Microsoft

SQL Server Plan Caching and Query Store

Module 8

Learning Units covered in this Module

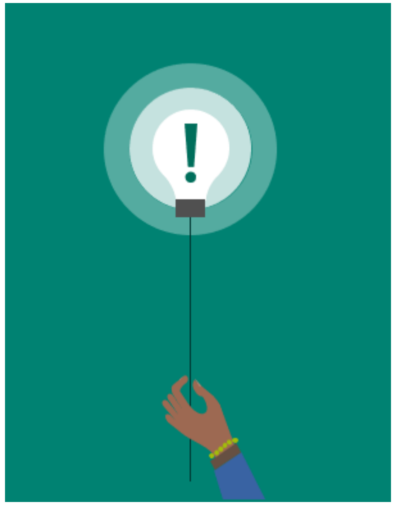
- Lesson 1: SQL Server Plan Cache
- Lesson 2: SQL Server Query Store

Lesson 1: SQL Server Plan Cache

Objectives

After completing this learning, you will be able to:

- Describe the purpose and contents of the plan cache.
- Query the plan cache using Dynamic Management Objects.
- Discuss the pros and cons of plan reuse.
- Explain why *ad hoc* SQL statements can be especially problematic.



The Plan Cache

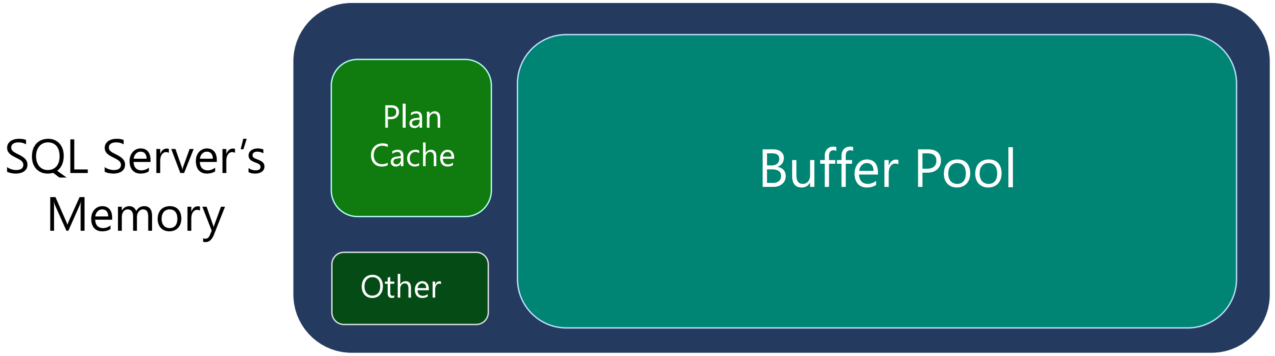
A pool of memory used to store query execution plans

Used by all databases in a SQL Server instance

Exists to avoid repeated optimization and compilation

Reuse of reduce optimization and compilation costs

Size and contents vary over time.



SQL Server's Memory

Plan Cache

Other

Buffer Pool

With a fixed amount of memory, when the plan cache grows the buffer pool shrinks and vice-versa

Plan Cache Contents

Object plans:

- Stored procedures, functions, triggers

SQL Plans:

- Prepared plans and *ad hoc* plans

Other:

- Bound trees and extended stored procedure references

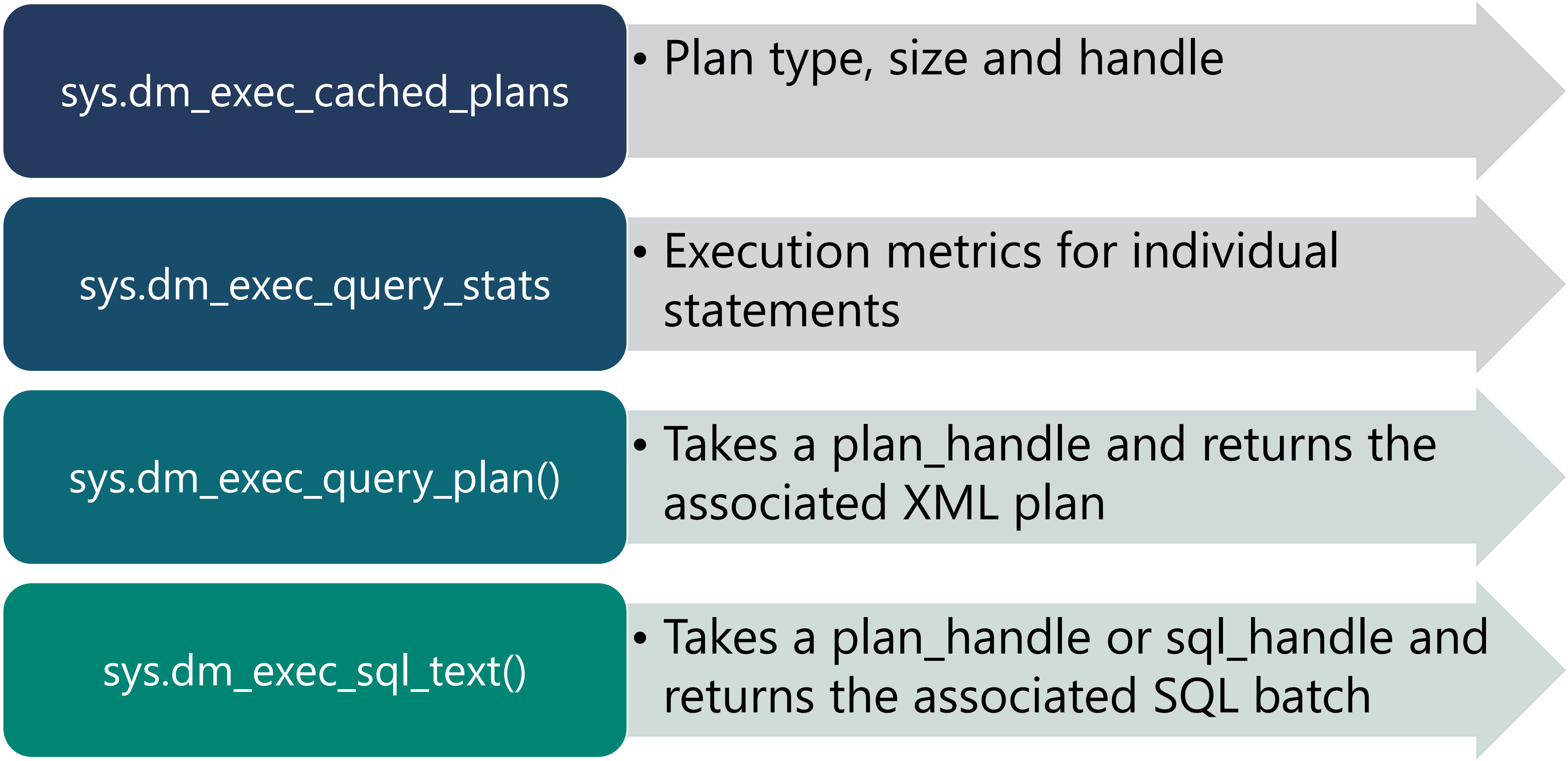
Granularity:

- Execution plans are per-statement (not object).

Plans per Statement:

- Multiple plans for a single statement may exist if differing execution contexts were used or if a parallel plan was generated.

Dynamic Management Objects



sys.dm_exec_cached_plans

- Plan type, size and handle

sys.dm_exec_query_stats

- Execution metrics for individual statements

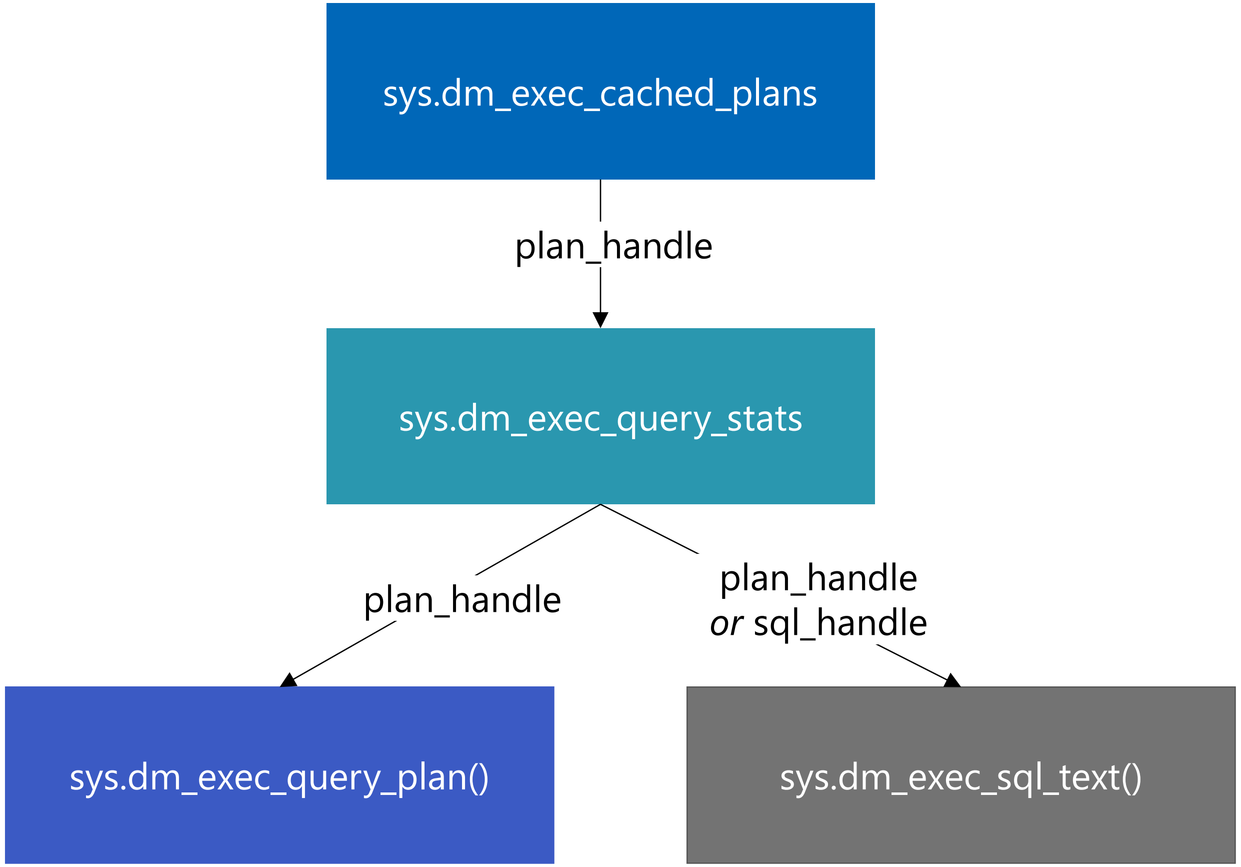
sys.dm_exec_query_plan()

- Takes a plan_handle and returns the associated XML plan

sys.dm_exec_sql_text()

- Takes a plan_handle or sql_handle and returns the associated SQL batch

Relationships between DMOs



sys.dm_exec_cached_plans

plan_handle

sys.dm_exec_query_stats

plan_handle

plan_handle
or sql_handle

sys.dm_exec_query_plan()

sys.dm_exec_sql_text()

Captured Metrics (sys.dm_exec_query_stats)

Partial listing

- Execution count
- Worker time
- Physical read
- Logical reads
- Elapsed time
- Row count
- Memory consumption
- Spills

Aggregate statistics

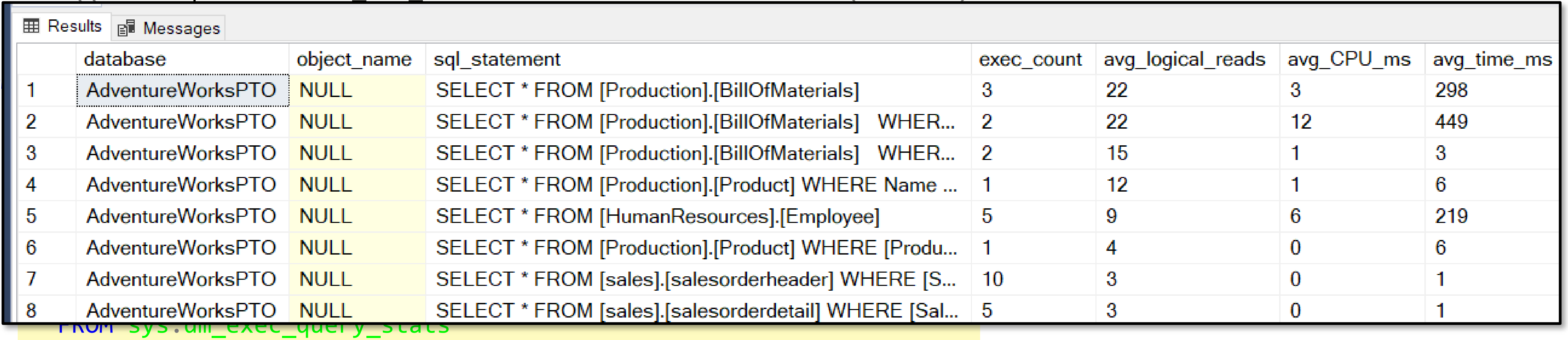
- Total
- Last
- Min
- Max

Compute Average: Avg = Total / Execution count

Mining the Plan Cache with T-SQL

Top 10 plans by logical reads

```
SELECT CASE st.dbid WHEN 32767 THEN 'resourcedb'
       WHEN NULL THEN 'NA' ELSE DB_NAME(st.dbid)END AS [database], OBJECT_NAME(st.objectid) AS object_name,
       SUBSTRING( st.text, ( qs.statement_start_offset / 2 ) + 1,
        (( CASE qs.statement_end_offset WHEN -1 THEN DATALENGTH(st.text)
FRO
       FROM sys.dm_exec_query_stats
       ORDER BY ( total_logical_reads / execution_count ) DESC ) AS qs
     CROSS APPLY sys.dm_exec_sql_text(qs.plan_handle) st
     CROSS APPLY sys.dm_exec_query_plan(qs.plan_handle) qp
ORDER BY qs.avg_logical_reads DESC
OPTION ( RECOMPILE );
```



	database	object_name	sql_statement	exec_count	avg_logical_reads	avg_CPU_ms	avg_time_ms
1	AdventureWorksPTO	NULL	SELECT * FROM [Production].[BillOfMaterials]	3	22	3	298
2	AdventureWorksPTO	NULL	SELECT * FROM [Production].[BillOfMaterials] WHER...	2	22	12	449
3	AdventureWorksPTO	NULL	SELECT * FROM [Production].[BillOfMaterials] WHER...	2	15	1	3
4	AdventureWorksPTO	NULL	SELECT * FROM [Production].[Product] WHERE Name ...	1	12	1	6
5	AdventureWorksPTO	NULL	SELECT * FROM [HumanResources].[Employee]	5	9	6	219
6	AdventureWorksPTO	NULL	SELECT * FROM [Production].[Product] WHERE [Produ...	1	4	0	6
7	AdventureWorksPTO	NULL	SELECT * FROM [sales].[salesorderheader] WHERE [S...	10	3	0	1
8	AdventureWorksPTO	NULL	SELECT * FROM [sales].[salesorderdetail] WHERE [Sal...	5	3	0	1

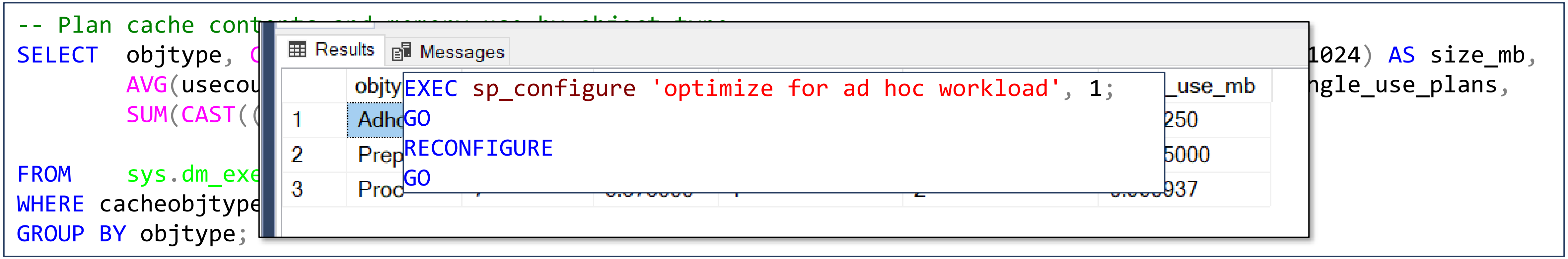
Cache Plan Bloat

Caused by *ad hoc* SQL

Heavy *ad hoc* workloads can bloat the plan cache

No benefit to caching single-use, ad hoc plans

Enable “optimize for ad hoc workload” to keep single-use plans out of the cache



```
-- Plan cache cont
SELECT   objtype, C                                              1024) AS size_mb,
         AVG(usecou                                              ngle_use_plans,
         SUM(CAST((

FROM     sys.dm_exe
WHERE cacheobjtype
GROUP BY objtype;
```

```
     objty                                    _use_mb
1    Adhc                                     250
2    Prep                                     5000
3    Proc                                     937
```

```
EXEC sp_configure 'optimize for ad hoc workload', 1;
GO
RECONFIGURE
GO
```

Clearing the Plan Cache

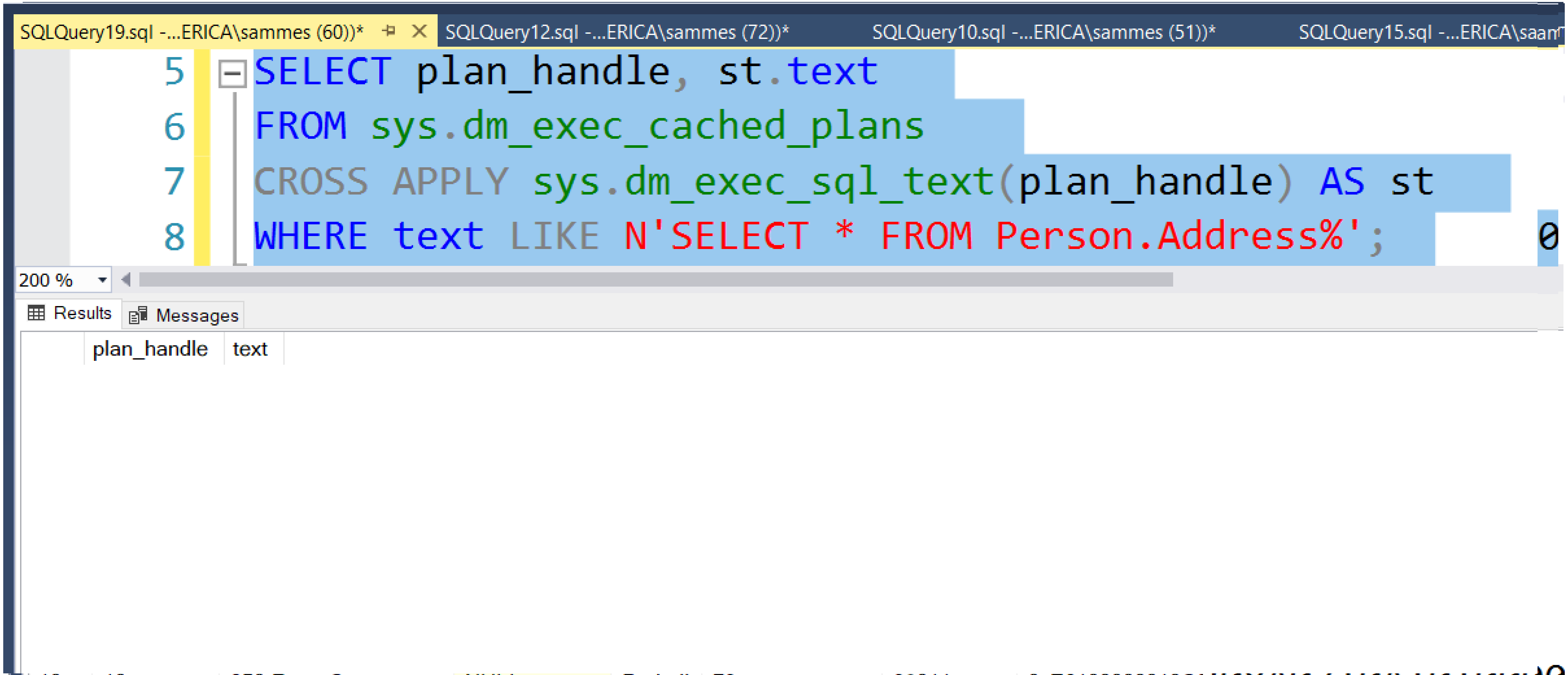
Not always the best option!

