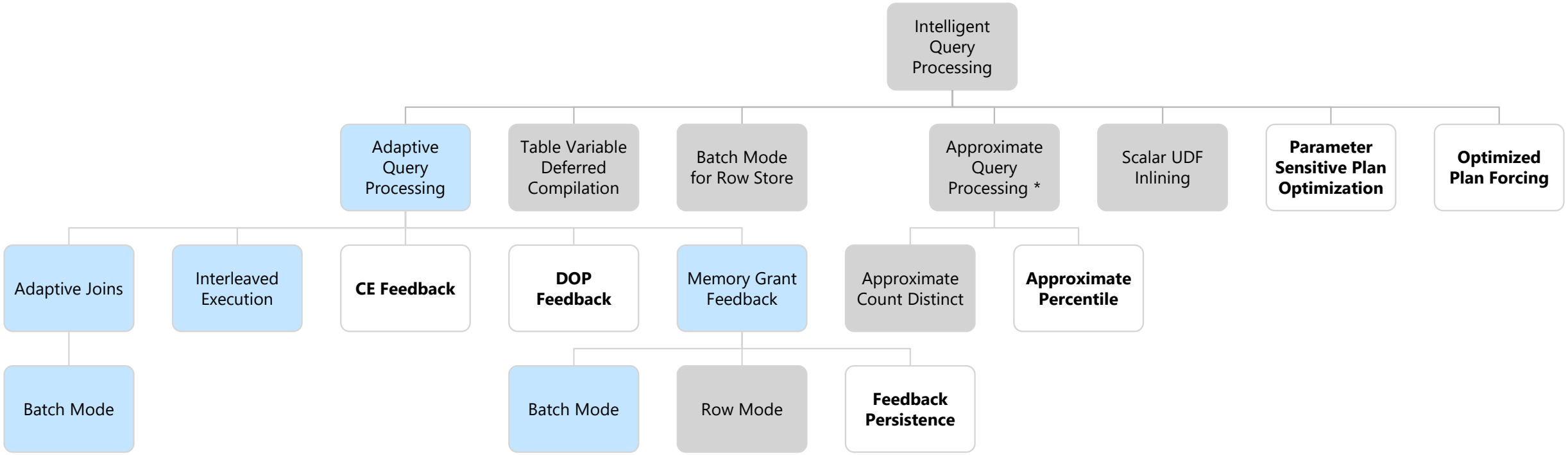
Azure SQL DB

SQL Server 2017 | SQL Server 2019

References: https://aka.ms/IQP and https://aka.ms/IQPDemos

Intelligent Query Processing

The Intelligent Query Processing feature family



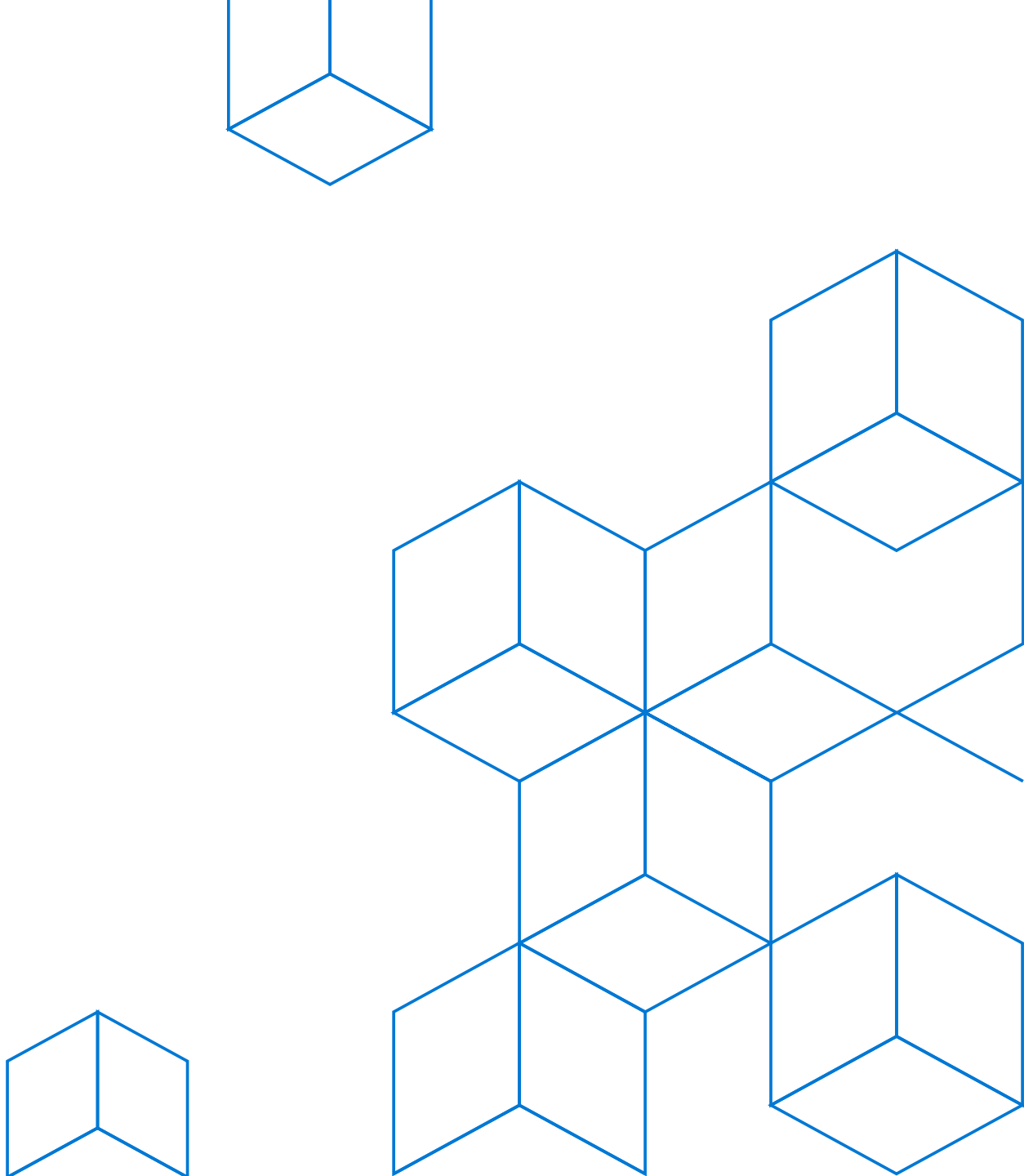
Azure SQL DB

SQL Server 2017 | SQL Server 2019 | SQL Server 2022

Parameter-sensitive Plan Optimization

2022 public preview in Azure SQL Database

SQL Server 2022 CTP 1.0



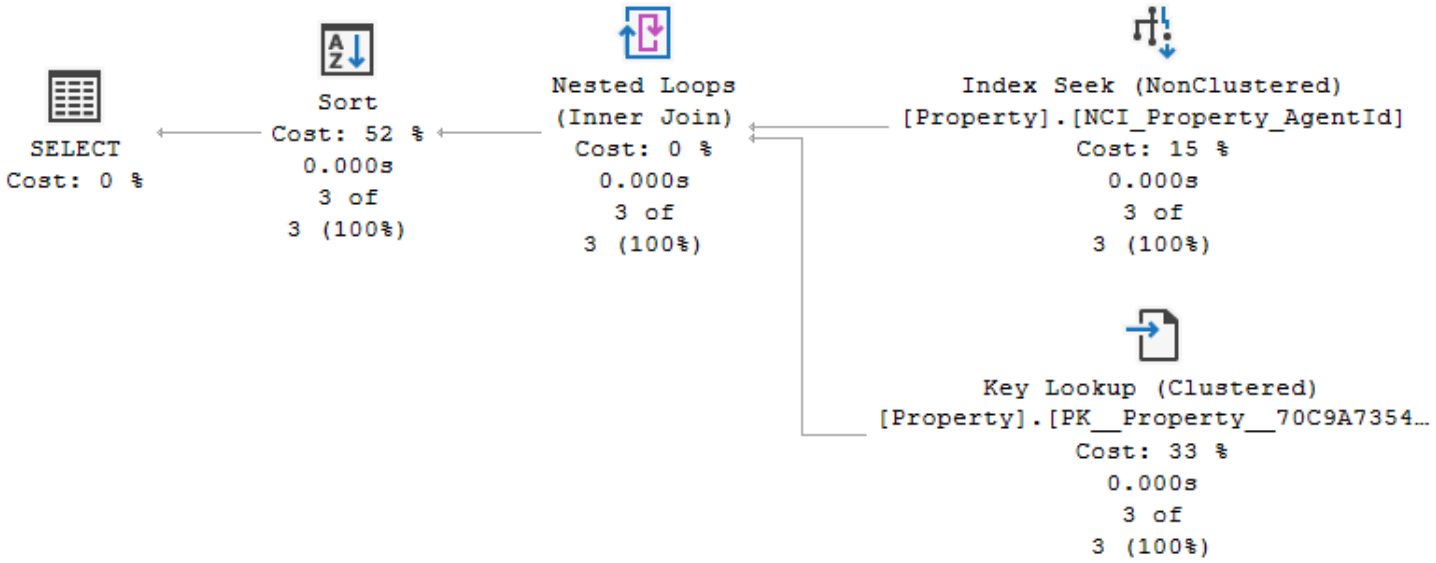
Parameter-sensitive plan problem

