Microsoft

SQL Server 2022 Database Engine Deep Dive - Part 1

Pedro Lopes
Principal Architect
@SQLPedro

At the end, please rate part 1 of this session

https://sqlb.it/?72014

Agenda

- SQL Server 2022 investment areas
- Improvements
  - Storage Engine
  - Availability
  - Relational Engine
  - Security

The next step for SQL Server

SQL Server 2016

Query Store

Polybase

Always Encrypted

Row Level Security

It just runs faster

Std Edition surface area

SQL Server 2017

SQL Server on Linux

Containers

Adaptive Query Processing

Automatic Tuning

Graph database

Machine Learning Services

SQL Server 2019

Data virtualization

Intelligent Query Processing

Accelerated Database Recovery

Data classification

SQL Server 2022

A hybrid, data and analytics platform built on industry-leading security, performance, and availability

Azure-enabled	Link feature in Azure SQL Managed Instance Synapse link for SQL Server Azure Purview policies
Industry leading database engine	SQL Server Ledger Large memory and concurrency scalability Multi-write replication
Object Storage Integration	Data virtualization for any data lake Object storage backup and restore
Intelligent database	Query store on by default with replica support Query store hints Intelligent Query Processing NextGen
Extending T-SQL	JSON data Enhanced T-SQL surface area Time series support

Storage Engine Improvements

tempdb Performance Critical for Scalability

- tempdb is one of four system databases
- SQL Server performance and scalability is often centered on tempdb health
- Important to optimize tempdb performance and apply best practices
- Important to track and address tempdb bottlenecks

tempdb Performance Critical for Scalability

SQL Server 2016 Improvements

- Setup experience has improved
- Trace Flag 1117 and 1118 are no longer required

SQL Server 2019 Improvements

- Memory-optimized tempdb metadata
- Concurrent PFS updates

tempdb Performance Critical for Scalability

SQL Server 2016 Improvements

- Setup experience has improved
- Trace Flag 1117 and 1118 are no longer required

SQL Server 2019 Improvements

- Memory-optimized tempdb metadata
- Concurrent PFS updates

SQL Server 2022 Improvements

- System page latch concurrency enhancements (GAM/SGAM)

Shrink Database with Low Priority

- Customers often need to reclaim data space
- Common for hosted environments (new database per customer)
- Shrink Database operations can cause concurrency issues
- Shrink Database WLP addresses this problem by waiting with less restrictive locking
- Similar to `ALTER INDEX WAIT_AT_LOW_PRIORITY`

```
DBCC SHRINKDATABASE (2, 20, NOTRUNCATE)
WITH WAIT_AT_LOW_PRIORITY (ABORT_AFTER_WAIT = SELF));
```

XML Compression

- XML data type is commonly used to store unstructured data
- Data compression only applies to in-row scenarios (row, page)
- XML Compression will compress the XML data type in Azure SQL and SQL Server 2022
- XML Compression can be specified during CREATE and ALTER of TABLE and INDEX statements
- `sp_estimate_data_compression_savings` will be expanded to estimate XML savings

```
ALTER TABLE Sales.StoreBIGXMLCopy REBUILD PARTITION = ALL
WITH (DATA_COMPRESSION = PAGE, XML_COMPRESSION = ON);
```

XML Compression Demo

Current Buffer Pool Scan Operations

- Operations that scan the buffer pool can be slow, especially on large memory machines, such as:
  - Creating a new databases
  - File drop operations
  - Backup/restore operations
  - Always On failover events
  - DBCC CHECKDB
  - Log restore operations
  - Internal operations (e.g., checkpoint)

Buffer Pool Parallel Scan

Benefits both small and large databases on large-memory machines

Customers running mission critical OLTP, and data warehouse environments will see the most benefit

Improvement adds diagnostics and telemetry for supportability and insights:

- Long Buffer Pool scans will be visible by the ERRORLOG
- Extended events will capture scan start/complete, errors, FlushCache, etc.

Buffer Pool Parallel Scan Demo

Buffer Pool Parallel Scan Performance Results

Setup:

HP DL580

2TB RAM

BP warmed up with 1TB of data (~140M buffers).

Scenario	Serial Scan	Parallel (2 tasks)	Parallel (16 tasks)	Improvement
FlushCache	7	3.3	0.5	15x faster with 16 parallel tasks.
ShutdownDB (Small)	6.2	3.3	0.5	
ShutdownDB (Large)	180	120	32	6x faster.
DBCC Check (Small)	30	15	2	15x faster. DBCC Check does 4 scans.
DBCC Check (Large)	55	30	5.4	10x faster DBCC Check does 4 scans.
CreateDB	7	3.4	0.5	Does FlushCache
BackupDB	7	3.4	0.5	Does FlushCache
RestoreLog	17	7.8	1.1	Does 2 FlushCache
DropCleanBuffers	100	51	29	4x faster

Units are elapsed time in seconds

Other Storage Engine improvements

Accelerated Database Recovery: single thread used for cleanup operations per database (from per instance only); overall performance, scalability, and better telemetry for troubleshooting

In Memory OLTP Enhancements: manual cleanup sproc; overall performance, scalability, and better telemetry for troubleshooting

Resumable ADD CONSTRAINT: powerful feature especially for large, mission critical environments

Availability Improvements

Intel QAT backup compression

- Leverages Intel® QuickAssist Technology (Intel® QAT) for improved backup performance
- Free-up processor cycles by offloading backup compression
- Reduce demands on processor
- Dramatically improves backup speed

```
--Enable at the server level
ALTER SERVER CONFIGURATION SET
HARWARE_OFFLOAD ON (QAT);
```

Intel QAT backup compression

- Leverages Intel® QuickAssist Technology (Intel® QAT) for improved backup performance
- Free-up processor cycles by offloading backup compression
- Reduce demands on processor
- Dramatically improves backup speed

```
--Backup database with compression level
BACKUP DATABASE testdb
FROM DISK='F:\SQLBACKUPS\testdb.bak'
WITH COMPRESSION
(ALGORITHM = 'QAT-DEFLATE');
```

Intel QAT backup compression

- Leverages Intel® QuickAssist Technology (Intel® QAT) for improved backup performance
- Free-up processor cycles by offloading backup compression
- Reduce demands on processor
- Dramatically improves backup speed

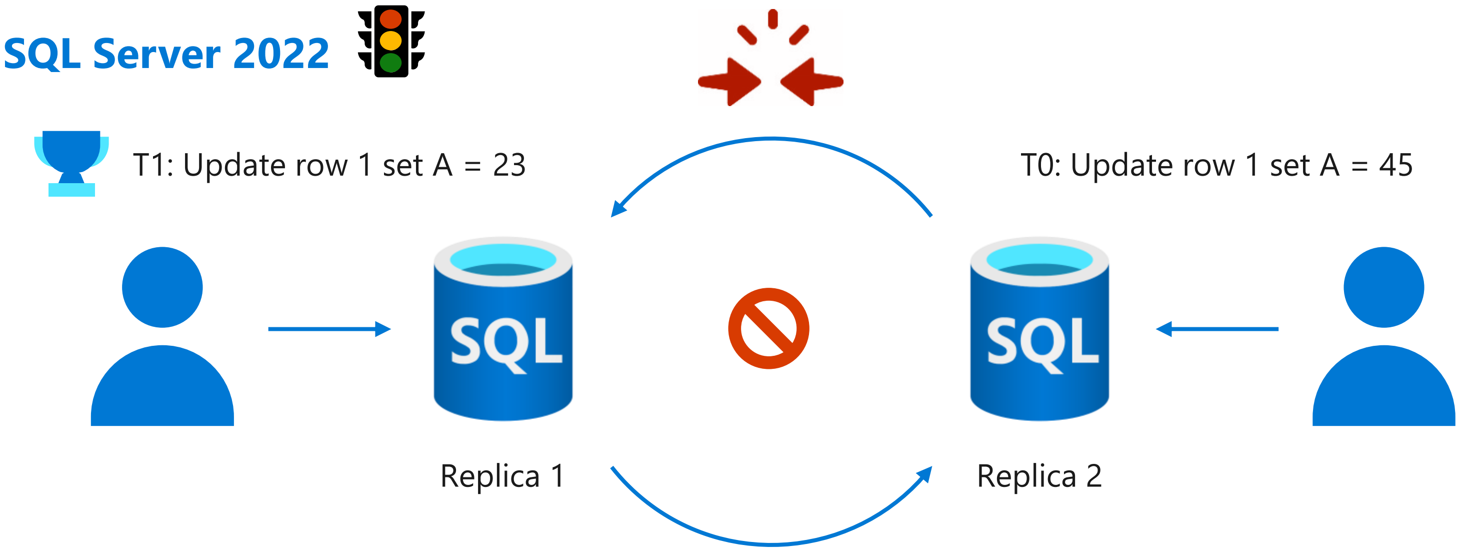
```
--Backup database with compression level
BACKUP DATABASE testdb
FROM DISK='F:\SQLBACKUPS\testdb.bak'
WITH COMPRESSION
(ALGORITHM = 'MS-XPRESS');
```

