Microsoft

SQL Server Query Tuning

Module 8

Learning Units covered in this Module

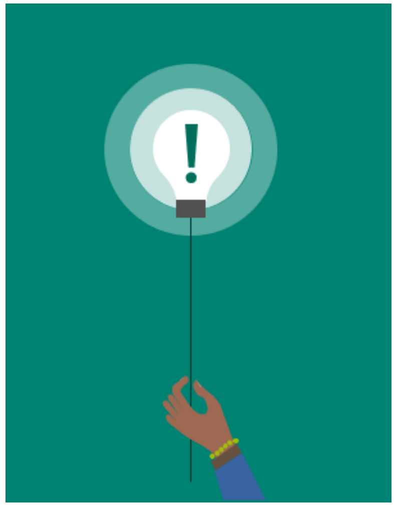
- Lesson 1: Sargable Expressions
- Lesson 2: Query Hints
- Lesson 3: Query Troubleshooting

Lesson 1: Sargable Expressions

Objectives

After completing this learning, you will be able to:

- Address SARGability Issues.
- Use computed columns for performance.
- Use constraints for performance.
- Understand parameter sniffing.

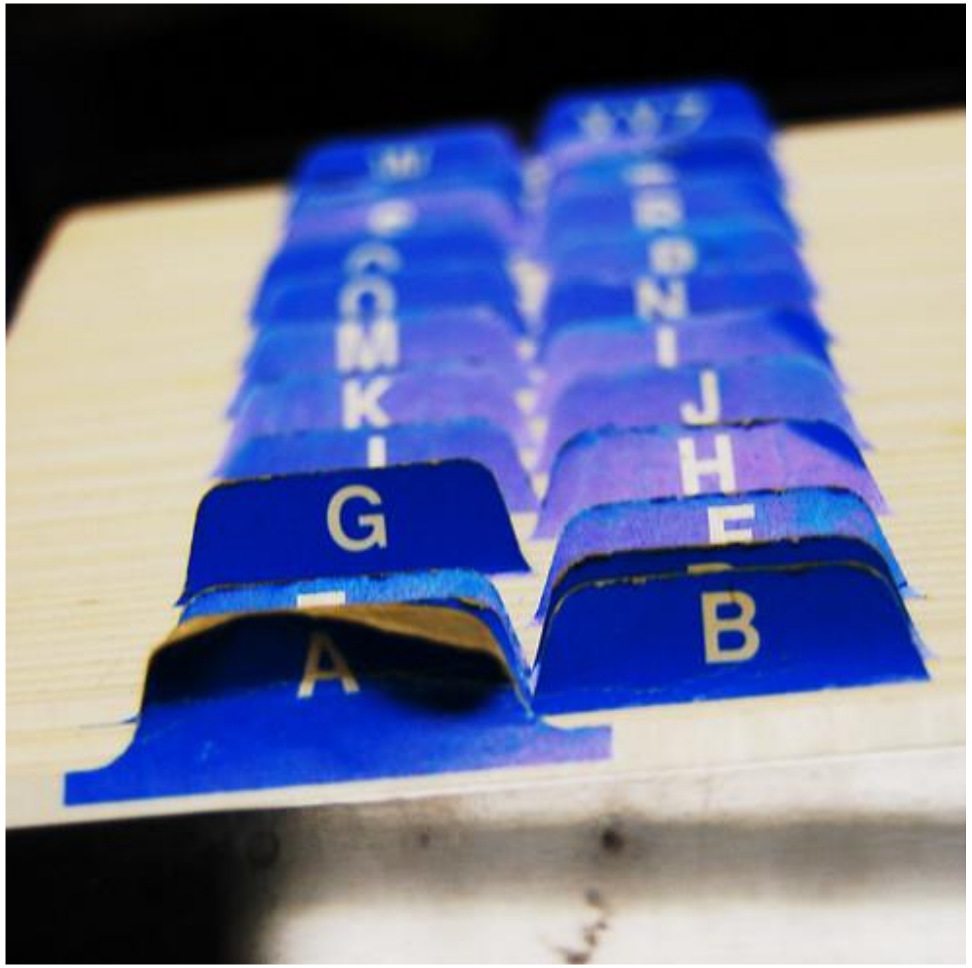


SARGability

What is this?

A SARGable item in a search predicate is able to use an index.

Non-SARGable expressions can significantly slow down queries.



