Lesson Objectives

- In this Lesson you will learn the following:
  - How Query Store Works
  - Use Cases and Benefits of Query Store
  - How to Use Query Store to Troubleshoot Issues
  - How Live Query Statistics Works
  - How to Leverage Live Query Statistics to Troubleshoot Running Queries

Introducing Query Store

Mission Critical Performance

SQL Server 2016 Mission-Critical Performance

Performance

Operational analytics
Insights on operational data; Works with in-memory OLTP and disk-based OLTP

In-memory OLTP enhancements
Greater T-SQL surface area, terabytes of memory supported, and greater number of parallel CPUs

Query Store
Monitor and optimize query plans

Native JSON
Expanded support for JSON data

Temporal database support
Query data as points in time

Security

Always encrypted
Sensitive data remains encrypted at all times with ability to query

Row-level security
Apply fine-grained access control to table rows

Dynamic data masking
Real-time obfuscation of data to prevent unauthorized access

Other enhancements
Audit success/failure of database operations

TDE support for storage of in-memory OLTP tables

Enhanced auditing for OLTP with ability to track history of record changes

Availability

Enhanced AlwaysOn
Three synchronous replicas for auto failover across domains

Round robin load balancing of replicas

Automatic failover based on database health

DTC for transactional integrity across database instances with AlwaysOn

Support for SSIS with AlwaysOn

Scalability

Enhanced database caching
Cache data with automatic, multiple TempDB files per instance in multi-core environments

SQL Server 2016 Mission-Critical Performance

Performance

Operational analytics
Insights on operational data; Works with in-memory OLTP and disk-based OLTP

In-memory OLTP enhancements
Greater T-SQL surface area, terabytes of memory supported, and greater number of parallel CPUs

Query Store
Monitor and optimize query plans

Native JSON
Expanded support for JSON data

Temporal database support
Query data as points in time

Security

Always encrypted
Sensitive data remains encrypted at all times with ability to query

Row-level security
Apply fine-grained access control to table rows

Dynamic data masking
Real-time obfuscation of data to prevent unauthorized access

Other enhancements
Audit success/failure of database operations

TDE support for storage of in-memory OLTP tables

Enhanced auditing for OLTP with ability to track history of record changes

Availability

Enhanced AlwaysOn
Three synchronous replicas for auto failover across domains

Round robin load balancing of replicas

Automatic failover based on database health

DTC for transactional integrity across database instances with AlwaysOn

Support for SSIS with AlwaysOn

Scalability

Enhanced database caching
Cache data with automatic, multiple TempDB files per instance in multi-core environments

Introducing Query Store

Usage Summary

“A Bad Plan is not the one which failed, but the one which succeeded at the Greatest Cost.”

~ Anonymous DBA

Lesson: SQL Server Query Store

Usage Scenarios and Architecture

When Performance is not Good…

Website / App is down

Database is not working

Temporary issues

Impossible to predict / root cause

System Upgrade

Regression caused by upgrade

Plan choice change can cause these problems

What are your Options Today?

- Most solutions are reactive in nature
  - Flush the bad plan from the cache with sp_recompile
  - Flush the entire plan cache with DBCC FREEPROCCACHE
  - Force the plan to recompile every time
  - Restart OS / SQL Server (It works for some reason?)
- Proactive solutions are challenging
  - Often takes a long time to even detect that there is a plan problem
  - Only the latest plan is stored in the cache
  - Need to catch both the good and the bad plan in order to troubleshoot
  - Information is stored in memory only
    - Reboot or memory pressure causes diagnostic data to be lost
    - No history or timing available – stats are aggregated for what is currently in cache

Why Plan Changes Happen..

- SQL Query Optimizer considers many plans
- When a plan is chosen it is cached and reused
- As your data changes, the optimizer might select a different plan as optimal
- Volume and Data Distribution can affect plan choices
- Sometimes a rare plan choice will be cached (The Parameter Sensitive Plan Problem)

Addressing Plan Choice Regressions

- First You Have to find the “Slow” Query
- Figuring out Why it is slow is not Easy
- You may not have enough information to fix it
- Even if you do know what it is supposed to be…
  - **Can you modify the query to hint it?**
  - **Can you figure out how to make a plan guide?**

Tackling the Problem – What Could We Do?

1. Store the history of plans for each query
2. Baseline the performance of each plan over time
3. Identify queries that have “become slower recently”
4. Find a way to force plans quickly and easily
5. Make sure this works across server restarts, upgrades, and query recompiles

This is what the Query Store does for you!

Introducing the Query Store

- Query Store persists execution plans per database
- Runtime stats store persists execution statistics per database
- New views and graphical interface allow you to quickly and easily troubleshoot query performance
  - Quickly find query plan performance regressions
  - Fix plan regressions by forcing a previous plan
  - Determine the number of times a query was executed in a given time window
  - Identify Top N Queries in the past X hours
  - Audit the history of query plans for a given query
  - Analyze the resource usage patterns for a particular database