The entire plan cache – all databases	• DBCC FREEPROCCACHE;
All plans of a specific type	• DBCC FREESYSTEMCACHE ('SQL Plans');
All plans for a single database	• ALTER DATABASE SCOPED CONFIGURATION CLEAR PROCEDURE_CACHE;
A specific plan	• ALTER DATABASE SCOPED CONFIGURATION CLEAR PROCEDURE_CACHE(<plan_handle>);

Performance impact as new plans are compiled

Clearing the Plan Cache

Not your first option!


SQLQuery19.sql -...ERICA\sammes (60))*
SQLQuery12.sql -...ERICA\sammes (72))*
SQLQuery10.sql -...ERICA\sammes (51))*
SQLQuery15.sql -...ERICA\saam
200 %
Results
Messages

```sql
SELECT plan_handle, st.text
FROM sys.dm_exec_cached_plans
CROSS APPLY sys.dm_exec_sql_text(plan_handle) AS st
WHERE text LIKE N'SELECT * FROM Person.Address%';
```

plan_handle	text

Plan Reuse

Requires that SQL statements match exactly

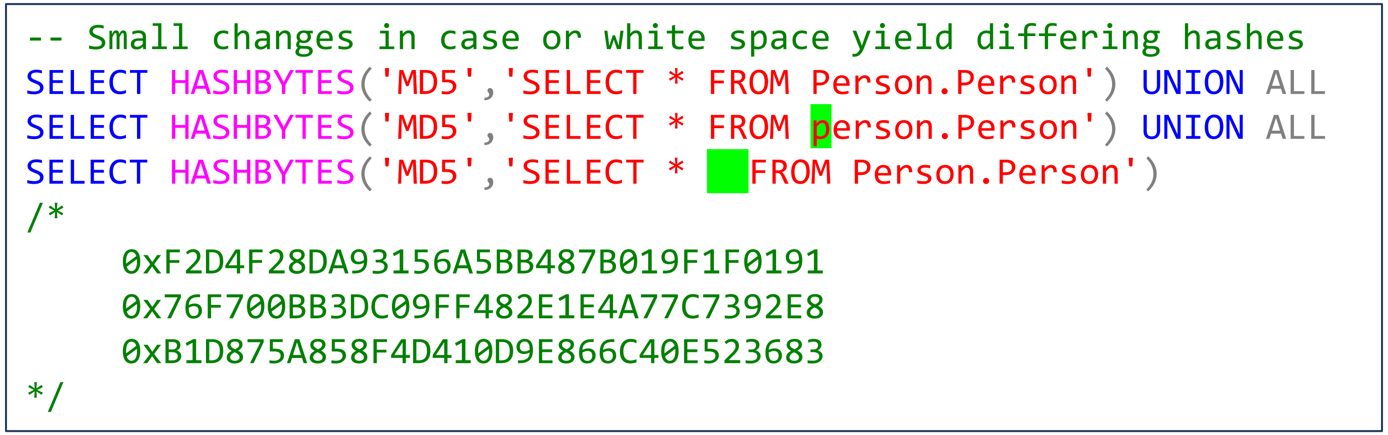
- Any difference in casing, white space or literal values will affect a hashed value

Less likely for *ad hoc* SQL statements

- Simple (on by default) or Forced Parameterization can improve reuse

Most easily achieved using:

- Stored procedures, Functions, Triggers
- Prepared statements and parameterized queries (sp_executesql)



```
-- Small changes in case or white space yield differing hashes
SELECT HASHBYTES('MD5','SELECT * FROM Person.Person') UNION ALL
SELECT HASHBYTES('MD5','SELECT * FROM person.Person') UNION ALL
SELECT HASHBYTES('MD5','SELECT *   FROM Person.Person')
/*
    0xF2D4F28DA93156A5BB487B019F1F0191
    0x76F700BB3DC09FF482E1E4A77C7392E8
    0xB1D875A858F4D410D9E866C40E523683
*/
```

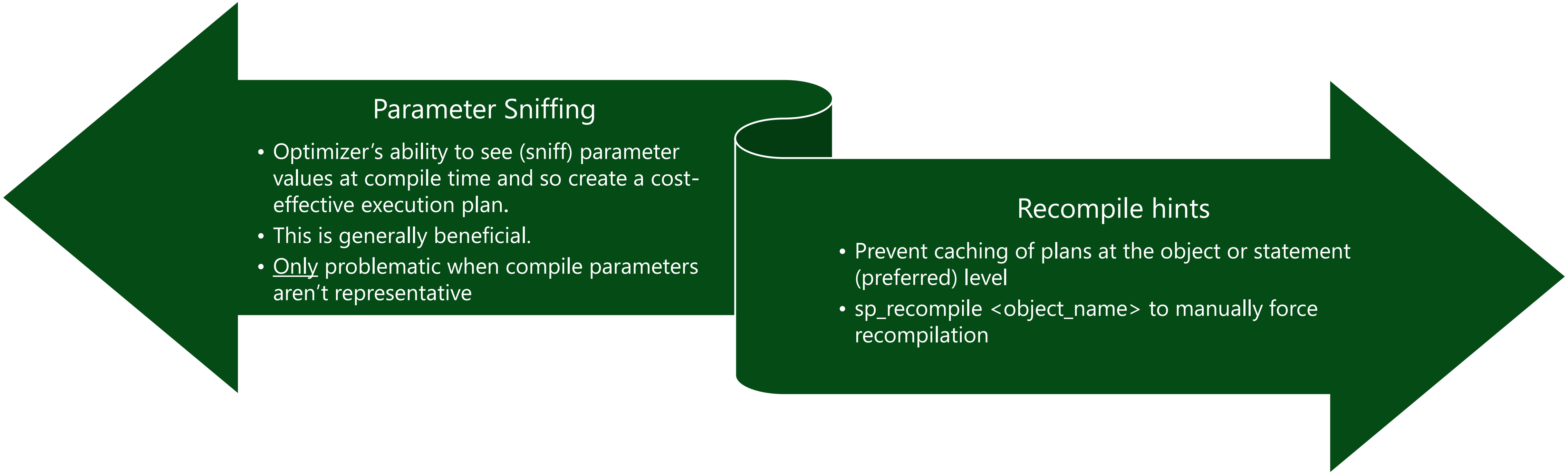
Plan Reuse

Benefit

- Improved performance as reuse saves time and CPU

Drawback

- Degraded performance when reused plan is not optimal for all parameter values



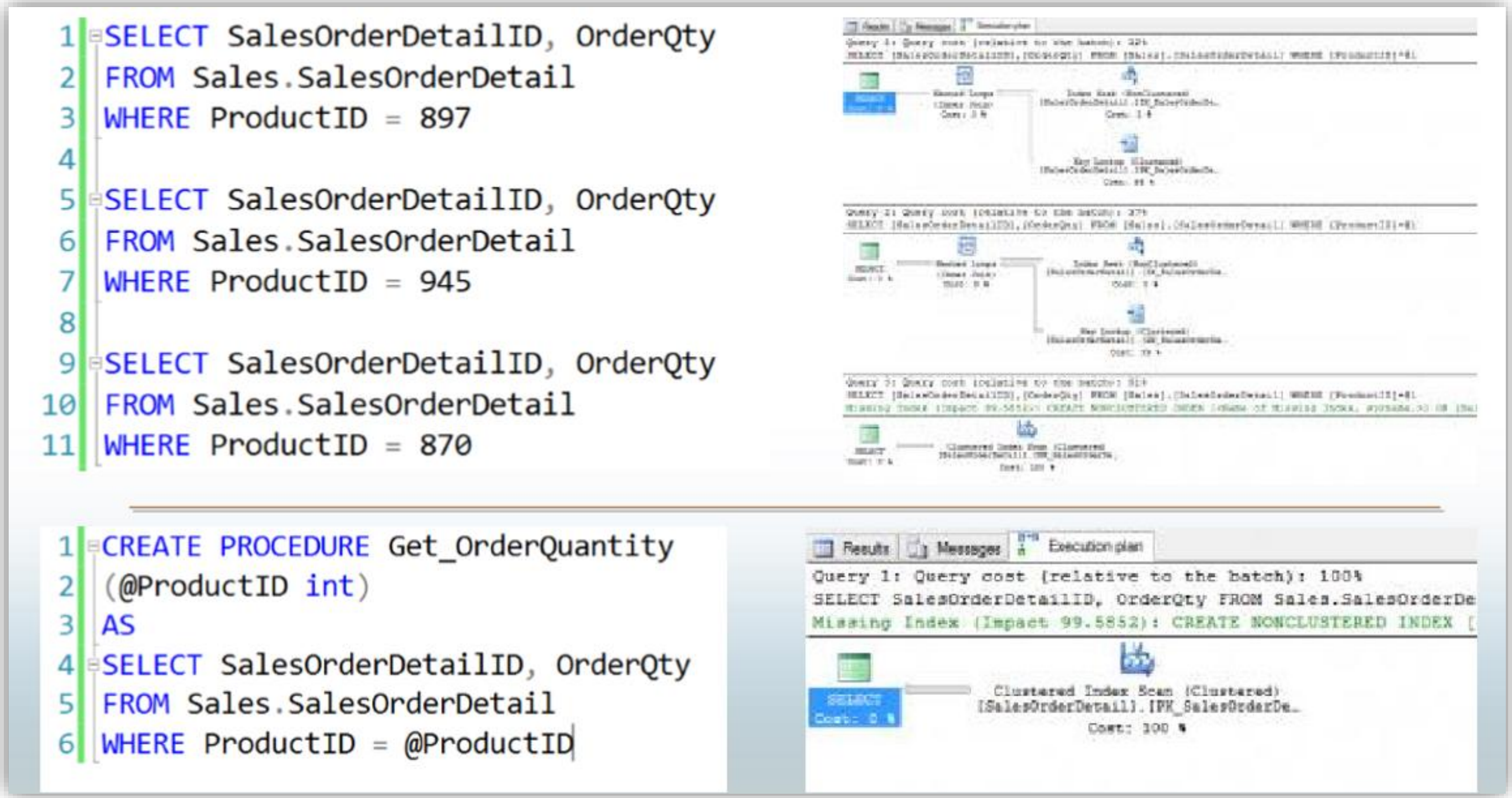
Parameter Sniffing

- Optimizer's ability to see (sniff) parameter values at compile time and so create a cost-effective execution plan.
- This is generally beneficial.
- Only problematic when compile parameters aren't representative

Recompile hints

- Prevent caching of plans at the object or statement (preferred) level
- sp_recompile <object_name> to manually force recompilation

Parameter Sniffing



```sql
SELECT SalesOrderDetailID, OrderQty
FROM Sales.SalesOrderDetail
WHERE ProductID = 897

SELECT SalesOrderDetailID, OrderQty
FROM Sales.SalesOrderDetail
WHERE ProductID = 945

SELECT SalesOrderDetailID, OrderQty
FROM Sales.SalesOrderDetail
WHERE ProductID = 870
```

```
Query 1: Query cost (relative to the batch): [illegible]
SELECT [SalesOrderDetailID],[OrderQty] FROM [Sales].[SalesOrderDetail] WHERE [ProductID]=@1

Query 2: Query cost (relative to the batch): [illegible]
SELECT [SalesOrderDetailID],[OrderQty] FROM [Sales].[SalesOrderDetail] WHERE [ProductID]=@1

Query 3: Query cost (relative to the batch): [illegible]
SELECT [SalesOrderDetailID],[OrderQty] FROM [Sales].[SalesOrderDetail] WHERE [ProductID]=@1
Missing Index (Impact 99.5852): CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname,>] ON [Sal
```

Results
Messages
Execution plan
SELECT
Nested Loops
(Inner Join)
Index Seek (NonClustered)
[SalesOrderDetail].[IX_SalesOrderDe...
Key Lookup (Clustered)
[SalesOrderDetail].[PK_SalesOrderDe...
Clustered Index Scan (Clustered)
[SalesOrderDetail].[PK_SalesOrderDe...

```sql
CREATE PROCEDURE Get_OrderQuantity
(@ProductID int)
AS
SELECT SalesOrderDetailID, OrderQty
FROM Sales.SalesOrderDetail
WHERE ProductID = @ProductID
```

```
Query 1: Query cost (relative to the batch): 100%
SELECT SalesOrderDetailID, OrderQty FROM Sales.SalesOrderDe
Missing Index (Impact 99.5852): CREATE NONCLUSTERED INDEX [
```

Results
Messages
Execution plan
SELECT
Cost: 0 %
Clustered Index Scan (Clustered)
[SalesOrderDetail].[PK_SalesOrderDe...
Cost: 100 %

SQL Server Execution Plan Recompilations

Overview

Most recompilations are required either for statement correctness or to obtain potentially faster query execution plan.

The engine detects changes that invalidate execution plan(s) and marks those as not valid. New plan must be recompiled for the next query execution.

Starting with SQL Server 2005, whenever a statement within a batch causes recompilation, only the statement inside the batch that triggers recompilation is recompiled.

SQL Server Execution Plan Recompilations

Recompilation reasons

Table / Index Changes

- Changes made to objects referenced by the query (ALTER TABLE and ALTER VIEW).
- Changing or dropping any indexes used by the execution plan.

Stored Procedures

- Changes made to a single procedure, which would drop all plans for that procedure from the cache (ALTER PROCEDURE).
- Explicit call to sp_recompile.
- Executing a stored procedure using the WITH RECOMPILE option.

Data Volume

- Updates on statistics used by the execution plan
- For tables with triggers, if the number of rows in the inserted or deleted tables grows significantly.

Other

- Large numbers of changes to keys (generated by statements from other users that modify a table referenced by the query).
- Temporary table changes

Demonstration

Caching and Parameter sniffing

- Caching and reuse of *ad hoc* vs. stored procedure query plans
- Parameter sniffing
- Querying the plan cache

Using Plan Guides

Designing and Implementing Plan Guides | Microsoft Learn

OBJECT

Match queries that execute in the context of Transact-SQL stored procedures, scalar functions, multi-statement table-valued functions, and DML triggers.

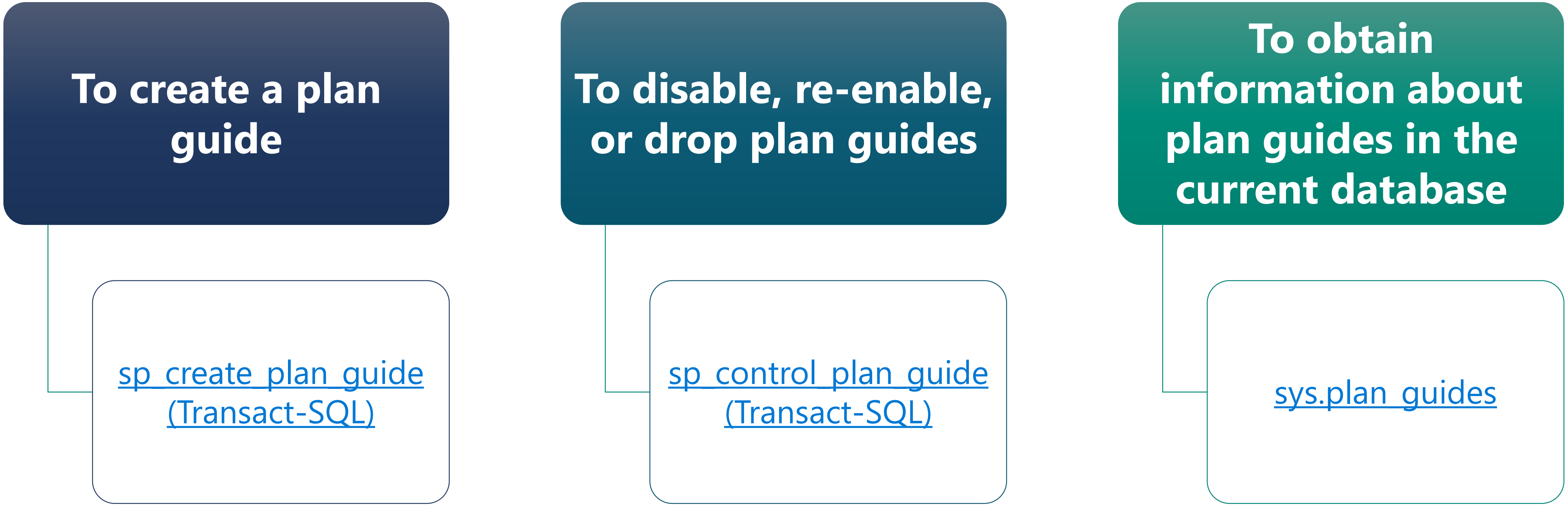
SQL

Match queries that execute in the context of stand-alone Transact-SQL statements and batches that are not part of a database object.

TEMPLATE

Match stand-alone queries that parameterize to a specified form. These plan guides are used to override the current PARAMETERIZATION database SET option of a database for a class of queries.