Non-SARGable Expressions

Functions

WHERE ABS(ProductID) = 771

WHERE UPPER(City) = “London”

WHERE UPPER(surname) = “SMITHS”

Calculations / Expressions

WHERE Sales.Price + Sales.Tax > 100

WHERE Sales.Price * (1 + Sales.TaxRate) > 100

Using leading wildcard with LIKE operator

WHERE Employee.FirstName LIKE '%L%' is non-SARGable

WHERE Employee.FirstName LIKE ‘L%' is SARGable

Non-SARGable Expressions

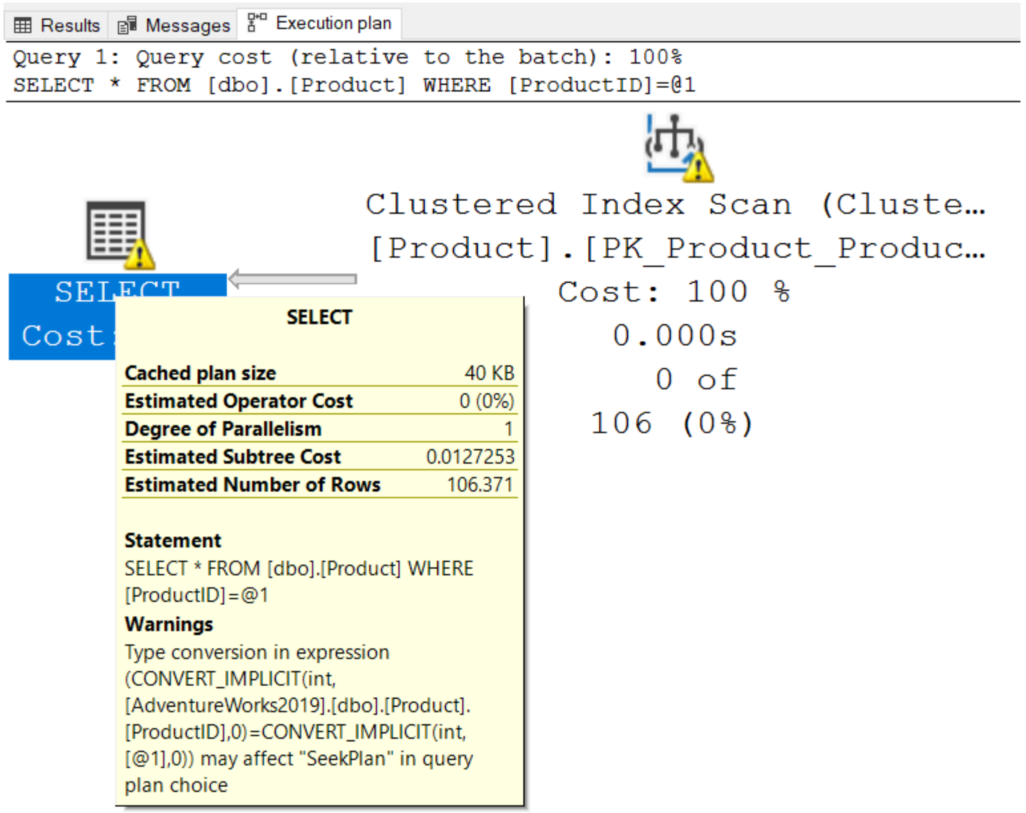
Continued

Implicit Conversions

ProductID is defined as **nvarchar(8)**

```
SELECT *
FROM [dbo].[Product]
WHERE [ProductID] = 7
```

```
SELECT *
FROM [dbo].[Product]
WHERE [ProductID] = N'7'
```



```
Query 1: Query cost (relative to the batch): 100%
SELECT * FROM [dbo].[Product] WHERE [ProductID]=@1
```

```
Clustered Index Scan (Cluste…
[Product].[PK_Product_Produc…
Cost: 100 %
0.000s
0 of
106 (0%)
```

SELECT

Cached plan size	40 KB
Estimated Operator Cost	0 (0%)
Degree of Parallelism	1
Estimated Subtree Cost	0.0127253
Estimated Number of Rows	106.371

Statement
SELECT * FROM [dbo].[Product] WHERE [ProductID]=@1

Warnings
Type conversion in expression (CONVERT_IMPLICIT(int,[AdventureWorks2019].[dbo].[Product].[ProductID],0)=CONVERT_IMPLICIT(int,[@1],0)) may affect "SeekPlan" in query plan choice

Using Computed Columns to Improve Performance

Resolving non-SARGable expressions

- Create computed column to Replace calculations / expressions.
- Create an index on computed column.