Key Usage Scenarios

Find and fix query plan regressions

Identify top resource consumers

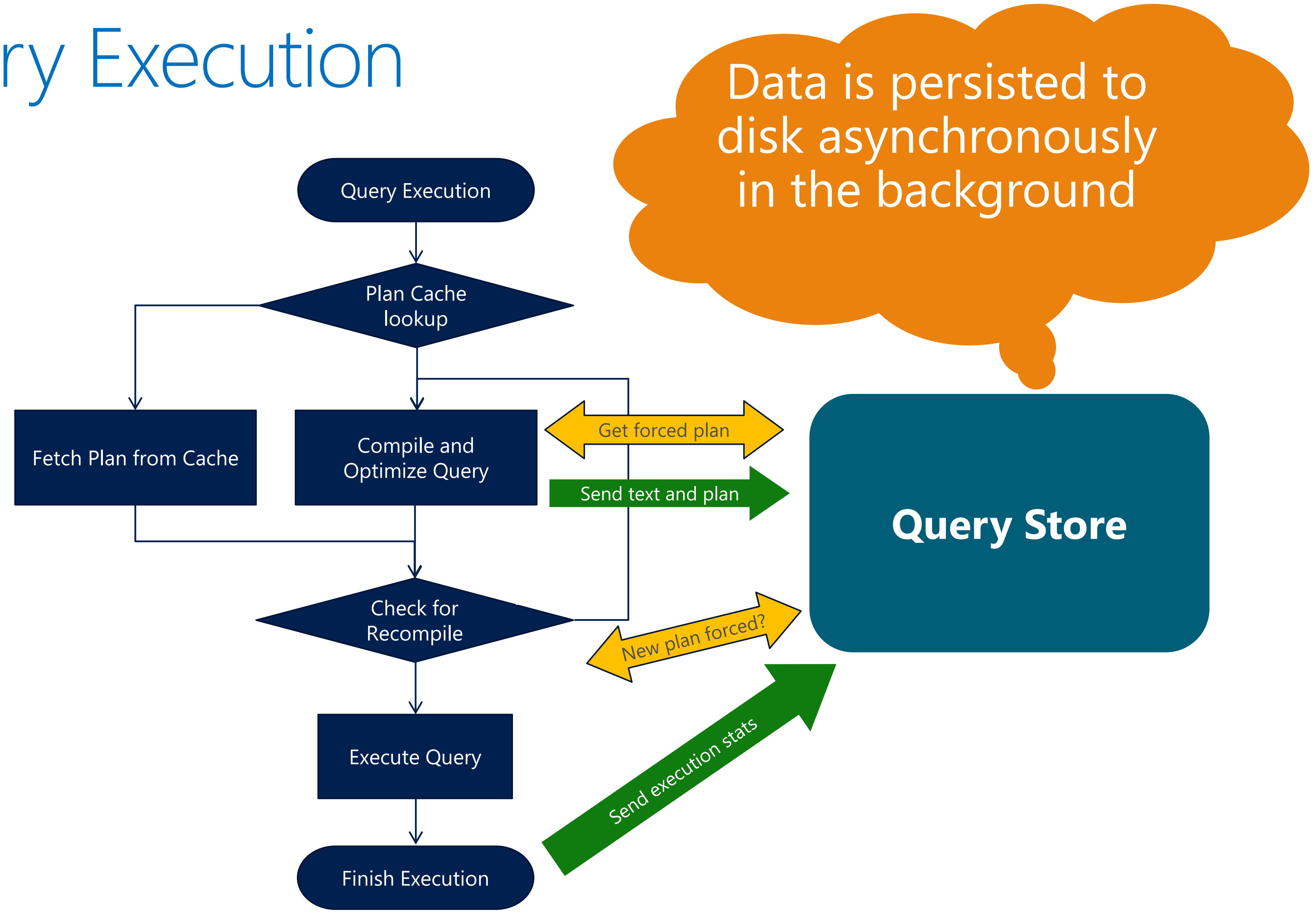
Reduce risks with server upgrade

Deep analysis of workload patterns/perf

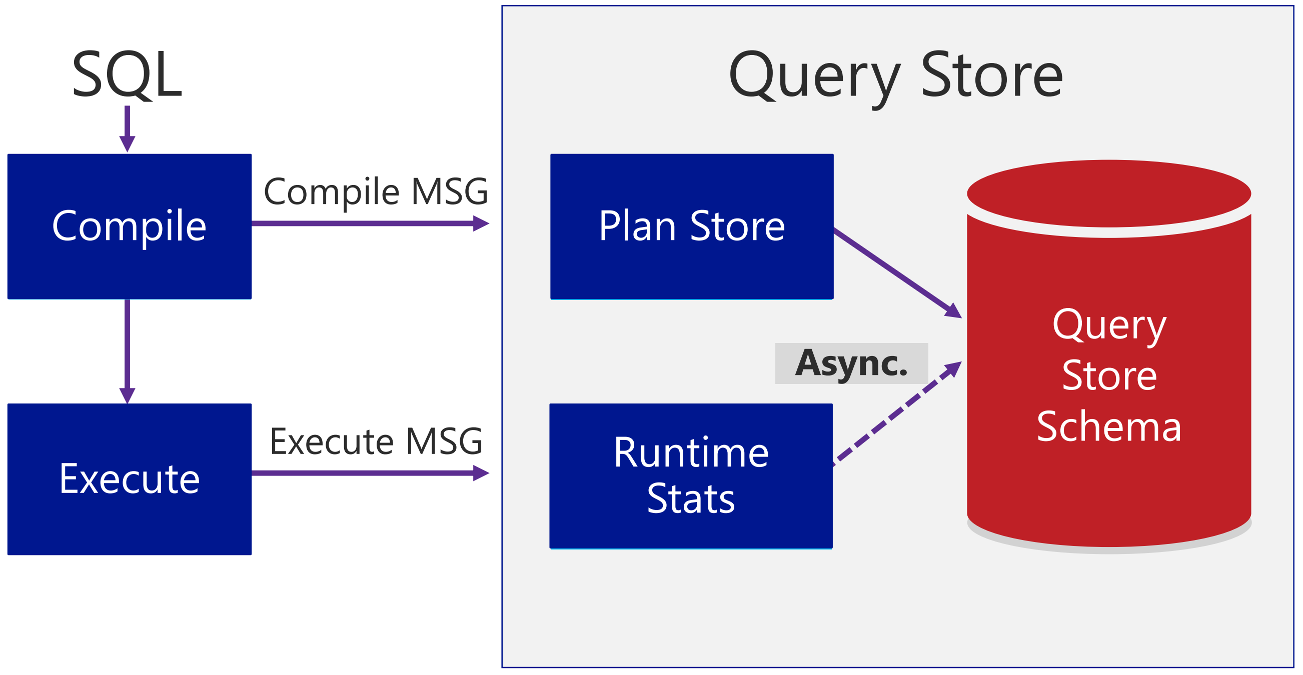
Short-term/tactical

Long-term/strategic

SQL Query Execution



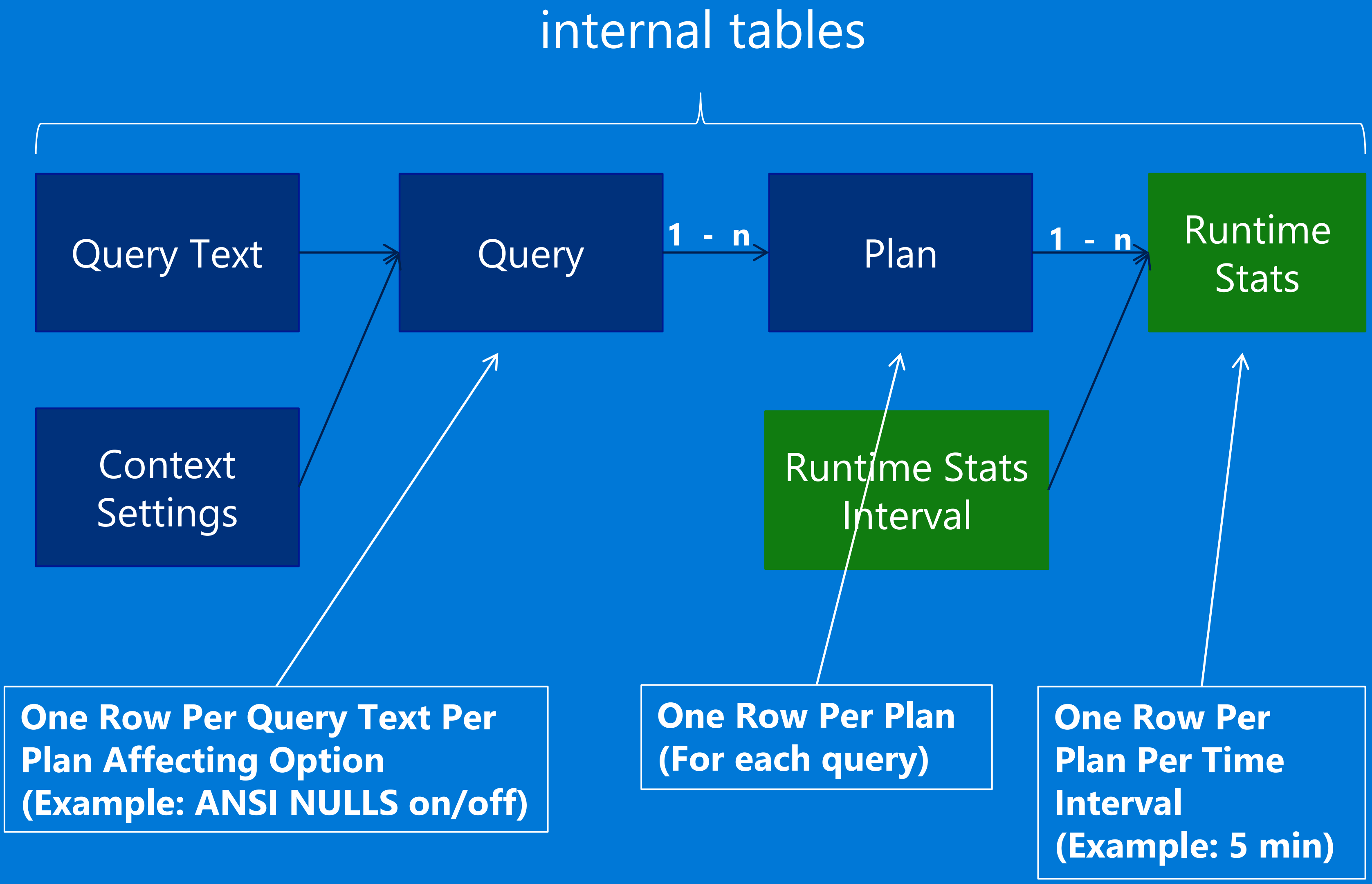
Query Store Architecture



Durability latency controlled by DB option
DATA_FLUSH_INTERNAL_SECONDS

- Collects Query Text and all relevant properties
- Stores all Plan Choices and Performance Metrics
- Works across restarts / upgrades / recompiles
- Dramatically lowers the bar for performance troubleshooting
- New DMVs support deeper analysis
- Intuitive and easy plan forcing

Query Store Schema Explained


internal tables
Query Text
Context Settings
Query
1 - n
Plan
1 - n
Runtime Stats
Runtime Stats Interval
One Row Per Query Text Per Plan Affecting Option (Example: ANSI NULLS on/off)
One Row Per Plan (For each query)
One Row Per Plan Per Time Interval (Example: 5 min)

exposed views

sys.

Compile stats:
query_store_query_text
query_context_settings
query_store_query
query_store_plan

Runtime stats:
query_store_runtime_stats_interval
query_store_runtime_stats

Query Store Details

- Plan and execution data are stored on disk in the user database
- Query Store is configurable