Using Plan Guides



To create a plan guide

sp_create_plan_guide (Transact-SQL)

To disable, re-enable, or drop plan guides

sp_control_plan_guide (Transact-SQL)

To obtain information about plan guides in the current database

sys.plan_guides

Plan Guide Parameters

```
EXEC sp_create_plan_guide @name, @stmt, @type, @module_or_batch, @params, @hints
```

@name – name of the plan guide

@stmt – a T-SQL statement or batch

@type – indicates the type of guide (OBJECT, SQL, or TEMPLATE)

@module_or_batch – the name of a module (i.e. a stored procedure)

@params – for SQL and TEMPLATE guides, a string of all parameters for a T-SQL batch to be matched by this plan guide

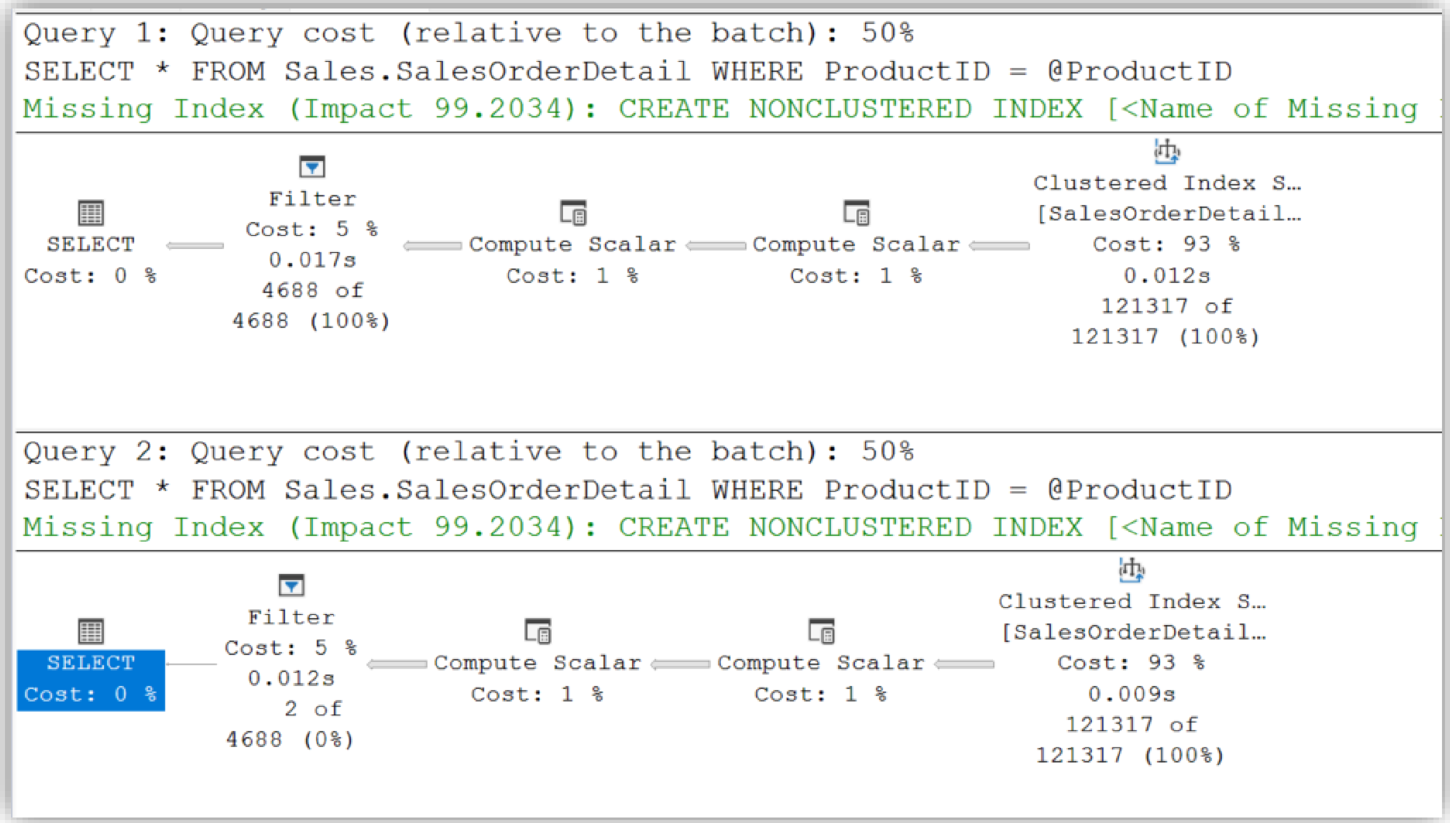
@hints – OPTION clause hint to attach to a query as defined in the @stmt parameter

Parameter Sensitive Plan example

```
--ProductID 870 returns 4,688 rows out of 121,317 total rows.
EXEC sp_executesql
@stmt = N'SELECT * FROM Sales.SalesOrderDetail WHERE ProductID =
@ProductID',
@params = N'@ProductID int', @ProductID = 870
GO

--ProductID 897 returns 2 rows out of 121,317 total rows.
EXEC sp_executesql
@stmt = N'SELECT * FROM Sales.SalesOrderDetail WHERE ProductID =
@ProductID',
@params = N'@ProductID int', @ProductID = 897
GO
```

Parameter Sniffing example



```
Query 1: Query cost (relative to the batch): 50%
SELECT * FROM Sales.SalesOrderDetail WHERE ProductID = @ProductID
Missing Index (Impact 99.2034): CREATE NONCLUSTERED INDEX [<Name of Missing
```

```
Query 2: Query cost (relative to the batch): 50%
SELECT * FROM Sales.SalesOrderDetail WHERE ProductID = @ProductID
Missing Index (Impact 99.2034): CREATE NONCLUSTERED INDEX [<Name of Missing
```

```
<ParameterList>
  <ColumnReference Column="@ProductID" ParameterDataType="int"
                   ParameterCompiledValue="(870)" ParameterRuntimeValue="(897)" />
</ParameterList>
```

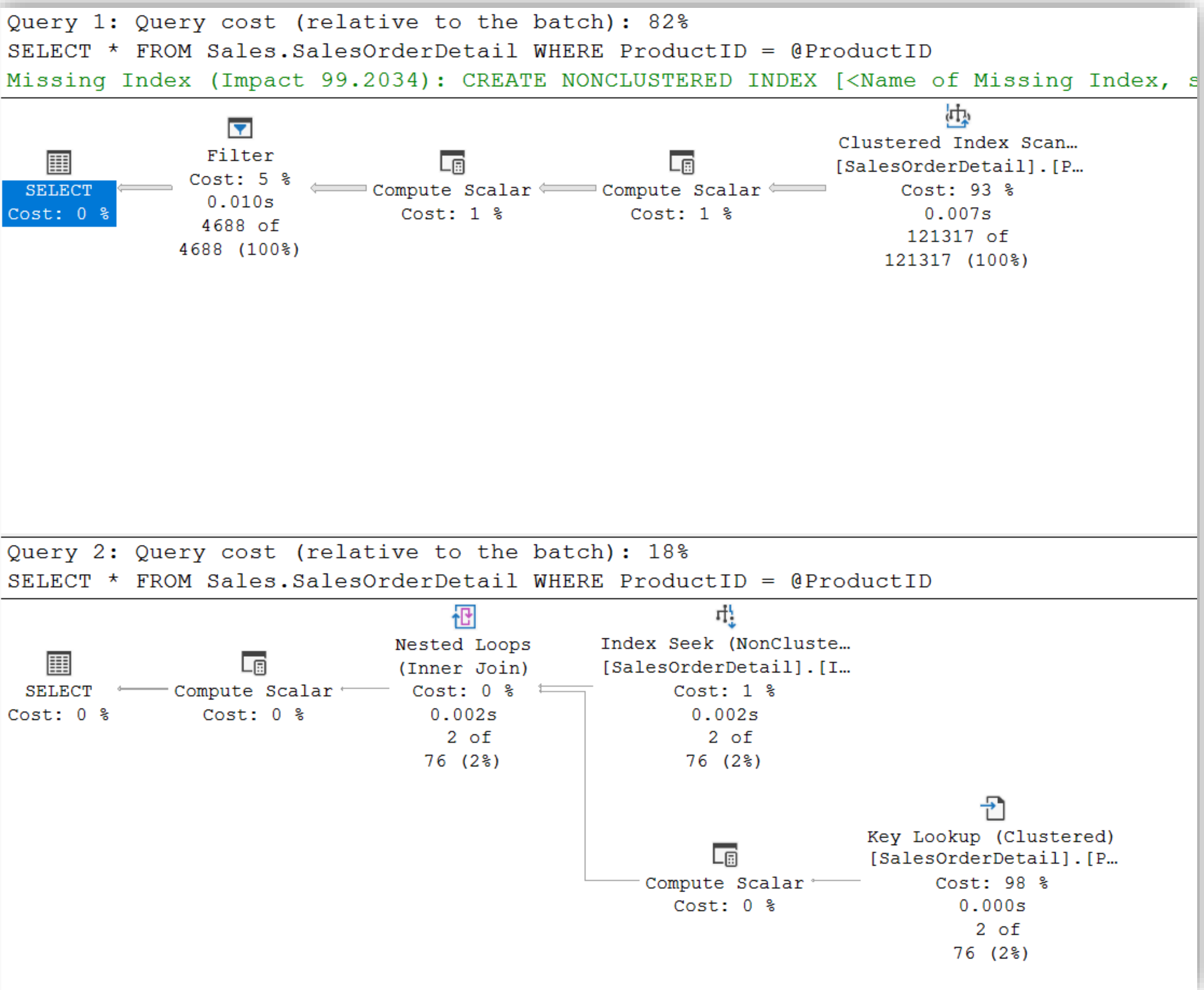
Creating a Plan Guide

```sql
--Create SQL Plan Guide to force a RECOMPILE.
EXEC sp_create_plan_guide
@name = N'SalesOrders_ProductID_Recompile',
@stmt = N'SELECT * FROM Sales.SalesOrderDetail WHERE ProductID = @ProductID',
@type = N'SQL',
@module_or_batch = NULL,
@params = N'@ProductID int',
@hints = N'OPTION (RECOMPILE)'
GO

--To see a list of plan guides stored on the database
SELECT * FROM sys.plan_guides
GO

--Disable plan guide
--@operation – a control option; one of DROP, DROP ALL, DISABLE, ENABLE
--@name – name of the plan guide to control
EXEC sp_control_plan_guide N'DISABLE', N'SalesOrders_ProductID_Recompile'
GO
```

Recompiled Plans



```
Query 1: Query cost (relative to the batch): 82%
SELECT * FROM Sales.SalesOrderDetail WHERE ProductID = @ProductID
Missing Index (Impact 99.2034): CREATE NONCLUSTERED INDEX [<Name of Missing Index, s
```

```
Query 2: Query cost (relative to the batch): 18%
SELECT * FROM Sales.SalesOrderDetail WHERE ProductID = @ProductID
```

Questions?

Knowledge Check

Is the size of the plan cache fixed?

How long do query plans remain in the plan cache?

Why is plan caching helpful?

Why are *ad hoc* query plans sometimes problematic?

What can be done to lessen the impact of an *ad hoc* workload?

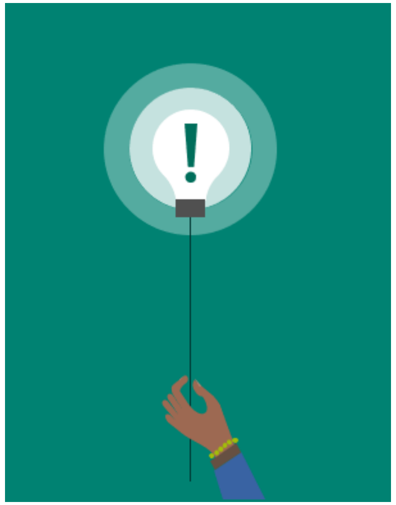
What can be done to address a parameter sniffing issue?

Lesson 2: SQL Server Query Store

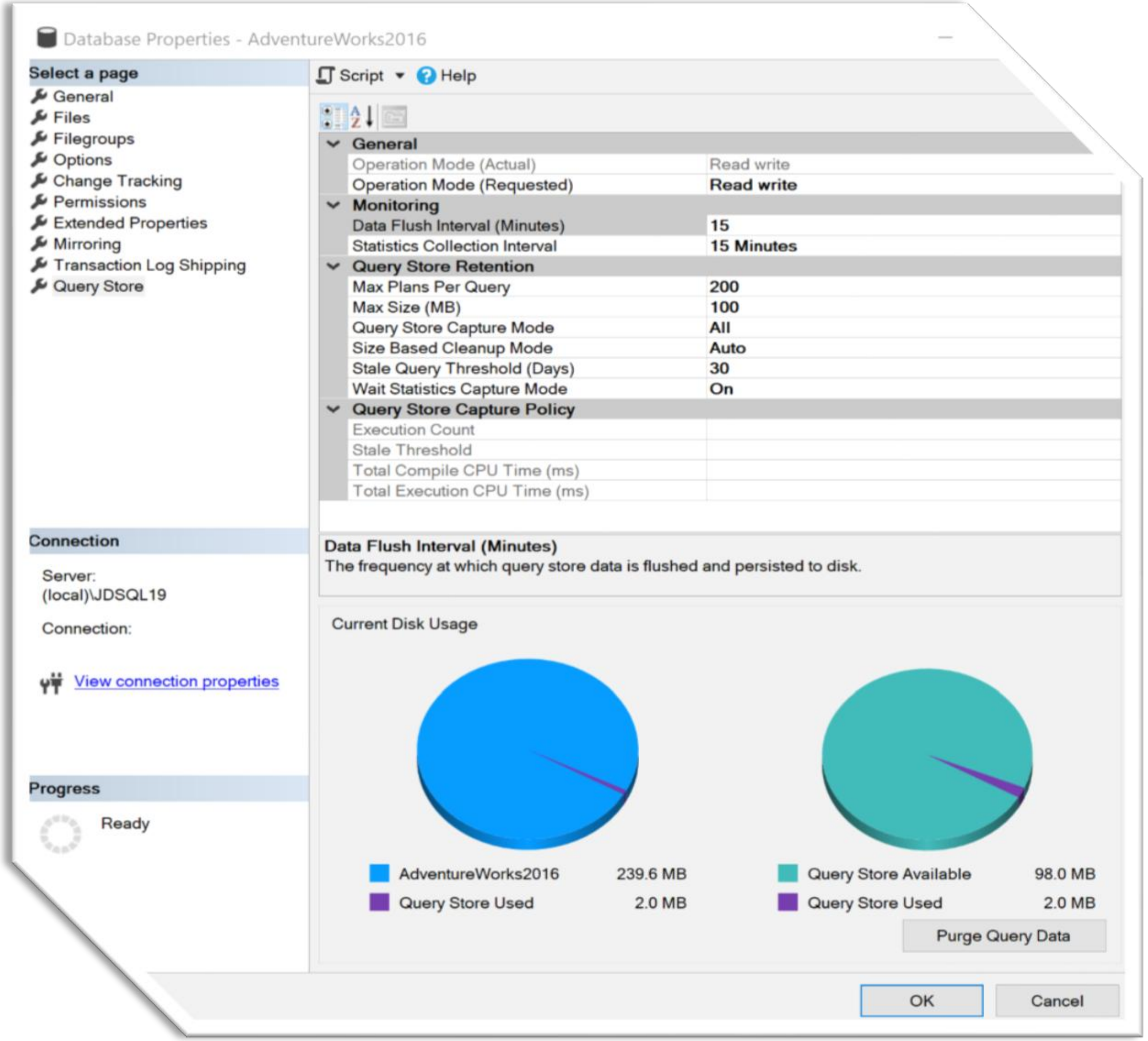
Objectives

After completing this learning, you will be able to:

- Understand what makes Query Store a valuable tool.
- Understand its key usage scenarios.
- Enable Query Store and configure it appropriately.
- List types of runtime data collected by the Query Store.
- Have a basic understanding of the built-in reports.
- Understanding Query Store Hints.



Introducing the Query Store



General	
Operation Mode (Actual)	Read write
Operation Mode (Requested)	Read write
Monitoring	
Data Flush Interval (Minutes)	15
Statistics Collection Interval	15 Minutes
Query Store Retention	
Max Plans Per Query	200
Max Size (MB)	100
Query Store Capture Mode	All
Size Based Cleanup Mode	Auto
Stale Query Threshold (Days)	30
Wait Statistics Capture Mode	On
Query Store Capture Policy	
Execution Count	
Stale Threshold	
Total Compile CPU Time (ms)	
Total Execution CPU Time (ms)	

Data Flush Interval (Minutes)
The frequency at which query store data is flushed and persisted to disk.

Server:
(local)\JDSQL19

Connection:

Ready

Current Disk Usage

AdventureWorks2016	239.6 MB
Query Store Used	2.0 MB
Query Store Available	98.0 MB
Query Store Used	2.0 MB

Query Store is set at the database level

Cannot be used for Master or TempDB system databases but can be enabled for the Model and MSDB system databases.

The user database stores the data in internal tables that can be accessed by using built-in Query Store views.

SQL Server retains this data until the space allocated to Query Store is full or manually purged.

Why use Query Store?