Multi-write replication

Multi-master writes for users across multiple locations

Challenge: I need automatic and logical conflict detection for multi-write replication

- Globally distributed database replicas for geo-localized writes
- Enhanced conflict detection for inserts and updates with Last Writer Wins (LWW) capabilities
- Ensures the last update is persisted across all replicas based on the UTC time of the operation.



Final Value?

A = 45 without LWW (highest replica wins regardless of timing)
A= 23 with LWW (last update wins regardless of replica ID)

Other Availability improvements

Cross platform improved support for snapshot backups: T-SQL support to freeze/thaw I/O for a database and backup metadata to improve coordinate snapshot backups (without VDI or VSS)

Parallel redo enhancements: faster database start up = faster failovers and reduced lag for Always On AGs

Improve DAG throughput: use multiple parallel data connections to speed up replication across DAG

Contained Always On Availability Groups: AG containing its own system databases such as MSDB, and can contain user logins, certificates and other user artifacts which were replicated to multiple AG replicas

Relational Engine Improvements

Query Store ON by default

For new databases only

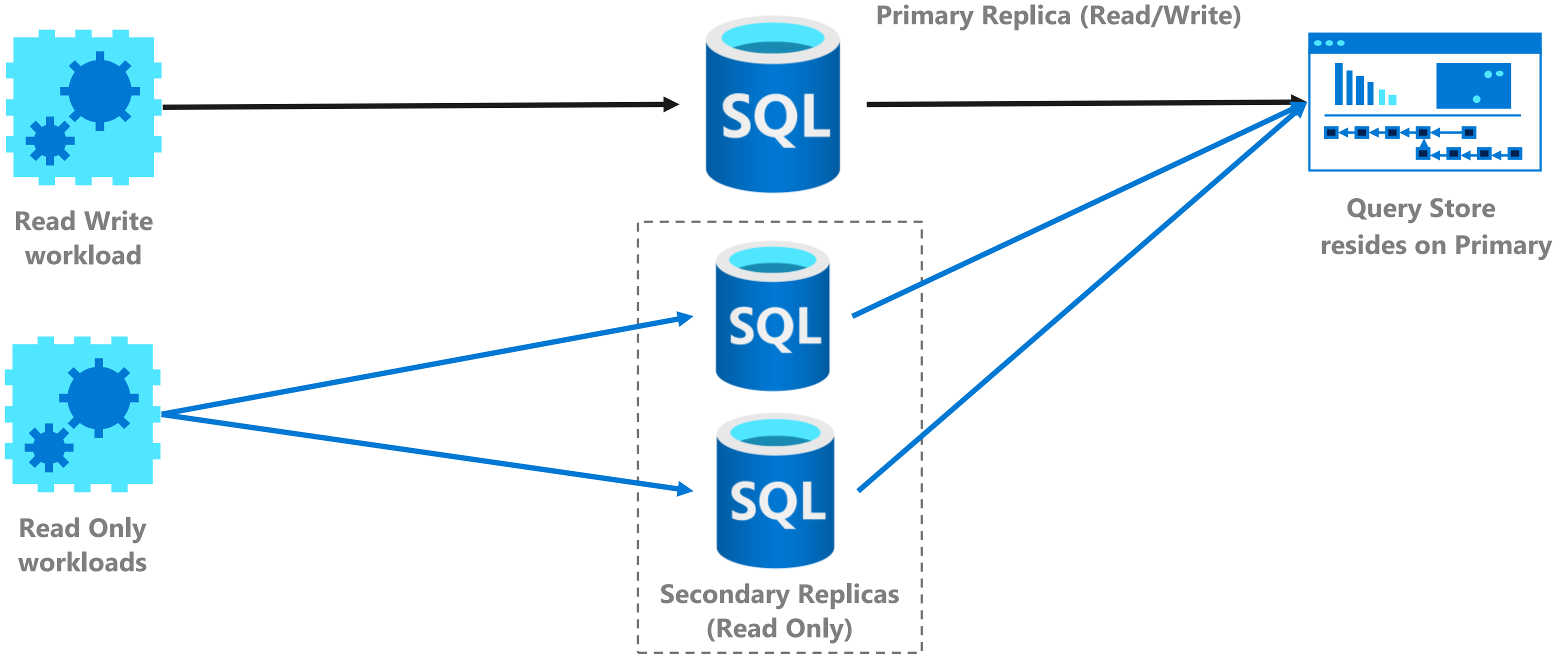
- SQL never changes database defaults when restoring/attaching to higher version engine.

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 for Secondary Replicas

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


Primary Replica (Read/Write)
SQL
Read Write workload
Query Store resides on Primary
SQL
SQL
Read Only workloads
Secondary Replicas (Read Only)

- Query data will be visible by role type or secondary 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

Third 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

Maintenance

`sp_query_store_flush_db` will apply only to the replicas on which it was executed

The following apply per replica set:

- `sp_query_store_remove_plan 2`
- `sp_query_store_remove_query 46006`
- `sp_query_store_reset_exec_stats 2`

If executed for a **secondary role**, action will be taken for all machines with that role
If executed from a **named secondary**, action will be taken for that secondary only

SQL Server 2022 Database Engine Deep Dive - Part 2

Pedro Lopes
Principal Architect
@SQLPedro

At the end, please rate part 2 of this session

https://sqlb.it/?72014

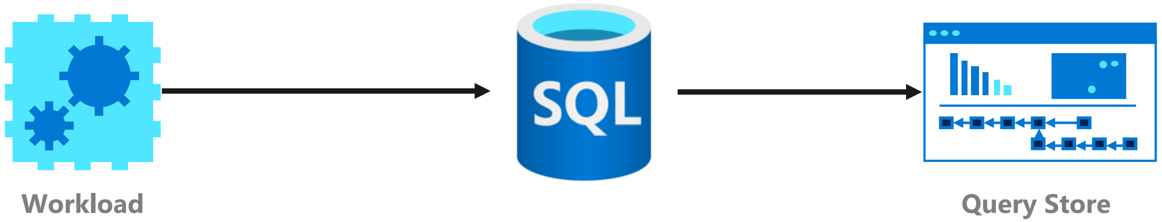
Agenda

- SQL Server 2022 investment areas
- Improvements
  - Storage Engine
  - Availability
  - Relational Engine
  - Security

Query Store hints

This feature provides a simple method for shaping query plans and behavior *without* changing application code

Leverages Query Store (on-by-default in Azure SQL Database) and greatly simplifies the overall performance tuning experience



