Microsoft

SQL Server Query Execution and Plans

Module 5

Learning Units covered in this Module

- Lesson 1: SQL Server Query Execution

- Lesson 2: SQL Server Query Plan Analysis

- Lesson 3: SQL Server Query Plan Analysis

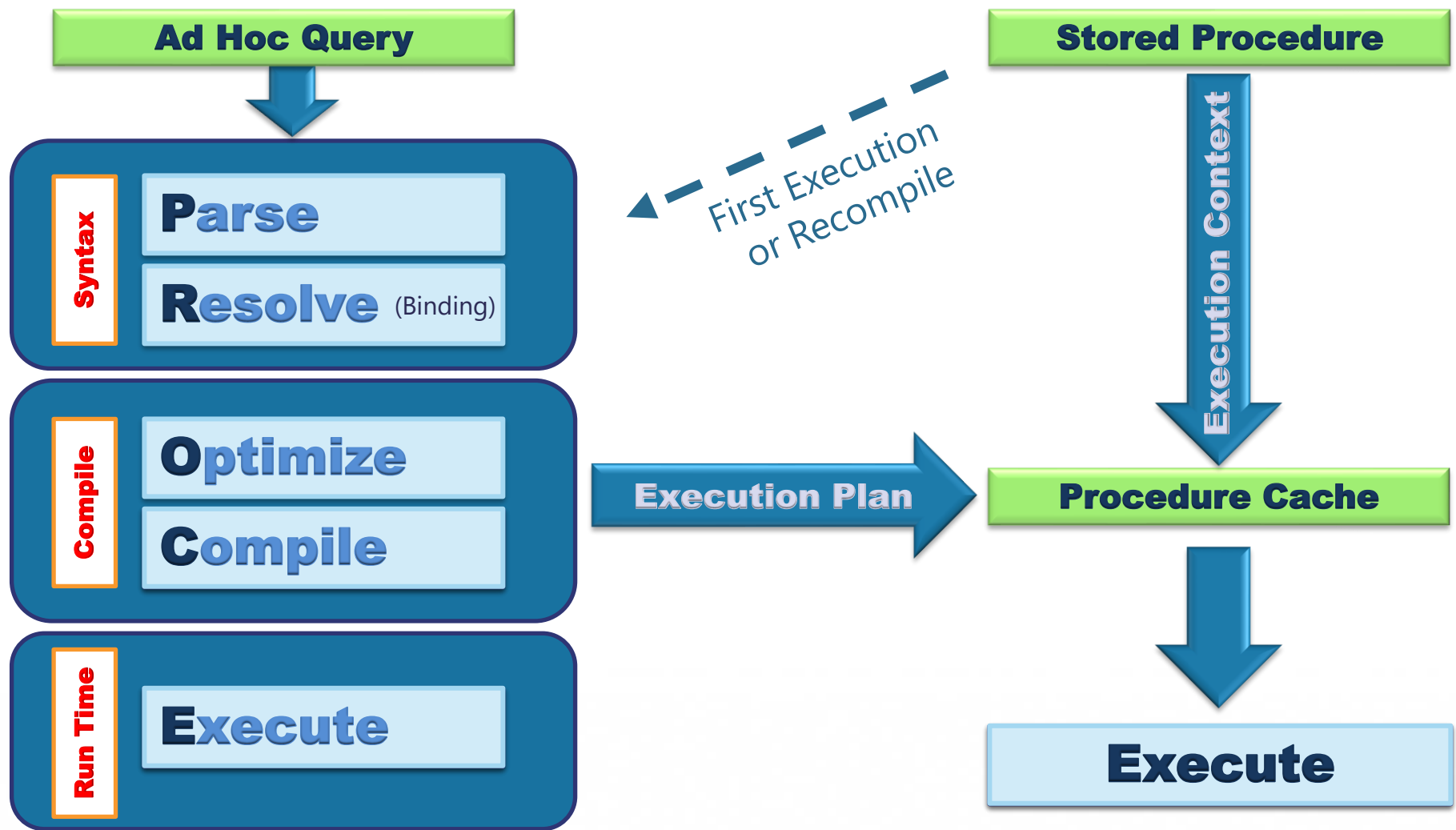- Lesson 4: Troubleshooting with Query Store

Lesson 1: SQL Server Query Execution

Objectives

After completing this learning, you will be able to:

- Explain Query Compilation and Optimization Process.

- Explain Query Execution Process.

- Explain Recompilation causes.

| empid	lastname	firstna...	title	titleofcourt...	birthdate
1	Davis	Sara	CEO	Ms.	1958-12-08 00:00:00.000
2	Funk	Don	Vice President, Sales	Dr.	1962-02-19 00:00:00.000
3	Lew	Judy	Sales Manager	Ms.	1973-08-30 00:00:00.000
4	Peled	Yael	Sales Representative	Mrs.	1947-09-19 00:00:00.000
5	Buck	Sven	Sales Manager	Mr.	1965-03-04 00:00:00.000

What does the binding step resolve?

User permissions are checked.

Does a cached plan exist?

Object names (Tables, Views, Columns, etc.) to see if they exist.

Resolve aliases of columns and tables

Data types and if implicit data type conversions are needed.

Query Optimizer Phases

SQL Server Query Optimization
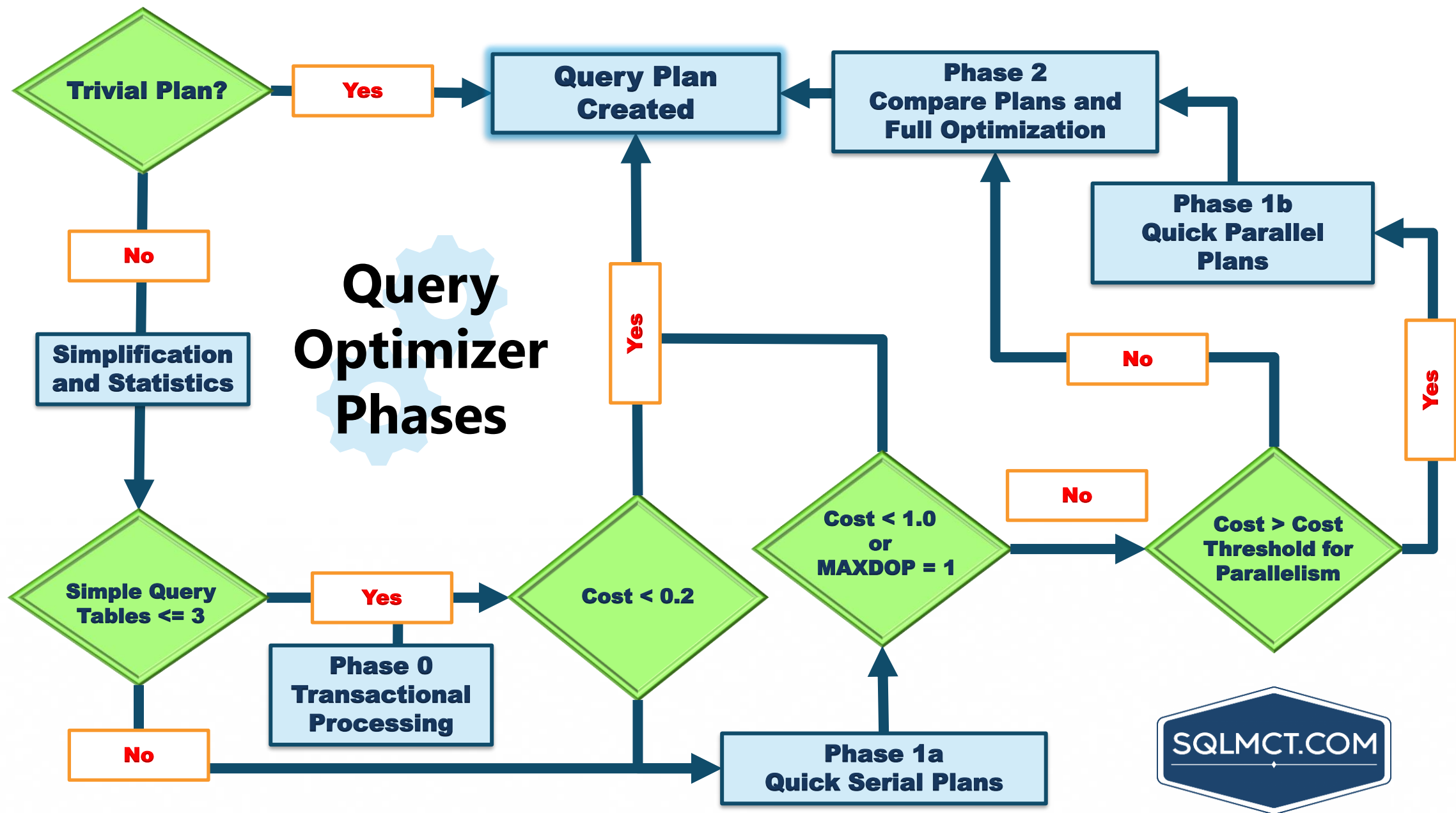
Query Simplification phases

