Microsoft Services

Advanced Query Store and AQP in SQL Server 2016/2017

Microsoft

SQL Server 2016 / 2017 Mission-Critical (DB Engine)

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

Live Query Statistics

Query Store
Monitor and optimize query plans

Automatic Database Tuning
Provides insight into potential query performance problems, recommends solutions, and can automatically fix identified problems

DMV Improvements

Adaptive Query Processing
A feature family that introduces a new generation of query processing improvements

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

Advanced Threat Detection
Ability to find unusual login patterns, track usage behavior in an auditing database, track SQL injection vulnerability, and more

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 / Platform

SQL Server 2017 on Linux

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

Stretch Database
Archive historical data transparently and securely to Azure

Queries stretch across local data as well as Azure data

Scalability

Machine Learning Services
R Scripting along with Python scripting from the SQL Server Engine

Graph DB Support
For modeling many-to-many relationships

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

New Programmatic Improvements
New TSQL Functionality, Maintenance Plan Improvements, New ALTER DATABASE Options
Expanded support for JSON data
New PolyBase query engine integrates SQL Server with external data in Hadoop or Azure Blob storage

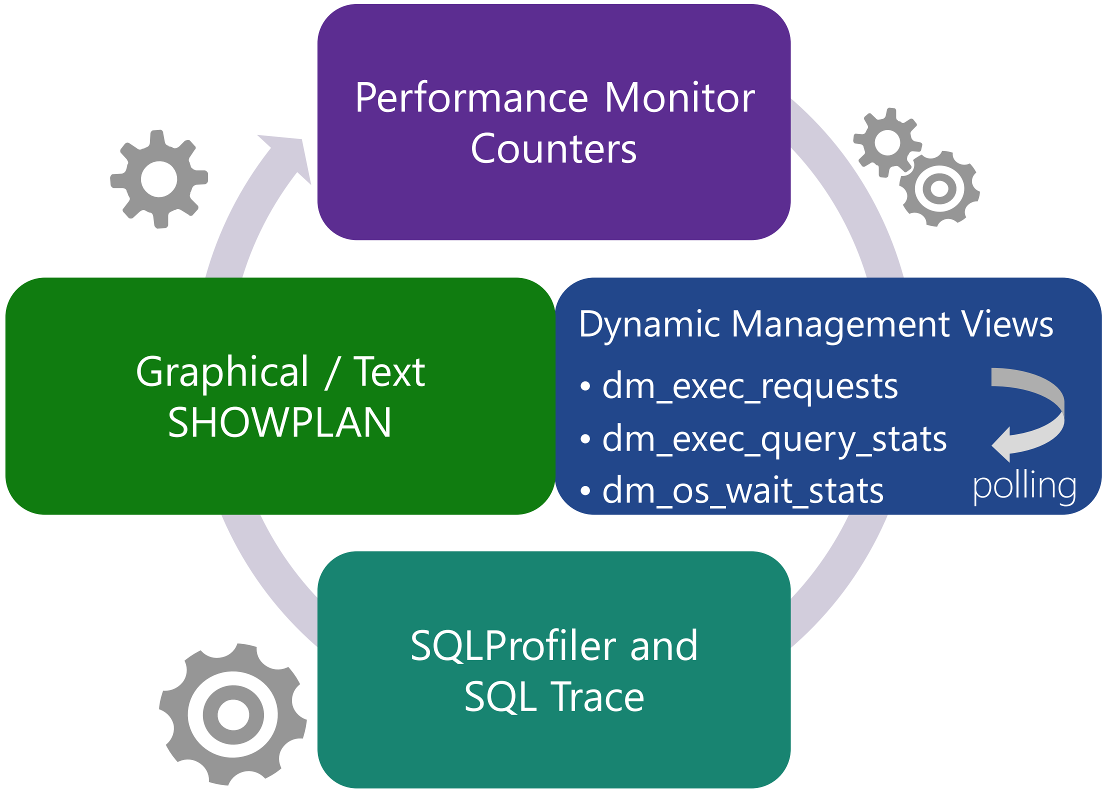
Temporal Database Support
Query data as points in time

New in SQL Server 2017

SQL Server 2016/2017 Monitoring and Tooling

Full SQLTrace Parity+ since 2012

Traditional Troubleshooting



- Extended Events is scalable
- Query Store is persisted and improving
- Performance Dashboard Reports
- Live Query Statistics
- Lightweight Query Profiling
- Expanded Query Plan Diagnostics
- SSMS Dump Analysis (Preview)
- SQL Server Vulnerability Assessment
- SQL Data Discovery and Classification

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

~ Anonymous DBA

Building Blocks: Plan Guides

How we got here..