Selectivity issues for inequalities

- SQL Server assumes 30% selectivity on inequality comparisons.
- Computed column will have more accurate statistics.
- No need to specify computed column.

Using Computed Columns to Improve Performance

Persisted Computed Columns

Specifies that the Database Engine will physically store the computed values in the table.

Marking a computed column as PERSISTED allows an index to be created on a computed column that is deterministic.

```
--Add a new computed column as persisted
ALTER TABLE [dbo].[ProductTest]
ADD stockValue AS (isnull(([UnitPrice] * ((1.0) - [UnitPriceDiscount])) *
[StockQty], (0.0))) PERSISTED;


--Create an index on new computed column to improve query performance
CREATE INDEX IX_StockValue ON [dbo].[ProductTest](stockValue);
```

Using Constraints to Improve Performance

Helping Query Optimizer to choose better plans

UNIQUE constraint	• DISTINCT property can be ignored • Extra columns in ORDER BY may be ignored
CHECK constraint	• CHECK constraints enforce domain integrity by limiting the values that are accepted by one or more columns.
Primary Key	• Special case of UNIQUE constraint
Foreign Key	• Declarative Referential Integrity (DRI) • Usually faster than triggers • Allows the query optimizer to identify unnecessary joins

Parameter Sniffing

SQL Server "sniffs" current parameter values during compilation or recompilation.

Parameter values are passed along to Query Optimizer, generating a more efficient execution plan.

The generated plan stays in query cache.

Parameter sniffing can be seen on the following types of batches:

- Stored procedures
- Queries submitted via sp_executesql
- Prepared queries

Parameter Sniffing

Mitigation

RECOMPILE

- This workaround trades compilation time and increased CPU for better plan quality.

OPTION (OPTIMIZE FOR…)

- This option requires a good understanding of optimal parameter values and associated plan characteristics.

OPTION (OPTIMIZE FOR UNKNOWN)

- Overrides the actual parameter value and instead use the density vector.

DISABLE_PARAMETER_SNIFFING

- Disables parameter sniffing entirely. This hint name is equivalent to:
  - TRACE FLAG 4136
  - Database Scoped Configuration setting PARAMETER_SNIFFING = OFF

KEEPFIXEDPLAN

- This workaround assumes that the good-enough common plan is the one in cache already.

USE PLAN

- Force the plan by explicitly using this query hint by rewriting the query and adding the hint in the query text. Or set a specific plan by using Query Store.

Demonstration

Query Tuning Sargability

Query Tuning using

- Sargability
- Constraints
- Computed Columns

Query Tuning
SARGability

- Applying SARGable expression to create better execution plans

Questions?

Knowledge Check

Give two examples of non-SARGable expressions.

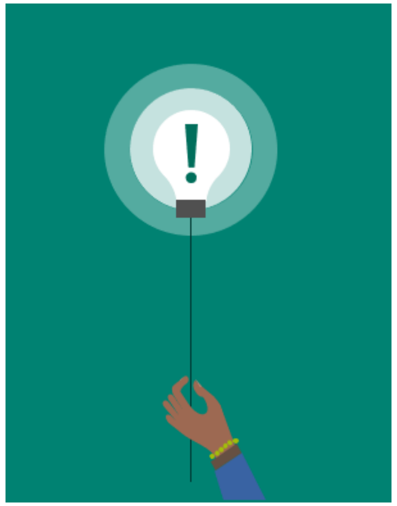
Explain three hints we can use to mitigate Parameter Sniffing.

Lesson 2: Query Hints

Objectives

After completing this lesson, you will be able to learn:

- Query hint types and how to use them



Hint Types

Query Hints

- Applied to the whole query
- Specified using the OPTION clause

Examples: MAXDOP, RECOMPILE, OPTIMIZE FOR

Table Hints

- Applied to single tables in the query

Examples: FORCESEEK, FORCESCAN, and INDEX

Join Hints

- Applied to specific joins only

Examples: LOOP, MERGE, HASH and REMOTE

Query Hints

Specify that the indicated hint(s) should be used throughout the query.

Affect all operators in the statement.