Parameter-sensitive Plan (PSP), a.k.a. Parameter-sniffing problem refers to a scenario where a **single** cached plan for a parameterized query is **not optimal for all** possible incoming parameter values

If the 1st compilation is not representative of most executions, you have a perceived “bad plan”

PSP today (example of Real Estate agents portfolio)

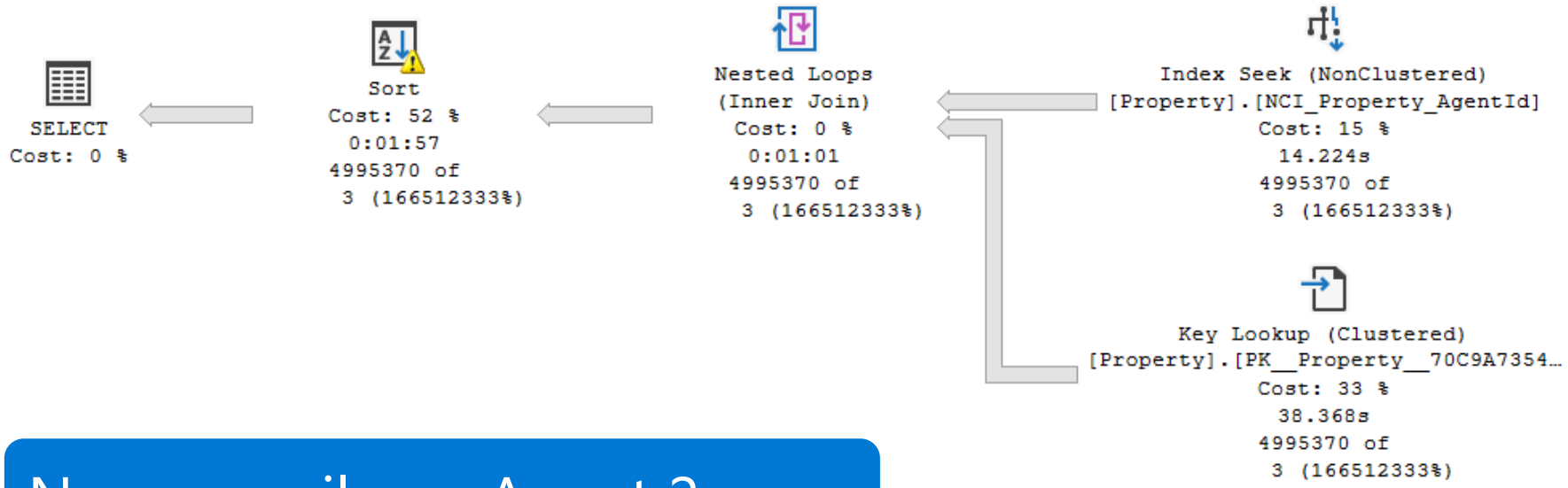
New compile on Agent 4



QueryTimeStats	
CpuTime	0
ElapsedTime	0

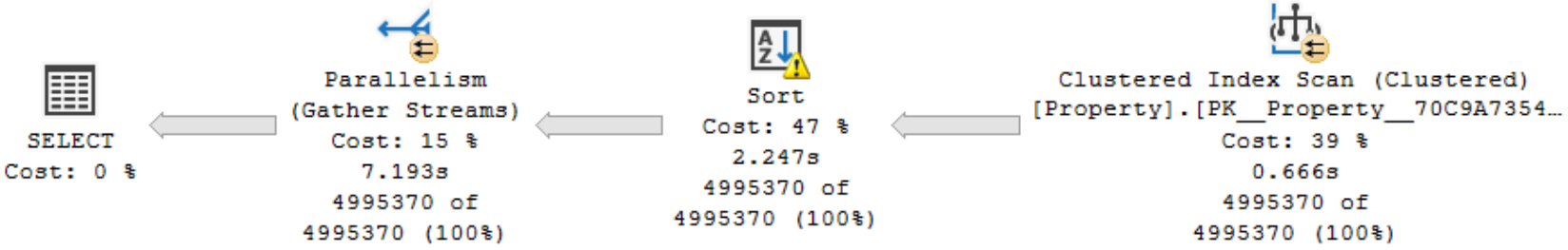
PSP today (example of Real Estate agents portfolio)