  - Settings such as MAX_SIZE_MB, QUERY_CAPTURE_MODE, CLEANUP_POLICY allow you to decide how much data you want to store for how long
  - Can be configured either via the SSMS GUI or T-SQL scripts
- Query Store can be viewed and managed via scripting or SSMS

What does Query Store Track?

- Query Texts begins at the first character of the first token of the statement; end at last character of last token
  - Comments before/after do not count
  - Spaces and comments inside *do* count
- Context_settings contains one row per unique combination of plan-affecting settings
  - Different SET options cause multiple plan entries in the Query Store
  - Plan caching/recompilation behavior unaffected

What Gets Captured?

- Query Texts
- Query Plans
- Runtime Statistics (Per unit of time, default 1 hour)
  - Count of executions of each captured plan
  - For each metric: average, last, min, max, stddev
  - Metrics: duration, cpu_time, logical_io_reads, logical_io_writes, physical_io_reads, clr_time, DOP, query_max_used_memory, rowcount
  - Data is recorded when a query execution ends

Demonstration: Enabling Query Store in SQL Server 2016 and View Query Store Properties

Lesson: Performance Features in SQL Server

Query Store Use Cases

Keeping Stability While Upgrading SQL Server

Upgrade to SQL vNext

Maintain Older Compatibility Level

Freeze plans (optional)