Example use cases for Query Store hints

Recompile a query on each execution

Cap the memory grant size for a bulk operation

Limit maximum degree of parallelism for specific queries

Use a Hash join instead of a Nested Loops join

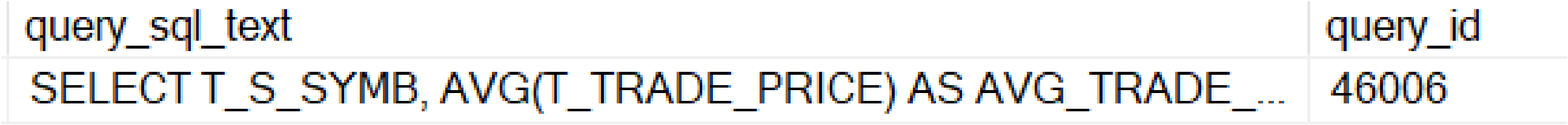
Use compatibility level 120 while keeping everything else 150

Disable optimizer rowgoal for a SELECT TOP n query

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:

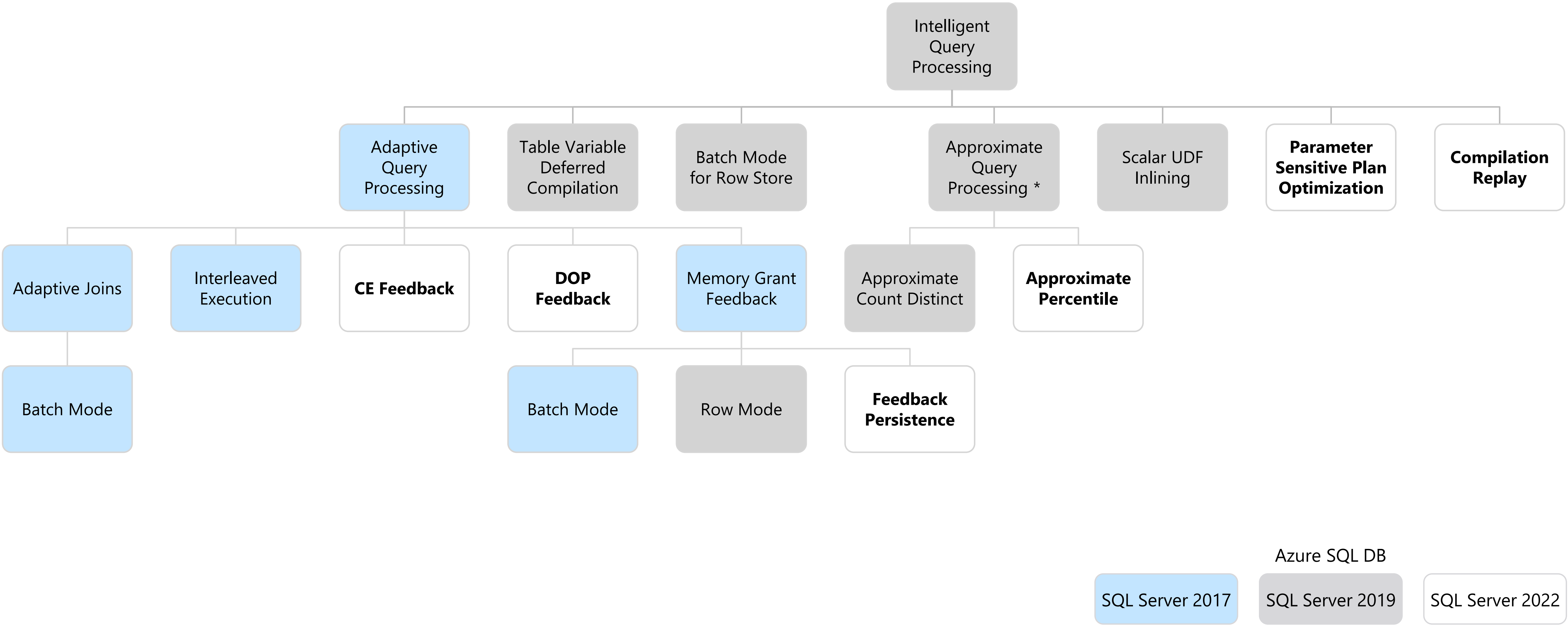
```
-- Setting a single query hint
EXEC sp_query_store_set_hints 46006, N'OPTION(MAXDOP 1)’;


-- 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;
```

Intelligent Query Processing

The Intelligent Query Processing feature family



Parameter-sensitive Plan Optimization

2022 public preview in Azure SQL Database

SQL Server 2022 CTP 1.0

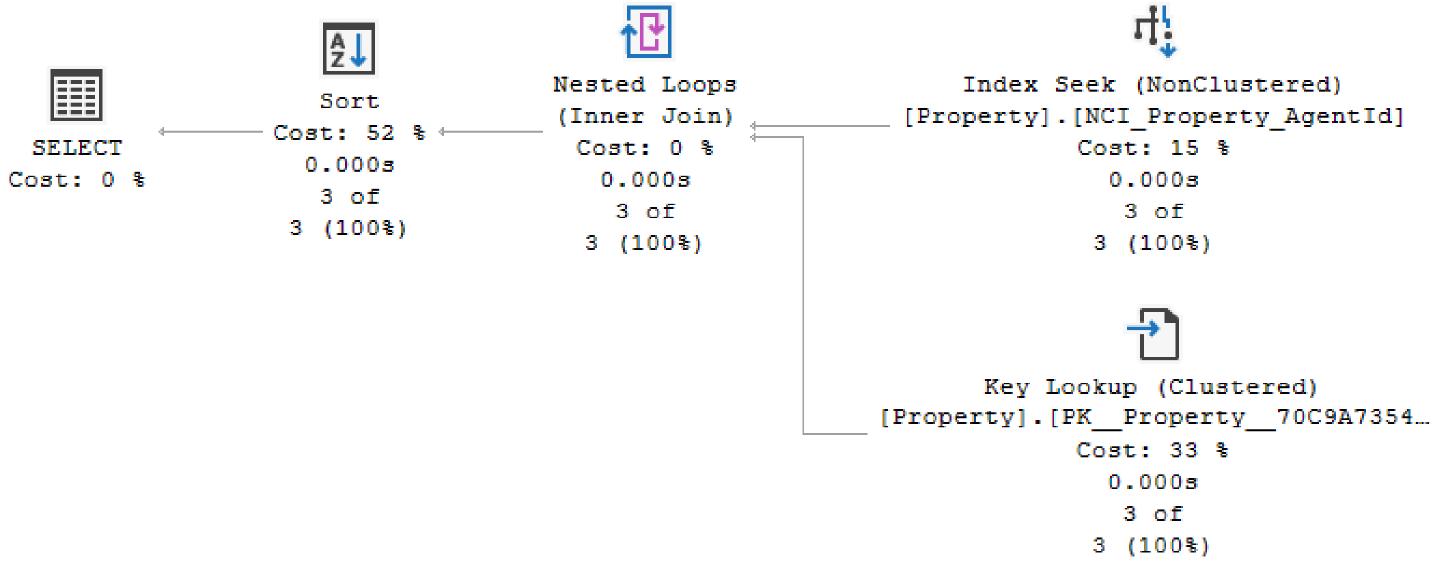
Parameter-sensitive plan problem

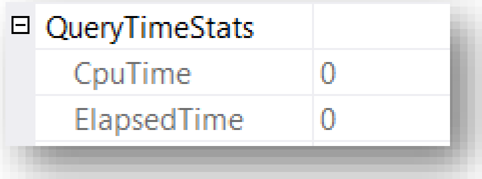
Parameter-sensitive Plan (PSP), a.k.a. Parameter-sniffing problem refers to a scenario where a **single** cached plan for a parameterized query is **not optimal for all** possible incoming parameter values