Why Use Plan Guides?

- Useful for tuning queries generated by 3rd party applications
- Plan guides work by keeping a list of queries on the server, along with the Hints you want to apply
- You need to provide SQL Server with the query you want to optimize and a query hint using the OPTION clause
- When the query is optimized, SQL Server will apply the hint requested in the plan guide definition

Plan Guides Stored Procedures

- Use the sp_create_plan_guide stored procedure to create a plan guide
- Use sp_control_plan_guide to drop enable or disable plan guides
- You can see which plan guides are defined in your database using the sys.plan_guides catalog view
- Note: When Using Plan Guides, you must match Query Text and Parameter Names exactly

Common Query Hints Used in Plan Guides

- OPTIMIZE FOR (Value, Unknown)
- RECOMPILE
- MAXDOP #
- FORCE ORDER
- USE PLAN
- NULL

Plan Guides from Cache

- Also known as ‘Plan Freezing’
- Prevents a current cached plan from changing
- Example:

```
-- Create a plan guide for the query by specifying the query plan in the plan
cache.
DECLARE @plan_handle varbinary(64);
DECLARE @offset int;
SELECT @plan_handle = plan_handle, @offset = qs.statement_start_offset
FROM sys.dm_exec_query_stats AS qs
CROSS APPLY sys.dm_exec_sql_text(sql_handle) AS st
CROSS APPLY sys.dm_exec_text_query_plan(qs.plan_handle,
qs.statement_start_offset,
qs.statement_end_offset) AS qp
WHERE text LIKE N'SELECT WorkOrderID, p.Name, OrderQty, DueDate%';

EXECUTE sp_create_plan_guide_from_handle
    @name =  N'Guide1',
    @plan_handle = @plan_handle,
    @statement_start_offset = @offset;
GO
-- Verify that the plan guide is created.
SELECT * FROM sys.plan_guides
WHERE scope_batch LIKE N'SELECT WorkOrderID, p.Name, OrderQty, DueDate%';
GO
```

USE PLAN

- Used to explicitly guide the optimizer to use a specific plan
- Accepts an XML Showplan as the parameter
- **Note:** Plans larger than 8KB cannot be used

```
SELECT *
FROM Sales.SalesOrderHeader h, Sales.SalesOrderDetail
OPTION (USE PLAN N'<ShowPlanXML
xmlns="http://schemas.microsoft.com/sqlserver/2004/07/showplan"
Version="0.5"
Build="9.00.1187.07">
  <BatchSequence>
    <Batch>
      <Statements>
   …
      </Statements>
    </Batch>
  </BatchSequence>
</ShowPlanXML>
')
```

.Now: Query Store

SQL Server 2016 / 2017
(Azure SQL DB)

Query and Query Plan Fingerprints

- Query Fingerprint
  - query_hash
  - Explicitly identifies a specific query in the cache.
  - *sys.dm_exec_requests*
  - *sys.dm_exec_query_stats*
- SQL Handle
  - sql_handle
  - Token for the SQL text that relates to a batch.
  - *sys.dm_exec_sql_text*
  - *sys.dm_exec_query_stats*
  - *sys.dm_exec_query_memory_grants*
- Query Plan Fingerprints
  - query_plan_hash
  - Useful to determine queries that share the same execution plan.
  - Can be used to determine if the query plan has changed.
  - *sys.dm_exec_requests*
  - *sys.dm_exec_query_stats*
- Plan Handle
  - plan_handle
  - Token for a cached execution plan.
  - *sys.dm_exec_query_plan*
  - *sys.dm_exec_cached_plans*

When performance is not good…

- Database is not working

Website / App is down

- Impossible to predict / root cause

Temporary Perf. issues

- Regression caused by upgrade

System Upgrade

Plan choice change can cause these problems

Have You Ever...?

...Had your system down/slowed down and everyone waiting for you to magically fix the problem ASAP?

...Upgraded on and had an issu on down?

...Had a prob performance and been unable to determine what ong?

Query Plan choice changes can cause all of these problems!