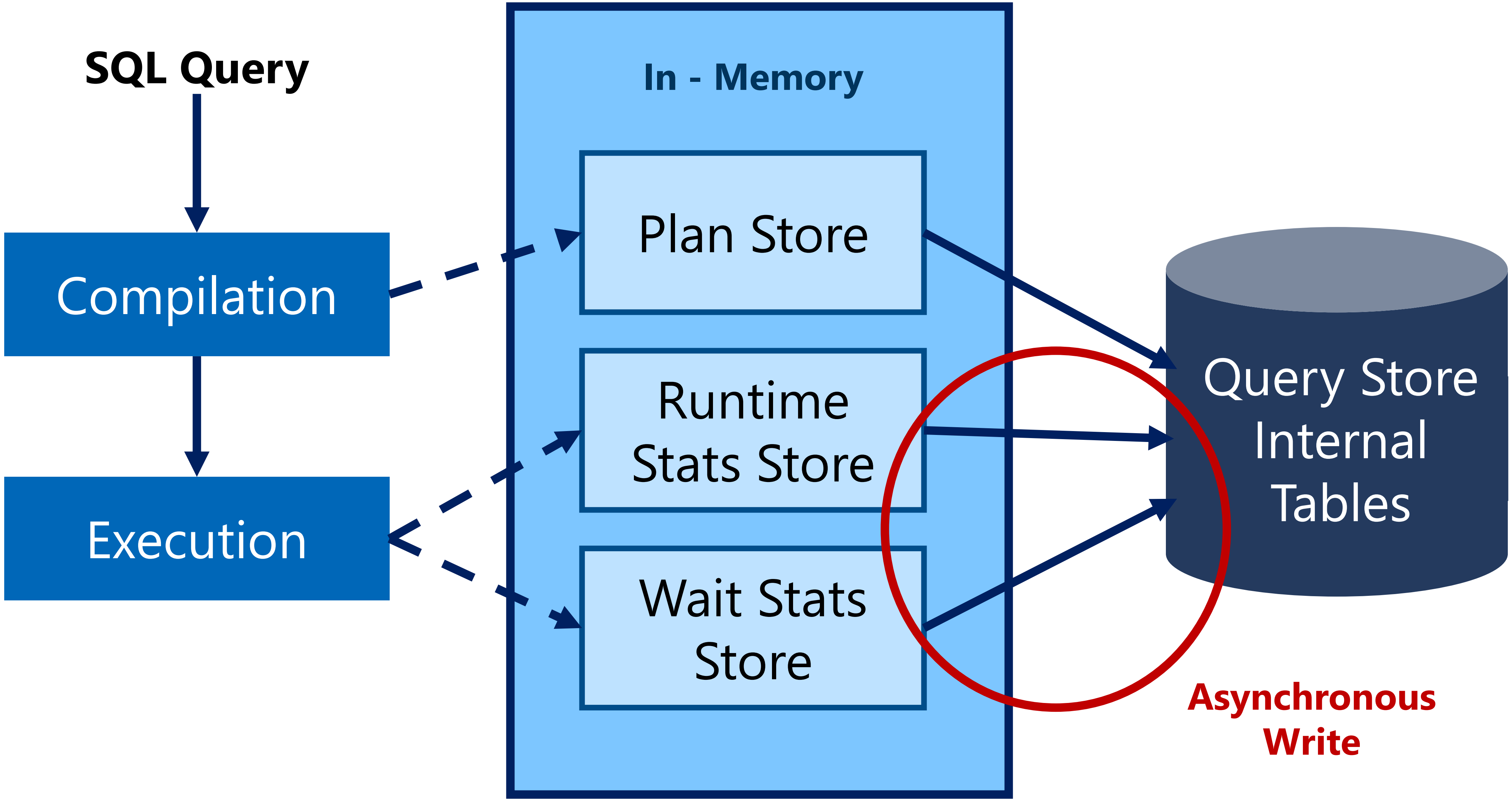
Before Query Store

- Requires manual proactive monitoring to identify execution plan problems.
- Only the latest plan was stored in the procedure cache
- Restart caused data to be lost
- Frequent recompiles of procedures or use of DBCC FREEPROCACHE
- No history or aggregated gathering of data available.

With Query Store

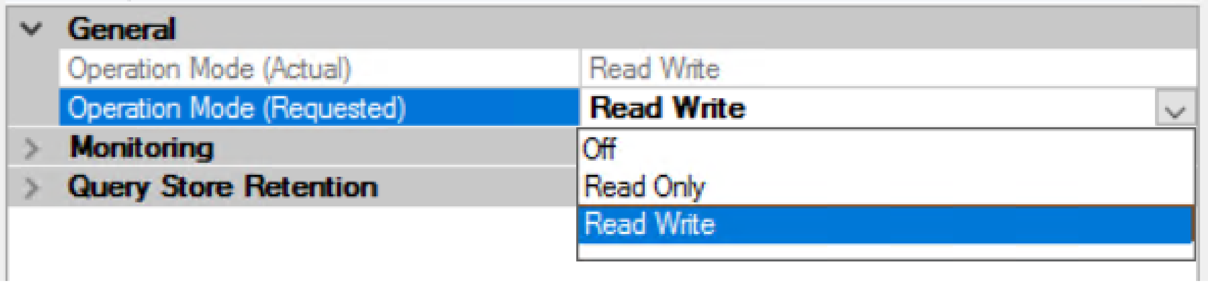
- It stores the history of the execution plans for each query
- It establishes a performance baseline for each plan over time
- It identifies queries that may have regressed
- It is possible to force plans quickly and easily
- It works across server restarts, upgrades, and query recompilation

How Query Store collects and stores data



Query Store Operation Modes

Operation Mode can be set under database properties



General

Operation Mode (Actual) Read Write

Operation Mode (Requested) Read Write

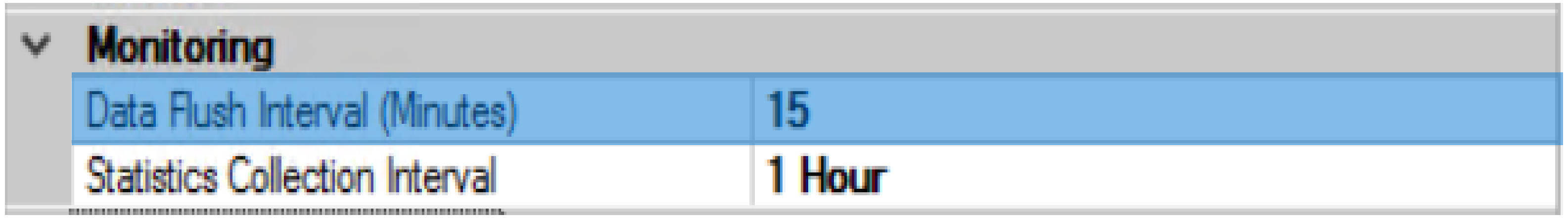
Operation Mode can be enabled two ways using T-SQL. If only using the ON option, the Mode defaults to **Read_Write**

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

ALTER DATABASE [AdventureWorksPTO] SET QUERY_STORE
(OPERATION_MODE = READ_WRITE);
```

Query Store Monitoring Settings

Data Flush Interval determines the frequency at which data written to the query store is persisted to disk. (Default is **15 Minutes**).



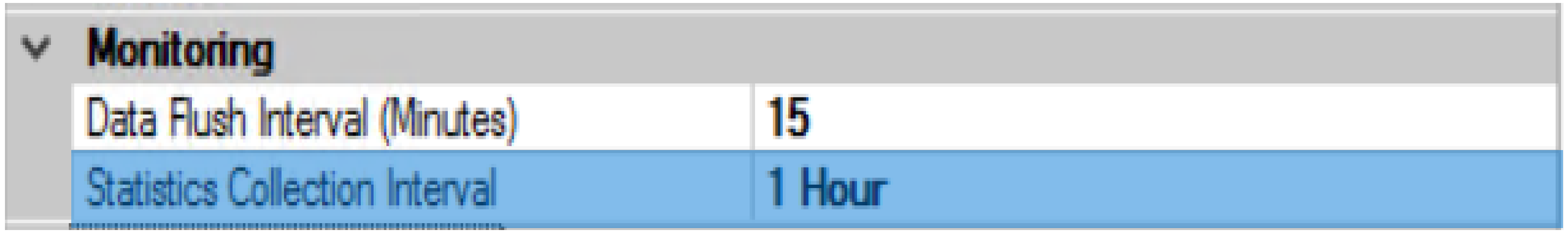
Monitoring

Data Flush Interval (Minutes)	15
Statistics Collection Interval	1 Hour

```
ALTER DATABASE [AdventureWorksPTO] SET QUERY_STORE
(INTERVAL_LENGTH_MINUTES = 60,
 DATA_FLUSH_INTERVAL_SECONDS = 900 );
```

Query Store Monitoring Settings

Statistics Collection Interval determines the time interval at which runtime execution statistics data is aggregated into the query store. Only the values of 1, 5, 10, 15, 60, and 1440 minutes is allowed. (Default is **60**).



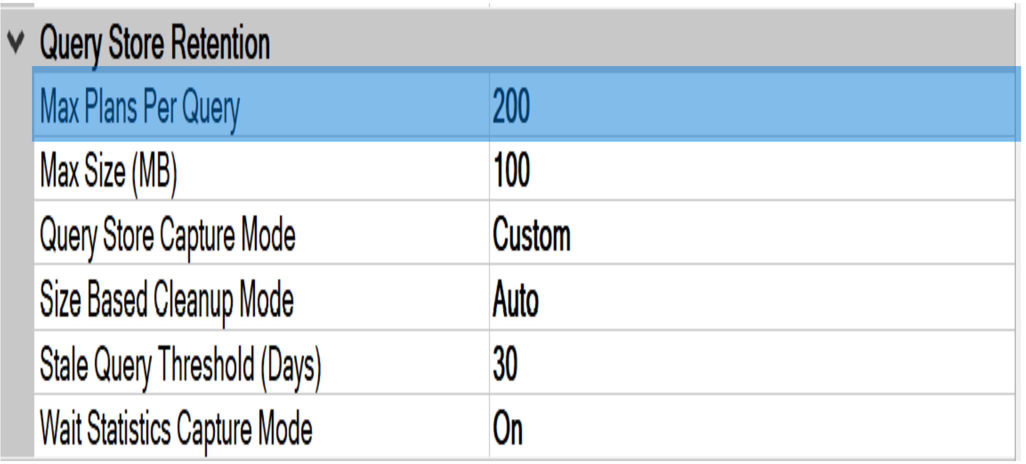
Monitoring

Data Flush Interval (Minutes)	15
Statistics Collection Interval	1 Hour

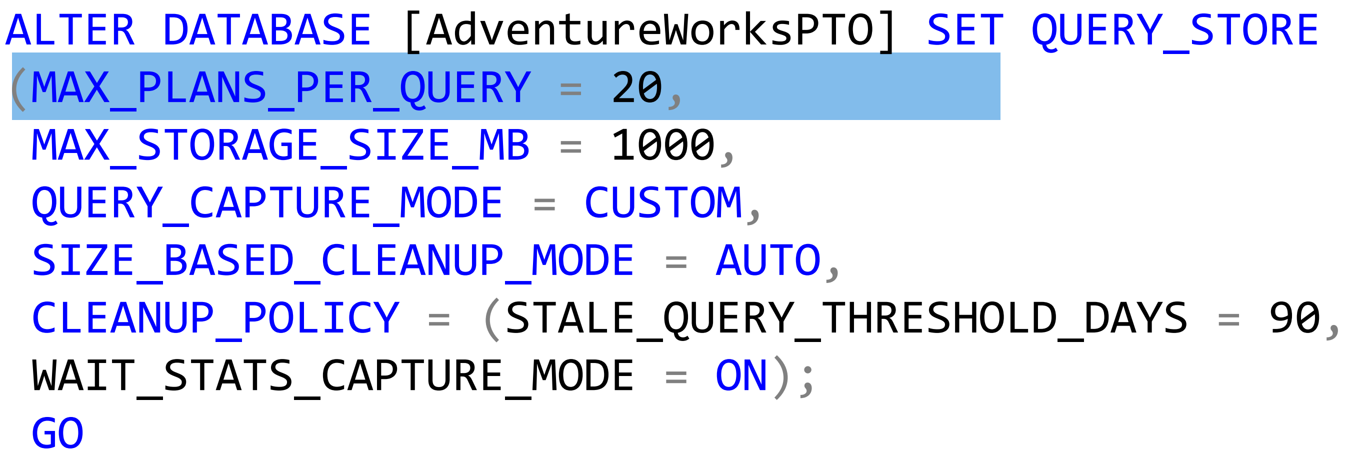
```
ALTER DATABASE [AdventureWorksPTO] SET QUERY_STORE
(INTERVAL_LENGTH_MINUTES = 60,
 DATA_FLUSH_INTERVAL_SECONDS = 900 );
```

Query Store Retention Settings

Max Plans Per Query is a new retention setting introduced in SQL Server 2017 and is an integer representing the maximum number of plans maintained for each query. (Default is **200**).



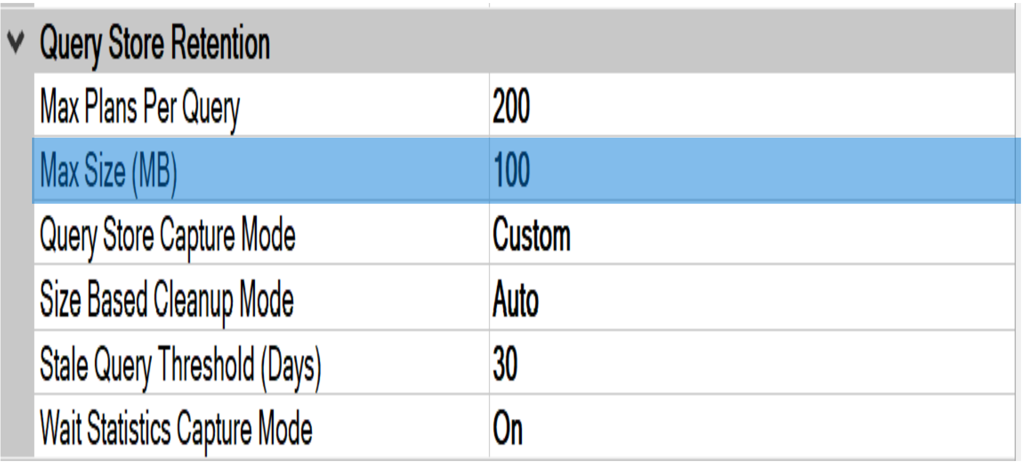
Query Store Retention	
Max Plans Per Query	200
Max Size (MB)	100
Query Store Capture Mode	Custom
Size Based Cleanup Mode	Auto
Stale Query Threshold (Days)	30
Wait Statistics Capture Mode	On



```
ALTER DATABASE [AdventureWorksPTO] SET QUERY_STORE
(MAX_PLANS_PER_QUERY = 20,
 MAX_STORAGE_SIZE_MB = 1000,
 QUERY_CAPTURE_MODE = CUSTOM,
 SIZE_BASED_CLEANUP_MODE = AUTO,
 CLEANUP_POLICY = (STALE_QUERY_THRESHOLD_DAYS = 90,
 WAIT_STATS_CAPTURE_MODE = ON);
 GO
```

Query Store Retention Settings

Max Size (MB) configures the maximum storage size for the query store. (Default is **100MB**) When the query store limit is reached, query store changes the state from read-write to read-only.

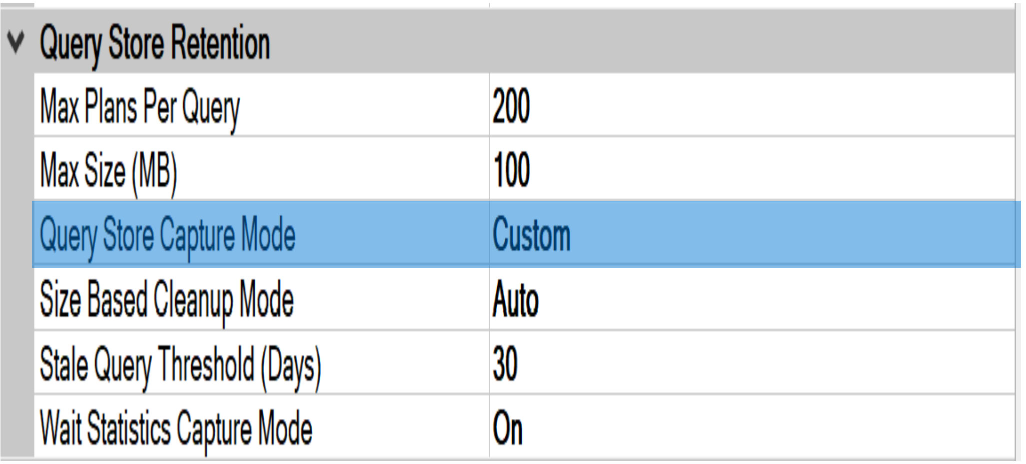


Query Store Retention	
Max Plans Per Query	200
Max Size (MB)	100
Query Store Capture Mode	Custom
Size Based Cleanup Mode	Auto
Stale Query Threshold (Days)	30
Wait Statistics Capture Mode	On

```
ALTER DATABASE [AdventureWorksPTO] SET QUERY_STORE
(MAX_PLANS_PER_QUERY = 20,
 MAX_STORAGE_SIZE_MB = 1000,
 QUERY_CAPTURE_MODE = CUSTOM,
 SIZE_BASED_CLEANUP_MODE = AUTO,
 CLEANUP_POLICY = (STALE_QUERY_THRESHOLD_DAYS = 90,
 WAIT_STATS_CAPTURE_MODE = ON);
 GO
```

Query Store Retention Settings

Query Store Capture Mode determines to capture all the queries (Default is **ALL**), or relevant queries based on execution count and resource consumption (**AUTO**) or stop capturing queries (**NONE**). SQL Server 2019 introduces an additional (**CUSTOM**) setting.

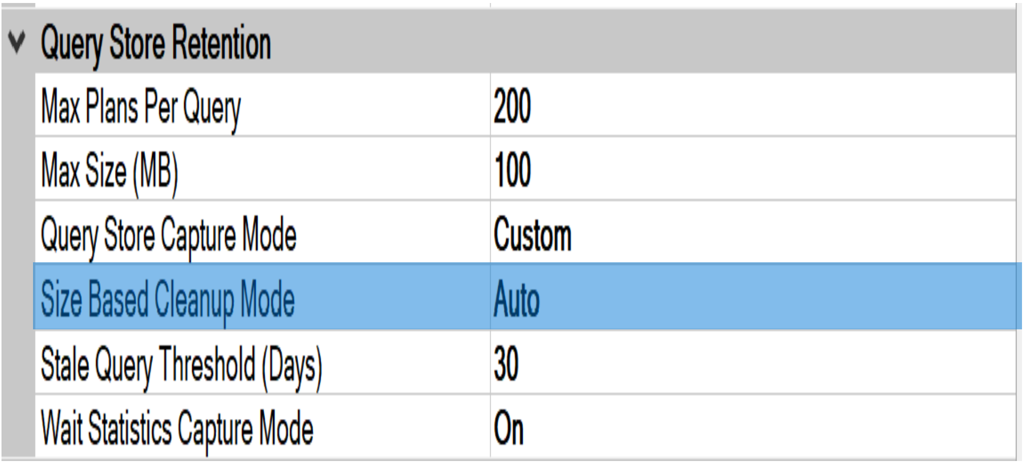


Query Store Retention	
Max Plans Per Query	200
Max Size (MB)	100
Query Store Capture Mode	Custom
Size Based Cleanup Mode	Auto
Stale Query Threshold (Days)	30
Wait Statistics Capture Mode	On