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

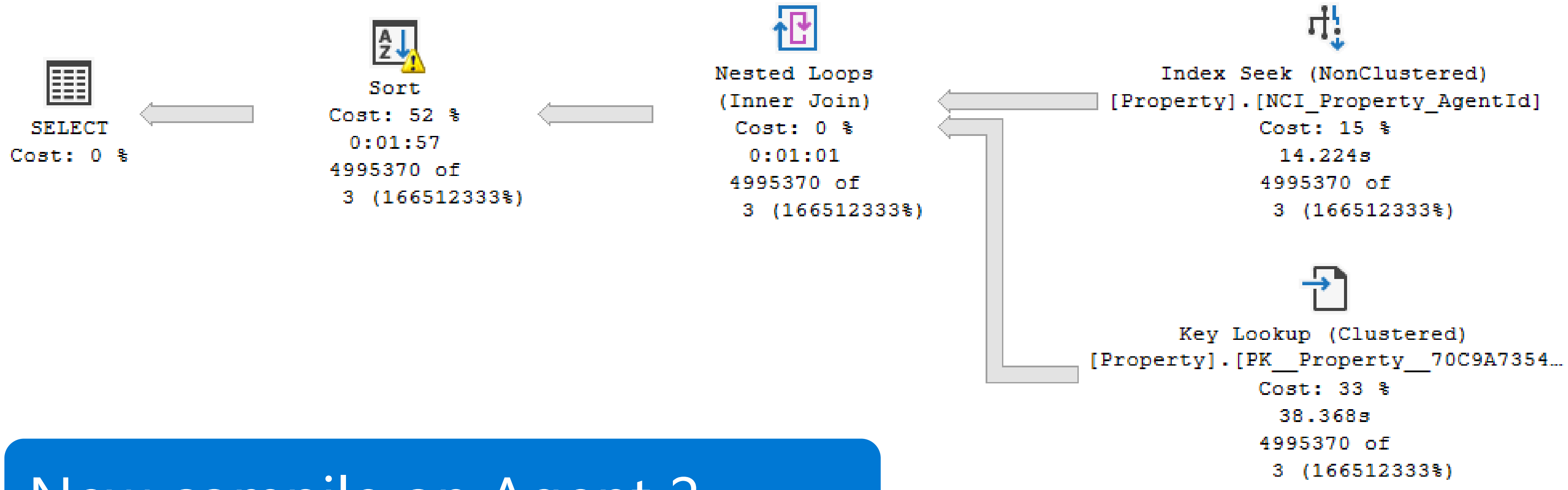




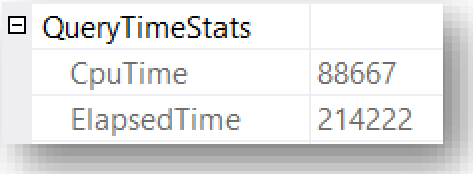
⊟ QueryTimeStats	
CpuTime	0
ElapsedTime	0

PSP today

Using cached plan for Agent 2

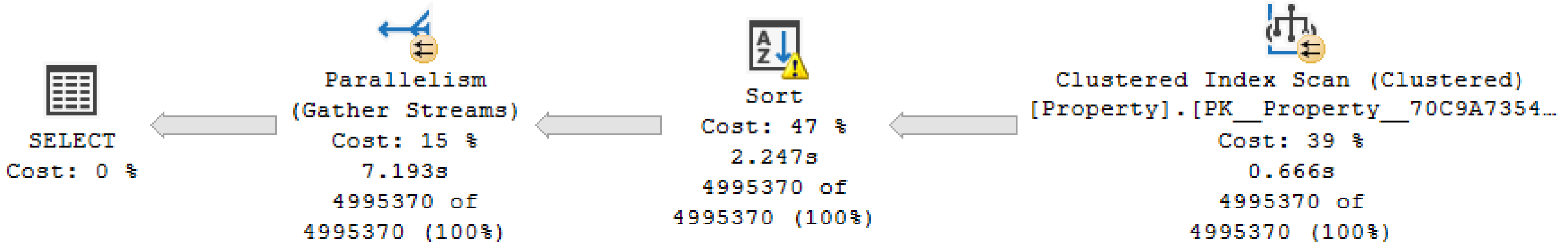


```
SELECT
Cost: 0 %

Sort
Cost: 52 %
0:01:57
4995370 of
 3 (166512333%)

Nested Loops
(Inner Join)
Cost: 0 %
0:01:01
4995370 of
 3 (166512333%)

Index Seek (NonClustered)
[Property].[NCI_Property_AgentId]
Cost: 15 %
14.224s
4995370 of
 3 (166512333%)

Key Lookup (Clustered)
[Property].[PK__Property__70C9A7354…
Cost: 33 %
38.368s
4995370 of
 3 (166512333%)
```

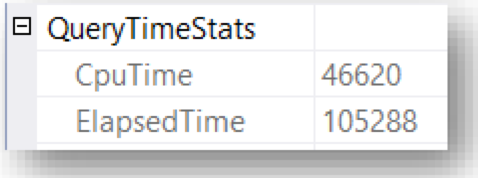


⊟ QueryTimeStats	
CpuTime	88667
ElapsedTime	214222

New compile on Agent 2



```
SELECT
Cost: 0 %

Parallelism
(Gather Streams)
Cost: 15 %
7.193s
4995370 of
4995370 (100%)

Sort
Cost: 47 %
2.247s
4995370 of
4995370 (100%)

Clustered Index Scan (Clustered)
[Property].[PK__Property__70C9A7354…
Cost: 39 %
0.666s
4995370 of
4995370 (100%)
```



⊟ 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 benefit
- Compatible with Query Store plan forcing

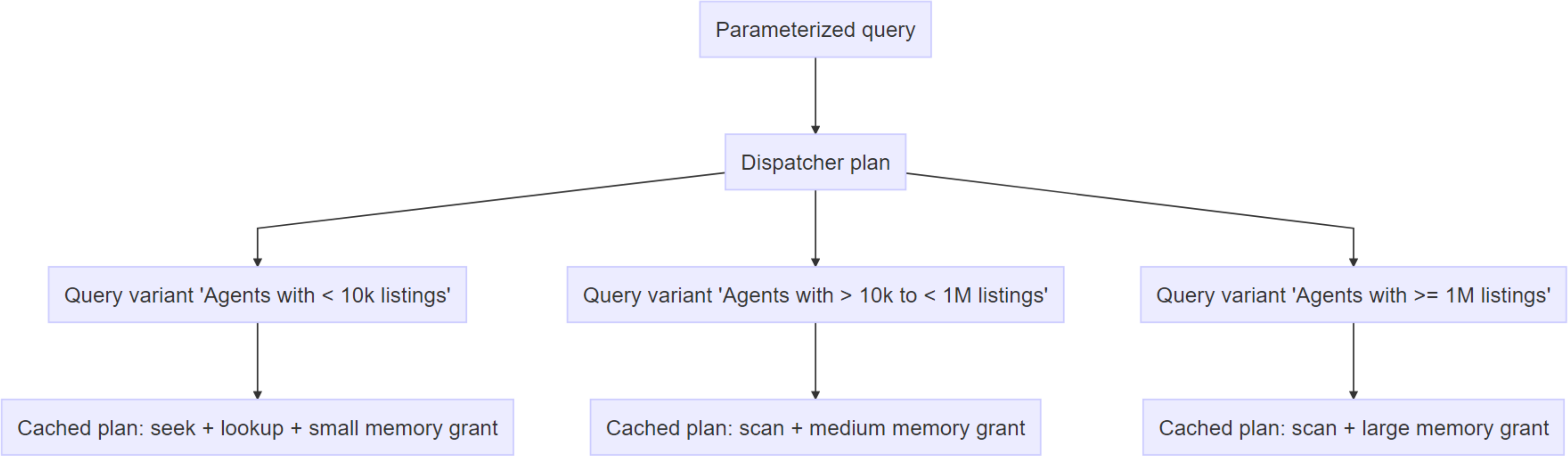
Predicate selection

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)