What are you doing 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 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 <u>data to be lost</u>
    - No history or timing available – stats are aggregated for what is currently in cache

Addressing Plan Choice Regressions

- First You Have to find the “Slow” Query
- Figuring out Why it is slow isn’t 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 “gotten 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

- Plan 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

Demonstration:
Enabling Query Store in SQL Server 2016

- Query Store Properties

Key Usage Scenarios

Find and fix query plan regressions

Identify top resource consumers

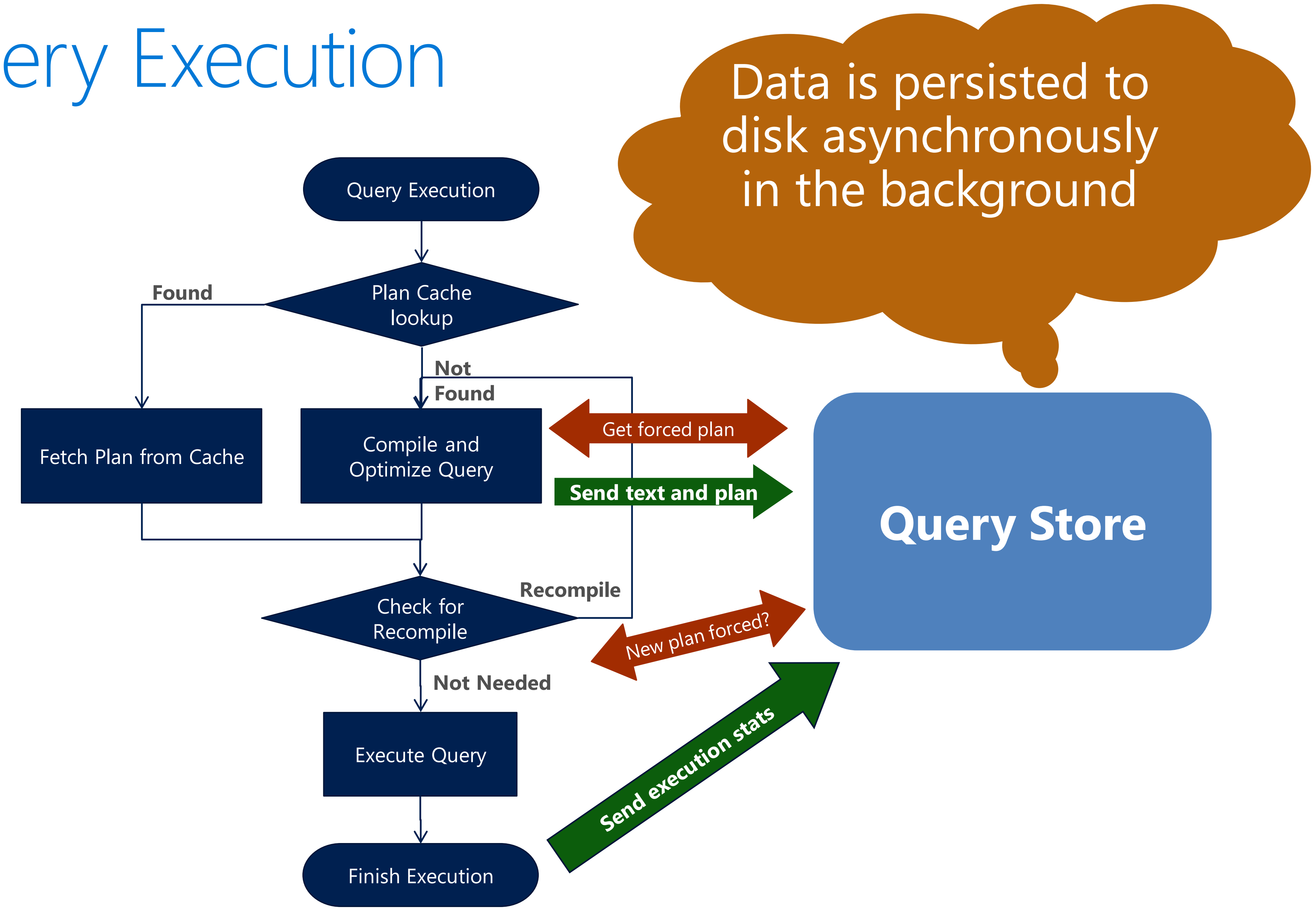
Reduce risks with server upgrade

Deep analysis of workload patterns/perf

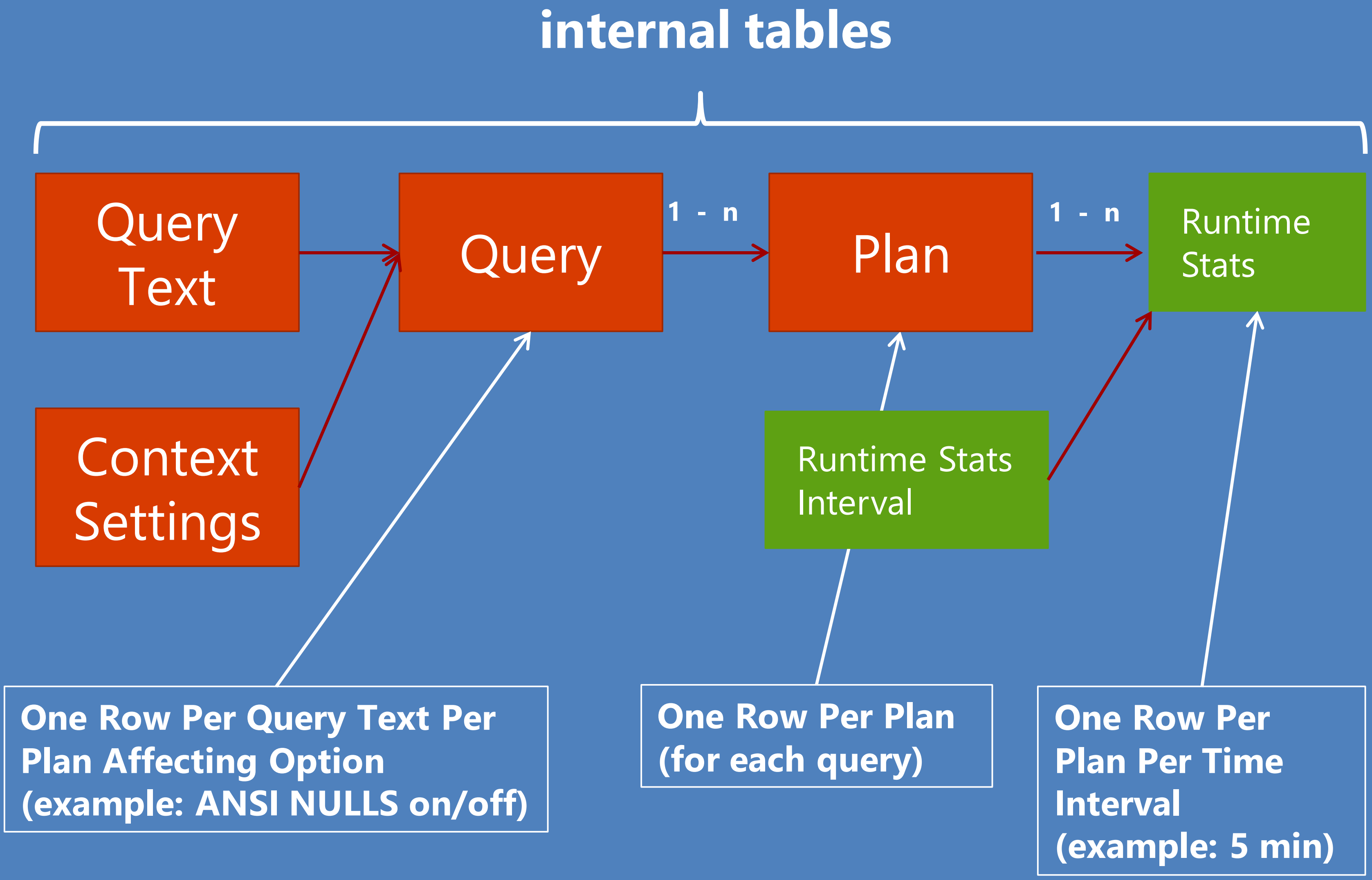
Short-term/tactical

Long-term/strategic

SQL Query Execution



Query Store 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
sys.query_store_wait_stats (2017)