→

Run Query Store and Establish a baseline

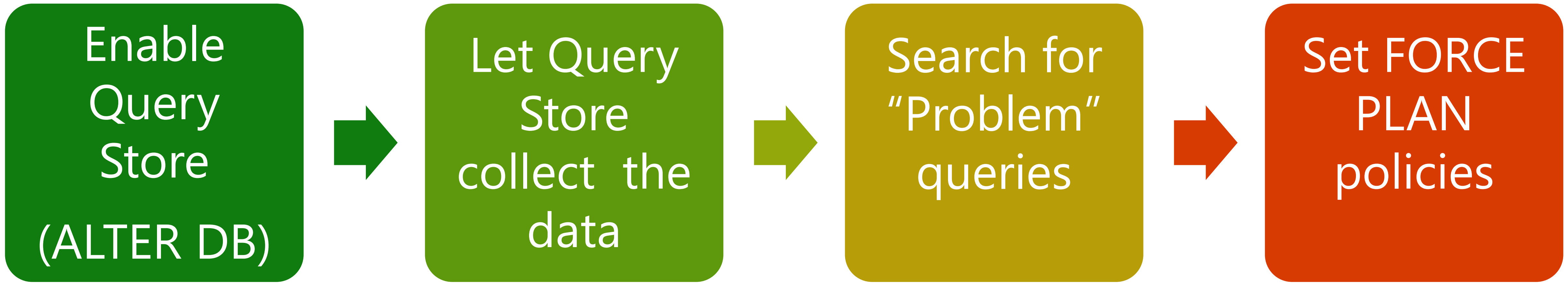
→

Move to a later Compatibility Level (ex. 130) and unfreeze plans

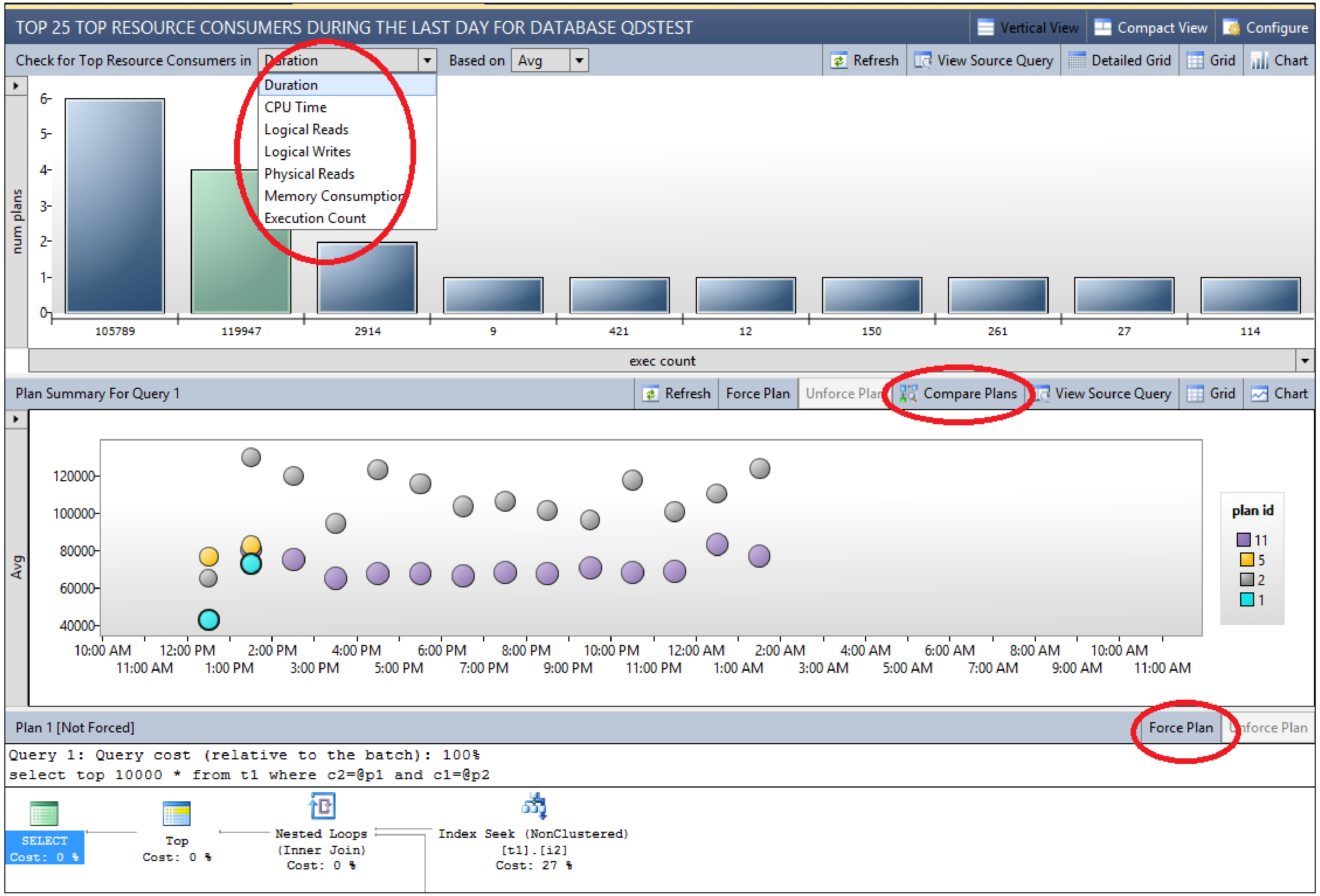
→

Monitor and fix regressions with plan forcing

Performance Troubleshooting with Query Store

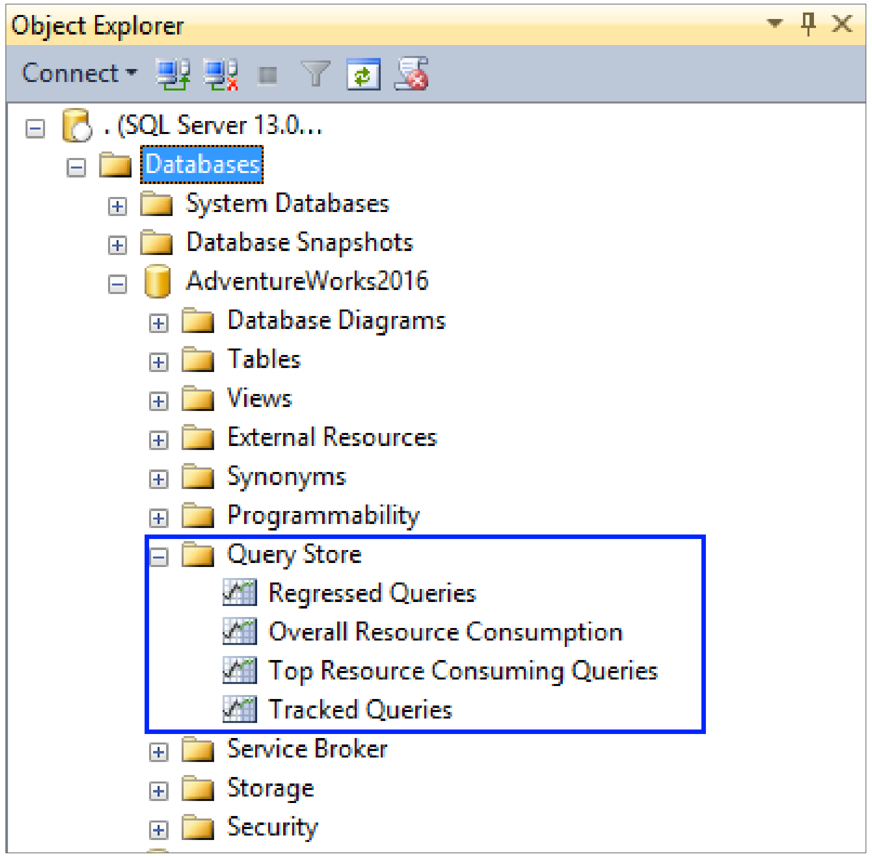

Enable Query Store
(ALTER DB)
Let Query Store collect the data
Search for “Problem” queries
Set FORCE PLAN policies

Monitoring Performance By Using the Query Store



```
Query 1: Query cost (relative to the batch): 100%
select top 10000 * from t1 where c2=@p1 and c1=@p2
```

Query Store Report Views



Object Explorer

Connect ▾

- . (SQL Server 13.0...
  - Databases
    - System Databases
    - Database Snapshots
    - AdventureWorks2016
      - Database Diagrams
      - Tables
      - Views
      - External Resources
      - Synonyms
      - Programmability
      - Query Store
        - Regressed Queries
        - Overall Resource Consumption
        - Top Resource Consuming Queries
        - Tracked Queries
      - Service Broker
      - Storage
      - Security

SSMS View	Scenario
Regressed Queries	Pinpoint queries for which execution metrics have recently regressed. Use this view to correlate observed performance problems in your application with the actual queries that needs to be fixed or improved.
Top Resource Consuming Queries	Choose an execution metric of interest and identify queries that had the most extreme values for a provided time interval. Use this view to focus your attention on the most relevant queries which have the biggest impact.
Tracked Queries	Track the execution of the most important queries in real-time. Typically, you use this view when you have queries with forced plans and you want to make sure that query performance is stable.
Overall Resource Consumption	Analyze the total resource consumption for the database for any of the execution metrics. Use this view to identify resource patterns (daily vs. nightly workloads) and optimize overall consumption for your database.

Working with Query Store

```
/* (6) Performance analysis using Query
Store views*/

SELECT q.query_id, qt.query_text_id,
qt.query_sql_text, SUM(rs.count_executions)
AS total_execution_count
FROM
sys.query_store_query_text qt JOIN
sys.query_store_query q ON qt.query_text_id
= q.query_text_id JOIN
sys.query_store_plan p ON q.query_id =
p.query_id JOIN