Constant Folding: Expressions with constant values are reduced

- **Quantity = 2 + 3** becomes **Quantity = 5**
- **10 < 20** becomes **True**

Contradiction Detection: Removes criteria that doesn't match table constraints

- **Constraint:** Age > 18
- **Contradiction:** WHERE Age < 18

Domain Simplification: Reduces complex ranges to simple ranges
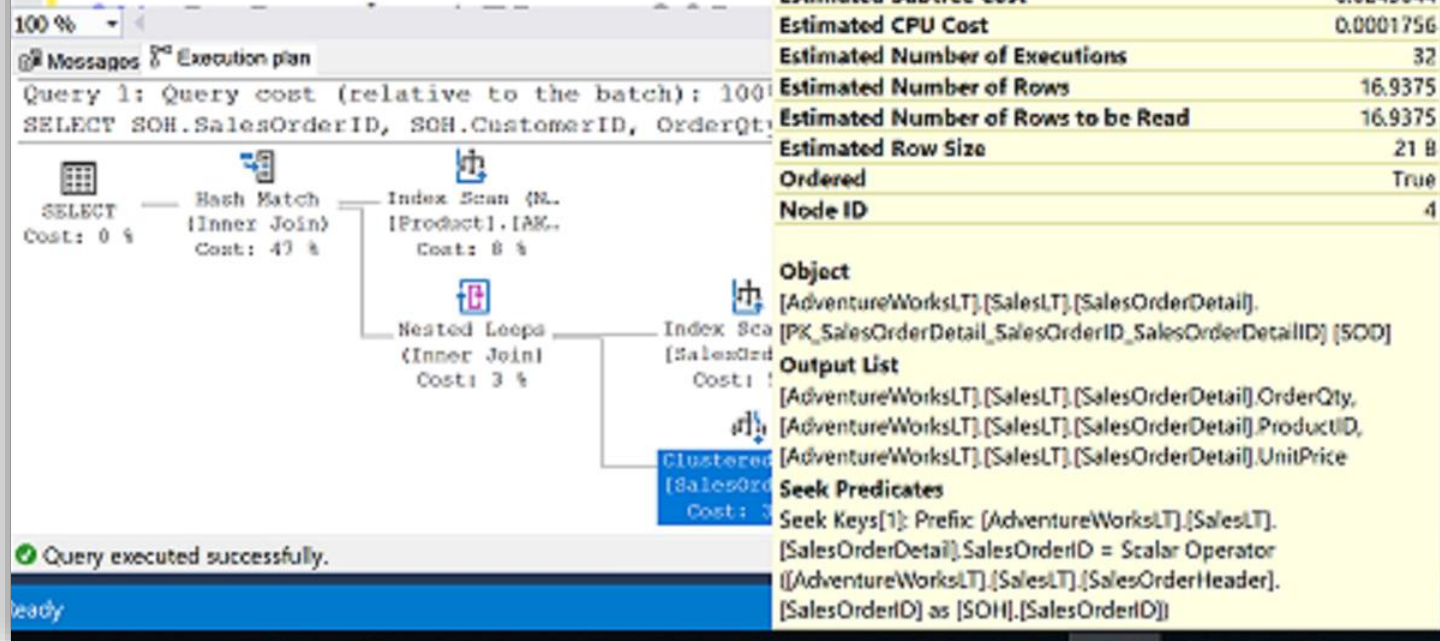
- **Complex range:** ID > 10 and ID < 20 or ID > 30 and < 50
- **Simplified range:** ID > 10 and < 50

Join Simplification: Removes redundant joins that are not necessary

Predicate Pushdown: Perform calculations only on rows returned

What is an Execution Plan?

How to see the query plan

Graphical execution plan

Estimated Execution Plan (Before Execution)

- The compiled plan.

Actual Execution Plan (After Execution)

- The same as the compiled plan plus its execution context.
- This includes runtime information available after the execution completes, such as execution warnings, or in newer versions of the Database Engine, the elapsed and CPU time used during execution.

Live Query Statistics (During Execution)

- The same as the compiled plan plus its execution context.
- This includes runtime information during execution progress and is updated every second. Runtime information includes for example the actual number of rows flowing through the operators.
- Enables rapid identification of potential bottlenecks.

SQL Server Execution Plan
Execution modes

Row mode execution

- Efficient for OLTP scenarios
- Used with traditional tables, where data is stored in row-wise format.
- Operators read all columns from qualifying rows based on predicate, and for each row it retrieves columns needed for the result set

Batch mode execution

- Efficient for Data Warehousing scenarios
- Used to process multiple rows together
- Closely integrated with, and optimized around the ColumnStore storage format
- Operators read only the columns required for the result, from group of rows together.
- Starting SQL Server 2019, Batch mode does not require a ColumnStore index as in the previous versions.
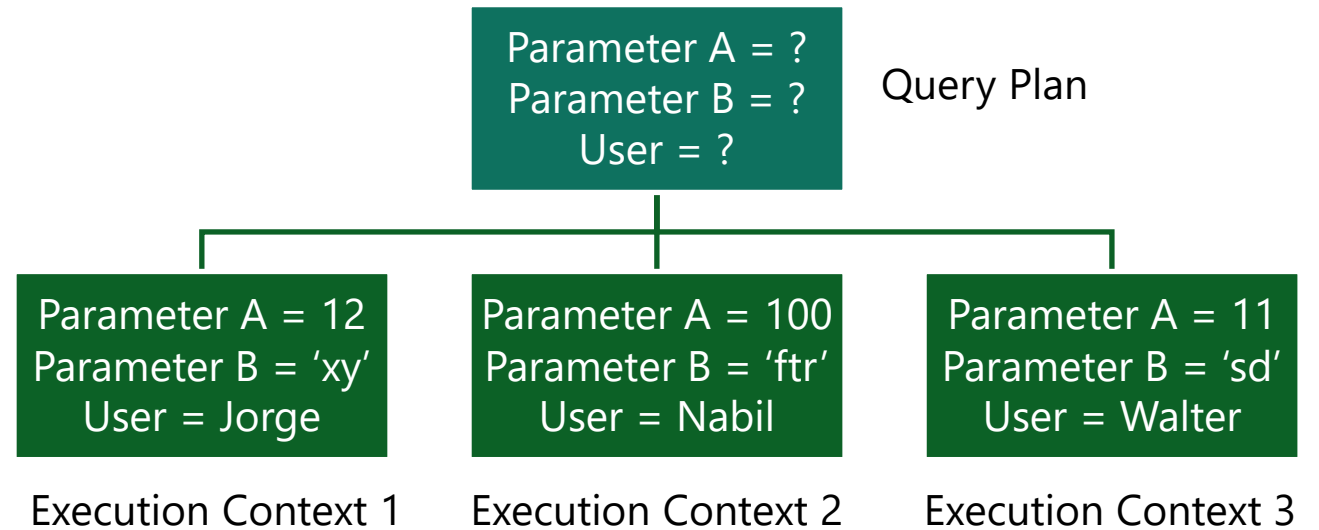
SQL Server Execution Plan

Main components

Compiled Plan (or Query Plan)

Compilation produces a query plan, which is a read-only data structure used by any number of users.

Execution Context

A data structure used to hold information specific to a query execution, such as parameter values.

Parameter A = ?
Parameter B = ?
User = ?

Query Plan

Parameter A = 12
Parameter B = 'xy'
User = Jorge

Parameter A = 100
Parameter B = 'ftr'
User = Nabil

Parameter A = 11
Parameter B = 'sd'
User = Walter

Execution Context 1

Execution Context 2

Execution Context 3

SQL Server Execution Plan Caching

Overview

Part of the memory pool used to store execution plans – also known as plan cache.

The plan cache has two stores for all compiled plans:

The **Object Plans** cache store (OBJCP) used for plans related to persisted objects (stored procedures, functions, and triggers).	The **SQL Plans** cache store (SQLCP) used for plans related to autoparameterized, dynamic, or prepared queries.

SQL Server compilation and execution

Concepts

Compilation

Process of creating a good enough query execution plan, as quickly as possible for a query batch.

Refer to both the compilation of non-DML constructs in SQL statements (control flow, DDL, etc.) and the process of Query Optimization.

Query Execution

Process of executing the plan that is created during query compilation and optimization.

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

Questions?

Lesson 2: SQL Server Query Plan Analysis

Objectives

After completing this learning, you will be able to:

- Read execution plans.

- Understand logical and physical join operators.

- Describe data access.

How to see the query plan

Graphical execution plan

Estimated Execution Plan (Before Execution)

- The compiled plan.

Actual Execution Plan (After Execution)

- The same as the compiled plan plus its execution context.
- This includes runtime information available after the execution completes, such as execution warnings, or in newer versions of the Database Engine, the elapsed and CPU time used during execution.

Live Query Statistics (During Execution)

- The same as the compiled plan plus its execution context.
- This includes runtime information during execution progress and is updated every second. Runtime information includes for example the actual number of rows flowing through the operators.
- Enables rapid identification of potential bottlenecks.

Contents of an Execution Plan

Sequence in which the source tables are accessed.

Methods used to extract data from each table.