Parameterized query
Dispatcher plan
Query variant 'Agents with < 10k listings'
Query variant 'Agents with > 10k to < 1M listings'
Query variant 'Agents with >= 1M listings'
Cached plan: seek + lookup + small memory grant
Cached plan: scan + medium memory grant
Cached plan: scan + large memory grant

Dispatchers and Query Variants

A **dispatcher plan** contains logic, called a **dispatcher expression**, which then maps to **query variants** based on predicate cardinality range boundary values

The dispatcher plan is built during initial optimization along with 1st variant, and determines the available scope for the last evaluated set of “at risk” predicates

Dispatcher plans are also automatically rebuilt if there are significant data distribution changes (for example resulting in different predicates being evaluated)

Query Variants

Each query variant will have its own query execution plan and is differentiated in Query Store

Query variants will have the same **query hash value** so customers can still determine the **aggregate resource usage** for queries that differ only by input values

Plans for a query variant in the same dispatcher will independently recompile as needed, the same way as is without the feature

PSP Optimization Demo

Force and Unforce plan

Same force and unforce plan procedures:

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

Considerations:

- If a variant is forced, dispatcher is not forced
- If a dispatcher is forced, this means only variants from that dispatcher are considered eligible for use
  - Previously forced variants in the same dispatcher are forced again
  - Previously forced variants from other dispatchers will become inactive, but retain “forced” status until such time as their dispatcher is forced again

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
	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)

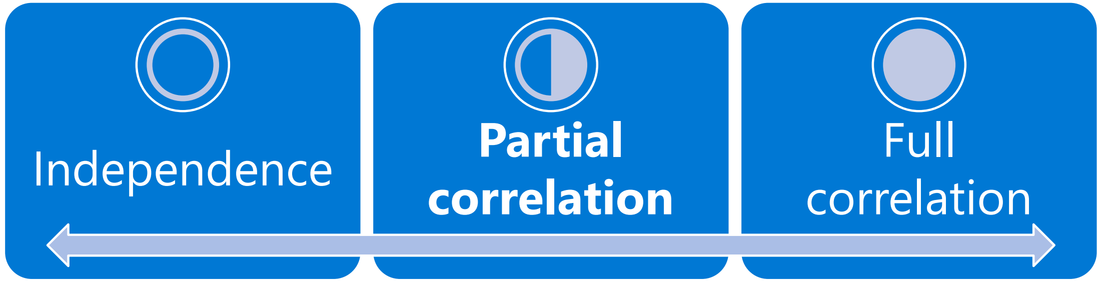
Repeating queries with cache-persistent plans

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.
- If next execution regresses, back off.
- Cancelled query = regression

CE Feedback – Correlation Analysis

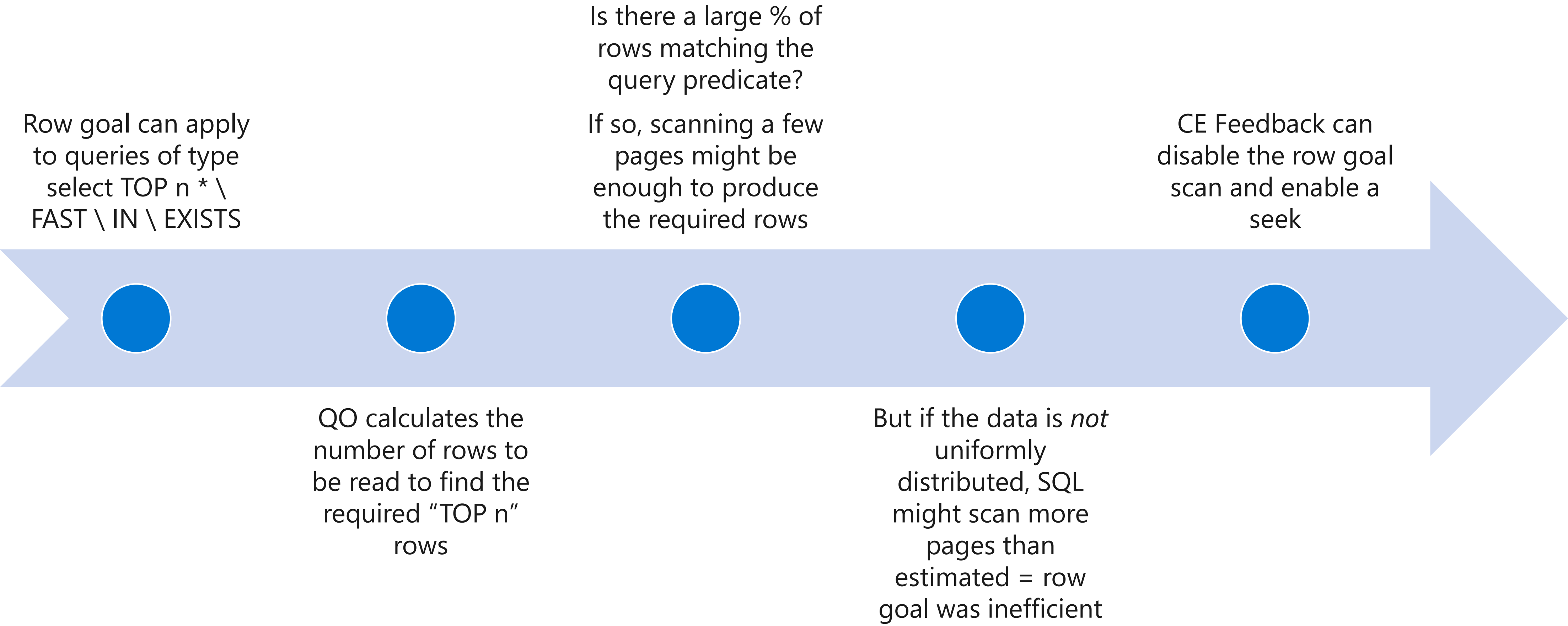
Correlation estimates can be **fully independent**, **partially independent** or **fully correlated**



When correlation is used, CEF will attempt to move the correlation to the correct direction one step at a time – based on the underestimate or overestimate

```sql
SELECT AddressID,
      AddressLine1,
      AddressLine2
FROM Person.Address
WHERE StateProvinceID = 9
      AND City = 'Dallas';
```

CE Feedback – Row goal Analysis



Row goal can apply to queries of type select TOP n * \ FAST \ IN \ EXISTS

QO calculates the number of rows to be read to find the required "TOP n" rows

Is there a large % of rows matching the query predicate?