sys.query_store_runtime_stats rs ON
p.plan_id = rs.plan_id
GROUP BY q.query_id, qt.query_text_id,
qt.query_sql_text
ORDER BY total_execution_count DESC

/* (7) Force plan for a given query */
exec sp_query_store_force_plan
12 /*@query_id*/, 14 /*@plan_id*/
```

- Database level features exposed through T-SQL
- ALTER DATABASE
- Catalog views (settings, compile & runtime stats)
- Stored Procedures used to force plans, clean up Query Store, and reset Query Store statistics

Key DMVs for Query Store

```
SELECT * FROM sys.query_store_query_text

SELECT * FROM sys.query_store_query

SELECT * FROM sys.query_store_plan

SELECT * FROM sys.query_store_runtime_stats
ORDER BY runtime_stats_id

SELECT * FROM
sys.query_store_runtime_stats_interval

SELECT * FROM sys.query_context_settings
```

- The DMVs shown here are enabled and populated for each database when Query Store is turned on

Demonstration: Using Query Store in SQL Server 2016

Lesson: Performance Features in SQL Server

Query Store Considerations and Best Practices

Troubleshooting Query Store

- Plan forcing does not always work
  - Example: If you drop an index, you cannot force a plan that uses it
- Query Store will revert to not forcing if it fails
  - This keeps the application working if the hint breaks
- You can see which plans are failing to force by looking at the Plan Table:

```sql
SELECT * FROM sys.query_store_plan
WHERE is_forced_plan = 1 AND force_failure_count > 0
```

Query Store Considerations for In Memory OLTP

- Runtime statistics collection is controlled with **sys.sp_xtp_control_query_exec_stats** and is not enabled by default
- **is_natively_compiled** field added to **sys.query_store_plan** to help finding queries generated by the native code compilation
- Memory grants metrics within **sys.query_store_runtime_stats** are not populated for natively compiled queries; their values are always 0
- Improving implementation of time-based cleanup (configured with **STALE_QUERY_THRESHOLD_DAYS**) to run in multiple transactions, holding database lock for a shorter period of time and thus minimize impact on customer workload

With Query Store…

- Obtain a full history of query execution
- Quickly pinpoint the most expensive queries
- Identify all query plan regressions
- Easily force better plan from history
- Perform server restarts without losing diagnostic data
- Safely perform upgrades or plan affecting operations