Using cached plan for Agent 2



QueryTimeStats	
CpuTime	88667
ElapsedTime	214222

New compile on Agent 2



QueryTimeStats	
CpuTime	46620
ElapsedTime	105288

PSP workarounds

RECOMPILE	OPTION (OPTIMIZE FOR…)	OPTION (OPTIMIZE FOR UNKNOWN)	Disable parameter sniffing entirely
KEEPFIXEDPLAN	Force a known plan	Nested procedures	Dynamic string execution

PSP Optimization

Enabled using Database Compatibility 160

Automatically enable multiple, active cached plans for a single parameterized statement

Cached execution plans will accommodate different data sizes based on the customer-provided runtime parameter value(s)

Design considerations

- If we generate too many plans, we could create cache bloat, so limit # of plans in cache
- Overhead of PSP optimization must not outweigh downstream benefit
- Compatible with Query Store plan forcing

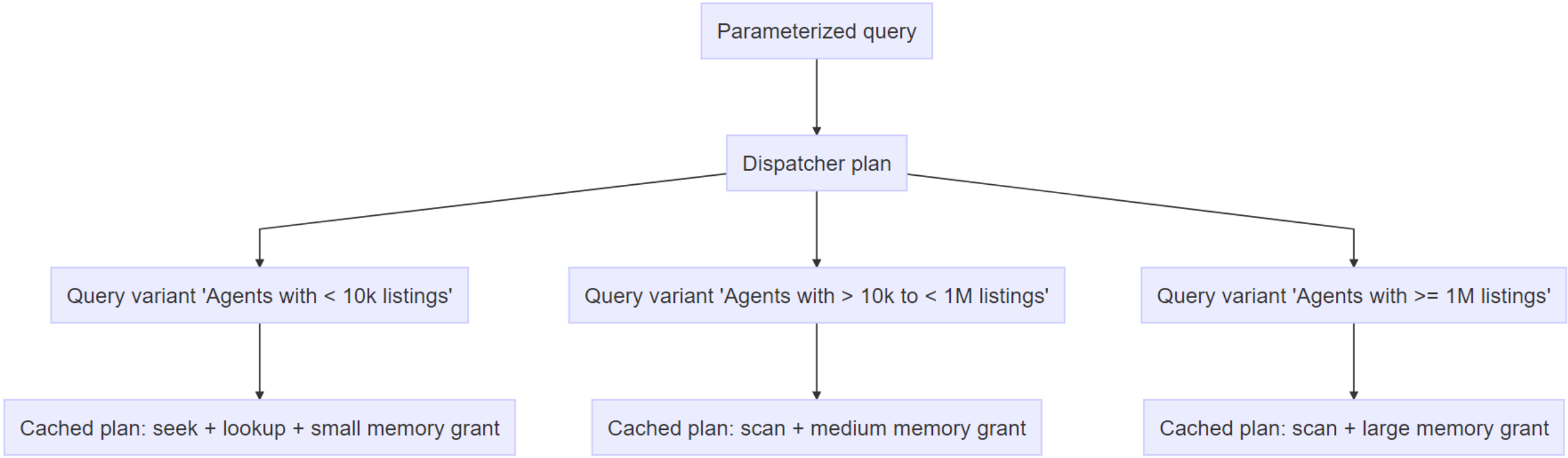
Predicate selection

During initial compilation we will evaluate the most “at risk” parameterized predicates (up to three out of all available)

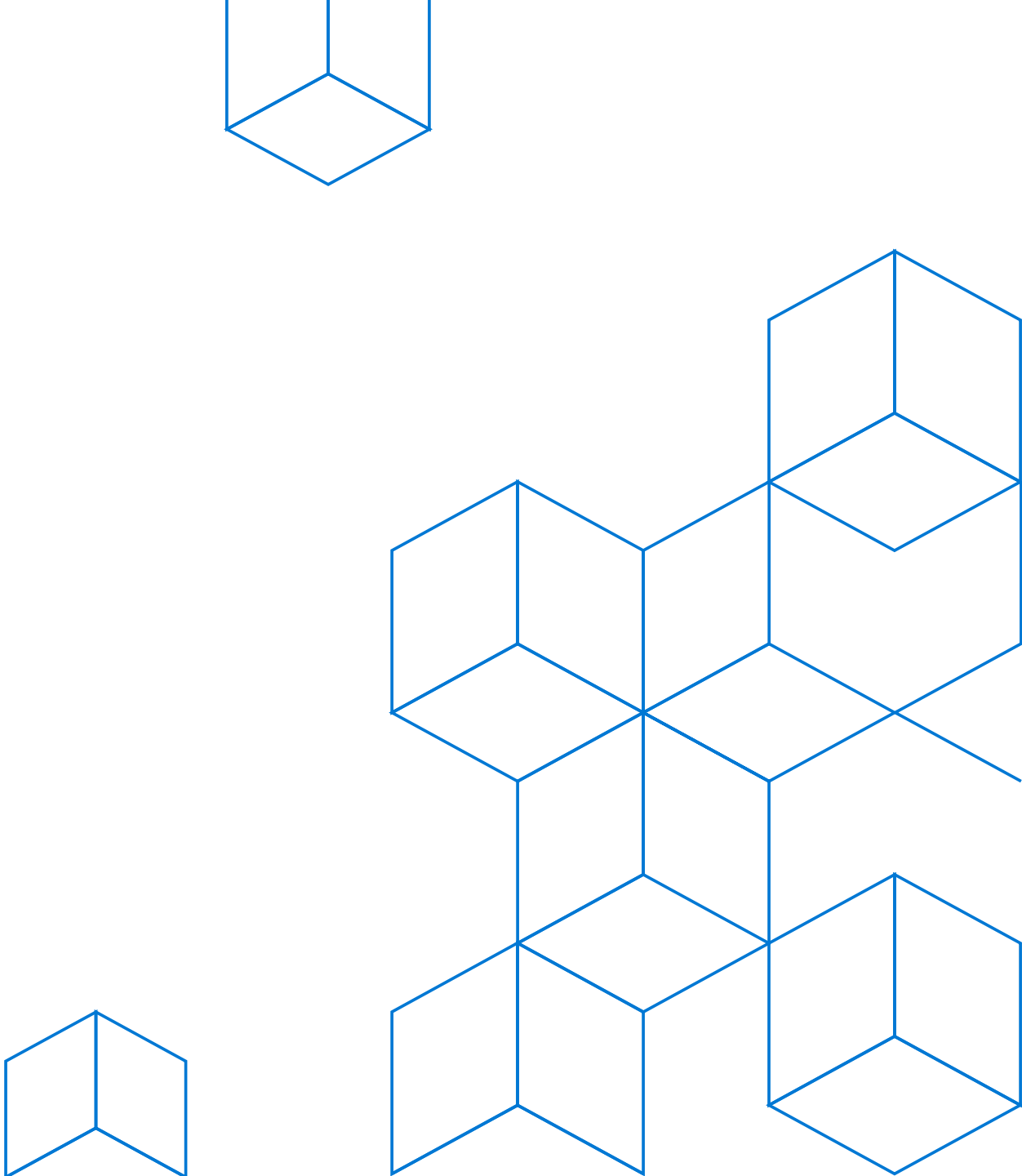
First version is scoped to equality predicates referencing statistics-covered columns; i.e., `WHERE AgentId = @AgentId`