```
ALTER DATABASE [AdventureWorksPTO] SET QUERY_STORE
(MAX_PLANS_PER_QUERY = 20,
 MAX_STORAGE_SIZE_MB = 1000,
 QUERY_CAPTURE_MODE = CUSTOM,
 SIZE_BASED_CLEANUP_MODE = AUTO,
 CLEANUP_POLICY = (STALE_QUERY_THRESHOLD_DAYS = 90,
 WAIT_STATS_CAPTURE_MODE = ON);
 GO
```

Query Store Retention Settings

Size Based Cleanup Mode determines whether the cleanup process will be automatically activated when the total amount of data gets close to the maximum size. (Default is **Auto**).

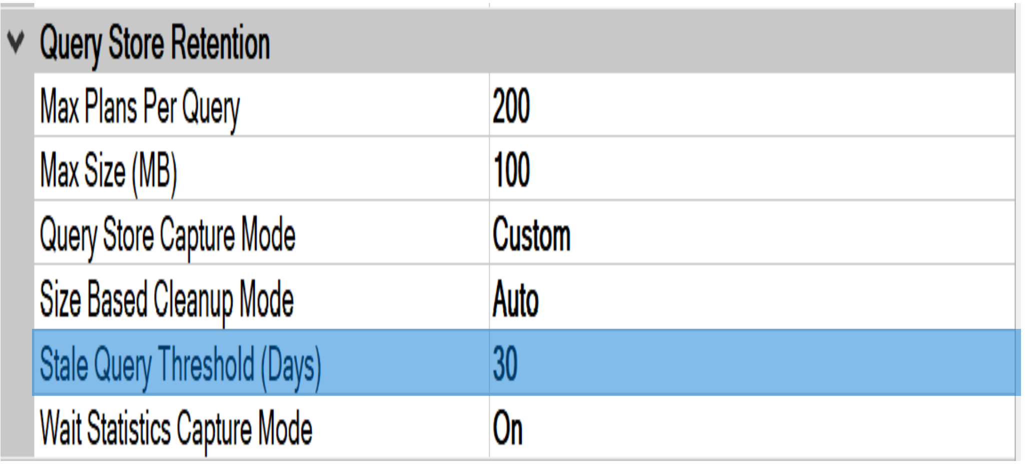


Query Store Retention	
Max Plans Per Query	200
Max Size (MB)	100
Query Store Capture Mode	Custom
Size Based Cleanup Mode	Auto
Stale Query Threshold (Days)	30
Wait Statistics Capture Mode	On

```
ALTER DATABASE [AdventureWorksPTO] SET QUERY_STORE
(MAX_PLANS_PER_QUERY = 20,
 MAX_STORAGE_SIZE_MB = 1000,
 QUERY_CAPTURE_MODE = CUSTOM,
 SIZE_BASED_CLEANUP_MODE = AUTO,
 CLEANUP_POLICY = (STALE_QUERY_THRESHOLD_DAYS = 90,
 WAIT_STATS_CAPTURE_MODE = ON);
 GO
```

Query Store Retention Settings

Stale Query Threshold (Days) determines the number of days to retain data in the query store. (Default is **30 days** and Maximum is **367 days**).



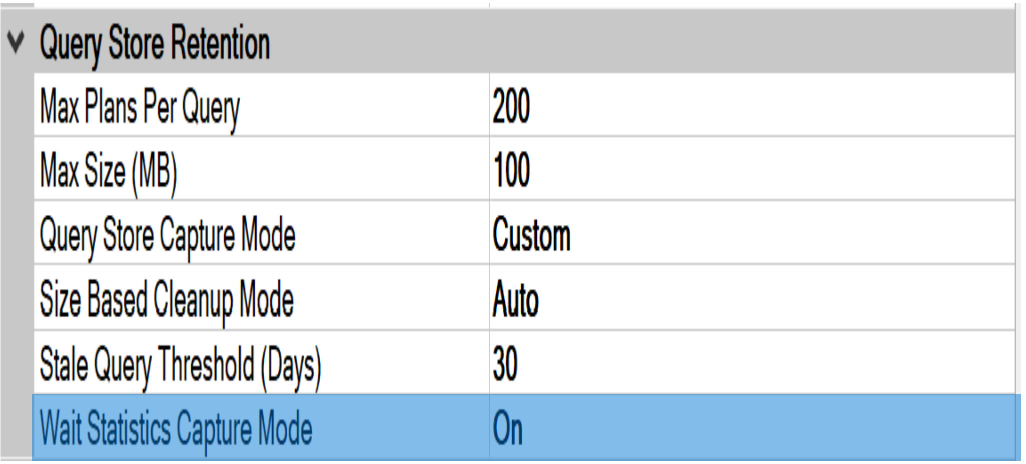
Query Store Retention	
Max Plans Per Query	200
Max Size (MB)	100
Query Store Capture Mode	Custom
Size Based Cleanup Mode	Auto
Stale Query Threshold (Days)	30
Wait Statistics Capture Mode	On

```
ALTER DATABASE [AdventureWorksPTO] SET QUERY_STORE
(MAX_PLANS_PER_QUERY = 20,
 MAX_STORAGE_SIZE_MB = 1000,
 QUERY_CAPTURE_MODE = CUSTOM,
 SIZE_BASED_CLEANUP_MODE = AUTO,
 CLEANUP_POLICY = (STALE_QUERY_THRESHOLD_DAYS = 90,
 WAIT_STATS_CAPTURE_MODE = ON);
 GO
```

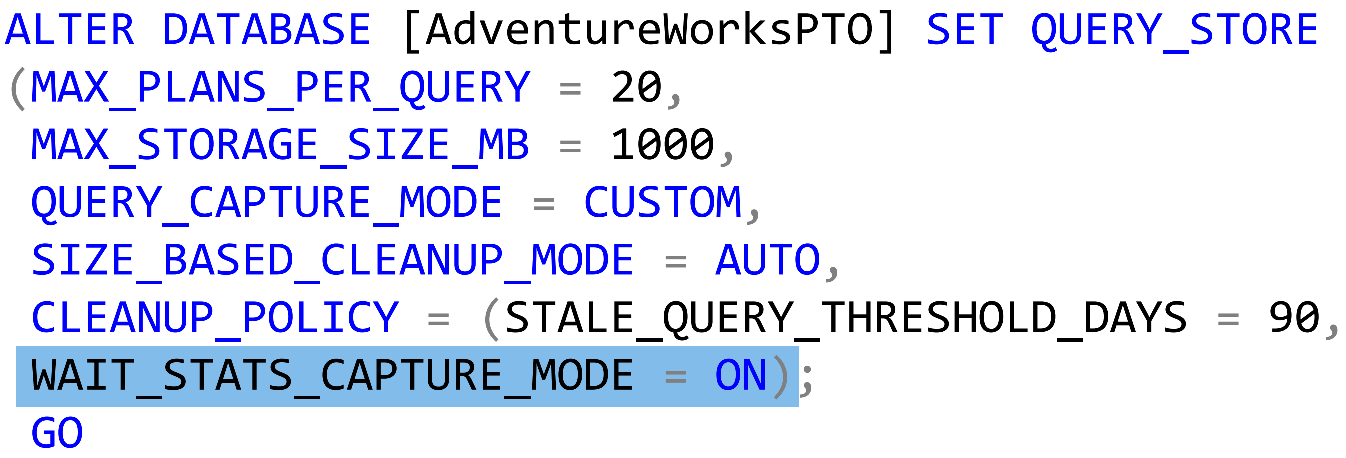
Query Store Retention Settings

Wait Statistics Capture Mode is a new retention setting introduced in SQL Server 2017 that controls if Query Store captures wait statistics information.

(Default = **ON**).



Query Store Retention	
Max Plans Per Query	200
Max Size (MB)	100
Query Store Capture Mode	Custom
Size Based Cleanup Mode	Auto
Stale Query Threshold (Days)	30
Wait Statistics Capture Mode	On

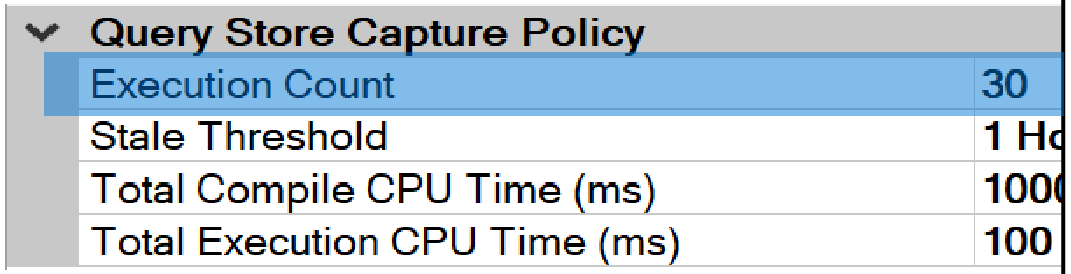


```
ALTER DATABASE [AdventureWorksPTO] SET QUERY_STORE
(MAX_PLANS_PER_QUERY = 20,
 MAX_STORAGE_SIZE_MB = 1000,
 QUERY_CAPTURE_MODE = CUSTOM,
 SIZE_BASED_CLEANUP_MODE = AUTO,
 CLEANUP_POLICY = (STALE_QUERY_THRESHOLD_DAYS = 90,
 WAIT_STATS_CAPTURE_MODE = ON);
 GO
```

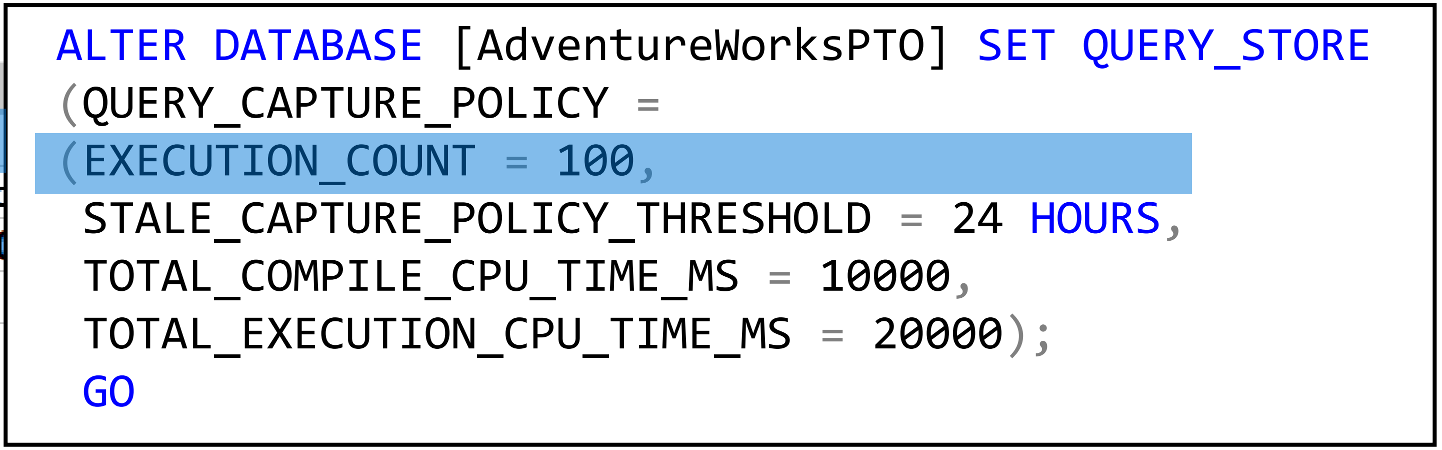
Query Store Capture Policy Settings

Introduced in SQL Server 2019 and available if the Query Store Capture Mode setting has been set to **CUSTOM.**

The value for the **EXECUTION COUNT** is the value a query must exceed within the Stale Threshold time period to be captured by the Query Store.



Query Store Capture Policy	
Execution Count	30
Stale Threshold	1 Ho
Total Compile CPU Time (ms)	1000
Total Execution CPU Time (ms)	100

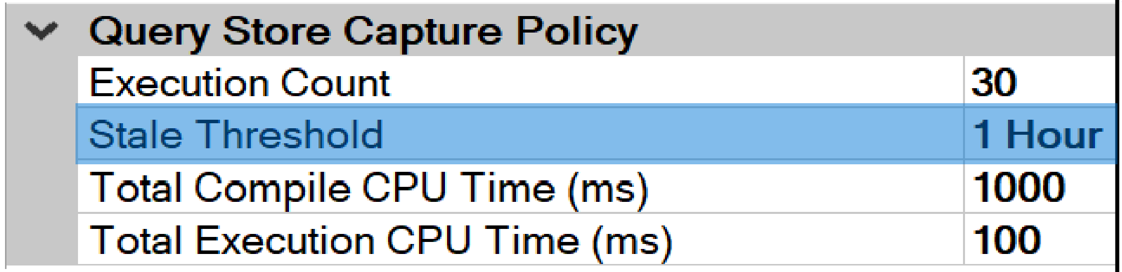


```
ALTER DATABASE [AdventureWorksPTO] SET QUERY_STORE
(QUERY_CAPTURE_POLICY =
(EXECUTION_COUNT = 100,
 STALE_CAPTURE_POLICY_THRESHOLD = 24 HOURS,
 TOTAL_COMPILE_CPU_TIME_MS = 10000,
 TOTAL_EXECUTION_CPU_TIME_MS = 20000);
 GO
```

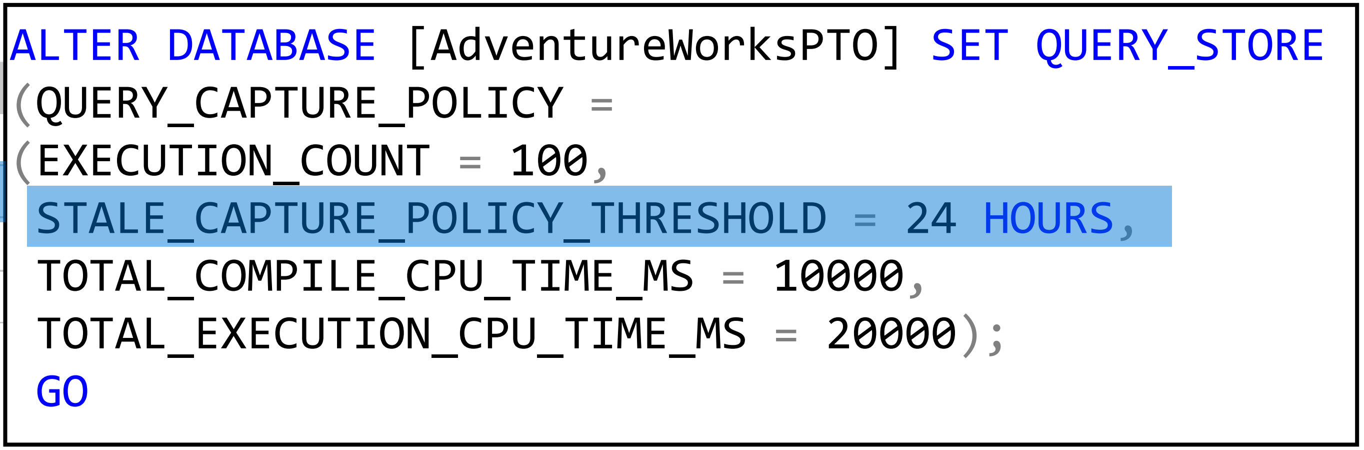
Query Store Capture Policy Settings

Introduced in SQL Server 2019 and available if the Query Store Capture Mode setting has been set to **CUSTOM.**

The value for the **Stale Threshold** can be from 1 hour up to 7 days. This setting specifies the time given to exceed the values of the three other settings for a query to be captured.



Query Store Capture Policy	
Execution Count	30
Stale Threshold	1 Hour
Total Compile CPU Time (ms)	1000
Total Execution CPU Time (ms)	100

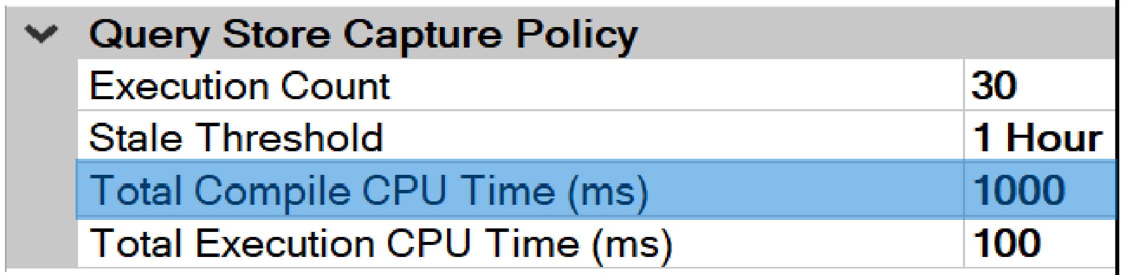


```
ALTER DATABASE [AdventureWorksPTO] SET QUERY_STORE
(QUERY_CAPTURE_POLICY =
(EXECUTION_COUNT = 100,
 STALE_CAPTURE_POLICY_THRESHOLD = 24 HOURS,
 TOTAL_COMPILE_CPU_TIME_MS = 10000,
 TOTAL_EXECUTION_CPU_TIME_MS = 20000);
 GO
```

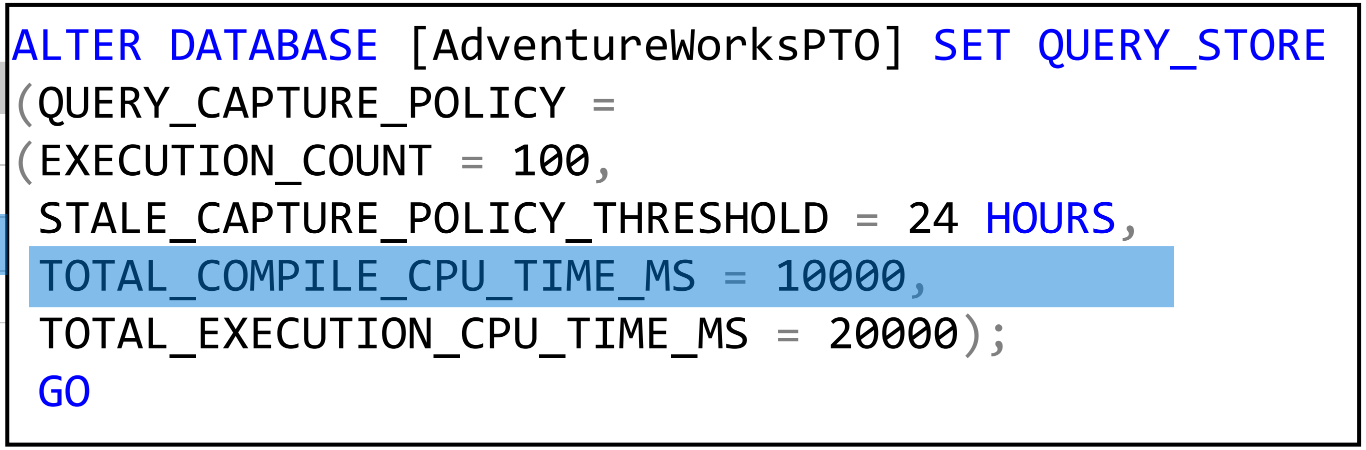
Query Store Capture Policy Settings

Introduced in SQL Server 2019 and available if the Query Store Capture Mode setting has been set to **CUSTOM.**

The value for the **Total Compile CPU Time (ms)** is the value in milliseconds that a query must exceed within the **Stale Threshold** time period to be captured by the Query Store.



Query Store Capture Policy	
Execution Count	30
Stale Threshold	1 Hour
Total Compile CPU Time (ms)	1000
Total Execution CPU Time (ms)	100

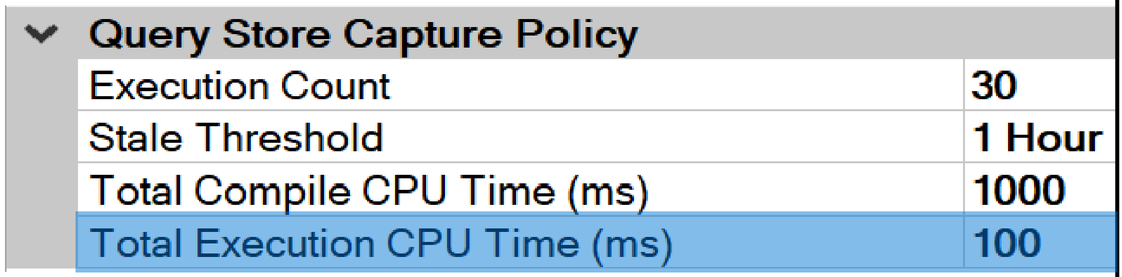