- OPTIMIZE FOR (Value/UNKNOWN)
- RECOMPILE
- FAST N
- FORCE ORDER
- KEEP PLAN
- MAXDOP
- PARAMETERIZATION
- USE HINT

USE HINT

Provides one or more additional hints to the query processor.

Additional hints are specified by a hint name **inside single quotation marks**.

FORCE_LEGACY_CARDINALITY_ESTIMATION

- Forces the Query Optimizer to use Cardinality Estimation model of SQL Server 2012 (11.x) and earlier versions. This hint name is equivalent to trace flag 9481.

DISABLE_ROW_MODE_MEMORY_GRANT_FEEDBACK

- Row mode memory grant feedback expands on the batch mode memory grant feedback feature by adjusting memory grant sizes for both batch and row mode operators.

DISALLOW_BATCH_MODE

- Disables batch mode execution.

ENABLE_QUERY_OPTIMIZER_HOTFIXES

- Enables Query Optimizer hotfixes (changes released in SQL Server Cumulative Updates and Service Packs). This hint name is equivalent to trace flag 4199.

Table Hints

Specified in the FROM clause of the DML statement.
Affect only the table or view referenced in that clause.

INDEX	FORCESEEK	FORCESCAN	NOLOCK
READPAST	XLOCK	HOLDLOCK	PAGLOCK

Join Hints

Specify a join strategy between two tables.

LOOP

HASH

MERGE

Demonstration

Query Hints

Questions?

Knowledge Check

Consider a query inside a Store Procedure that appears to be using an inefficient plan, after migration to SQL Server 2019 from SQL Server 2012.

This query used to have a good plan before the migration. How can a query hint be used to remediate this performance issue?

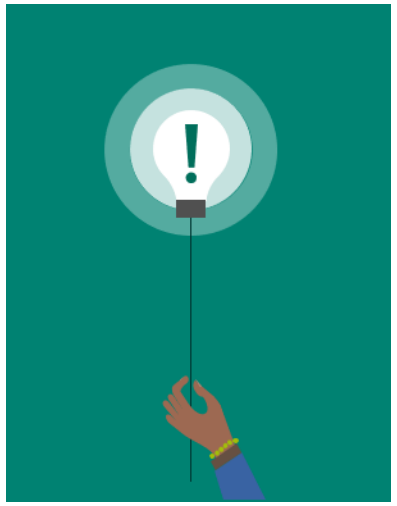
Explain one case a MAXDOP hint can be used to improve query performance.