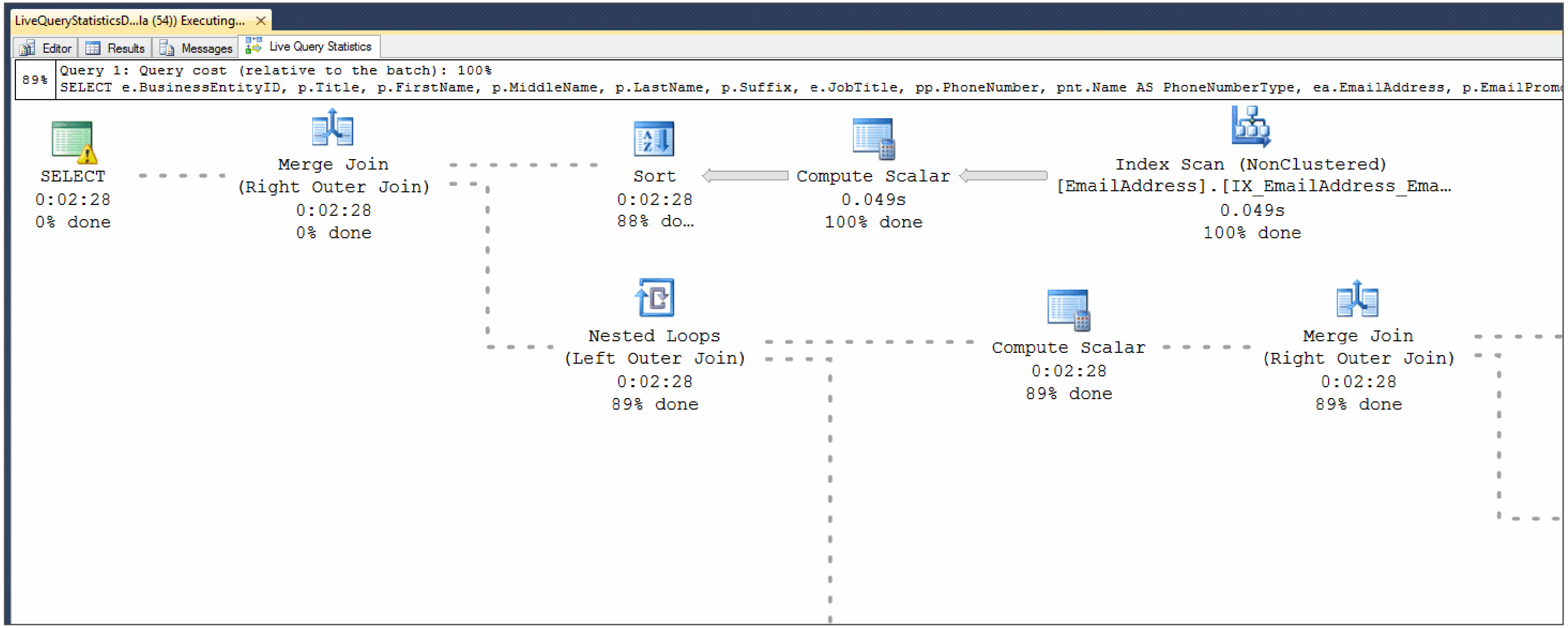
Query Store Best Practices

- Use the Latest SQL Server Management Studio
- Use Query Performance Insight in Azure SQL Database
- Keep Query Store Adjusted to your Workload
- Keep the Most Relevant Data in Query Store
- Check the Status of Forced Plans Regularly
- Avoid Renaming Databases if you have Queries with Forced Plans

Lesson: Performance Features in SQL Server

Live Query Statistics

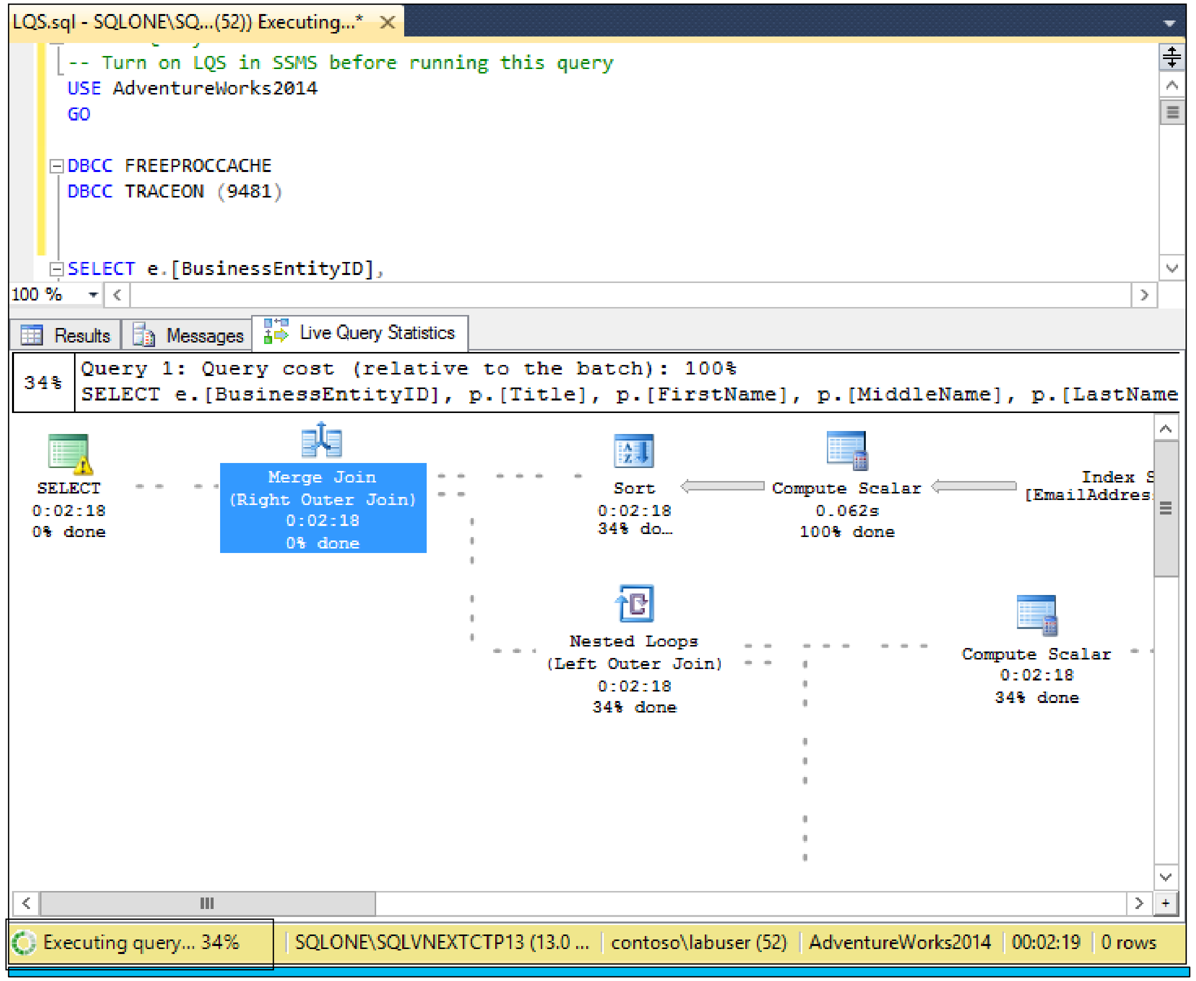
Introducing Live Query Execution Plans



```
89%  Query 1: Query cost (relative to the batch): 100%
     SELECT e.BusinessEntityID, p.Title, p.FirstName, p.MiddleName, p.LastName, p.Suffix, e.JobTitle, pp.PhoneNumber, pnt.Name AS PhoneNumberType, ea.EmailAddress, p.EmailProm
```