Runtime stats:

query_store_runtime_stats_interval
query_store_runtime_stats

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_store_wait_stats

SELECT * FROM sys.query_context_settings
```

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

Query Store Details

- Plans and execution data are stored on disk in the user database
  - Query store data persists reboots, upgrades, restores etc.
  - Plans and statistics are tracked at the database level rather than the server level
- 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 start 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 “queries” 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*

Keeping stability while upgrading to SQL Sever 2016/2017

Upgrade to SQL vNext

Keep 110/120 CompatLevel

Freeze plans (optional)

→

Run Query Store (establish perf. baseline)

→

Move to 130 Compat Level and unfreeze plans

→

Monitor perf. and fix regressions with plan forcing

Troubleshooting with Query Store

Enable Query Store

(ALTER DB)

→

Let Query Store collect the data

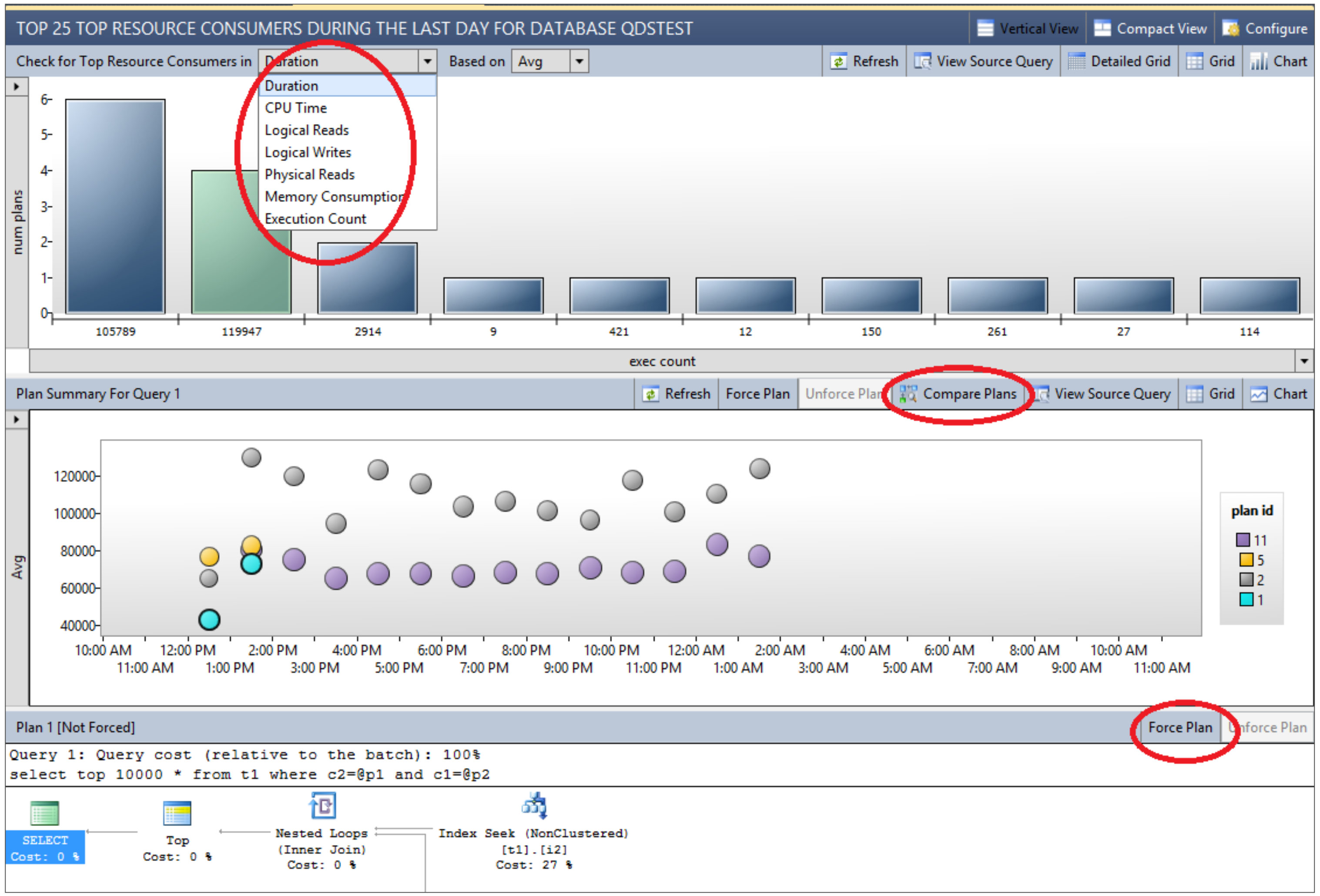
→

Search for "Problem" queries

→

Set FORCE PLAN policies

Monitoring Performance with Query Store



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

- The Query Store feature provides DBAs with insight on query plan choice and performance

Working with Query Store