Lesson 3: Query Troubleshooting

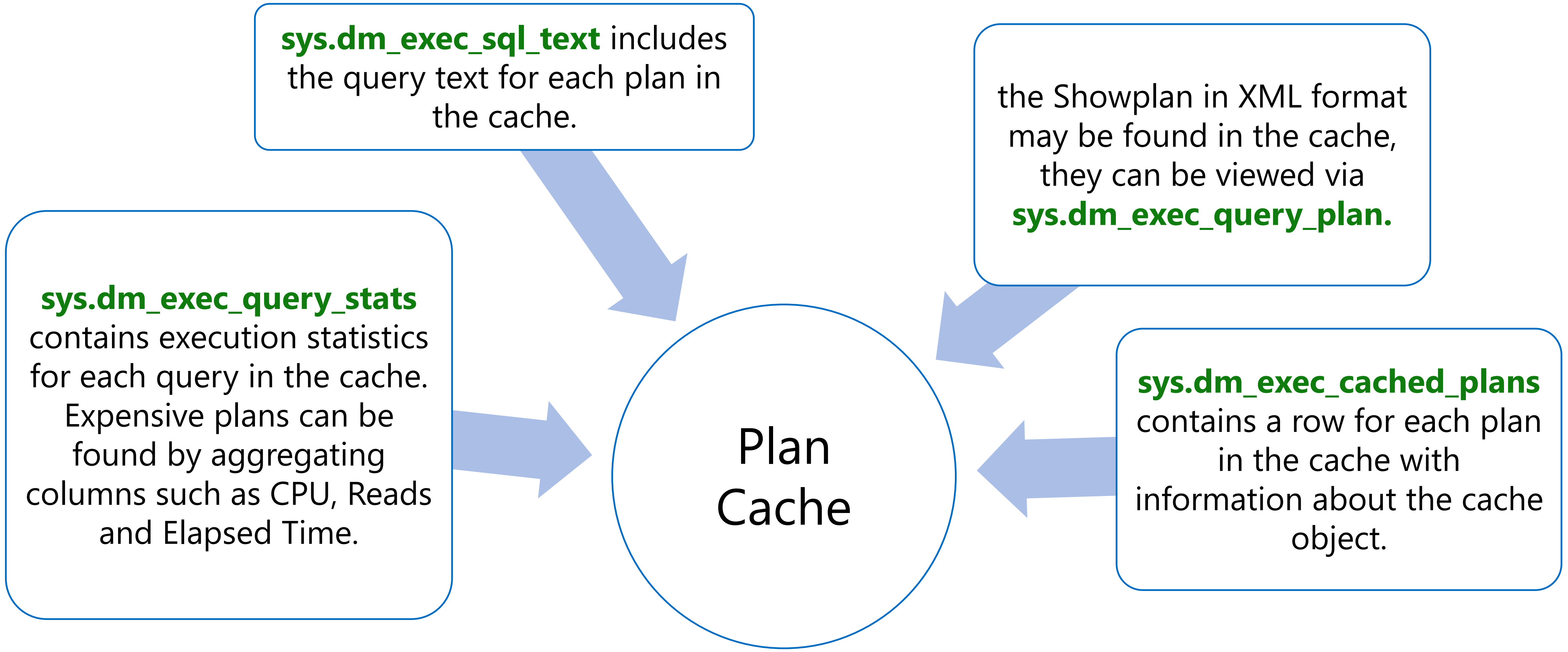
Objectives

After completing this learning, you will be able to:

- View the plan cache
- Use tools for troubleshooting



Queries in the Plan Cache



sys.dm_exec_sql_text includes the query text for each plan in the cache.

the Showplan in XML format may be found in the cache, they can be viewed via **sys.dm_exec_query_plan.**

sys.dm_exec_query_stats contains execution statistics for each query in the cache. Expensive plans can be found by aggregating columns such as CPU, Reads and Elapsed Time.

sys.dm_exec_cached_plans contains a row for each plan in the cache with information about the cache object.

Queries in the Plan Cache

```sql
SELECT
   last_execution_time, total_worker_time AS [Total CPU Time], execution_count, total_worker_time
   / execution_count AS [Avg CPU Time], text, q.query_plan
FROM
   sys.dm_exec_query_stats AS qs
   CROSS APPLY sys.dm_exec_sql_text (qs.sql_handle) AS st
   CROSS APPLY sys.dm_exec_query_plan (qs.plan_handle) AS q;
```

Procedures in the Plan Cache

sys.dm_exec_procedure

- Returns aggregate performance statistics for cached stored procedures.

```
SELECT
        db_name(database_id) AS database_name,
        object_name(object_id, database_id) AS proc_name, st.text, ps.*, qp.query_plan
FROM
        sys.dm_exec_procedure_stats ps
        CROSS APPLY sys.dm_exec_sql_text(ps.sql_handle) st
        CROSS APPLY sys.dm_exec_query_plan(ps.plan_handle) qp
```

Triggers in the Plan Cache

sys.dm_exec_trigger_stats

- Returns aggregate performance statistics for cached triggers.

```
SELECT
        db_name(database_id) AS database_name,
        object_name(object_id, database_id) AS proc_name, st.text, ts.*, qp.query_plan
FROM
        sys.dm_exec_trigger_stats ts
        CROSS APPLY sys.dm_exec_sql_text(ts.sql_handle) st
        CROSS APPLY sys.dm_exec_query_plan(ts.plan_handle) qp
```