How data is joined

Use of temporary worktables and sorts

Estimated rowcount, iterations, and costs from each operator

Actual rowcount and iterations

How to see the query plan

Text and XML

	Command	Execute query?	Include estimated row counts & stats (Estimated Query Plan)	Include actual row counts & stats (Actual Query Plan)
Text Plan	SET SHOWPLAN_TEXT ON	No	No	No
	SET SHOWPLAN_ALL ON	No	Yes	No
	SET STATISTICS PROFILE ON	Yes	Yes	Yes
XML Plan	SET SHOWPLAN_XML ON	No	Yes	No
	SET STATISTICS PROFILE XML	Yes	Yes	Yes

SSMS Graphical Plan
Execution Flow
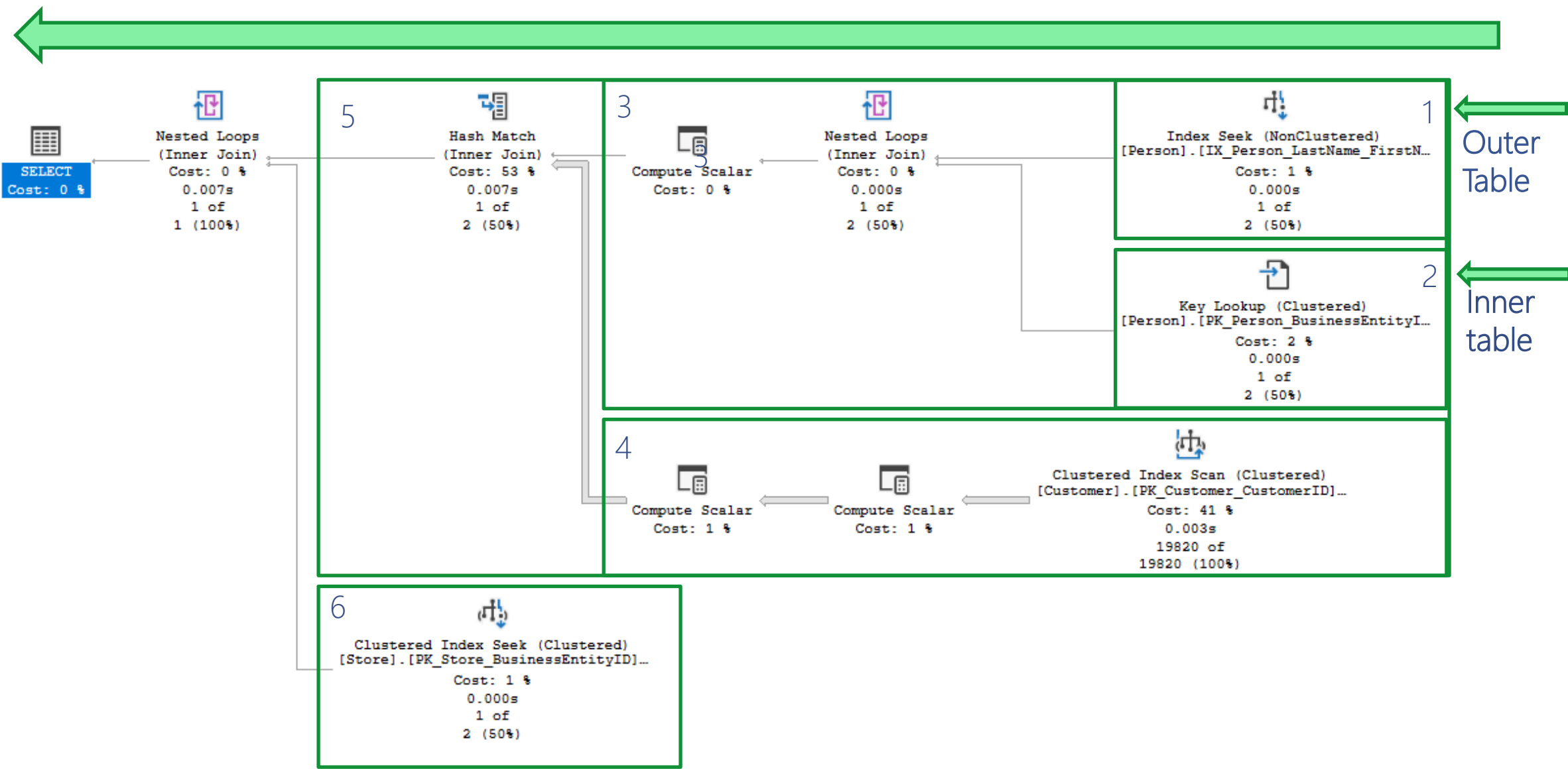
SSMS Graphical Plan

Data flow between the operators and statistical data of each operator

Statistical data for the operator
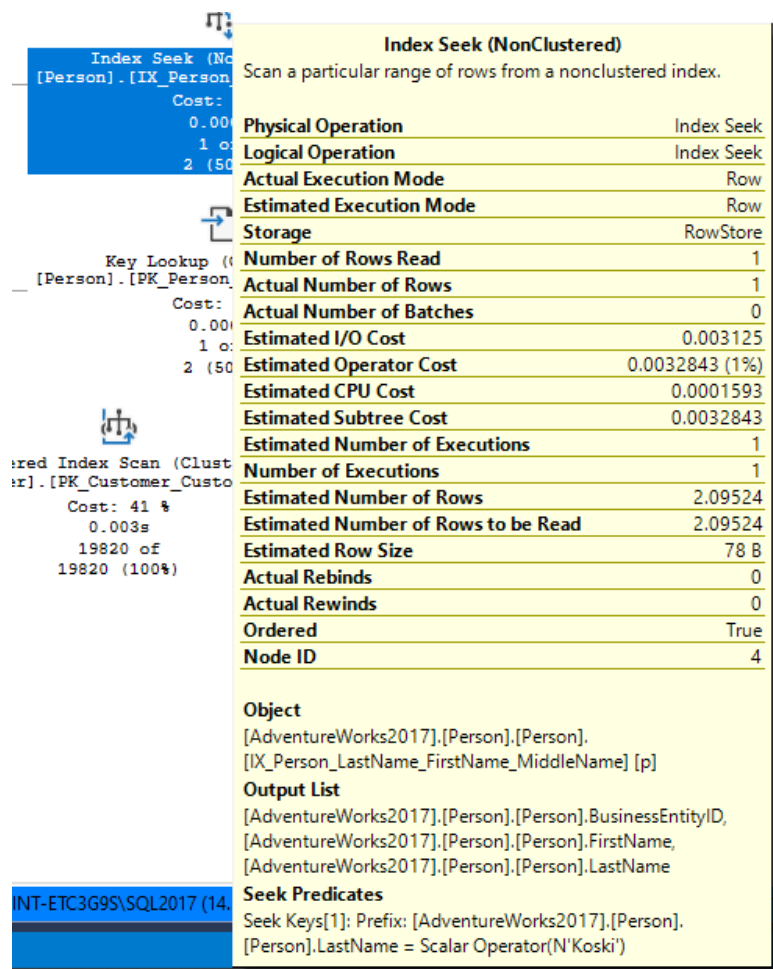
Data flow statistics
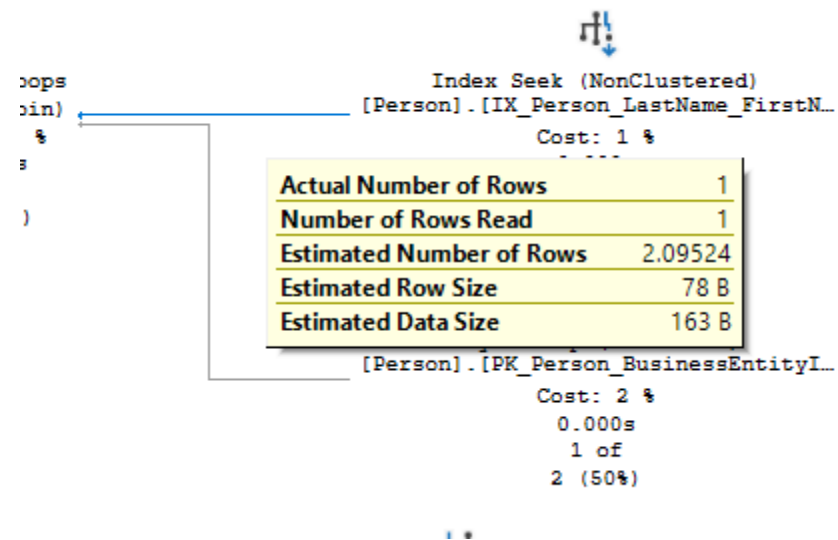
SSMS Graphical Plan

Properties sheet
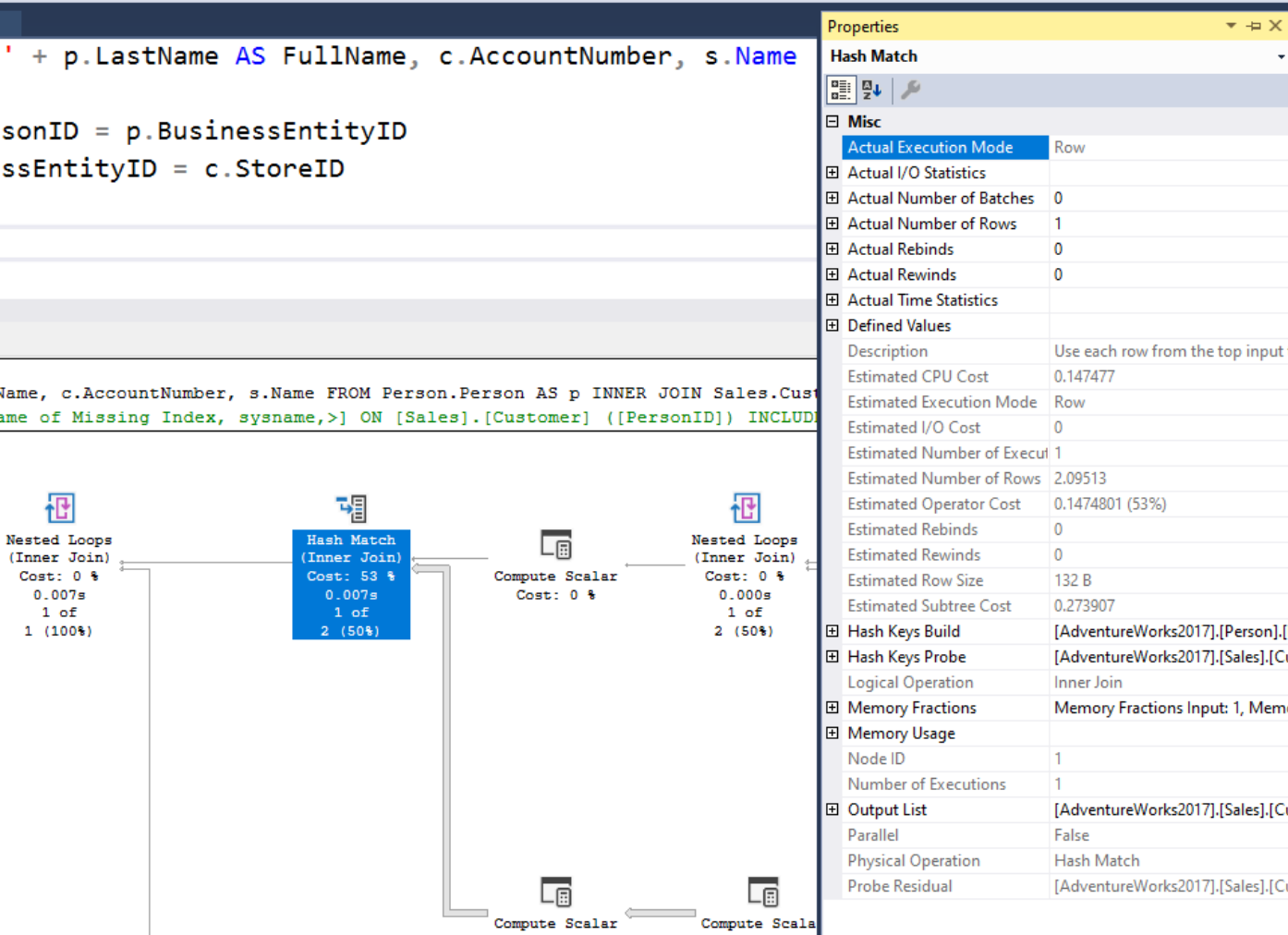
Management Studio Properties sheet includes even more detailed information about each operator and about the overall query plan.