Uses the column statistics histogram(s) to identify non uniform distributions

Boundary value selection (example of Real Estate agents portfolio)



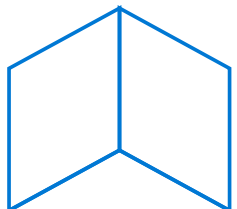
PSP Optimization Demo

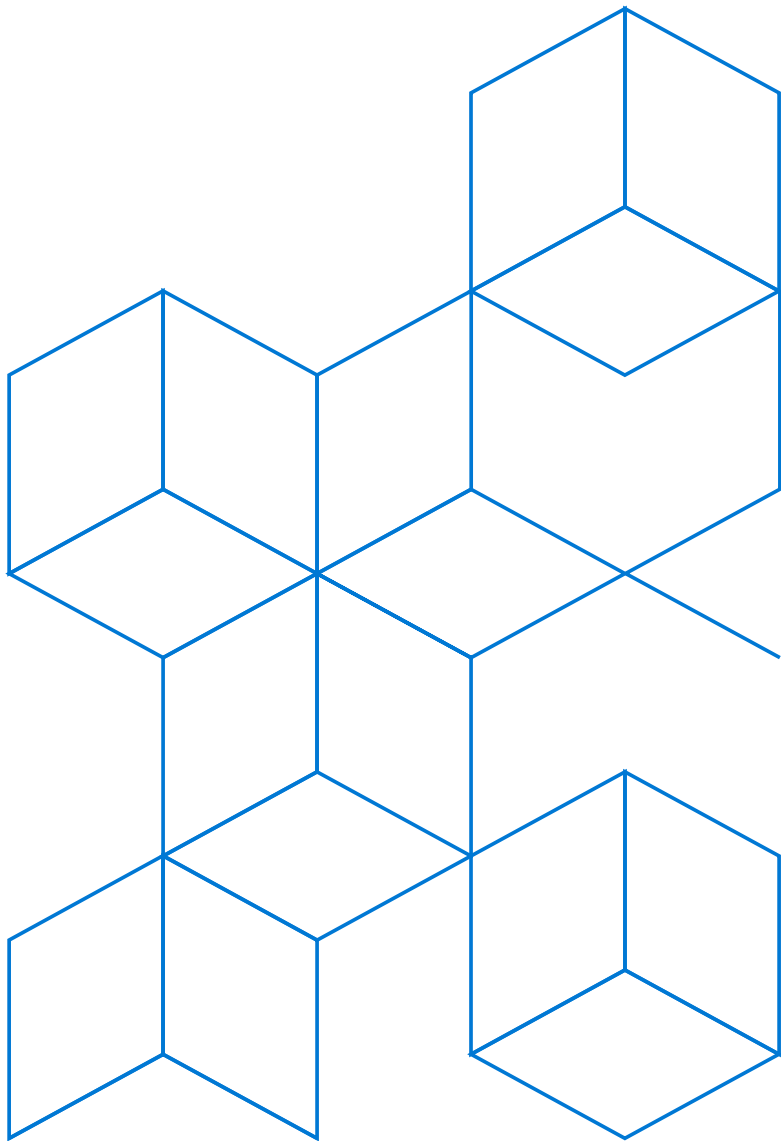


Memory Grant Feedback – Persistence and Percentile

2022 public preview in Azure SQL Database

SQL Server 2022 CTP 1.1





Memory Grant Feedback (MGF)

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 remove spills and improve concurrency for repeating queries