Troubleshooting Tools

Extended Events

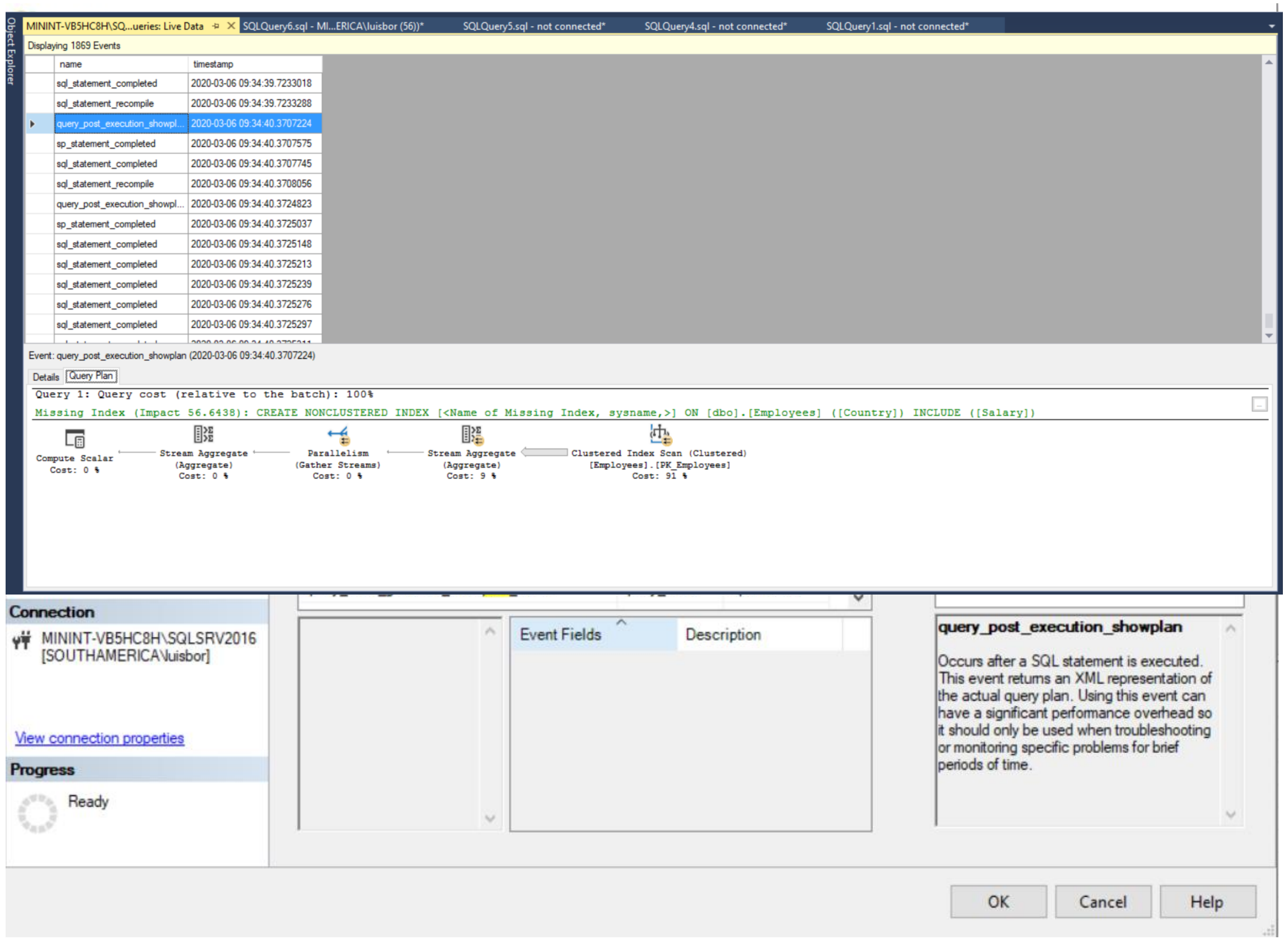
Identifying slow running queries

This architecture enables users to collect as much or as little data as is necessary to troubleshoot or identify a performance problem.

Extended Events is configurable, and it scales very well.

Some events of interest:

- sp_statement_completed
- sql_statement_completed
- rpc_completed
- sql_batch_completed
- wait_info
- large_cardinality_misestimate
- query_post_execution_showplan


MININT-VB5HC8H\SQ...ueries: Live Data
SQLQuery6.sql - MI...ERICA\luisbor (56))*
SQLQuery5.sql - not connected*
SQLQuery4.sql - not connected*
SQLQuery1.sql - not connected*
Object Explorer

Displaying 1869 Events

name	timestamp
sql_statement_completed	2020-03-06 09:34:39.7233018
sql_statement_recompile	2020-03-06 09:34:39.7233288
query_post_execution_showpl...	2020-03-06 09:34:40.3707224
sp_statement_completed	2020-03-06 09:34:40.3707575
sql_statement_completed	2020-03-06 09:34:40.3707745
sql_statement_recompile	2020-03-06 09:34:40.3708056
query_post_execution_showpl...	2020-03-06 09:34:40.3724823
sp_statement_completed	2020-03-06 09:34:40.3725037
sql_statement_completed	2020-03-06 09:34:40.3725148
sql_statement_completed	2020-03-06 09:34:40.3725213
sql_statement_completed	2020-03-06 09:34:40.3725239
sql_statement_completed	2020-03-06 09:34:40.3725276
sql_statement_completed	2020-03-06 09:34:40.3725297