If so, scanning a few pages might be enough to produce the required rows

But if the data is *not* uniformly distributed, SQL might scan more pages than estimated = row goal was inefficient

CE Feedback can disable the row goal scan and enable a seek

```
SELECT TOP 1 t1.*
FROM Sales.SalesOrderHeader AS t1
INNER JOIN Sales.SalesOrderDetail AS t2 ON t1.SalesOrderID = t2.SalesOrderID
```

CE Feedback – Containment Types

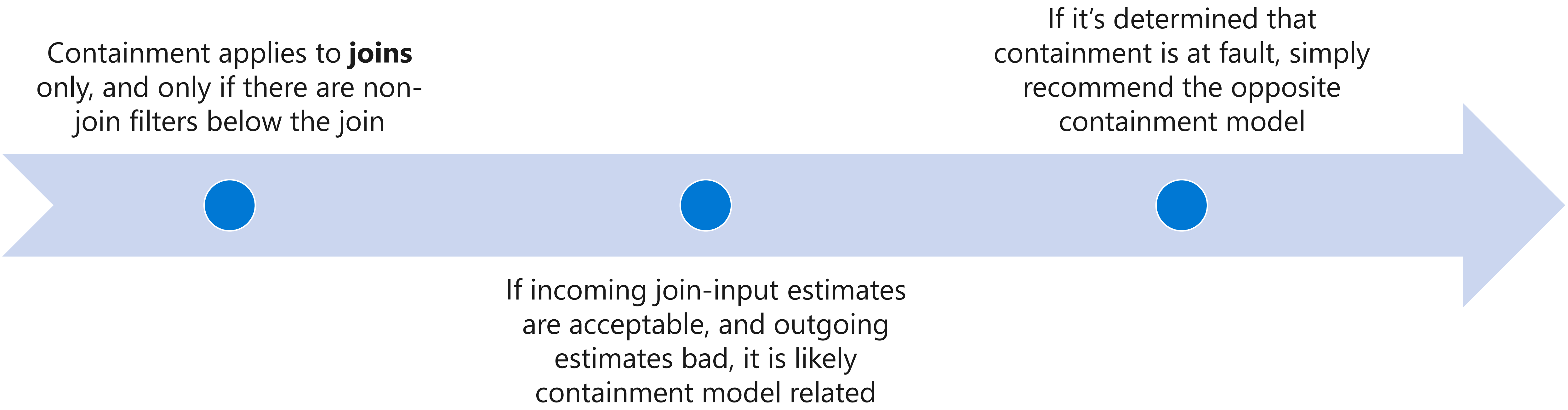
Simple containment assumes that join predicates are fully correlated.

- Estimate join selectivity based on the input relations only – using the already scaled-up or down estimates of any non-join filter predicates on the joined tables
- Summary: first estimate filters and then join

Base containment assumes no correlation between join predicates and downstream filters (including downstream joins).

- Estimate join selectivity based on the base table properties before applying the selectivity of non-join filters
- Summary: first estimate join and then filters

CE Feedback – Containment Analysis



Containment applies to **joins** only, and only if there are non-join filters below the join

If incoming join-input estimates are acceptable, and outgoing estimates bad, it is likely containment model related

If it’s determined that containment is at fault, simply recommend the opposite containment model

```
SELECT *
FROM FactCurrencyRate AS f
INNER JOIN DimDate AS d ON f.DateKey = d.DateKey
WHERE d.MonthNumberOfYear = 7 AND f.CurrencyKey = 3 AND f.AverageRate > 1
```

CE Feedback Demo

Auto update stats Wait Low Priority

SQL updates statistics automatically as needed to reflect changes in the underlying data distribution

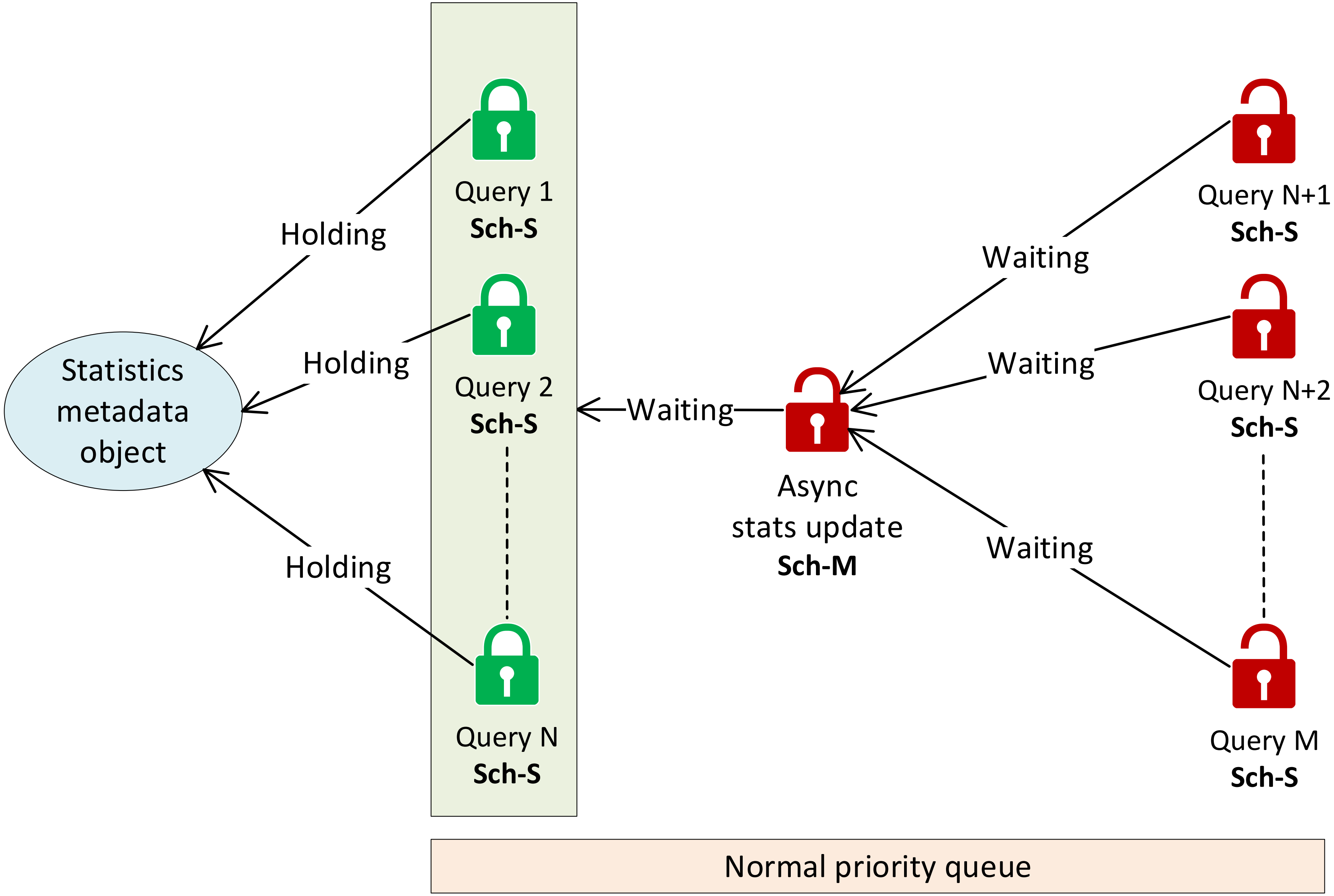
This helps the Query Optimizer generate better plans

However, extra time added to some short query executions due to stats update may be an overhead: that’s why we have AUTO_UPDATE_STATISTICS_ASYNC