Batch mode in Database Compatibility 140, Row mode in Database Compatibility 150

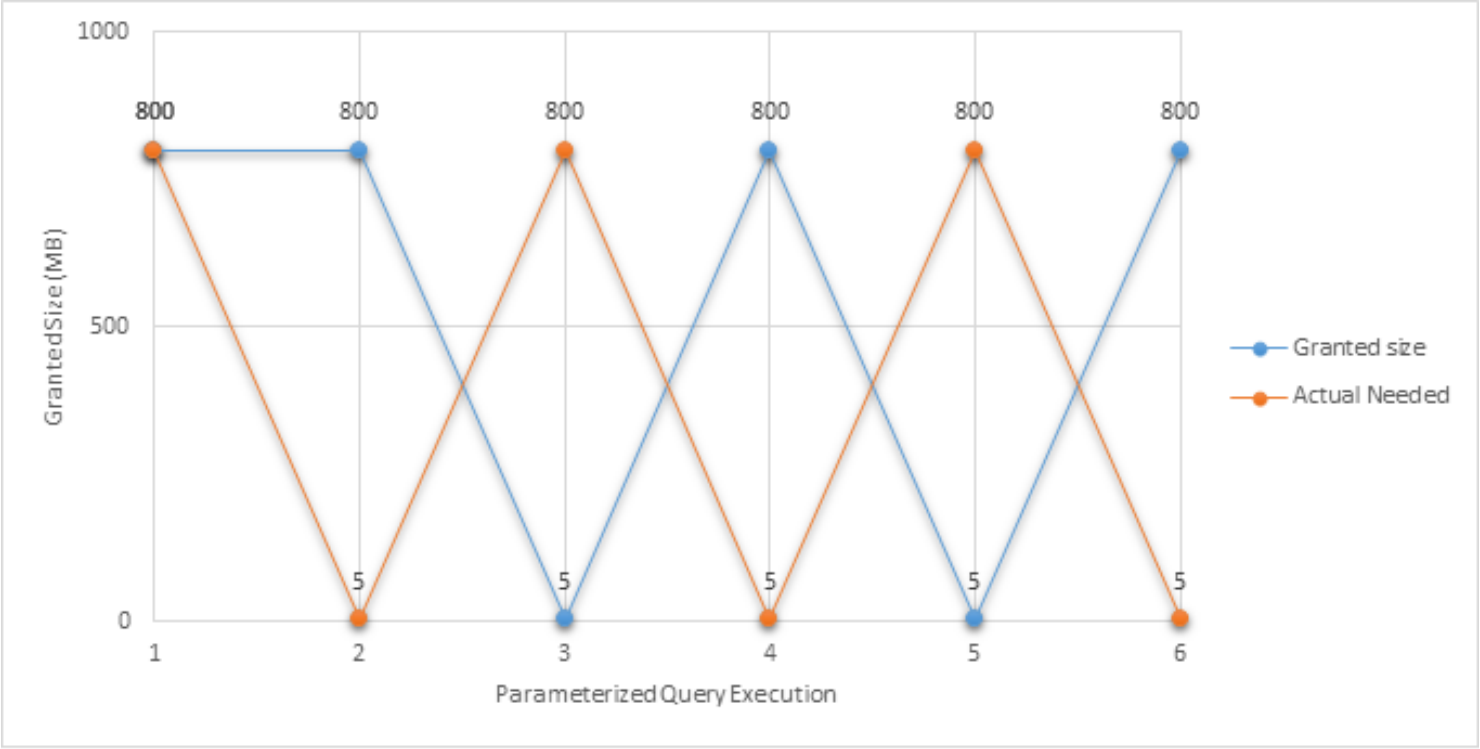
MGF improvements (1/2): Persistence

Problem: Cache Eviction	Feedback is not persisted if the plan is evicted from cache or failover Record of how to adjust memory is lost and must re-learn
Solution: Persist the feedback	Persist the memory grant feedback in the Query Store

MGF improvements (2/2): Fixing Oscillation

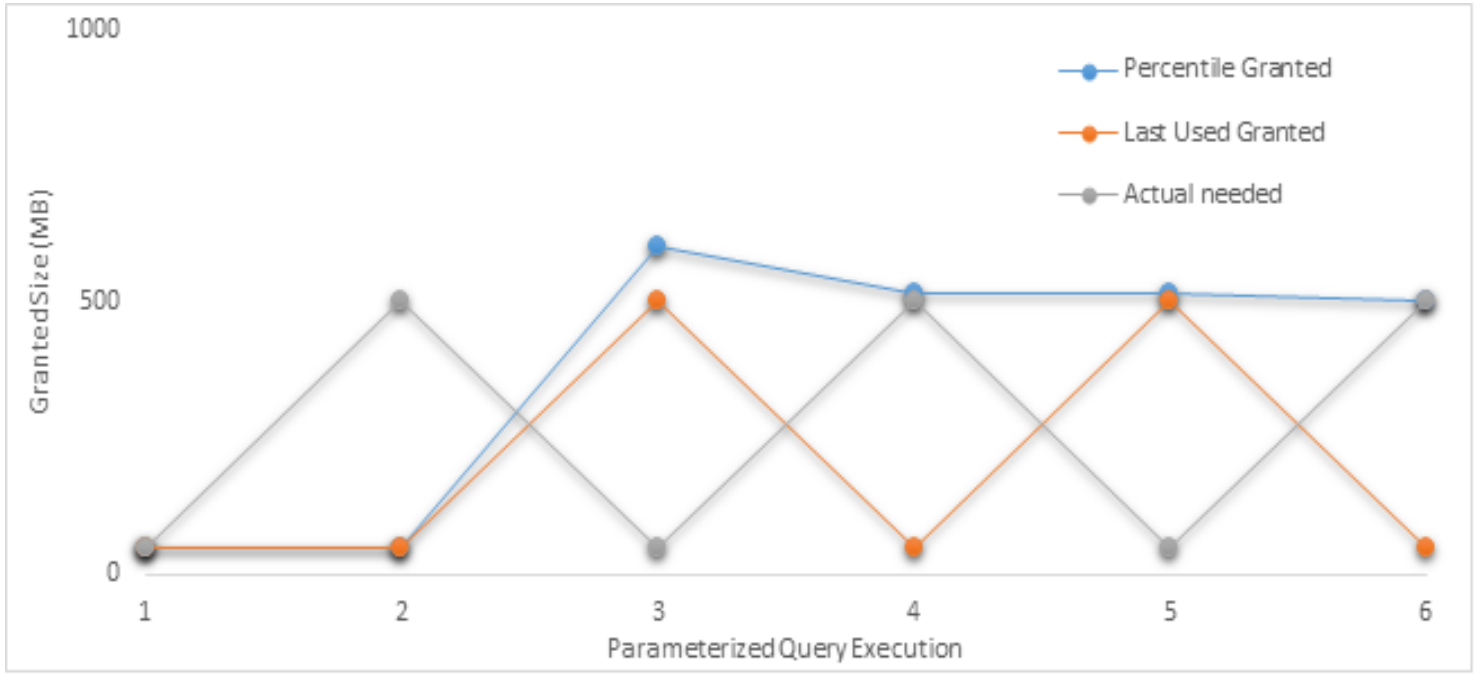
MGF adjusts memory grants based on **last execution feedback**

Feedback can be disabled if memory grant requirements keep oscillating (e.g. PSP)



MGF improvements (2/2): Fixing Oscillation

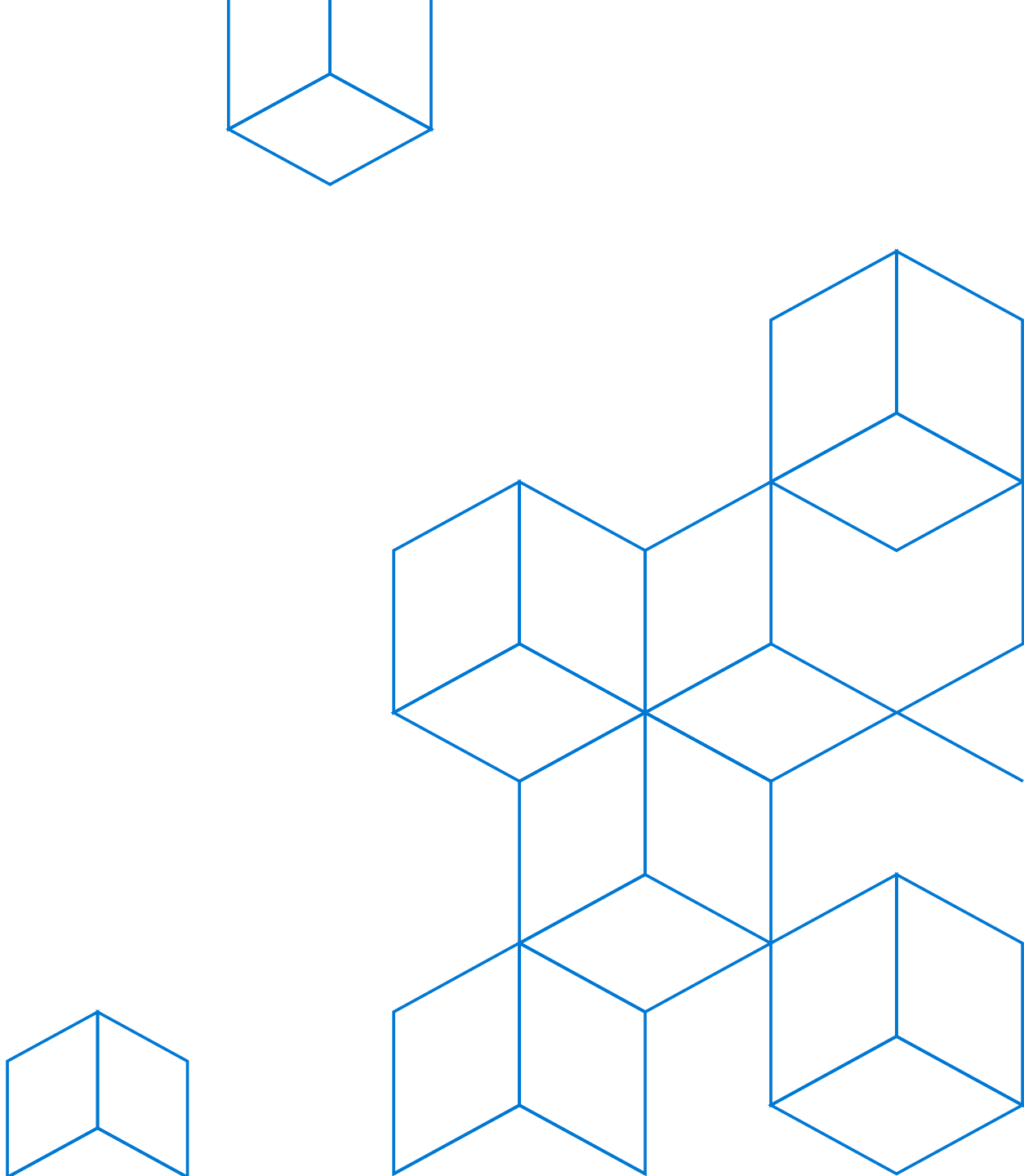
Smooth the grant size values based on past execution usage history and try to optimize for minimizing spills using a **percentile-based calculation**