```sql
/* (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*/
```

```sql
);

/* (4) Clear all Query Store data */
ALTER DATABASE MyDB SET QUERY_STORE CLEAR;

/* (5) Turn OFF Query Store */
ALTER DATABASE MyDB SET QUERY_STORE = OFF;
```

- DB-level feature exposed through T-SQL extensions
- ALTER DATABASE
- Catalog views (settings, compile & runtime stats)
- Stored Procs (plan forcing, query/plan/stats cleanup)

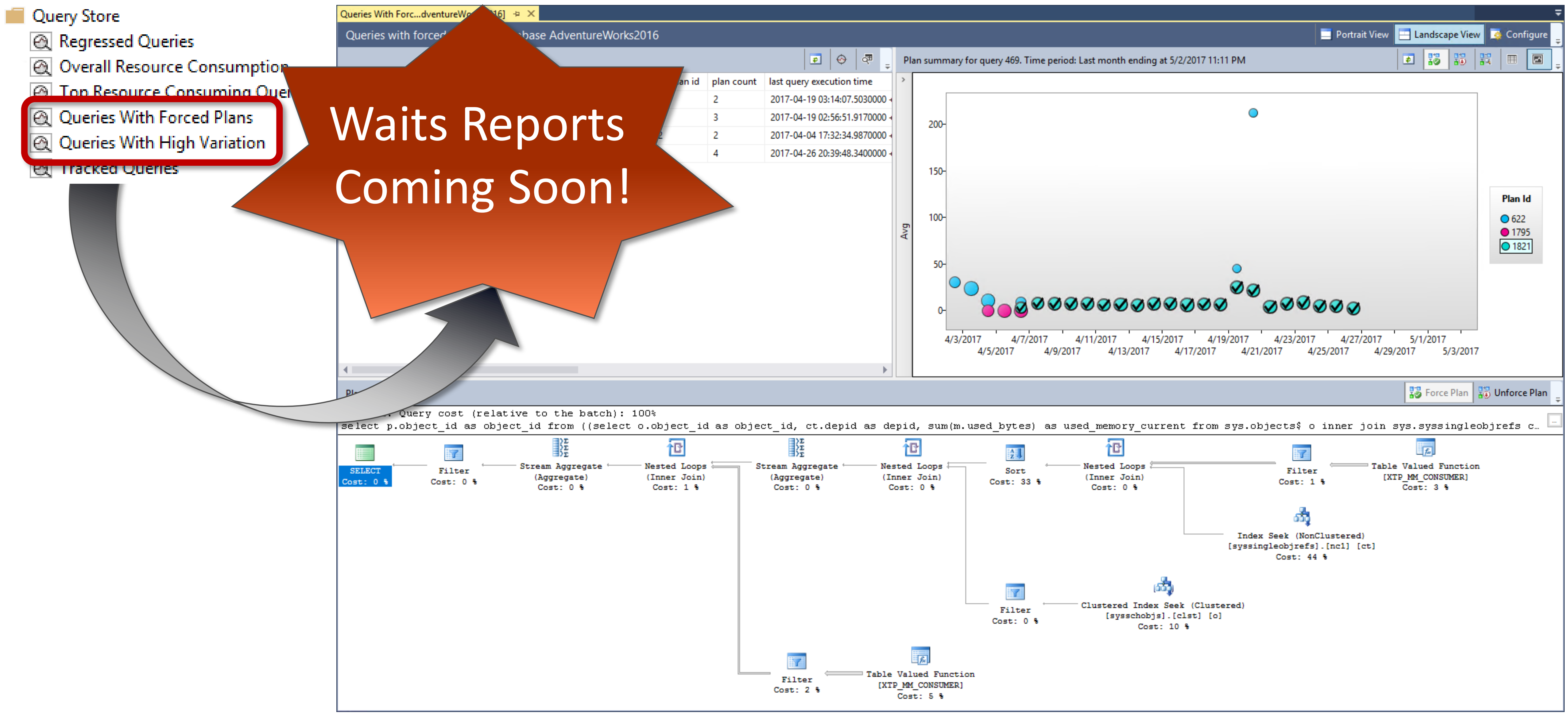
Troubleshooting Query Store

- Plan forcing does not always work
  - Example: If you drop an index, you can't 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:

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

Demonstration:
Using Query Store in SQL Server 2016 / 2017

Queries with Forced Plans in SQL Server 2017


Query Store
Regressed Queries
Overall Resource Consumption
Top Resource Consuming Quer
Queries With Forced Plans
Queries With High Variation
Tracked Queries
Waits Reports Coming Soon!
Queries With Forc...dventureWo 16]
Queries with forced base AdventureWorks2016
Portrait View
Landscape View
Configure
Plan summary for query 469. Time period: Last month ending at 5/2/2017 11:11 PM
Avg
Plan Id
● 622
● 1795
● 1821
Force Plan
Unforce Plan
SELECT Cost: 0 %
Filter Cost: 0 %
Stream Aggregate (Aggregate) Cost: 0 %
Nested Loops (Inner Join) Cost: 1 %
Stream Aggregate (Aggregate) Cost: 0 %
Nested Loops (Inner Join) Cost: 0 %
Sort Cost: 33 %
Nested Loops (Inner Join) Cost: 0 %
Filter Cost: 1 %
Table Valued Function [XTP_MM_CONSUMER] Cost: 3 %
Index Seek (NonClustered) [syssingleobjrefs].[nc1] [ct] Cost: 44 %
Filter Cost: 0 %
Clustered Index Seek (Clustered) [sysschobjs].[clst] [o] Cost: 10 %
Filter Cost: 2 %
Table Valued Function [XTP_MM_CONSUMER] Cost: 5 %

an id	plan count	last query execution time
	2	2017-04-19 03:14:07.5030000
	3	2017-04-19 02:56:51.9170000
	2	2017-04-04 17:32:34.9870000
	4	2017-04-26 20:39:48.3400000