```
ALTER DATABASE [AdventureWorksPTO] SET QUERY_STORE
(QUERY_CAPTURE_POLICY =
(EXECUTION_COUNT = 100,
 STALE_CAPTURE_POLICY_THRESHOLD = 24 HOURS,
 TOTAL_COMPILE_CPU_TIME_MS = 10000,
 TOTAL_EXECUTION_CPU_TIME_MS = 20000);
 GO
```

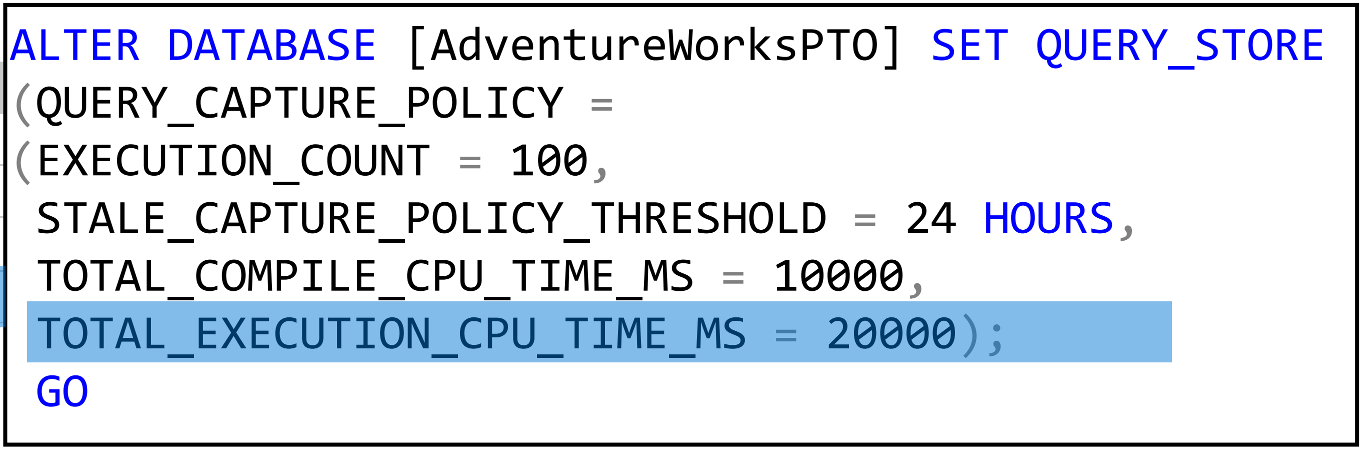
Query Store Capture Policy Settings

Introduced in SQL Server 2019 and available if the Query Store Capture Mode setting has been set to **CUSTOM.**

The value for the **Total Execution CPU Time (ms)** is the value in milliseconds that a query must exceed within the **Stale Threshold** time period to be captured by the Query Store.



Query Store Capture Policy	
Execution Count	30
Stale Threshold	1 Hour
Total Compile CPU Time (ms)	1000
Total Execution CPU Time (ms)	100

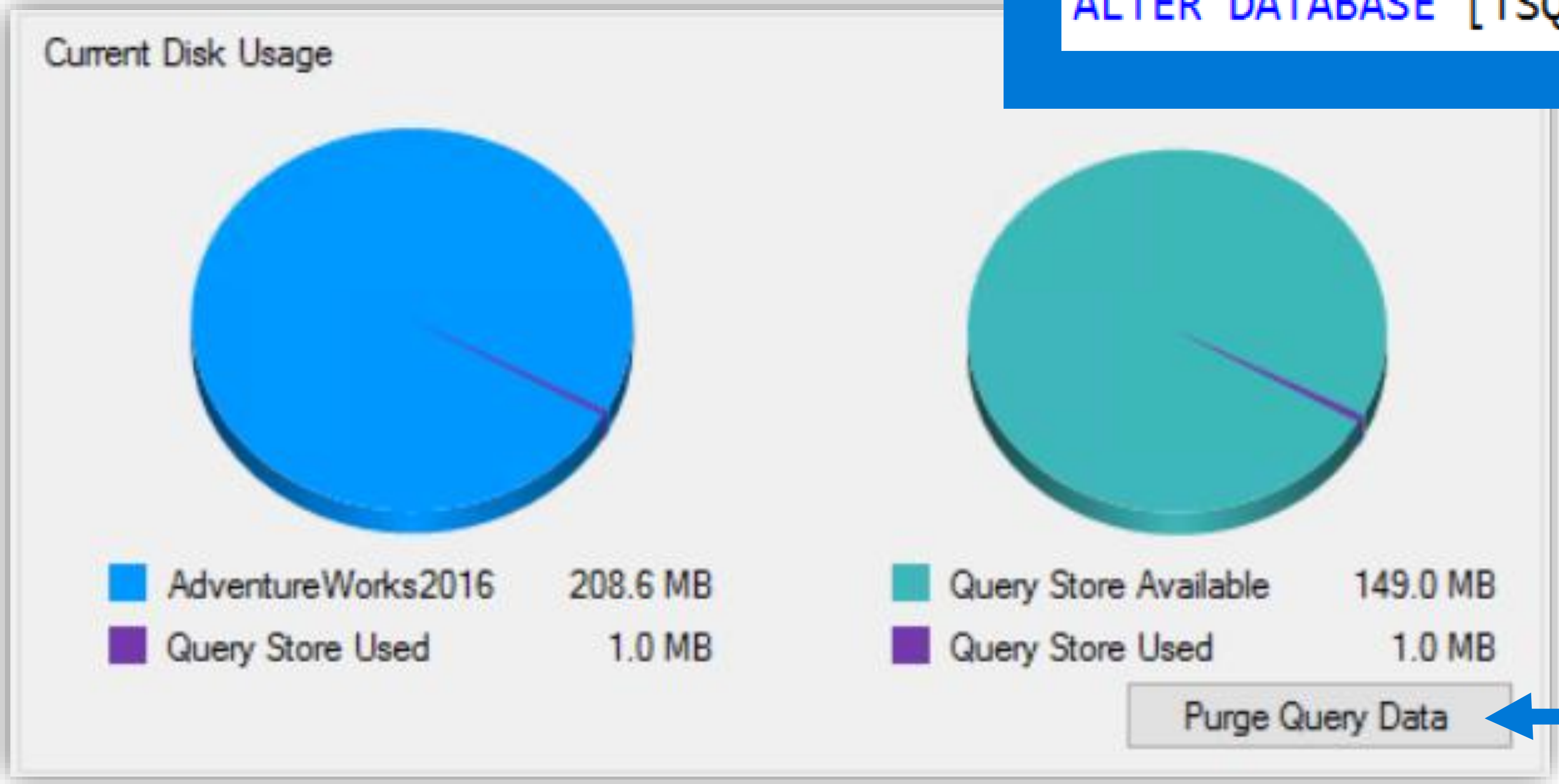


```
ALTER DATABASE [AdventureWorksPTO] SET QUERY_STORE
(QUERY_CAPTURE_POLICY =
(EXECUTION_COUNT = 100,
 STALE_CAPTURE_POLICY_THRESHOLD = 24 HOURS,
 TOTAL_COMPILE_CPU_TIME_MS = 10000,
 TOTAL_EXECUTION_CPU_TIME_MS = 20000);
 GO
```

Purge Query Data

Data can be manually purged from the Query Store.

```
ALTER DATABASE [TSQL] SET QUERY_STORE CLEAR;
```

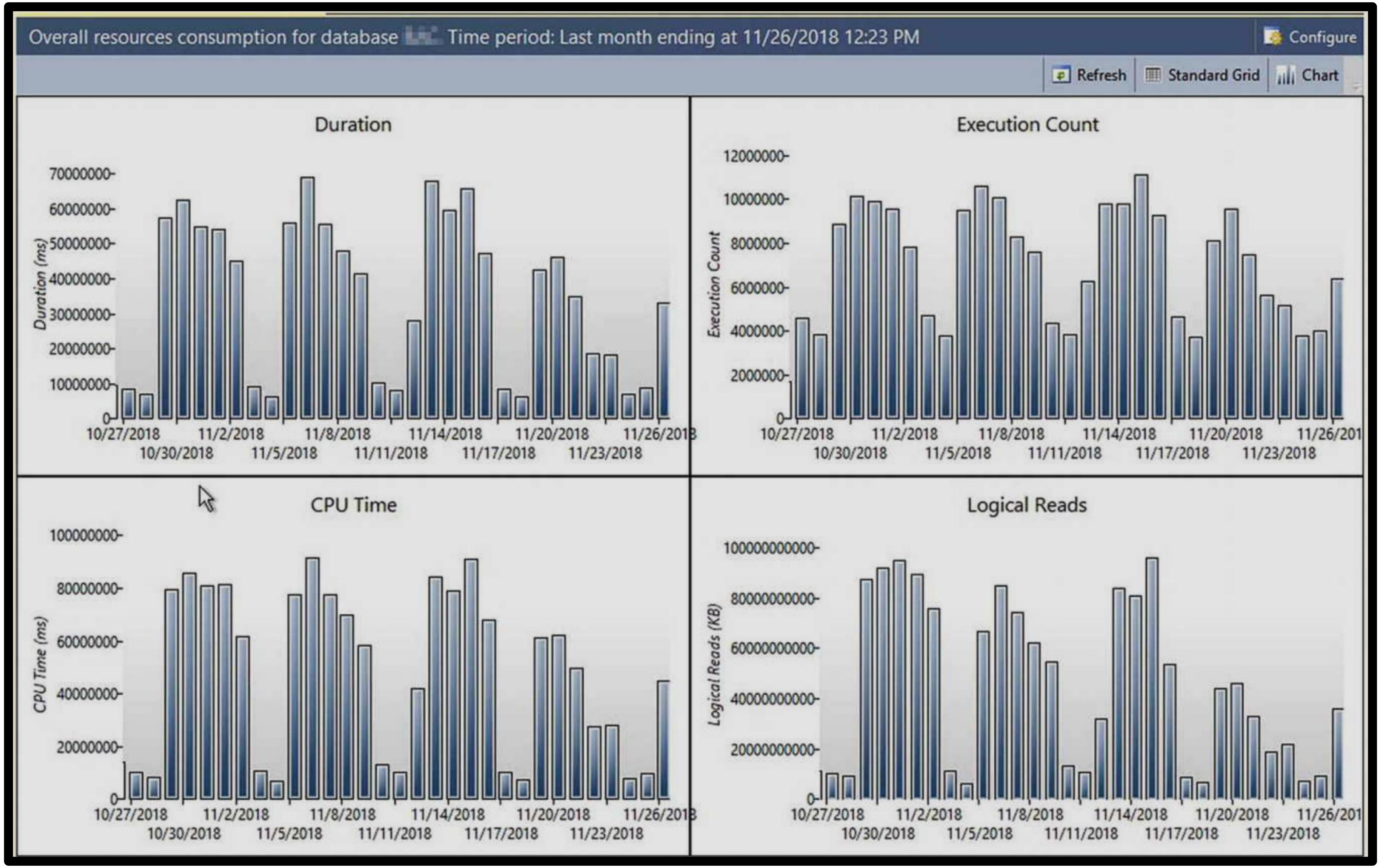


Built-in Reports

Regressed queries	• Automatic detection of queries that have begun executing more slowly
Overall resource consumption	• Historic view across 4 performance metrics of your choice
Top Resource-consuming queries	• Which queries are the costliest to execute
Queries with forced plans	• Check on the performance of queries using forced plans
Queries with high variation	• Inconsistently performing queries may need tuning
Query wait statistics	• Identify performance bottlenecks
Tracked queries	• Focus on plans and metrics for a single query

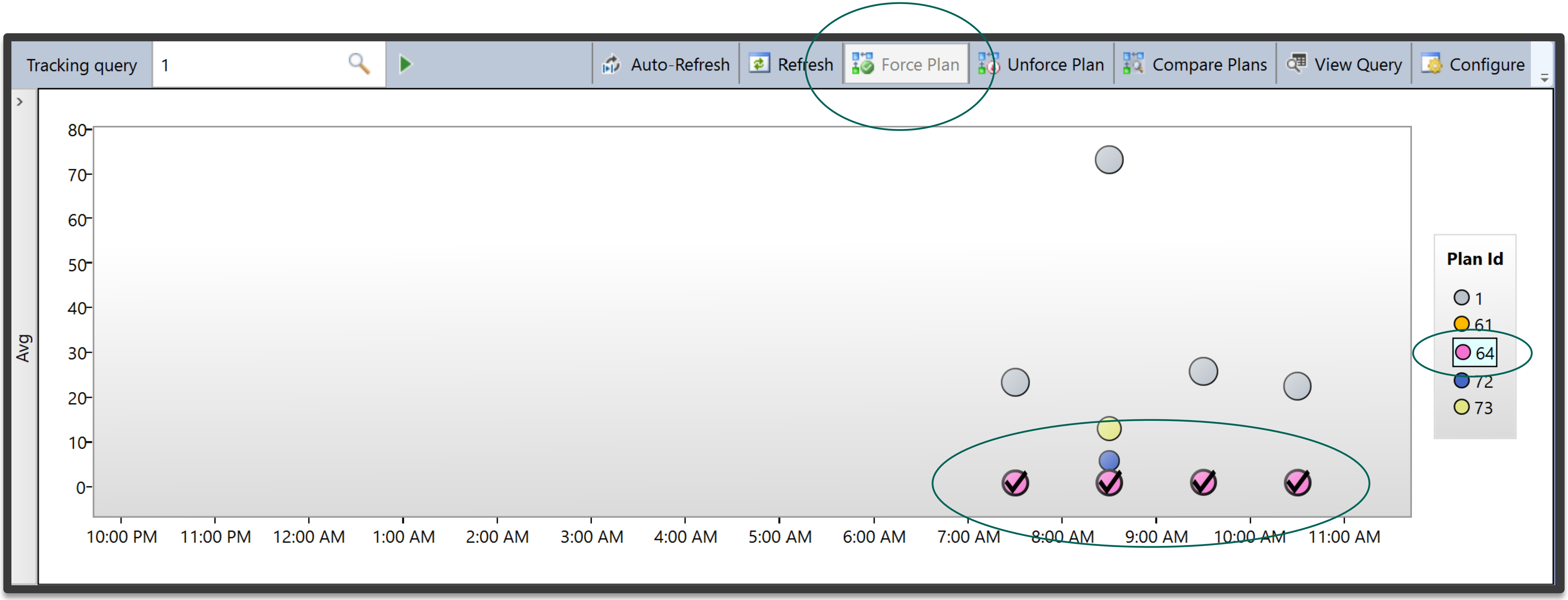
```sql
-- Permission to view reports
GRANT VIEW DATABASE STATE TO <UserName>;
```

Establishing a Baseline

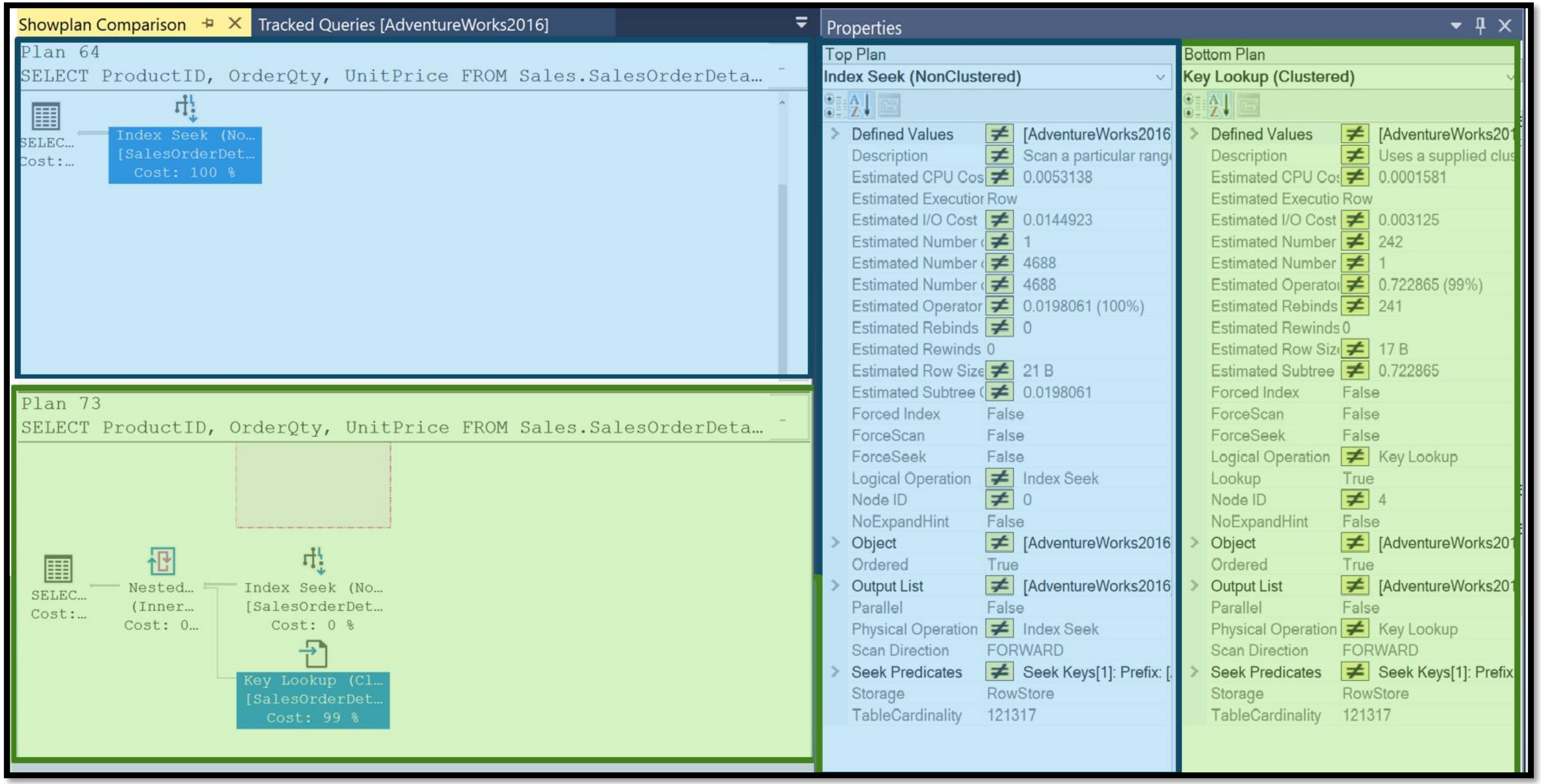


Overall resources consumption for database [redacted] Time period: Last month ending at 11/26/2018 12:23 PM

Force Plan


Tracking query
1
Auto-Refresh
Refresh
Force Plan
Unforce Plan
Compare Plans
View Query
Configure
Avg
80
70
60
50
40
30
20
10
0
10:00 PM
11:00 PM
12:00 AM
1:00 AM
2:00 AM
3:00 AM
4:00 AM
5:00 AM
6:00 AM
7:00 AM
8:00 AM
9:00 AM
10:00 AM
11:00 AM
Plan Id
1
61
64
72
73

Plan Compare