Use the most recent version of Management Studio as every new version display more detailed information about the Query Plan when examining the plan in graphical mode.

SSMS Graphical Plan

Live Query Statistics

View CPU/memory usage, execution time, and query progress

Allows drill down to live operator-level statistics, such as:
- Number of generated rows
- Elapsed time
- Operator progress
- Live warnings

This feature is primarily intended for troubleshooting purposes.

Using this feature can moderately slow the overall query performance.
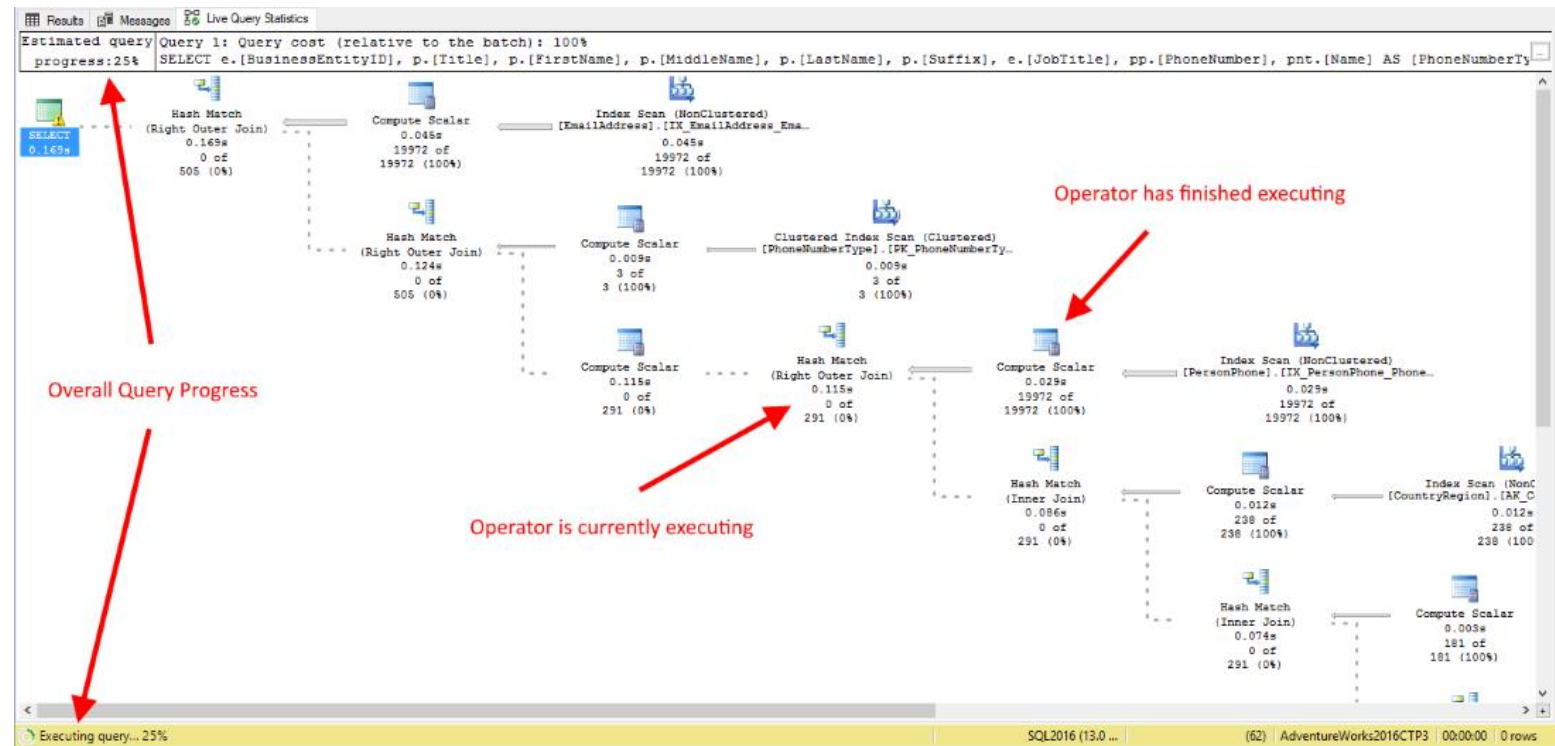
Execution Plan

Notable operators

Operators describe how SQL Server executes a query. The query optimizer uses operators to build a query plan to create the result specified in the query.

Table scan

Clustered index scan

Clustered index seek

RID lookup

Key lookup

ColumnStore Index Scan

Nonclustered index scan

Nonclustered index seek

Sort

Table spool

Index spool
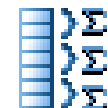
Stream Aggregation

Execution Plan Table Operators

Data stored in a Heap is not stored in any order and normally does not have a Primary Key.

Clustered Index data is stored in sorted order by the Clustering key. In many cases, this is the same value as the Primary Key.

Using a WHERE statement on an Index could possibly have the Execution Plan seek the Index instead of scan.

Table Scan
[BankAccounts]
Cost: 100 %

Clustered Index Scan (Cluste…
[BankAccounts].[pk_acctID]
Cost: 100 %

Clustered Index Seek (Cluste…
[BankAccounts].[pk_acctID]
Cost: 100 %

Execution Plan Join Operators (Code)

```sql
SELECT SOH.SalesOrderID, SOH.CustomerID,
    OrderQty, UnitPrice, P.Name
FROM SalesLT.SalesOrderHeader as SOH
JOIN SalesLT.SalesOrderDetail AS SOD
ON SOH.SalesOrderID = SOD.SalesOrderID
JOIN SalesLT.Product AS P
ON P.ProductID = SOD.ProductID
```

Execution Plan Join Operators

A Merge Join is useful if both table inputs are in the same sorted order on the same value.

Merge Join
(Inner Join)
Cost: 39 %

A Hash Match is used when the tables being joined are not in the same sorted order.

Hash Match
(Inner Join)
Cost: 47 %

A Nested Loop is use when a small (outer) table is used to lookup a value in a larger (inner) table.

Nested Loops
(Inner Join)
Cost: 3 %

What to look for in the query plan

Warnings	• Information about possible issues with the plan
Top Left Operator	• Overall properties of the plan
Expensive Operators	• Look from most expensive to least expensive
Data Flow Statistics	• Thicker arrows mean more data is being passed
Nested Loop Operator	• Possible to create index that covers query
Scans vs Seeks	• Not necessarily bad, but could indicate I/O issues
Skewed Estimates	• Statistics could be stale or invalid

Demonstration

Query Plan Analysis

- Use the graphical execution plan and IO statistics to tune a query.

- Explore Live Query Statistics.

Questions?

Lesson 3: SQL Server Plan Cache Internals

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

Is	Is used by	Exists to	Size and contents
• A pool of memory used to store query execution plans	• All databases in a SQL Server instance	• Avoid repeated optimization and compilation • Reuse of plans offers performance benefits, as Optimization and compilation costs time and CPU	• Vary over time • Can grow (within limits) to hold more and/or larger plans • Can shrink under memory pressure, by evicting plans

SQL Server's Memory

Plan Cache

Buffer Pool

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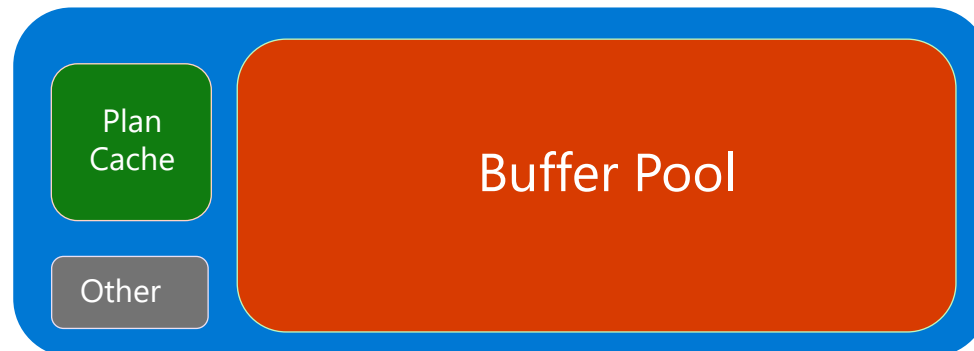
With a fixed amount of memory, when the plan cache grows the buffer pool shrinks and vice-versa