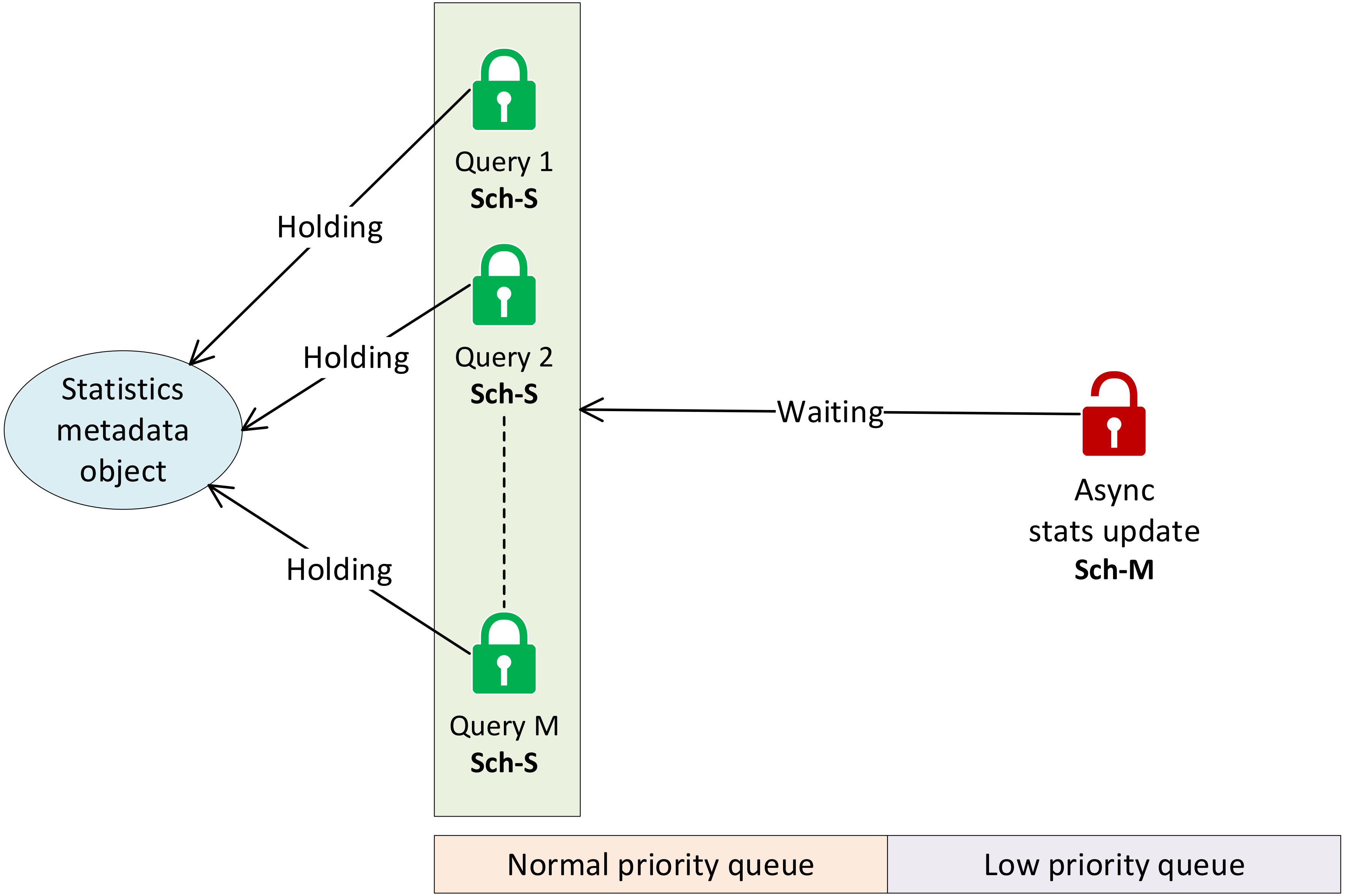
Stale statistics are then updated on a background thread asynchronously: may still generate blocking

SQL Server 2022 allows async stats wait for the Sch-M lock to be low priority (DB scoped config)

Normal async auto update stats



Async auto update stats Wait Low Priority



Other Relational Engine improvements

Allow column drop with existing stats: opt-in to stats that can get dropped if column is dropped

WINDOW clause: allows specifying window components using a named window to use in OVER clauses directly (DB compat 160 only)

```
SELECT SUM(OrderQty)
OVER(PARTITION BY SalesOrderID)
AS TotalOrderQty
FROM Sales.SalesOrderDetail;
```

→

```
SELECT SUM(OrderQty) OVER
WinSales AS TotalOrderQty
FROM Sales.SalesOrderDetail
WINDOW WinSales AS (PARTITION
BY SalesOrderID);
```

Other Relational Engine improvements

GREATEST and LEAST (local var, columns, expressions)

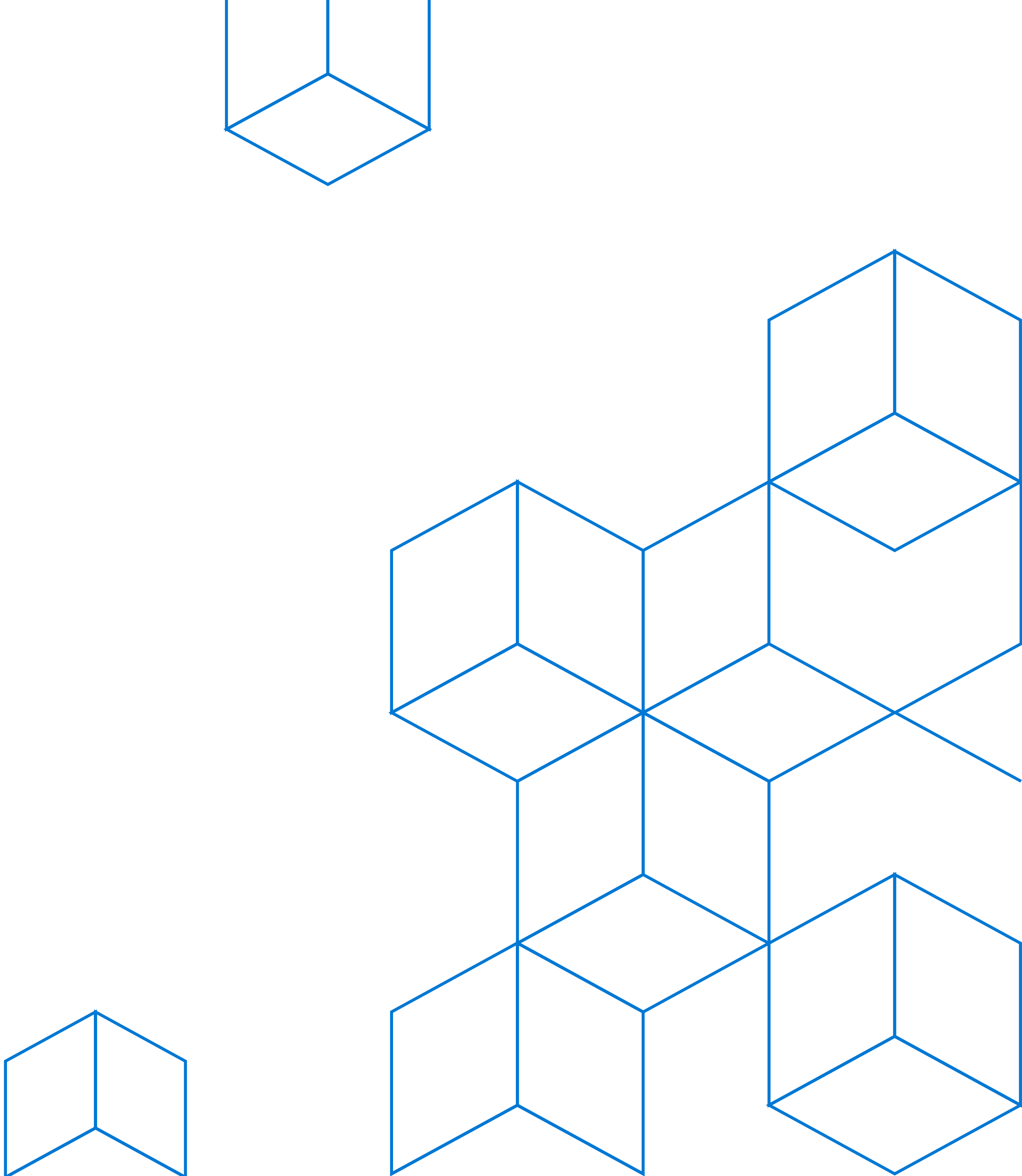
```
SELECT GREATEST ( '6.62', 3.1415, N'7' ) AS GreatestVal;

SELECT LEAST ( '6.62', 3.1415, N'7' ) AS LeastVal;
GO
```