Live Query Statistics

Collect actual metrics about query while running



LQS.sql - SQLONE\SQ...(52)) Executing...*

```
-- Turn on LQS in SSMS before running this query
USE AdventureWorks2014
GO

DBCC FREEPROCCACHE
DBCC TRACEON (9481)


SELECT e.[BusinessEntityID],
```

```
Query 1: Query cost (relative to the batch): 100%
SELECT e.[BusinessEntityID], p.[Title], p.[FirstName], p.[MiddleName], p.[LastName
```

Executing query... 34% | SQLONE\SQLVNEXTCTP13 (13.0 ... | contoso\labuser (52) | AdventureWorks2014 | 00:02:19 | 0 rows

- View CPU/memory usage, execution time, query progress, etc.
- Enables rapid identification of potential bottlenecks for troubleshooting query performance issues.
- Allows drill down to live operator level statistics:
  - Number of generated rows
  - Elapsed time
  - Operator progress
  - Live warnings, etc.

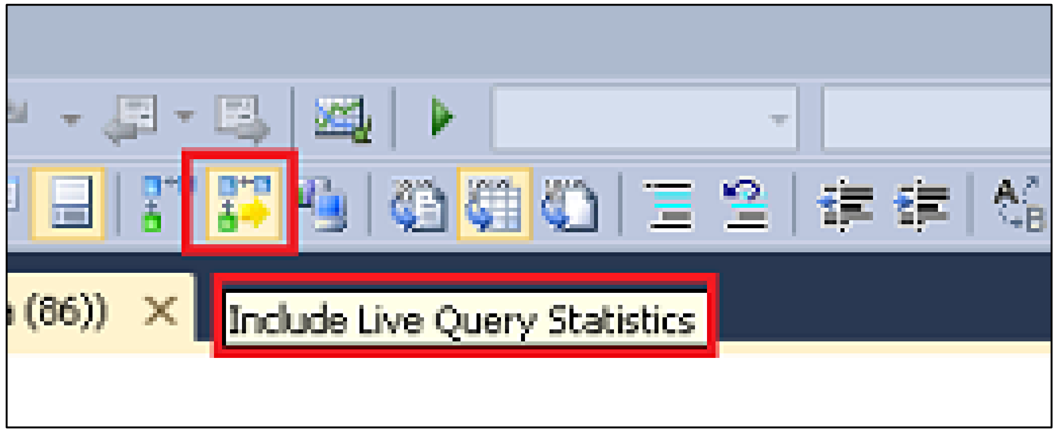
Live Query Statistics

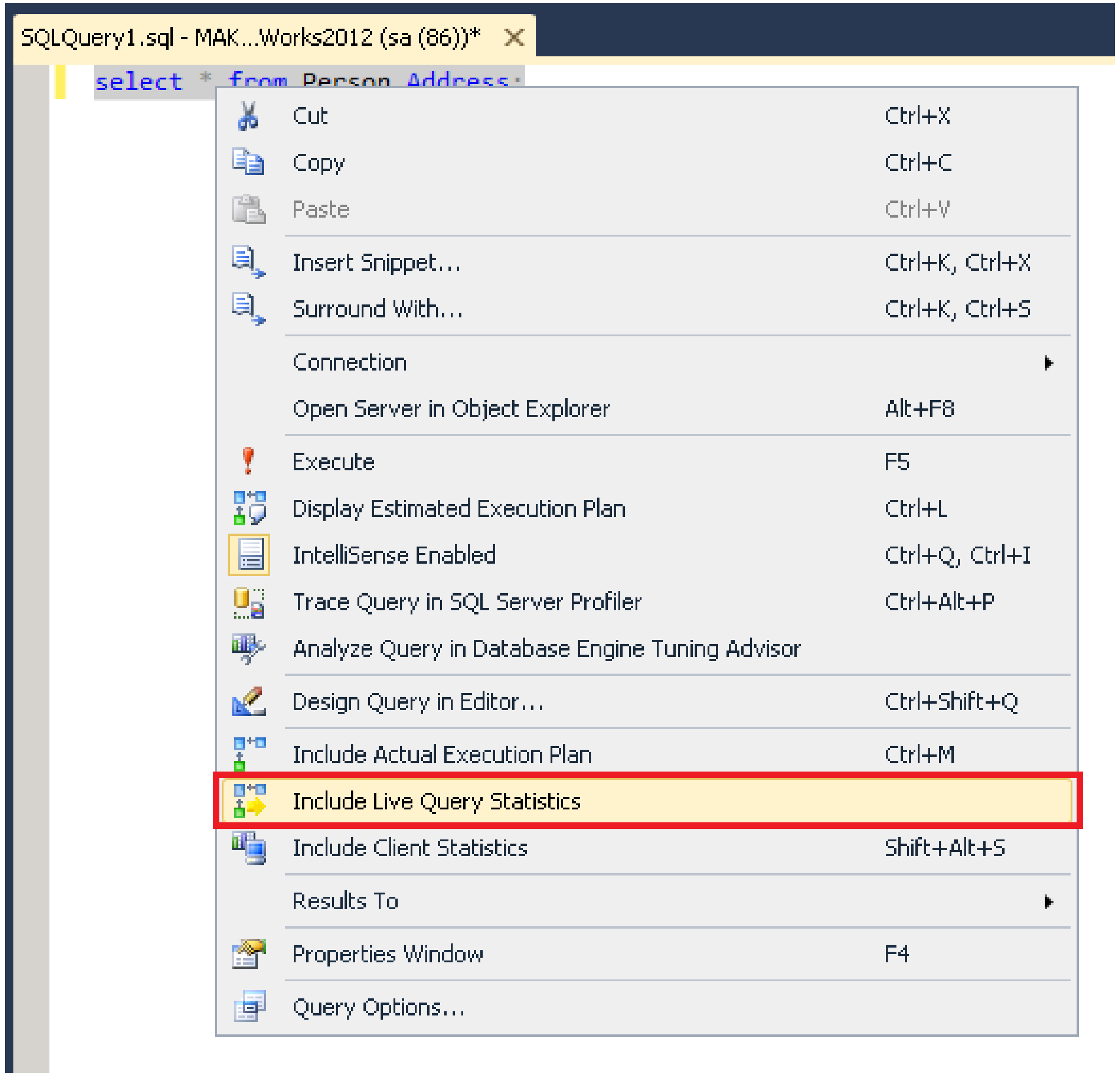
- Information available in the sys.dm_exec_query_profiles DMV
- Query must be executing to view LQS (cannot save and share)
- Accessible via SSMS query window or Activity Monitor
- DMV is available in SQL 2014, query statistics work against 2014 if you have SQL 2016 SSMS
- Slight performance overhead to monitor live statistics