Event: query_post_execution_showplan (2020-03-06 09:34:40.3707224)

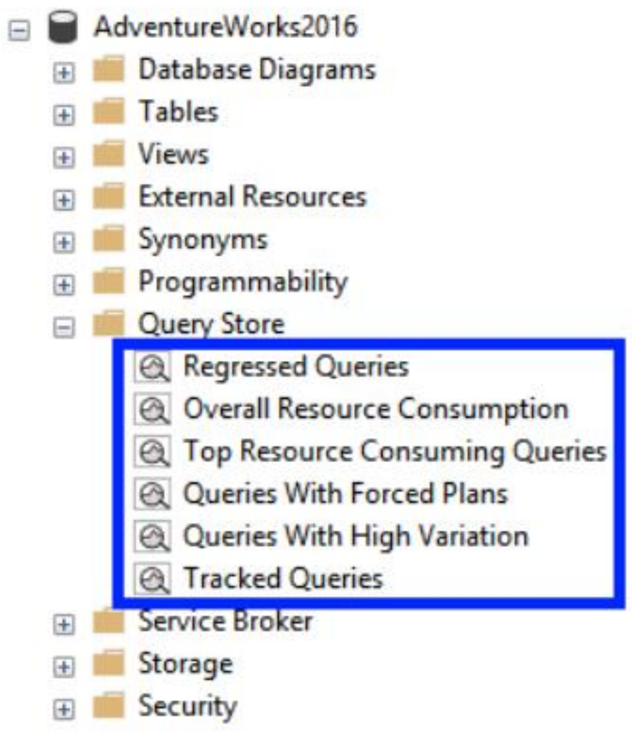
```
Query 1: Query cost (relative to the batch): 100%
Missing Index (Impact 56.6438): CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname,>] ON [dbo].[Employees] ([Country]) INCLUDE ([Salary])
```

query_post_execution_showplan

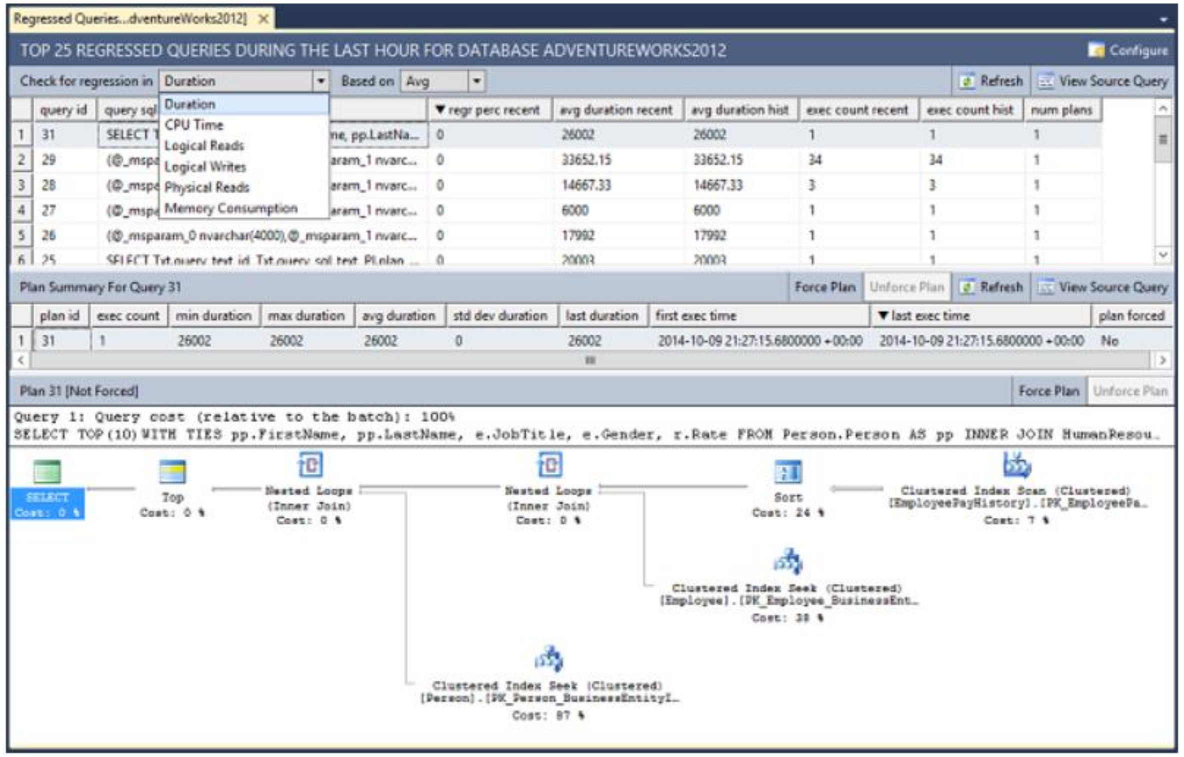
Occurs after a SQL statement is executed. This event returns an XML representation of the actual query plan. Using this event can have a significant performance overhead so it should only be used when troubleshooting or monitoring specific problems for brief periods of time.