CE Feedback

2022 public preview in Azure SQL Database

SQL Server 2022 CTP 1.0



Cardinality Estimation today

Cardinality estimation (CE) is the process by which the Query Optimizer derives the **estimated # of rows** for a query plan

CE models are based on **assumptions about data distribution** and expected usage. To know more about cardinality estimation, refer to https://aka.ms/sqlCE

The cardinality estimation process sometimes makes incorrect assumptions which lead to poor plan quality

One model doesn't fit all scenarios

Introducing CE Feedback

Learn which CE model assumptions are optimal over time and then apply the historically "correct" assumption.	CE Feedback will **identify** model-related assumptions and will evaluate whether they are accurate for repeating queries
	If an assumption looks incorrect, we'll test a new CE model assumption and **verify** if it helps
	If it helps, we'll **replace** the current cached plan
Addresses scenarios not yet handled by other IQP features that can cause perceived regressions:	Independence vs. Correlation assumptions (how multiple predicates over same table correlate)
	Join Containment assumptions (simple vs base)
	Row Goal

CE Feedback

Not a new “new CE”

Enabled using Database Compatibility 160

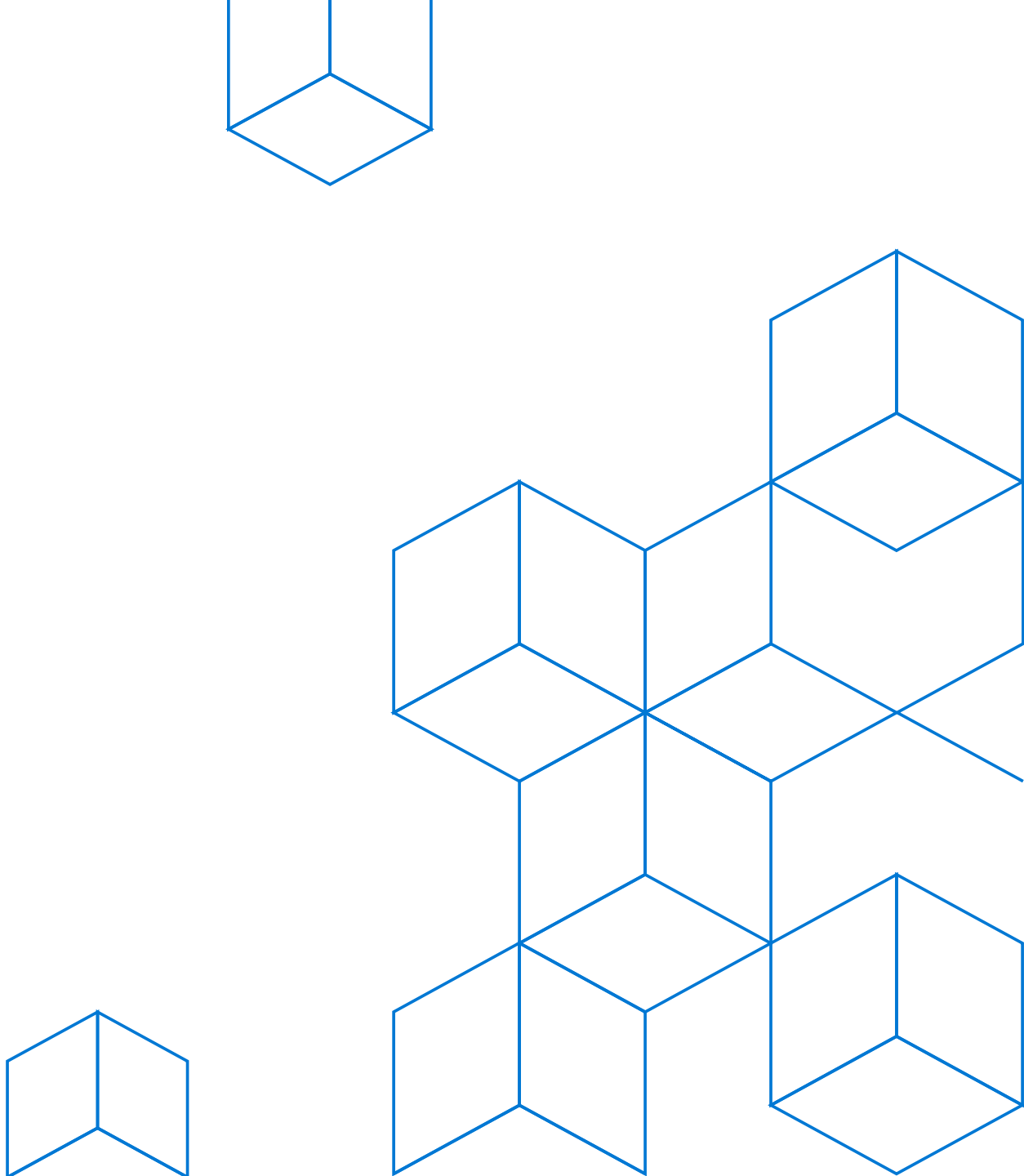
CEF will only apply feedback in the presence of *significant model estimation errors* resulting in performance drops (e.g. orders of magnitude off)