Enabling Live Query Statistics

- The statistics profile infrastructure must be enabled before live query statistics can capture information about the progress of queries
  - Specifying Include Live Query Statistics in SSMS enables the statistics for the current query session
  - Execute SET STATISTICS XML ON; or SET STATISTICS PROFILE ON; in the target session
  - You can enable from Activity Monitor to view other sessions
- Natively compiled stored procedures are not supported

Enabling Live Query Statistics



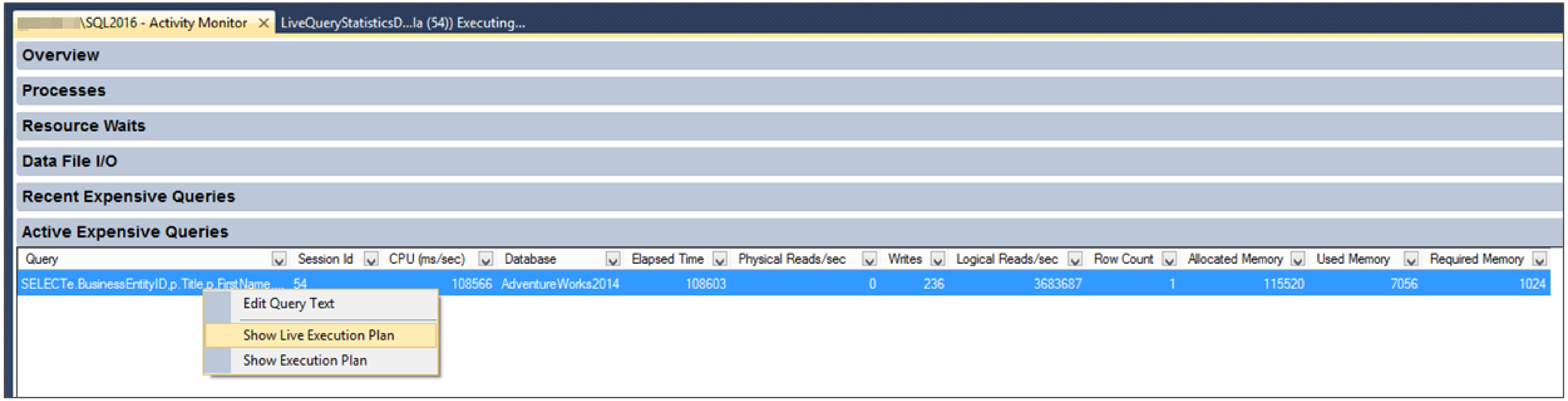


SQLQuery1.sql - MAK...Works2012 (sa (86))*

```
select * from Person.Address;
```

Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Insert Snippet... Ctrl+K, Ctrl+X
Surround With... Ctrl+K, Ctrl+S
Connection
Open Server in Object Explorer Alt+F8
Execute F5
Display Estimated Execution Plan Ctrl+L
IntelliSense Enabled Ctrl+Q, Ctrl+I
Trace Query in SQL Server Profiler Ctrl+Alt+P
Analyze Query in Database Engine Tuning Advisor
Design Query in Editor... Ctrl+Shift+Q
Include Actual Execution Plan Ctrl+M
Include Live Query Statistics
Include Client Statistics Shift+Alt+S
Results To
Properties Window F4
Query Options...

Monitor Currently Executing Queries



[redacted]\SQL2016 - Activity Monitor × LiveQueryStatisticsD...la (54)) Executing...

Overview

Processes

Resource Waits

Data File I/O

Recent Expensive Queries

Active Expensive Queries

Query	Session Id	CPU (ms/sec)	Database	Elapsed Time	Physical Reads/sec	Writes	Logical Reads/sec	Row Count	Allocated Memory	Used Memory	Required Memory
SELECTe.BusinessEntityID,p.Title,p.FirstName,...	54	108566	AdventureWorks2014	108603	0	236	3683687	1	115520	7056	1024

References

- What's New in Database Engine https://msdn.microsoft.com/en-us/library/bb510411.aspx
- What's New in Database Engine – Query Store (2016) https://msdn.microsoft.com/en-us/library/bb510411.aspx#QueryStore
- Using the Query Store with In-Memory OLTP https://msdn.microsoft.com/en-us/library/mt590480.aspx
- Best Practices with the Query Store

  https://msdn.microsoft.com/en-us/library/mt604821.aspx
- Monitoring Performance By Using the Query Store

  https://msdn.microsoft.com/en-us/library/dn817826.aspx
- Live Query Statistics https://msdn.microsoft.com/en-us/library/dn831878.aspx