```
Query cost (relative to the batch): 100%
select p.object_id as object_id from ((select o.object_id as object_id, ct.depid as depid, sum(m.used_bytes) as used_memory_current from sys.objects$ o inner join sys.syssingleobjrefs c…
```

Demonstration: Query Store in SQL Server 2017

- Query Store Waits
- Queries with Forced Plans
- Queries with High Variations

SQL Server 2017 – Modern and Intelligent

Query Store – Wait Stats and "Cloud Learnings"

Automatic Tuning and Plan Correction

Query Plan Analysis in SSMS

Adaptive Query Processor

SQL Server 2017 – Query Store Improvements

- New Query Store Reports
- Automatic Tuning Feature Support

```
ALTER DATABASE AdventureWorks2017
    SET AUTOMATIC_TUNING ( FORCE_LAST_GOOD_PLAN = ON );
```

- DBCC CLONEDATABASE flushes statistics while cloning to avoid missing query store runtime statistics
- New DMVs
  - sys.query_store_wait_stats
  - sys.dm_db_tuning_recommendations
  - sys.database_automatic_tuning_mode
  - sys.database_automatic_tuning_options

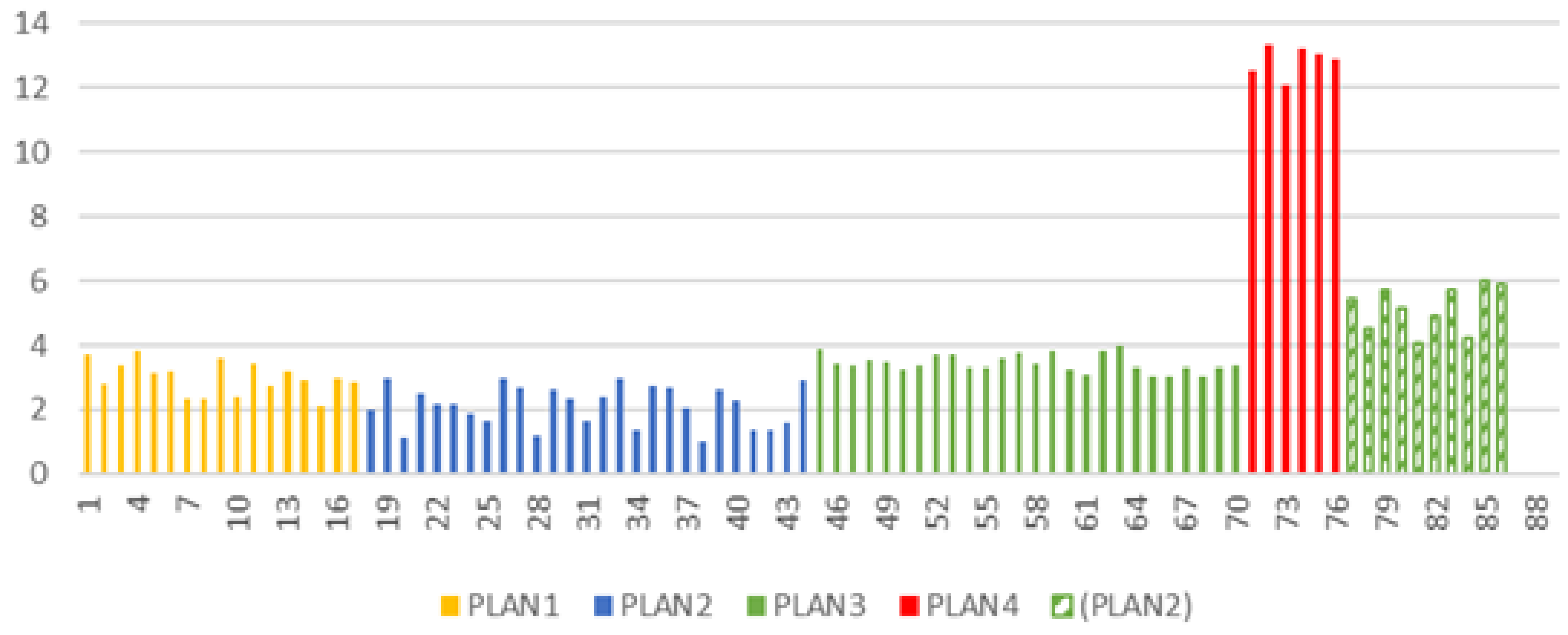
SQL Server 2017 Automatic Tuning