Dynamic Management Views and Functions

Category	Description
sys.dm_exec_%	Execution and connection information
sys.dm_os_%	Operating system related information
sys.dm_tran_%	Transaction management information
sys.dm_io_%	I/O related information
sys.dm_db_%	Database information

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
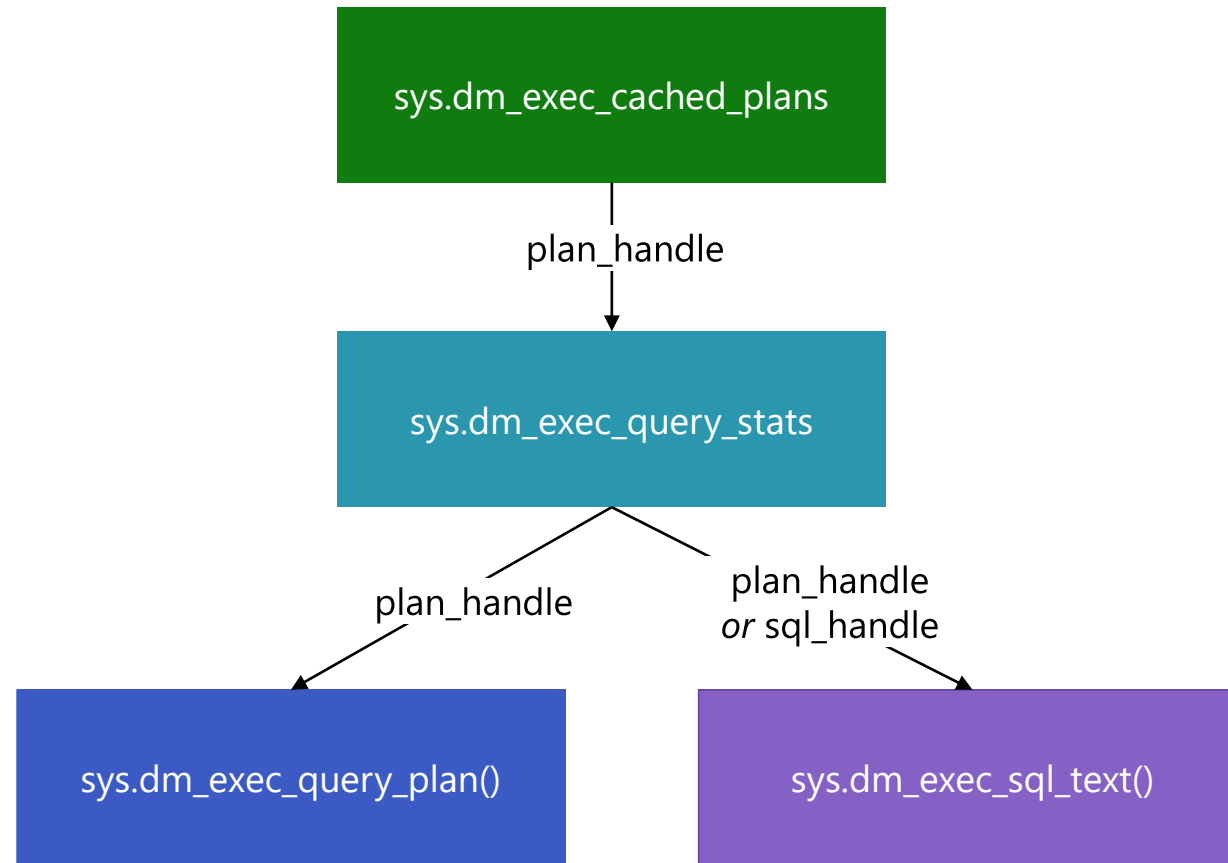
Captured Metrics (sys.dm_exec_query_stats)

Partial listing

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

Aggregate statistics
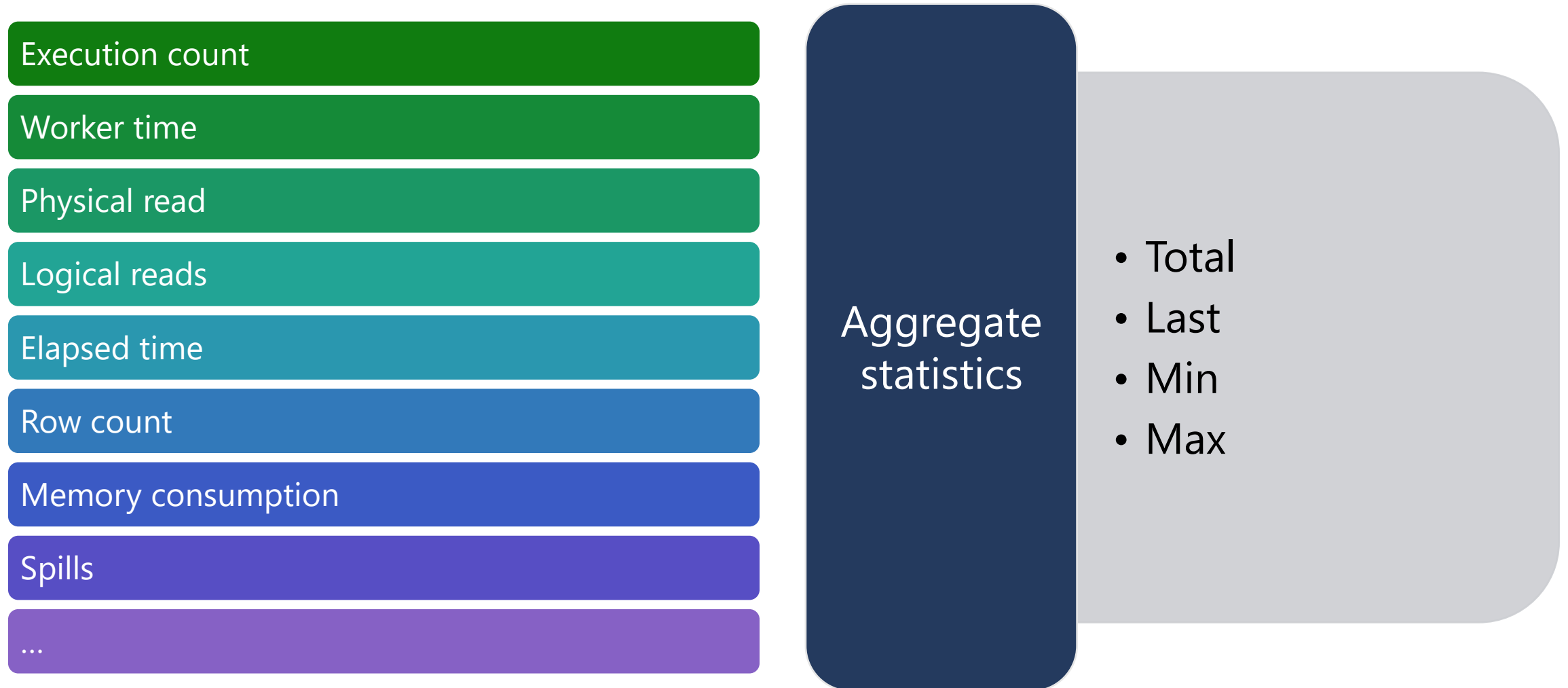
- Total
- Last
- Min
- Max

Compute Average: Avg = Total / Execution count

Mining the Plan Cache with T-SQL

Top 10 plans by logical reads

```sql
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

	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

```sql
      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 );
```

Vary the subquery ORDER BY metrics to identify top queries by other measures:
- total_worker_time/execution_count
- total_elapsed_time/execution_count
- execution_count
- *etc.*

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

```sql
-- Plan cache contents and memory use by object type
SELECT  objtype, C            1024) AS size_mb,
        AVG(usecou                                          _use_mb,
        SUM(CAST((                                          ngle_use_plans,

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

Results | Messages

	objty				
1	Adho				250
2	Prep				5000
3	Proc			2	0.000937

```sql
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!