STRING_SPLIT Ordinal (parameter and new column added for programmatic handling)

```
SELECT * FROM STRING_SPLIT('B-I-T-S', '-', 1);
```

value	ordinal
B	1
I	2
T	3
S	4

Security Improvements

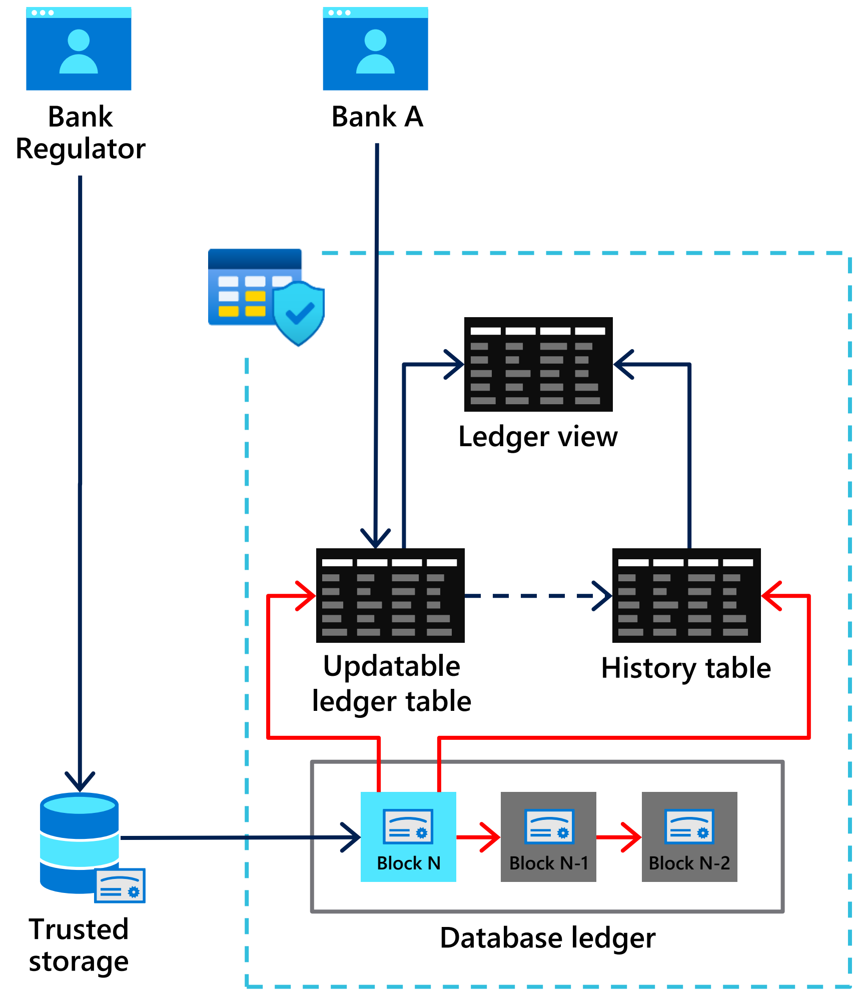


SQL Server ledger

Tamper-evidence track record of data over time

Challenge: I want the power of blockchain in a centralized system like SQL Server

- Use a cryptographically hashed ledger to protect data from tampering and by malicious actors
- Built into SQL Server with T-SQL
- Establish digital trust in a centralized system using blockchain technology.
- Attest to other parties that data integrity has not been compromised



Azure AD Authentication

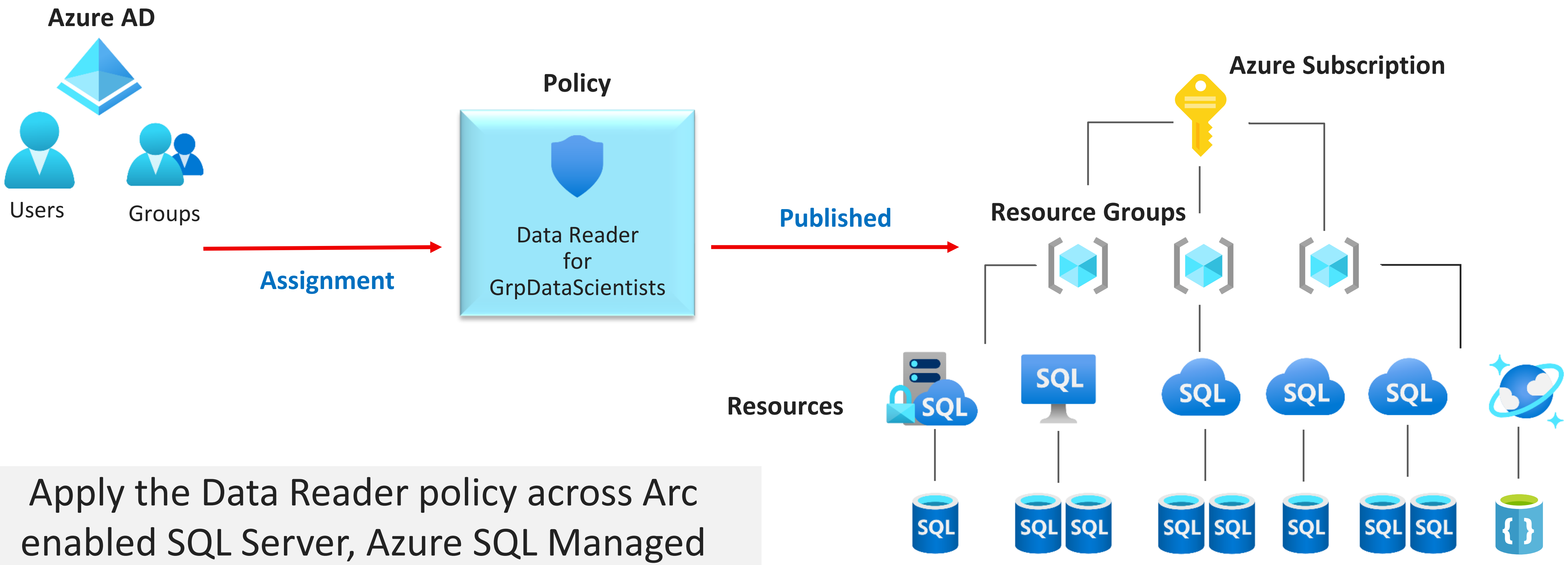
New authentication option for SQL Server instance

Allows to access Azure AD to authenticate and enables MFA scenarios

Automated setup using Azure portal and Azure Arc agent

Setup Azure AD administrator the same way Azure SQL does

Access Control at scale: using policies by Azure Purview



Apply the Data Reader policy across Arc enabled SQL Server, Azure SQL Managed Instance, Azure SQL Database and Cosmos DB

Other Security Features

Always Encrypted with secure enclaves: new query patterns, including ORDER BY, JOIN and GROUP BY on encrypted columns using enclaves

Crypto enhancements: import/backup/create certificates from PFX; Database Master Key backup/restore to/from Azure Blob Storage; crypto improvements related to system-generated certificates and hashing algorithm usage

New granular permissions and roles: new permissions to help implement the PoLP; Ownership-chaining covered by new permission

New granular permissions for DDM: GRANT/DENY UNMASK permission at schema, table, and column-level (same as Azure)

Support for TLS 1.3: moving from negotiated to strict encryption established by the connection string

Microsoft Defender for SQL Server: easier setup experience for SQL Server 2022

Learn more

Learn more about SQL Server 2022

aka.ms/sqlserver2022

Register to apply for the SQL Server 2022 Early Adoption Program and stay informed

aka.ms/EAPSignup

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/

At the end, please rate this 2-part session

https://sqlb.it/?7207

https://sqlb.it/?72014