```
Plan 64
SELECT ProductID, OrderQty, UnitPrice FROM Sales.SalesOrderDeta…
```

```
Plan 73
SELECT ProductID, OrderQty, UnitPrice FROM Sales.SalesOrderDeta…
```

Top Plan

Index Seek (NonClustered)

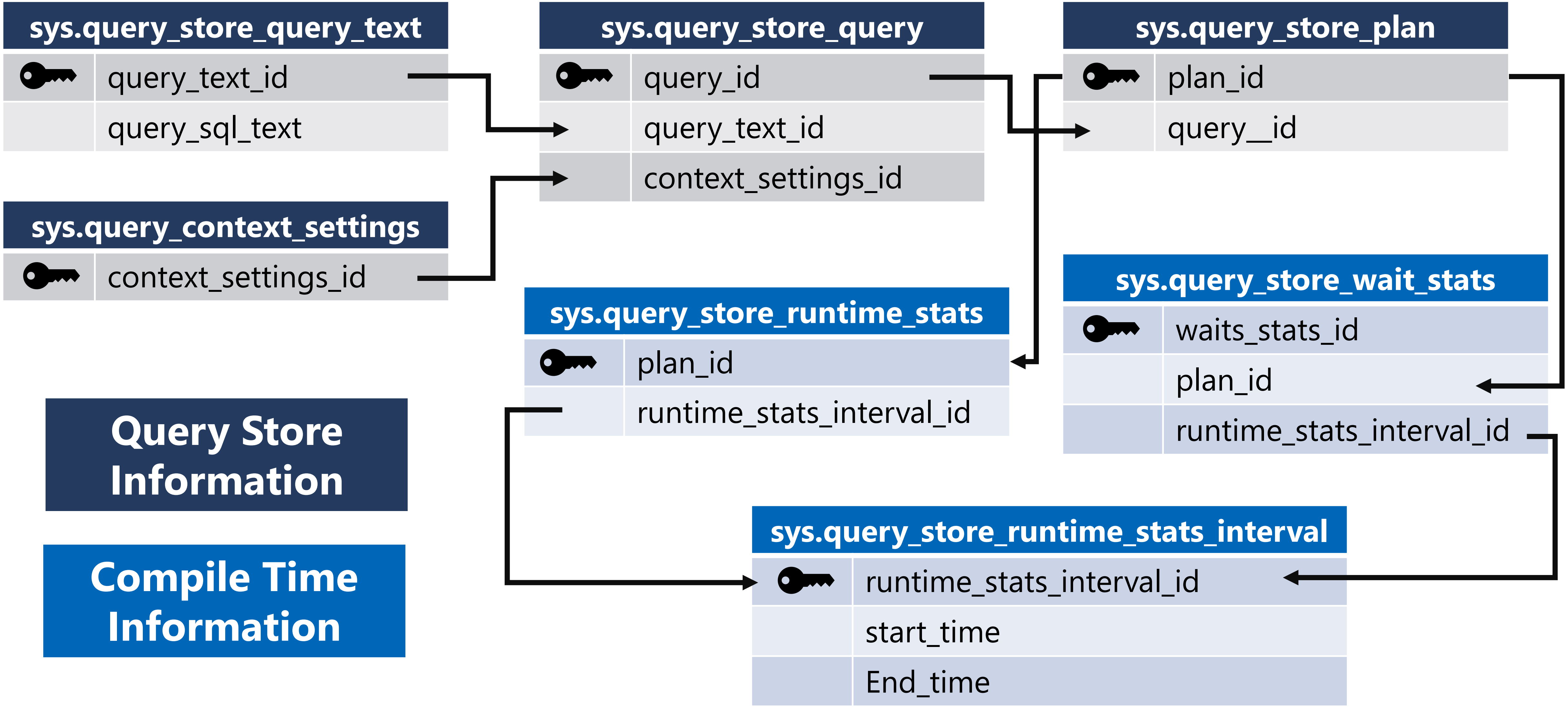
Defined Values	≠	[AdventureWorks2016
Description	≠	Scan a particular range
Estimated CPU Cos	≠	0.0053138
Estimated Executior		Row
Estimated I/O Cost	≠	0.0144923
Estimated Number (	≠	1
Estimated Number (	≠	4688
Estimated Number (	≠	4688
Estimated Operator	≠	0.0198061 (100%)
Estimated Rebinds	≠	0
Estimated Rewinds		0
Estimated Row Size	≠	21 B
Estimated Subtree (	≠	0.0198061
Forced Index		False
ForceScan		False
ForceSeek		False
Logical Operation	≠	Index Seek
Node ID	≠	0
NoExpandHint		False
Object	≠	[AdventureWorks2016
Ordered		True
Output List	≠	[AdventureWorks2016
Parallel		False
Physical Operation	≠	Index Seek
Scan Direction		FORWARD
Seek Predicates	≠	Seek Keys[1]: Prefix: [
Storage		RowStore
TableCardinality		121317

Bottom Plan

Key Lookup (Clustered)

Defined Values	≠	[AdventureWorks201
Description	≠	Uses a supplied clus
Estimated CPU Co:	≠	0.0001581
Estimated Executio		Row
Estimated I/O Cost	≠	0.003125
Estimated Number	≠	242
Estimated Number	≠	1
Estimated Operato	≠	0.722865 (99%)
Estimated Rebinds	≠	241
Estimated Rewinds		0
Estimated Row Siz	≠	17 B
Estimated Subtree	≠	0.722865
Forced Index		False
ForceScan		False
ForceSeek		False
Logical Operation	≠	Key Lookup
Lookup		True
Node ID	≠	4
NoExpandHint		False
Object	≠	[AdventureWorks201
Ordered		True
Output List	≠	[AdventureWorks201
Parallel		False
Physical Operation	≠	Key Lookup
Scan Direction		FORWARD
Seek Predicates	≠	Seek Keys[1]: Prefix
Storage		RowStore
TableCardinality		121317

Query Store Catalog Views



	sys.query_store_query_text
🔑	query_text_id
	query_sql_text

	sys.query_context_settings
🔑	context_settings_id

	sys.query_store_query
🔑	query_id
	query_text_id
	context_settings_id

	sys.query_store_plan
🔑	plan_id
	query__id

	sys.query_store_runtime_stats
🔑	plan_id
	runtime_stats_interval_id

	sys.query_store_wait_stats
🔑	waits_stats_id
	plan_id
	runtime_stats_interval_id

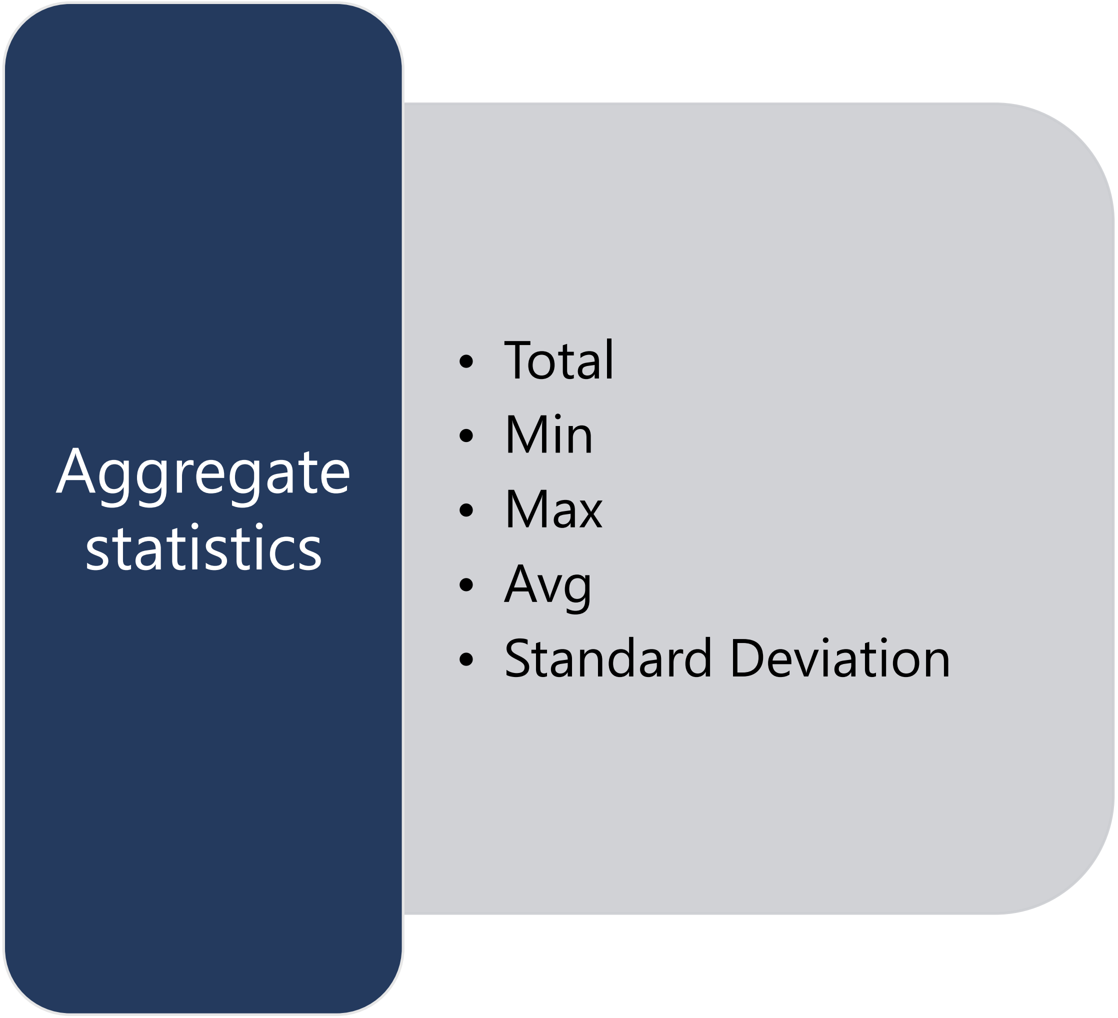
	sys.query_store_runtime_stats_interval
🔑	runtime_stats_interval_id
	start_time
	End_time

Query Store Information

Compile Time Information

Runtime Metrics and Statistics

- Execution count
- Duration
- CPU
- Logical reads
- Logical writes
- Physical reads
- CLR Time
- DOP
- Memory consumption
- Row Count
- Log memory used
- Tempdb memory used
- Wait time



Aggregate statistics

- Total
- Min
- Max
- Avg
- Standard Deviation

Using Query Store Catalog Views

Finding the TOP 10 most frequently executed SQL Server Queries in the Query Store.

```sql
SELECT TOP 10 t.query_sql_text, q.query_id
FROM sys.query_store_query_text as t
JOIN sys.query_store_query as q
ON t.query_text_id = q.query_text_id
JOIN sys.query_store_plan as p
ON q.query_id = p.query_id
JOIN sys.query_store_runtime_stats as rs
ON p.plan_id = rs.plan_id
WHERE rs.count_executions >1
GROUP BY t.query_sql_text, q.query_id
ORDER BY SUM(rs.count_executions)
```

Query Store read replica support for Availability Groups

New feature in SQL Server 2022 – currently in preview

Execution metrics for queries run on secondary replicas

Data is sent from secondaries back to the primary replica

Persisted in the primary replica’s Query Store

You must enable trace flag 12606 before you can enable Query Store for secondary replicas.

Considerations

- Sharing bandwidth with outgoing transaction records
- A shared Query Store will be larger
- Impact of *ad hoc* workloads run on secondary replicas

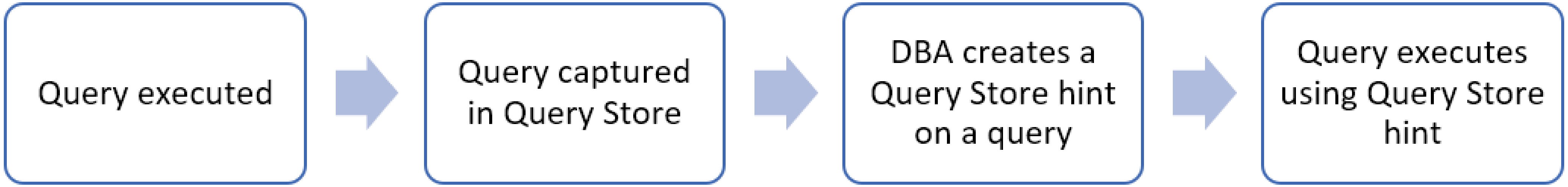
Query Store hints

Query Store hints are available in Azure SQL Database and Azure SQL Managed Instance. Query Store hints are also a feature introduced to SQL Server in SQL Server 2022 (16.x).

As the name suggests, this feature extends and depends on the **Query Store**.

Not all query hints are supported. Here is a list of **Supported query hints.**

Because the SQL Server Query Optimizer typically selects the best execution plan for a query, we recommend only using hints as a last resort. For more information, **Query Hints**.



Query executed

Query captured in Query Store

DBA creates a Query Store hint on a query

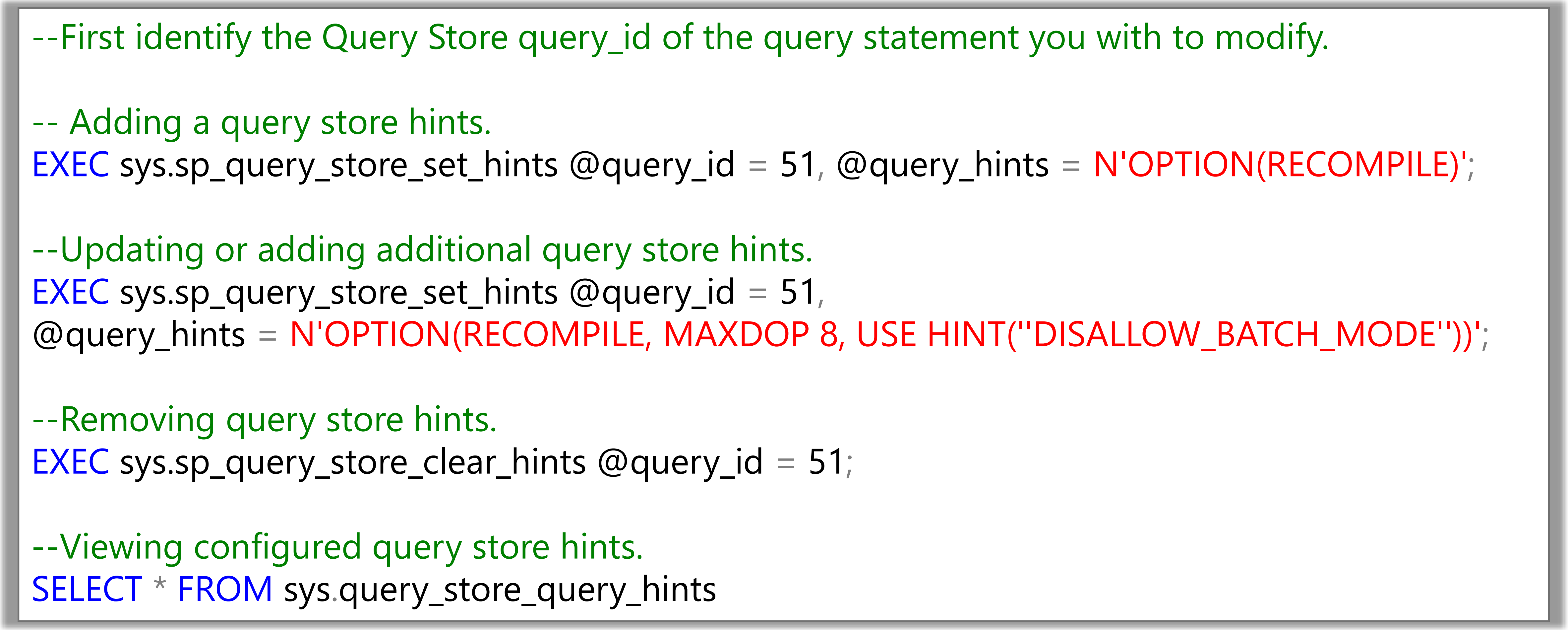
Query executes using Query Store hint

Query Store hints – Use Cases

Use Cases

- When code can’t be changed
- Override other hints/plan guides
- Recompile a query on each execution.
- Cap the memory grant size for a bulk insert operation.
- Limit the maximum degree of parallelism when updating statistics.
- 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.

Setting, clearing, and viewing query store hints.



```
--First identify the Query Store query_id of the query statement you with to modify.