Troubleshooting Tools

Query Store



- AdventureWorks2016
  - Database Diagrams
  - Tables
  - Views
  - External Resources
  - Synonyms
  - Programmability
  - Query Store
    - Regressed Queries
    - Overall Resource Consumption
    - Top Resource Consuming Queries
    - Queries With Forced Plans
    - Queries With High Variation
    - Tracked Queries
  - Service Broker
  - Storage
  - Security



Regressed Queries...dventureWorks2012]

TOP 25 REGRESSED QUERIES DURING THE LAST HOUR FOR DATABASE ADVENTUREWORKS2012

Check for regression in: Duration — Based on: Avg

Duration / CPU Time / Logical Reads / Logical Writes / Physical Reads / Memory Consumption

	query id	query sql	▼ regr perc recent	avg duration recent	avg duration hist	exec count recent	exec count hist	num plans
1	31	SELECT T…ne, pp.LastNa...	0	26002	26002	1	1	1
2	29	(@_mspa…aram_1 nvarc...	0	33652.15	33652.15	34	34	1
3	28	(@_mspa…aram_1 nvarc...	0	14667.33	14667.33	3	3	1
4	27	(@_mspa…aram_1 nvarc...	0	6000	6000	1	1	1
5	26	(@_msparam_0 nvarchar(4000),@_msparam_1 nvarc...	0	17992	17992	1	1	1
6	25	SELECT Txt.query_text_id, Txt.query_sql_text, Pl.plan_...	0	20003	20003	1	1	1

Plan Summary For Query 31

	plan id	exec count	min duration	max duration	avg duration	std dev duration	last duration	first exec time	▼ last exec time	plan forced
1	31	1	26002	26002	26002	0	26002	2014-10-09 21:27:15.6800000 +00:00	2014-10-09 21:27:15.6800000 +00:00	No

Plan 31 [Not Forced]

```
Query 1: Query cost (relative to the batch): 100%
SELECT TOP(10) WITH TIES pp.FirstName, pp.LastName, e.JobTitle, e.Gender, r.Rate FROM Person.Person AS pp INNER JOIN HumanResou…
```

Configure
Refresh
View Source Query
Force Plan
Unforce Plan
SELECT Cost: 0 %
Top Cost: 0 %
Nested Loops (Inner Join) Cost: 0 %
Nested Loops (Inner Join) Cost: 0 %
Sort Cost: 24 %
Clustered Index Scan (Clustered) [EmployeePayHistory].[PK_EmployeePa… Cost: 7 %
Clustered Index Seek (Clustered) [Employee].[PK_Employee_BusinessEnt… Cost: 38 %
Clustered Index Seek (Clustered) [Person].[PK_Person_BusinessEntityI… Cost: 87 %

Provides you with insight on query plan choice and performance.

It simplifies performance troubleshooting by helping you quickly find performance differences caused by query plan changes.

Query Store automatically captures a history of queries, plans, and runtime statistics, and retains these for your review.

It separates data by time windows so you can see database usage patterns and understand when query plan changes happened on the server.

Demonstration

Query Troubleshooting using DMVs

Query Troubleshooting

- Applying query troubleshooting techniques with DMVs, and Extended Events

Questions?

Knowledge Check

Name two Dynamic Management Views (DMVs) used to obtain information about cached execution plans.

What are three events can be used when monitoring SQL Server for query performance?

Microsoft