```
ALTER DATABASE CURRENT
SET AUTOMATIC_TUNING (FORCE_LAST_GOOD_PLAN = ON);
```

Detect with **dm_db_tuning_ recommendations** and force manually

Turn on Auto and system corrects

Reverts back to
“Last Known Good”



Perfect to help with parameter sniffing

Demonstration:
Automatic Tuning in SQL Server 2017

- Query Store Waits
- Queries with Forced Plans
- Queries with High Variations

Adaptive Query Processing

A Family of Corrective Processing…

Risks of Misestimation

Slow Query Response Time Due to Bad Plans

Excessive Resource Utilization (CPU, Memory, IO)

Reduced Throughput and Concurrency

T-SQL Refactoring for Off-Model Statements

Cardinality Estimation and Plan Quality

```
SELECT [fo].[Order Key], [fo].[Description], [fo].[Package], [fo].[Quantity],
   [foo].[OutlierEventQuantity]
FROM   [Fact].[OrderHistory] AS [fo]
INNER JOIN [Fact].[WhatIfOutEventQuantity]('Mild Recession', '1-01-2013', '10-15-2014’)
             AS [foo] ON [fo].[Order Key] = [foo].[Order Key]
               AND [fo].[City Key] = [foo].[City Key]
               AND [fo].[Customer Key] = [foo].[Customer Key]
               AND [fo].[Stock Item Key] = [foo].[Stock Item Key]
               AND [fo].[Order Date Key] = [foo].[Order Date Key]
               AND [fo].[Picked Date Key] = [foo].[Picked Date Key]
               AND [fo].[Salesperson Key] = [foo].[Salesperson Key]
               AND [fo].[Picker Key] = [foo].[Picker Key]
INNER JOIN [Dimension].[Stock Item] AS [si]
      ON [fo].[Stock Item Key] = [si].[Stock Item Key]
WHERE [si].[Lead Time Days] > 0
      AND [fo].[Quantity] > 50;
```

Adaptive Query Processing (SQL 2017)

Interleaved Execution

- Materialize estimates for multi-statement table valued functions (MSTVFs)
- Downstream operations will benefit from the corrected MSTVF cardinality estimate

Batch-mode Memory Grant Feedback

- Adjust memory grants based on execution feedback
- Remove spills and improve concurrency for repeating queries

Batch-mode Adaptive Joins

- Defer the choice of hash join or nested loop until after the first join input has been scanned
- Uses nested loop for small inputs, hash joins for large inputs

Demonstration: Adaptive Query Processing

- Interleaved Execution
- Batch-Mode Memory Grant Feedback
- Batch-Mode Adaptive Join

Microsoft