```
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 Reuse

Requires that SQL statements match exactly
- <u>Any</u> 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)

```sql
-- 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	Drawback
• Improved performance as reuse saves time and CPU	• 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
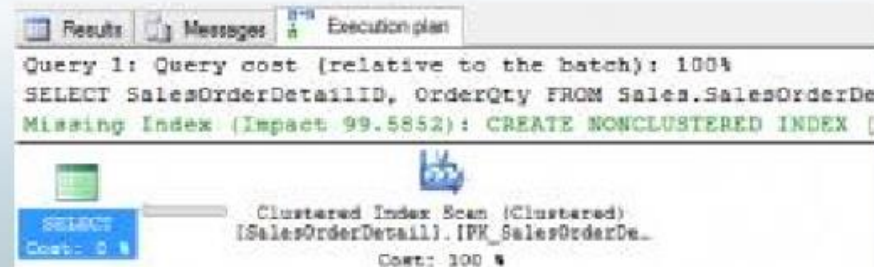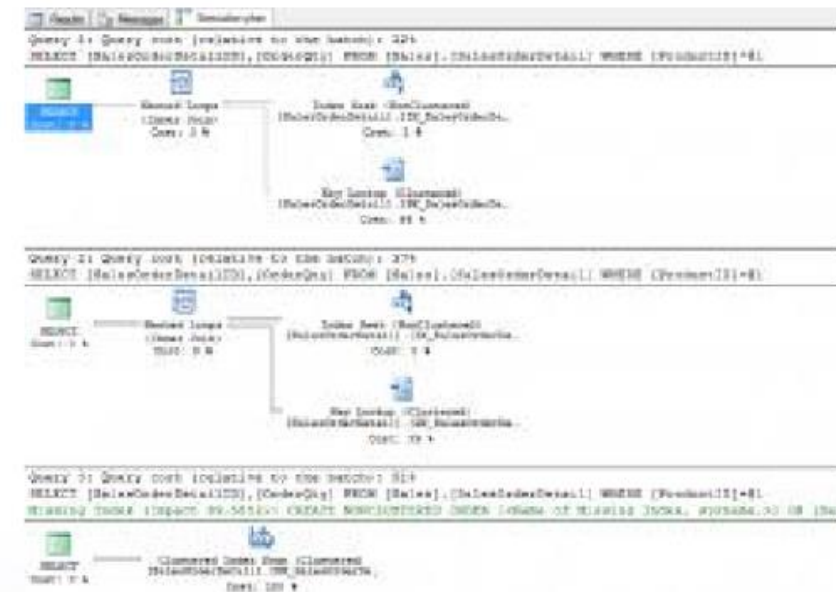
Knowledge Check

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 4: SQL Server Query Store

Introducing the Query Store
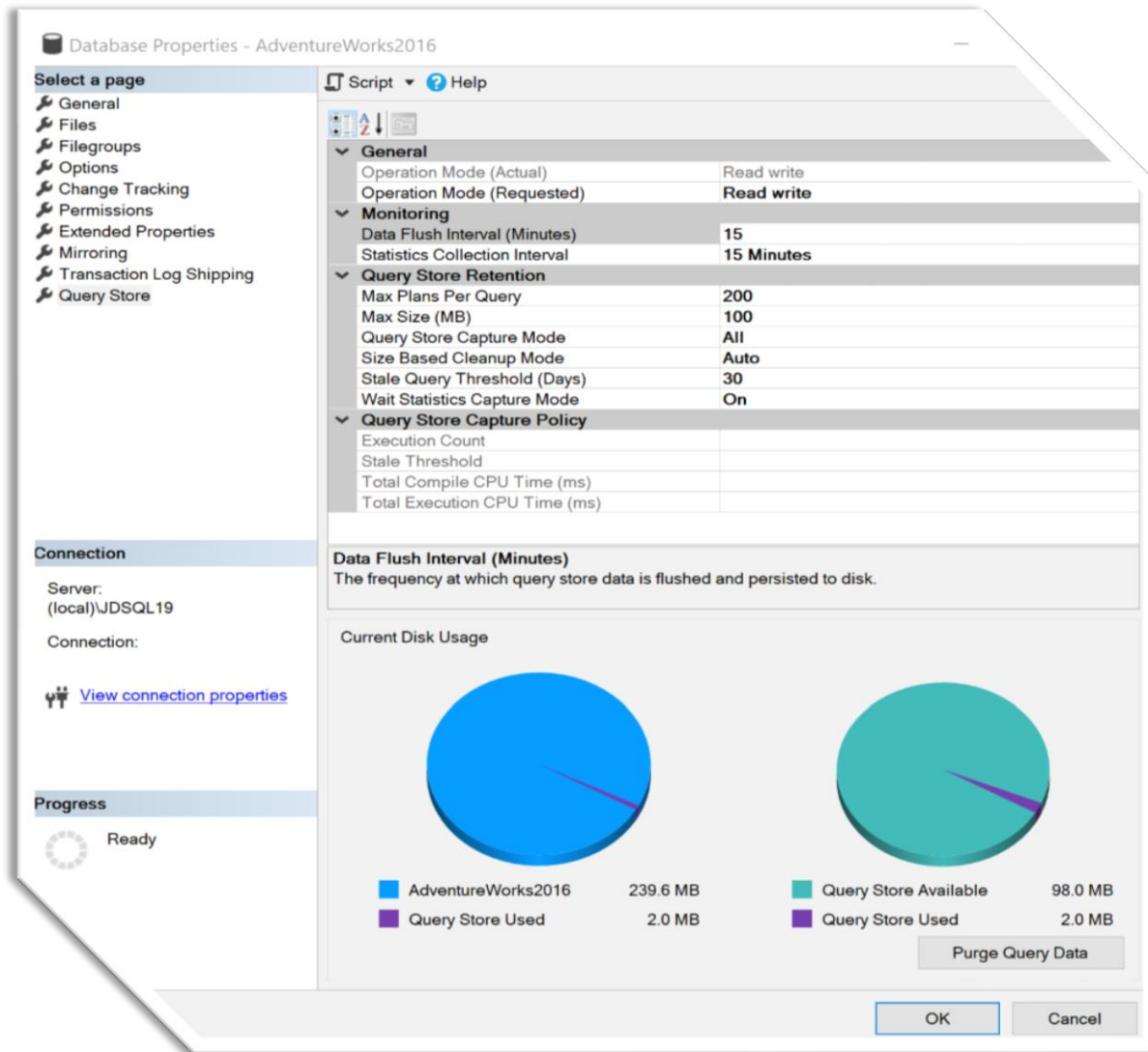
Query Store is set at the database level

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	With 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.	• 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

Query Store Operation Modes

Operation Mode can be set under database properties

General	
Operation Mode (Actual)	Read Write
Operation Mode (Requested)	**Read Write**
Monitoring	Off
Query Store Retention	Read Only
	Read Write
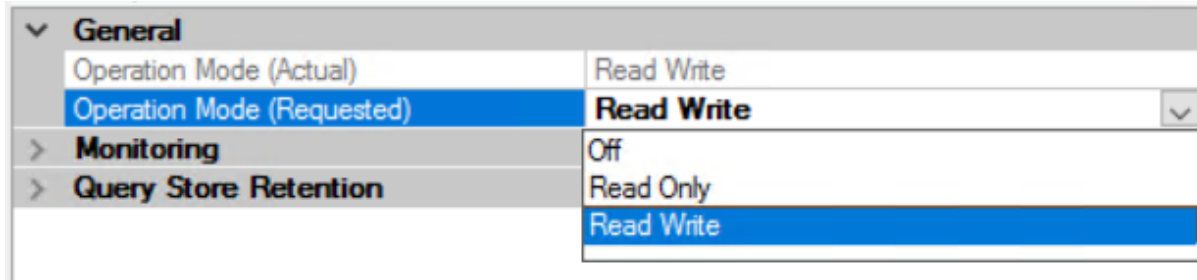
Operation Mode can be enabled two ways using T-SQL. If only using the ON option, the Mode defaults to **Read_Write**

```sql
ALTER DATABASE [AdventureWorks2016] SET QUERY_STORE = ON;


ALTER DATABASE [AdventureWorks2016] 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

```sql
ALTER DATABASE [AdventureWorks2016] SET QUERY_STORE
(INTERVAL_LENGTH_MINUTES = 1,
DATA_FLUSH_INTERVAL_SECONDS = 60)
```

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 [AdventureWorks2016] SET QUERY_STORE
(INTERVAL_LENGTH_MINUTES = 1,
DATA_FLUSH_INTERVAL_SECONDS = 60)
```

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

```sql
ALTER DATABASE AdventureWorks2016 SET QUERY_STORE
(MAX_PLANS_PER_QUERY = 200,
 MAX_STORAGE_SIZE_MB = 100,
 QUERY_CAPTURE_MODE = AUTO,
 SIZE_BASED_CLEANUP_MODE = AUTO,
 CLEANUP_POLICY = (STALE_QUERY_THRESHOLD_DAYS = 367),
 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

```sql
ALTER DATABASE AdventureWorks2016 SET QUERY_STORE
(MAX_PLANS_PER_QUERY = 200,
 MAX_STORAGE_SIZE_MB = 100,
 QUERY_CAPTURE_MODE = AUTO,
 SIZE_BASED_CLEANUP_MODE = AUTO,
 CLEANUP_POLICY = (STALE_QUERY_THRESHOLD_DAYS = 367),
 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

```sql
ALTER DATABASE AdventureWorks2016 SET QUERY_STORE
(MAX_PLANS_PER_QUERY = 200,
 MAX_STORAGE_SIZE_MB = 100,
 QUERY_CAPTURE_MODE = AUTO,
 SIZE_BASED_CLEANUP_MODE = AUTO,
 CLEANUP_POLICY = (STALE_QUERY_THRESHOLD_DAYS = 367),
 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

```sql
ALTER DATABASE AdventureWorks2016 SET QUERY_STORE
(MAX_PLANS_PER_QUERY = 200,
 MAX_STORAGE_SIZE_MB = 100,
 QUERY_CAPTURE_MODE = AUTO,
 SIZE_BASED_CLEANUP_MODE = AUTO,
 CLEANUP_POLICY = (STALE_QUERY_THRESHOLD_DAYS = 367),
 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

```sql
ALTER DATABASE AdventureWorks2016 SET QUERY_STORE
(MAX_PLANS_PER_QUERY = 200,
 MAX_STORAGE_SIZE_MB = 100,
 QUERY_CAPTURE_MODE = AUTO,
 SIZE_BASED_CLEANUP_MODE = AUTO,
 CLEANUP_POLICY = (STALE_QUERY_THRESHOLD_DAYS = 367),
 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

```sql
ALTER DATABASE AdventureWorks2016 SET QUERY_STORE
(MAX_PLANS_PER_QUERY = 200,
 MAX_STORAGE_SIZE_MB = 100,
 QUERY_CAPTURE_MODE = AUTO,
 SIZE_BASED_CLEANUP_MODE = AUTO,
 CLEANUP_POLICY = (STALE_QUERY_THRESHOLD_DAYS = 367),
 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 Hour
Total Compile CPU Time (ms)	1000
Total Execution CPU Time (ms)	100