-- Adding a query store hints.
EXEC sys.sp_query_store_set_hints @query_id = 51, @query_hints = N'OPTION(RECOMPILE)';

--Updating or adding additional query store hints.
EXEC sys.sp_query_store_set_hints @query_id = 51,
@query_hints = N'OPTION(RECOMPILE, MAXDOP 8, USE HINT(''DISALLOW_BATCH_MODE''))';

--Removing query store hints.
EXEC sys.sp_query_store_clear_hints @query_id = 51;

--Viewing configured query store hints.
SELECT * FROM sys.query_store_query_hints
```

Troubleshooting Using the Query Store

LAB

Questions?

Knowledge Check

If upgrading from SQL Server 2012 to 2019. Which report should figure prominently in your upgrade plans?

In a report’s bar chart what does each bar represent?

Which report can help troubleshoot a parameter sniffing issue?

Querying the wait statistics DMV it returns high PAGEIOLATCH waits. Which report can help identify queries with high IO wait times?

Someone has dropped an index needed by a forced plan. What happens the next time the query executes? What happens if the index is recreated?

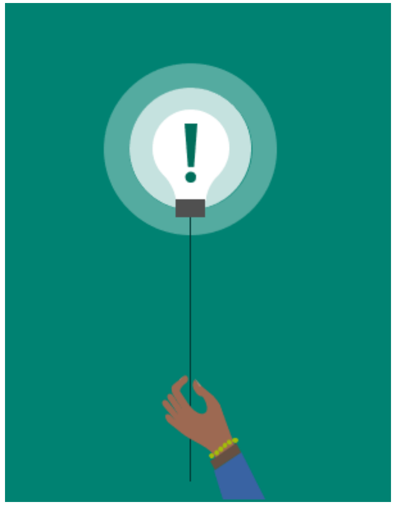
Questions?

Lesson 3: SQL Server Intelligent Query Processing

Objectives

After completing this learning, you will be able to:

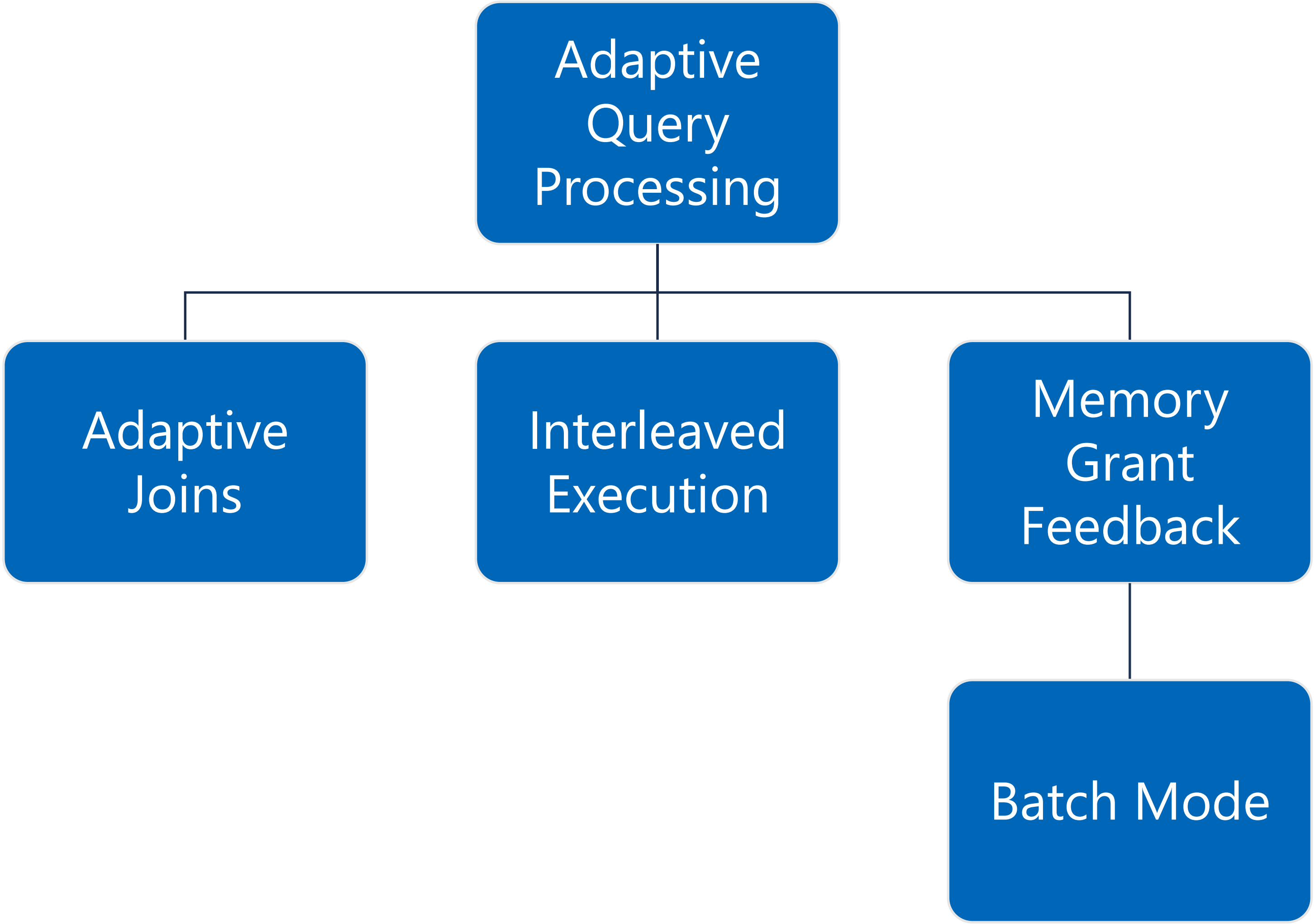
- Understand the Intelligent query processing features.
- Enable/disable Intelligent query processing features.
- A detailed version of this lesson is in Module 10.



A History of Intelligent Query Processing

- Adaptive Query Processing (2017)
- Intelligent Query Processing (2019)
- New Features of IQP (2022)

Adaptive Query Processing (2017)



Adaptive Query Processing

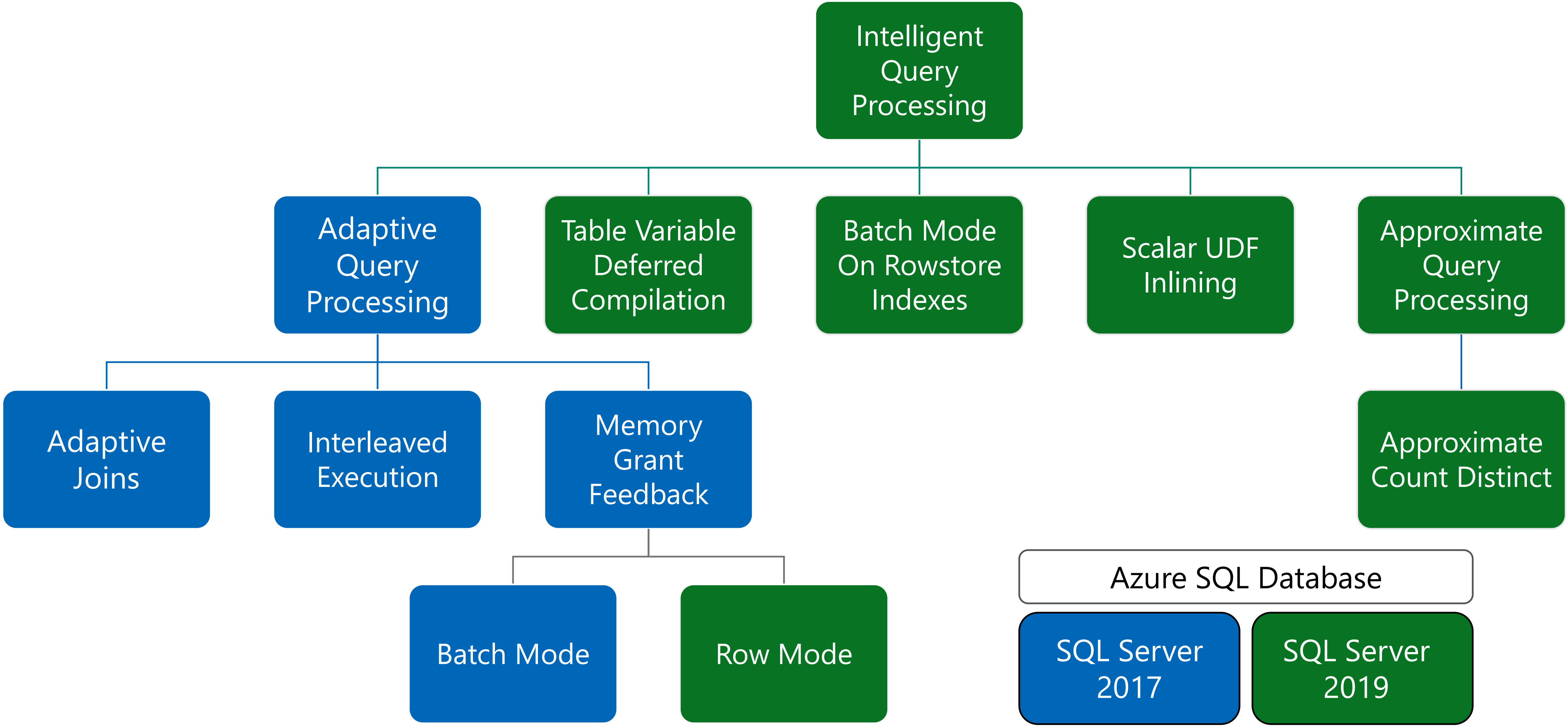
Adaptive Joins

Interleaved Execution

Memory Grant Feedback

Batch Mode

Intelligent Query Processing (2019)



Intelligent Query Processing

Adaptive Query Processing

Table Variable Deferred Compilation

Batch Mode On Rowstore Indexes

Scalar UDF Inlining

Approximate Query Processing

Adaptive Joins

Interleaved Execution

Memory Grant Feedback

Approximate Count Distinct

Batch Mode

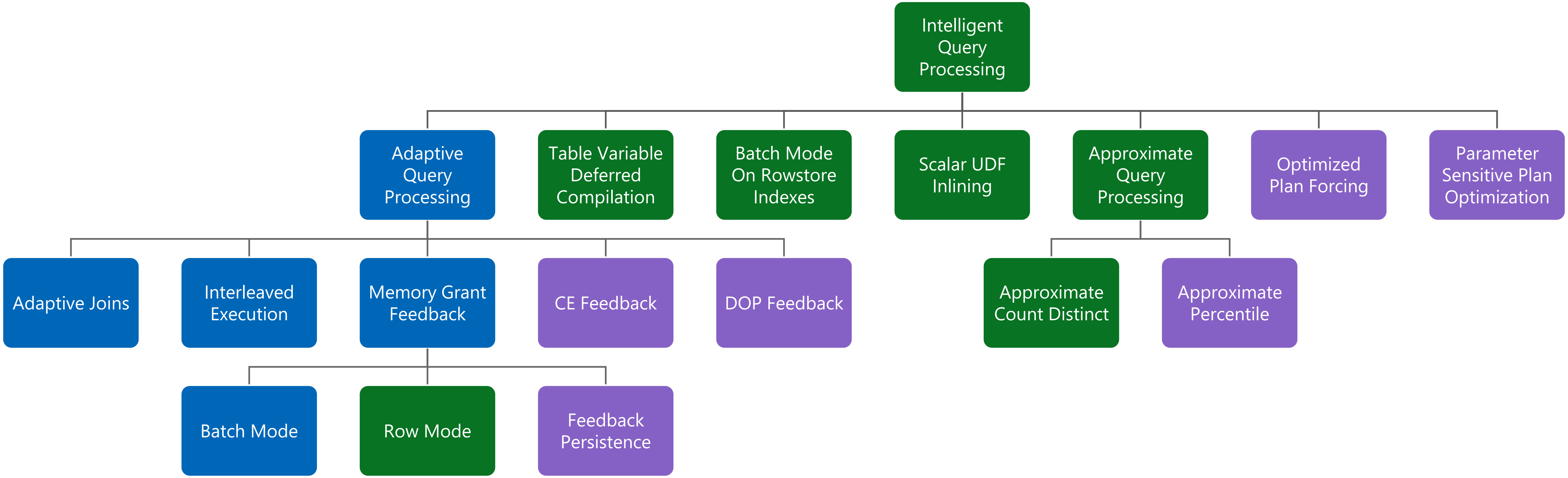
Row Mode

Azure SQL Database

SQL Server 2017

SQL Server 2019

Intelligent Query Processing (2022)



- Intelligent Query Processing
  - Adaptive Query Processing
    - Adaptive Joins
    - Interleaved Execution
    - Memory Grant Feedback
      - Batch Mode
      - Row Mode
      - Feedback Persistence
    - CE Feedback
    - DOP Feedback
  - Table Variable Deferred Compilation
  - Batch Mode On Rowstore Indexes
  - Scalar UDF Inlining
  - Approximate Query Processing
    - Approximate Count Distinct
    - Approximate Percentile
  - Optimized Plan Forcing
  - Parameter Sensitive Plan Optimization

Azure SQL Database

2017 | 2019 | 2022

https://aka.ms/IQP

Enabling and Disabling – Database Level

For SQL Server 2017 Features

- Enabled by default in Compatibility level 140 or higher
- To disable change compatibility level to 130 or lower

For SQL Server 2019 Features

- Enabled by default in Compatibility level 150 or higher
- To disable change compatibility level to 140 or lower

For SQL Server 2022 Features

- Enabled by default in Compatibility level 160 or higher
- To disable change compatibility level to 150 or lower

Enabling and Disabling – Database Level

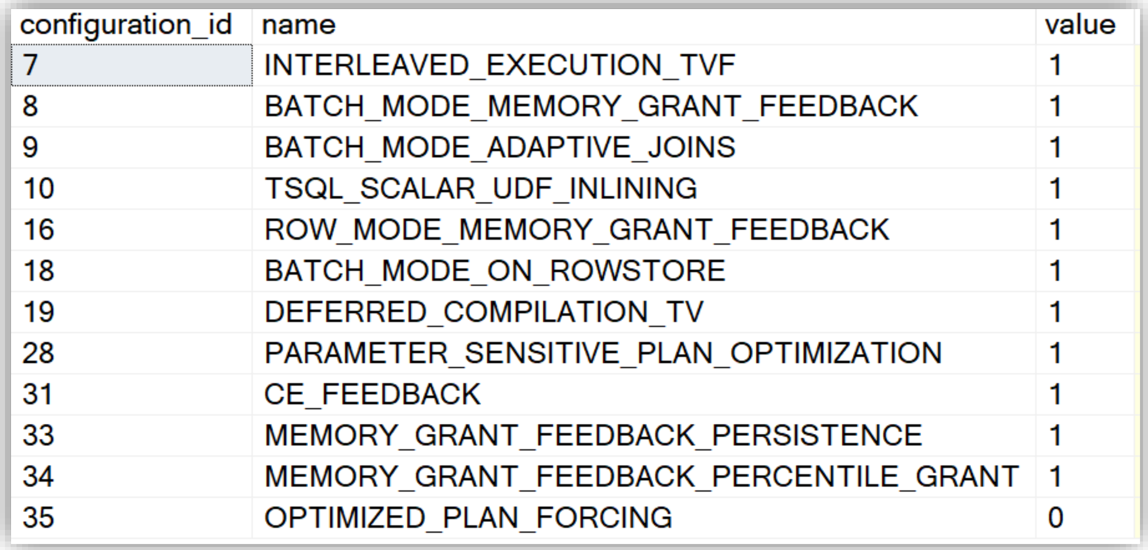
Different settings for 2017 vs Azure SQL, SQL Server 2019 and higher

```
ALTER DATABASE SCOPED CONFIGURATION SET DISABLE_BATCH_MODE_ADAPTIVE_JOINS = ON|OFF;
```

```
ALTER DATABASE SCOPED CONFIGURATION SET BATCH_MODE_ADAPTIVE_JOINS = ON|OFF;
```

To get a list of Database Scoped Configuration settings

```
SELECT * From sys.database_scoped_configurations;
```



configuration_id	name	value
7	INTERLEAVED_EXECUTION_TVF	1
8	BATCH_MODE_MEMORY_GRANT_FEEDBACK	1
9	BATCH_MODE_ADAPTIVE_JOINS	1
10	TSQL_SCALAR_UDF_INLINING	1
16	ROW_MODE_MEMORY_GRANT_FEEDBACK	1
18	BATCH_MODE_ON_ROWSTORE	1
19	DEFERRED_COMPILATION_TV	1
28	PARAMETER_SENSITIVE_PLAN_OPTIMIZATION	1
31	CE_FEEDBACK	1
33	MEMORY_GRANT_FEEDBACK_PERSISTENCE	1
34	MEMORY_GRANT_FEEDBACK_PERCENTILE_GRANT	1
35	OPTIMIZED_PLAN_FORCING	0

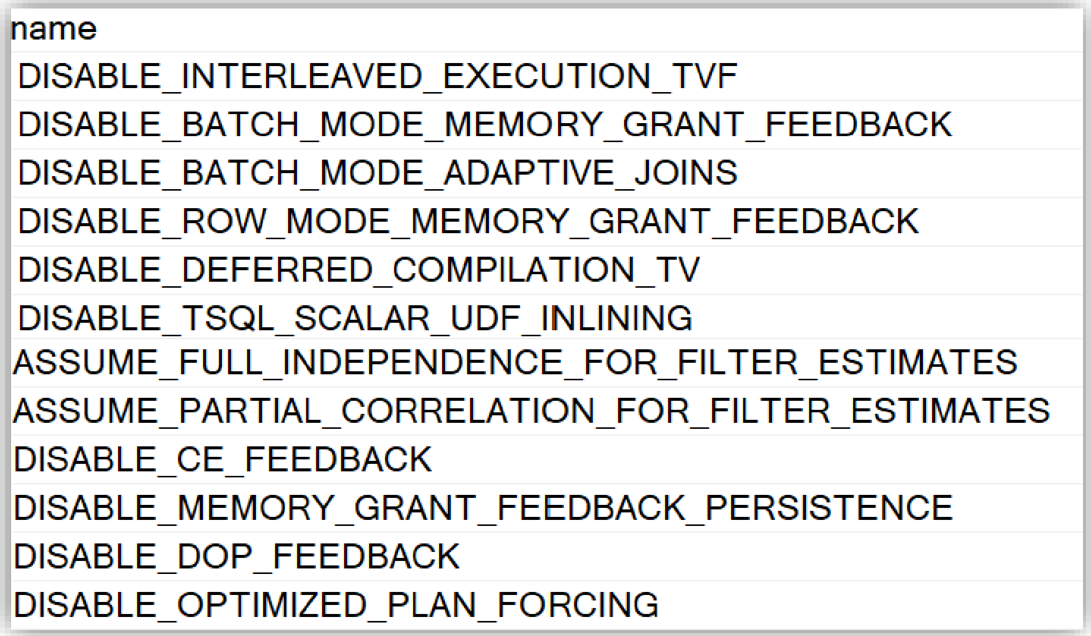
Enabling and Disabling – Statement Level

You can disable features at the statement scope if necessary.

```
<statement>
OPTION (USE HINT('DISABLE_BATCH_MODE_ADAPTIVE_JOINS'));
```

To get a list of valid query use hints

```
SELECT * FROM sys.dm_exec_valid_use_hints;
```



name
DISABLE_INTERLEAVED_EXECUTION_TVF
DISABLE_BATCH_MODE_MEMORY_GRANT_FEEDBACK
DISABLE_BATCH_MODE_ADAPTIVE_JOINS
DISABLE_ROW_MODE_MEMORY_GRANT_FEEDBACK
DISABLE_DEFERRED_COMPILATION_TV
DISABLE_TSQL_SCALAR_UDF_INLINING
ASSUME_FULL_INDEPENDENCE_FOR_FILTER_ESTIMATES
ASSUME_PARTIAL_CORRELATION_FOR_FILTER_ESTIMATES
DISABLE_CE_FEEDBACK
DISABLE_MEMORY_GRANT_FEEDBACK_PERSISTENCE
DISABLE_DOP_FEEDBACK
DISABLE_OPTIMIZED_PLAN_FORCING

Questions?

Microsoft