Repeating queries with cache-persistent plans

Adjusted through USE HINT query hints + hint support in Query Store.
- Will honor any hard-coded query hints if used

Only verified feedback is persisted (Query Store).
- If next execution regresses, back off.
- Cancelled query = regression

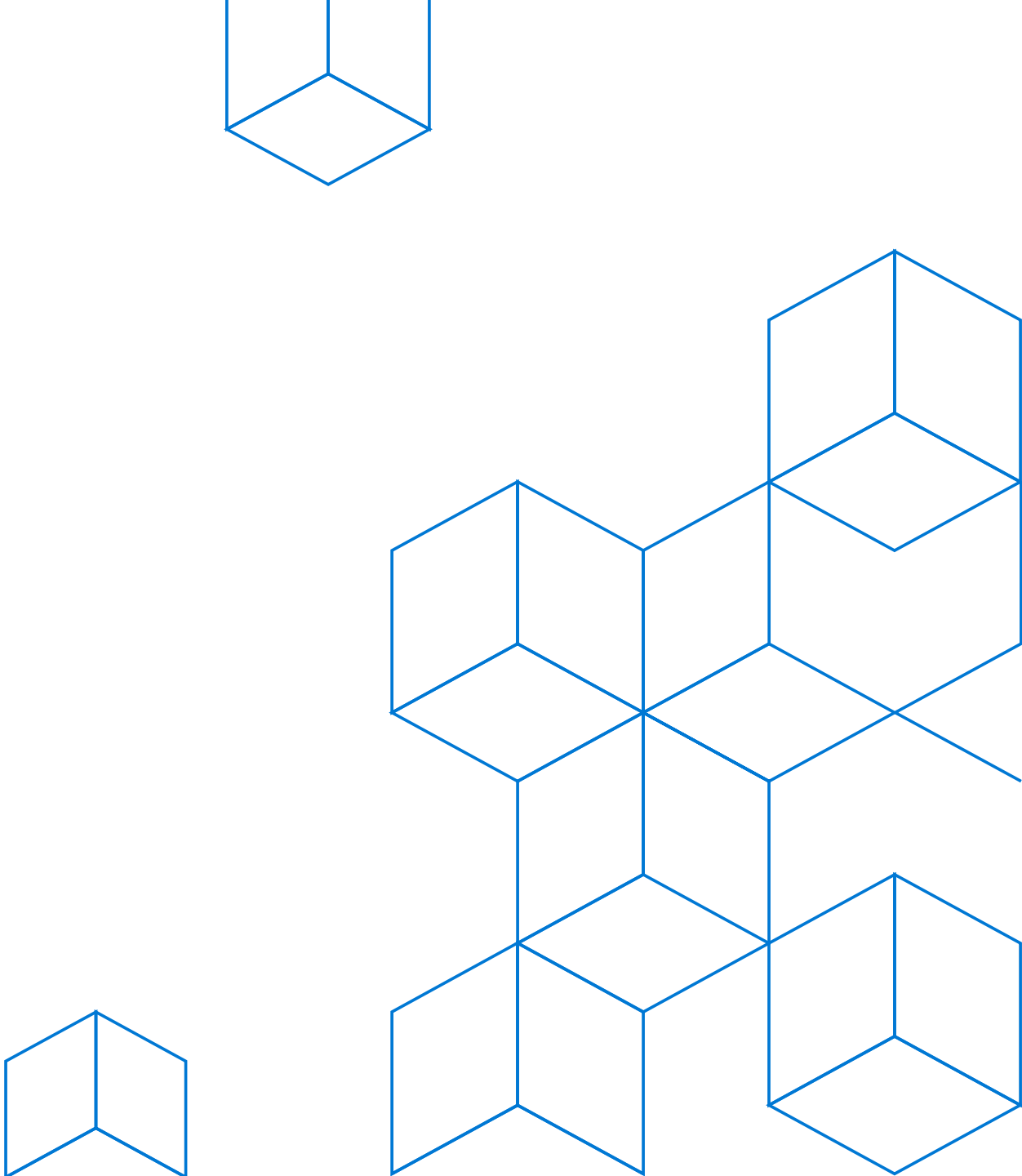
CE Feedback Demo



DOP Feedback

2022 public preview in Azure SQL Database

SQL Server 2022 CTP 1.4



Parallelism today

Parallelism is often beneficial for reporting / analytical queries or otherwise large amounts of data

OLTP-centric queries could suffer when time spent coordinating all threads outweighs the advantages of using a parallel plan

Before SQL Server 2019, default value for maxdop = 0: use all available schedulers if a query is eligible for parallelism

With SQL Server 2019, default is calculation at setup time based on available processors; 8 in Azure SQL Database

DOP Feedback

DOP Feedback will **identify** parallelism inefficiencies for repeating queries, based on CPU time, elapsed time, and waits

If parallelism usage is inefficient, **lower** from whatever is the configured DOP for next execution (min DOP = 2), and **verify** if it helps

Only verified feedback is persisted (Query Store).
- If next execution regresses, back to last good known DOP
- Cancelled query = regression

Considerations:

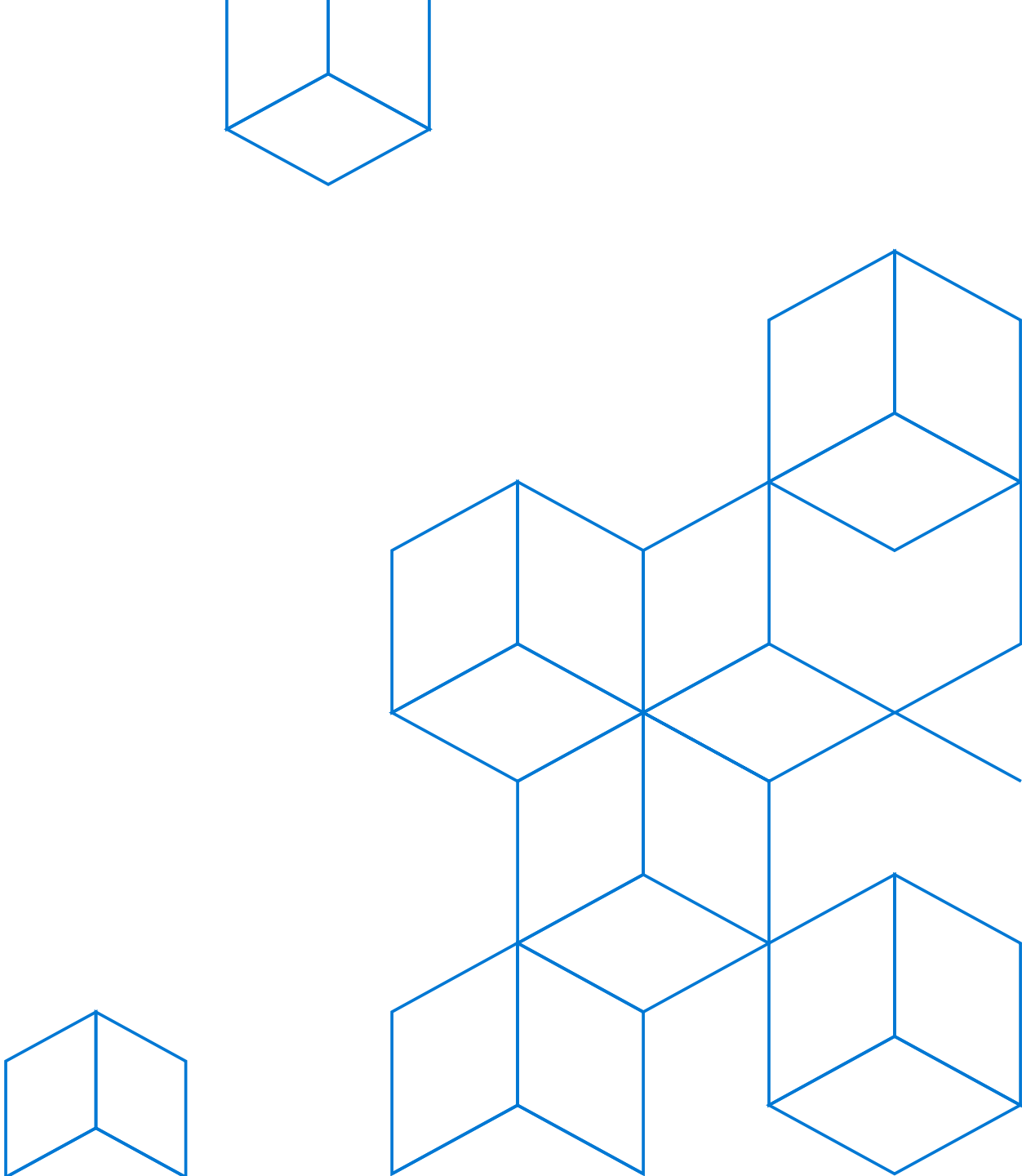
- Goal is to increase concurrency and reduce waits significantly, even if it slightly increases elapsed time
- Adjusting DOP doesn't recompile plans
- Current stable feedback is re-verified upon plan recompilation and may readjust back up, or continue down, but never above MAXDOP setting
- Will uphold MAXDOP hints as ceiling

Optimized Plan Forcing

using Query Compilation Replay

2022 public preview in Azure SQL Database

SQL Server 2022 CTP 1.3

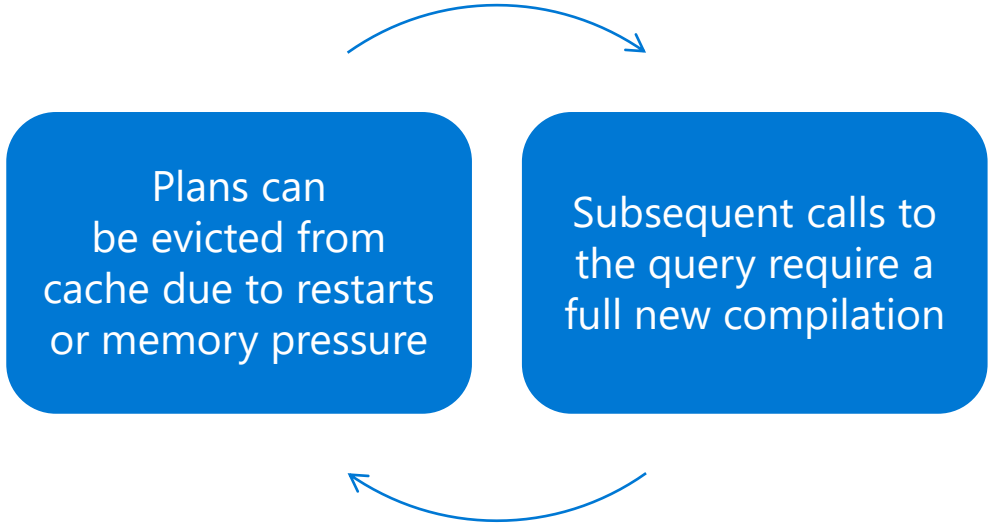


Query compilation today

Query optimization and compilation is a multi-phased process of quickly generating a “good-enough” query execution plan

Query execution time includes compilation. Can be time and resource consuming (CPU, memory)

To reduce compilation overhead for repeating queries, SQL caches query plans for re-use



Optimized Plan Forcing using query compilation replay

Compilation Replay will store a *compilation replay script* (CRS) that persists key compilation steps in Query Store (not user visible)

v1 targets previously forced plans through Query Store and Automatic Plan Correction

Uses those previously-recorded CRS to quickly reproduce and cache the original forced plan **at a fraction of the original compilation cost**

Compatible with Query Store hints and secondary replica support

Approximate QP

Approximate QP was introduced back in SQL Server 2019 and Azure SQL to enable operations on large data sets with minimal resource usage

Scenarios where responsiveness and scalability is more critical than absolute precision:

- KPI and telemetry dashboards
- Data science exploration
- Anomaly detection
- Big data analysis and visualization

Approximate QP: Approximate Percentile (CTP 1.3)

Continuous: approximate value based on a continuous distribution of the column value. Result is interpolated and might not be equal to any of the specific values in the column

`PERCENTILE_CONT` → `APPROX_PERCENTILE_CONT`

Discrete: smallest approximate column value >= designated percentile. Result is equal to a specific column value

`PERCENTILE_DISC` → `APPROX_PERCENTILE_DISC`

Implicitly covers `APPROX_MEDIAN` use case

Equivalent of `APPROX_PERCENTILE*(0.5)`

Still nondeterministic functions: may return different results each time, even with identical input values and no changes to data

Learn more

Learn more about SQL Server 2022

aka.ms/sqlserver2022

Watch a technical deep-dive on SQL Server 2022

aka.ms/sqlserver2022mechanics

Don’t miss us on Data Exposed

aka.ms/dataexposed

Microsoft

Continue your SQL journey on Microsoft Learn

The next step to expanding your technical skills

Sharpen the technical skills you need to advance your career with free courses and designated learning paths.

58% career opportunities improvement

58% of certified professionals report Azure certifications have helped them improve their career progression opportunities *

Advance your career in cloud data engineering

Build in-demand skills

Increase your productivity and efficiency

Get started at aka.ms/azuresqlfundamentals

* Nigel Frank Microsoft Azure Salary Survey (June 2020) https://www.nigelfrank.com/microsoft-azure-salary-survey/

Please rate this session

https://sqlb.it/?7187