```sql
ALTER DATABASE AdventureWorks2016 SET QUERY_STORE
(QUERY_CAPTURE_POLICY =
  (EXECUTION_COUNT = 100,
  STALE_CAPTURE_POLICY_THRESHOLD = 24 Hours,
  TOTAL_COMPILE_CPU_TIME_MS = 2000,
  TOTAL_EXECUTION_CPU_TIME_MS = 1000));
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 AdventureWorks2016 SET QUERY_STORE
(QUERY_CAPTURE_POLICY =
 (EXECUTION_COUNT = 100,
  STALE_CAPTURE_POLICY_THRESHOLD = 24 Hours,
  TOTAL_COMPILE_CPU_TIME_MS = 2000,
  TOTAL_EXECUTION_CPU_TIME_MS = 1000));
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

```sql
ALTER DATABASE AdventureWorks2016 SET QUERY_STORE
(QUERY_CAPTURE_POLICY =
 (EXECUTION_COUNT = 100,
 STALE_CAPTURE_POLICY_THRESHOLD = 24 Hours,
 TOTAL_COMPILE_CPU_TIME_MS = 2000,
 TOTAL_EXECUTION_CPU_TIME_MS = 1000));
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

```sql
ALTER DATABASE AdventureWorks2016 SET QUERY_STORE
(QUERY_CAPTURE_POLICY =
 (EXECUTION_COUNT = 100,
 STALE_CAPTURE_POLICY_THRESHOLD = 24 Hours,
 TOTAL_COMPILE_CPU_TIME_MS = 2000,
 TOTAL_EXECUTION_CPU_TIME_MS = 1000));
GO
```

Purge Query Data

Data can be manually purged from the Query Store.


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

Current Disk Usage

AdventureWorks2016	208.6 MB
Query Store Used	1.0 MB

Query Store Available	149.0 MB
Query Store Used	1.0 MB

Purge Query Data
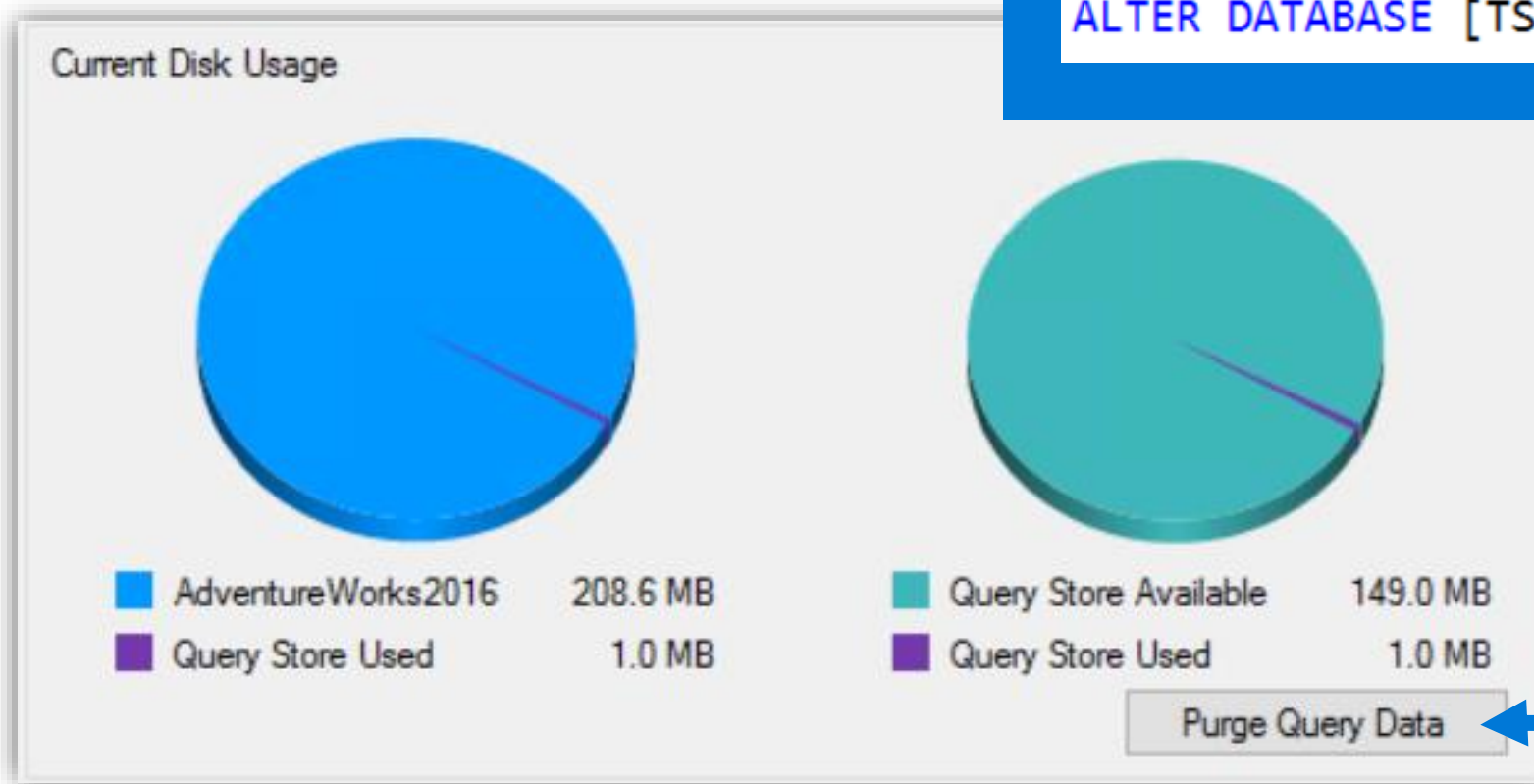
Query Store Reports

Regressed Queries: Use this dashboard to review queries that might have regressed because of execution plan changes
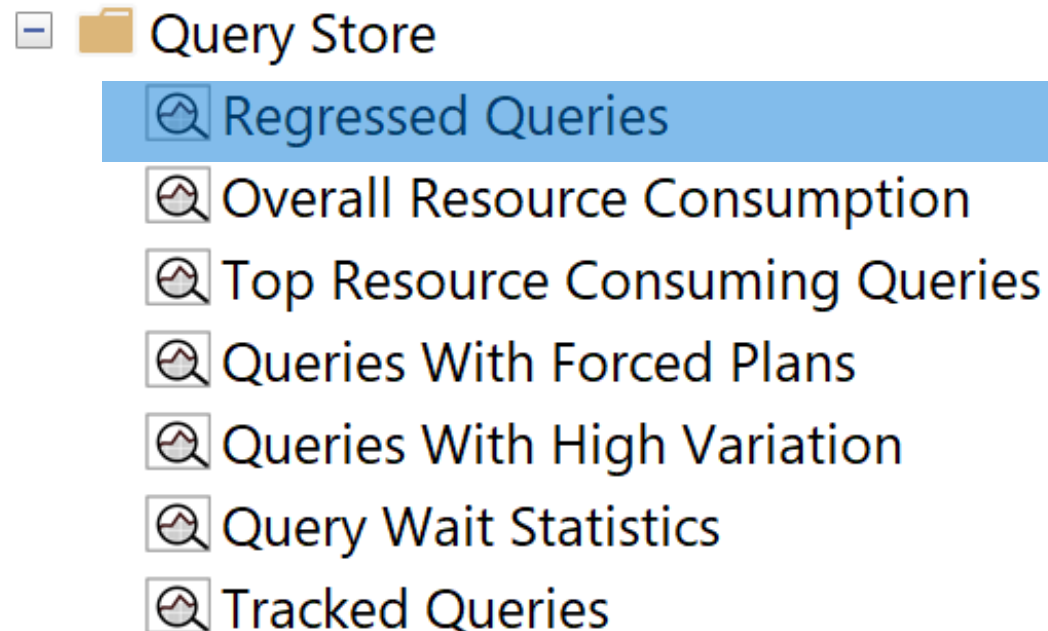
Query Store Reports

Overall Resource Consumption: Use this dashboard to visualize overall resource consumption during the last month in four charts: duration, execution count, CPU time, and logical reads
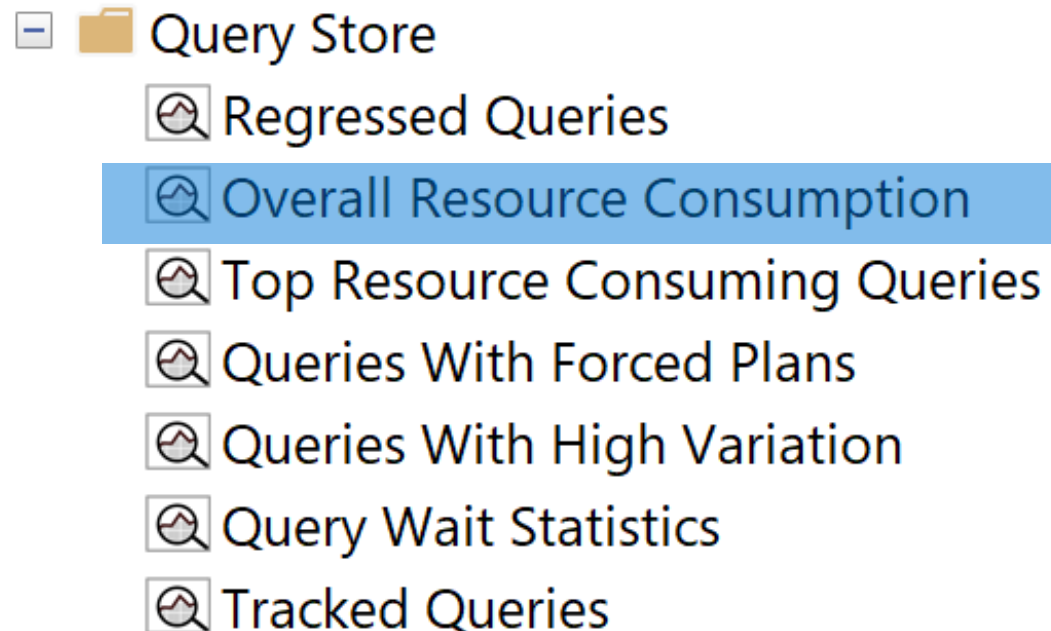
Query Store Reports

Top Resource Consuming Queries: Use this dashboard to review queries in the set of top 25 resource consumers during the last hour
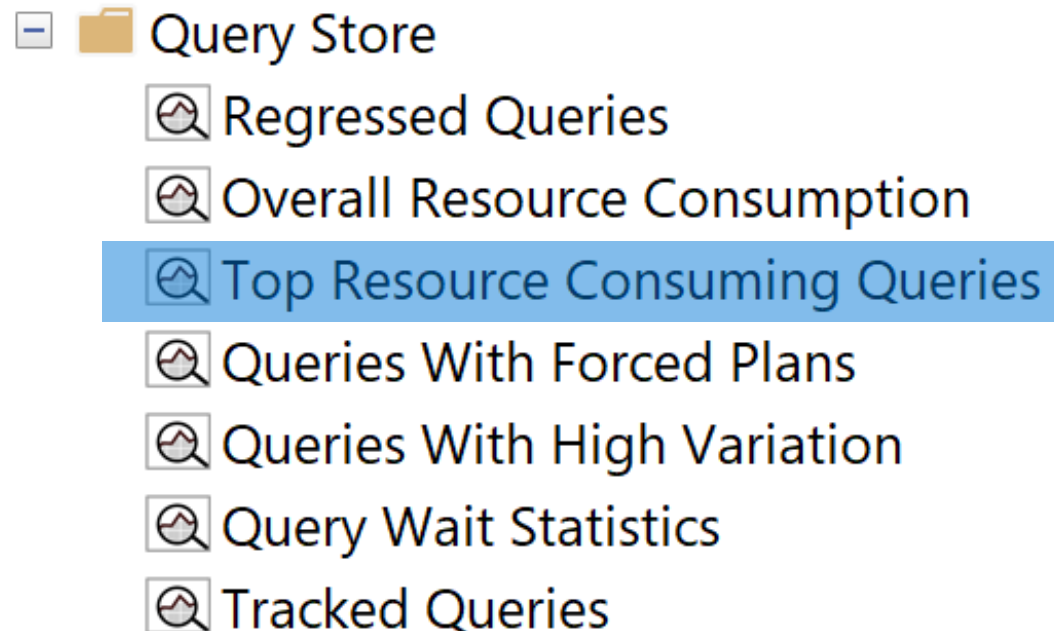
Query Store Reports

Queries With Forced Plans: Used to isolate queries that have been given a forced plan. Requires SQL Server 2016 SP1 or later.
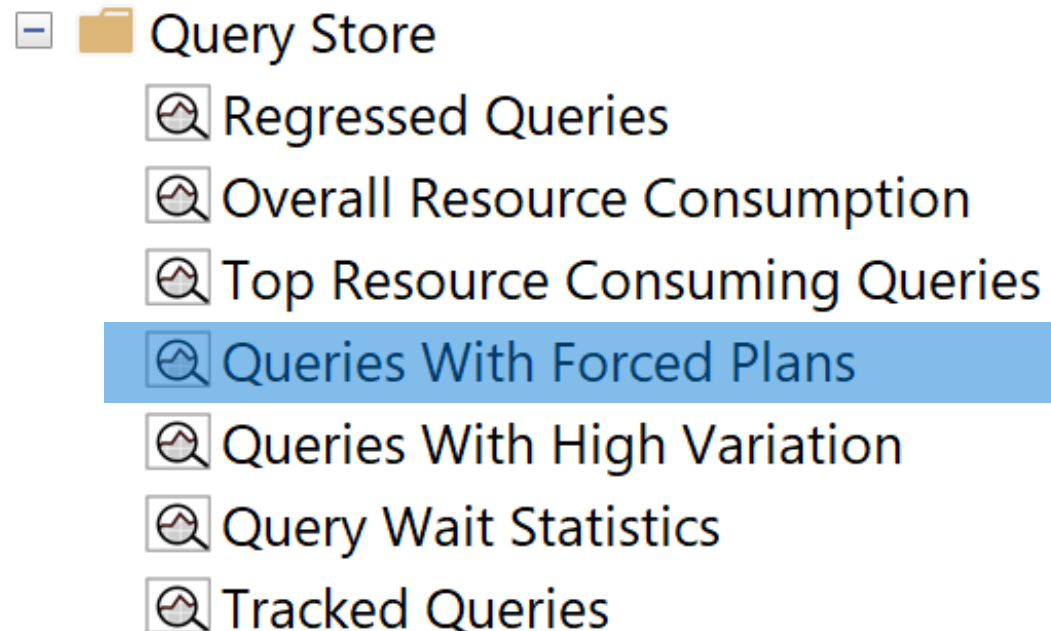
Query Store Reports

Queries With High Variation: Used to locate queries with high variation in query execution. Useful to locate queries with parameterization problems. Requires SQL Server 2016 SP1 or later.
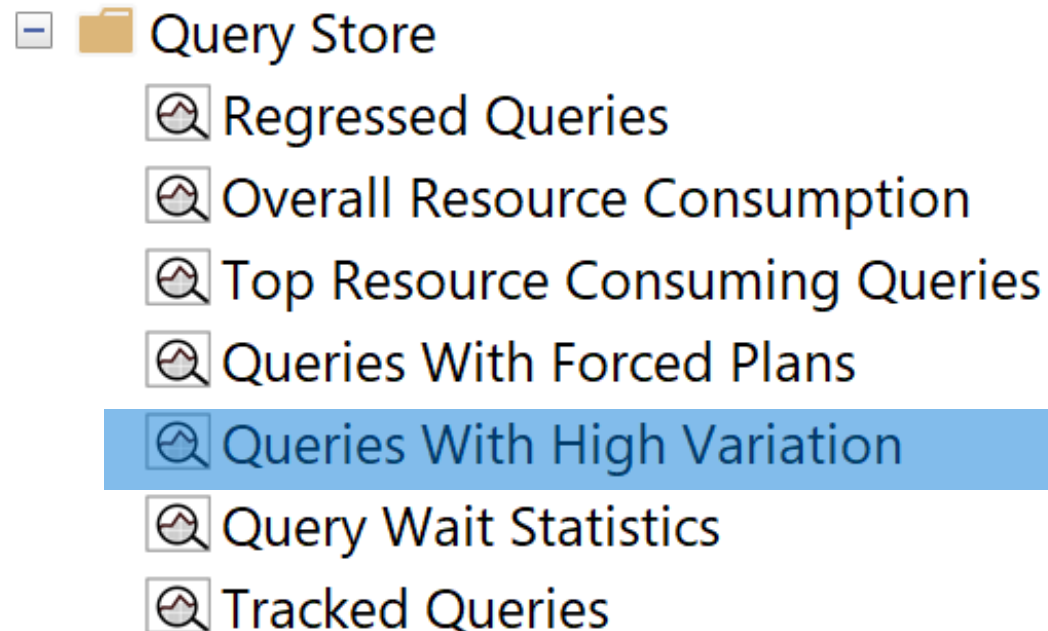
Query Store Reports

Query Wait Statistics shows a bar chart containing the top wait categories in the Query Store. Use the drop down at the top to select an aggregate criteria for the wait time: avg, max, min, std dev, and **total** (default). Requires SQL Server 2017.
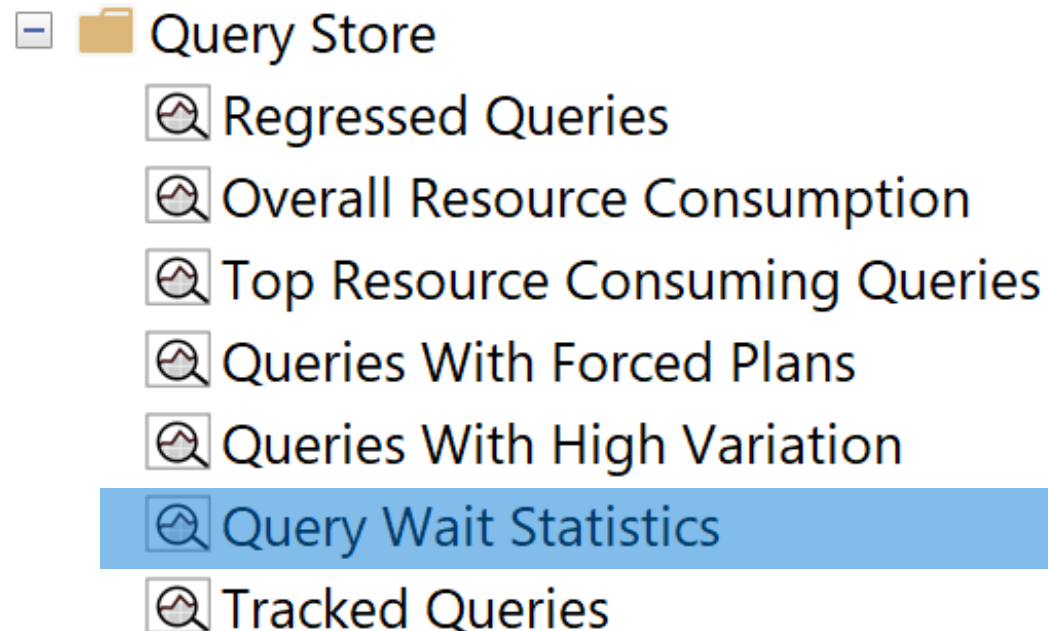
Query Store Reports

Tracked Queries: Use this dashboard to monitor the execution plans and regression of a specify query

Establishing a Baseline

Force Plan

Plan Compare

Query Store Catalog Views

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)
```