Making the Most of Query Store in the Real World

Jeff Iannucci

Who in the world is Jeff Iannucci?

StraightPath Solutions

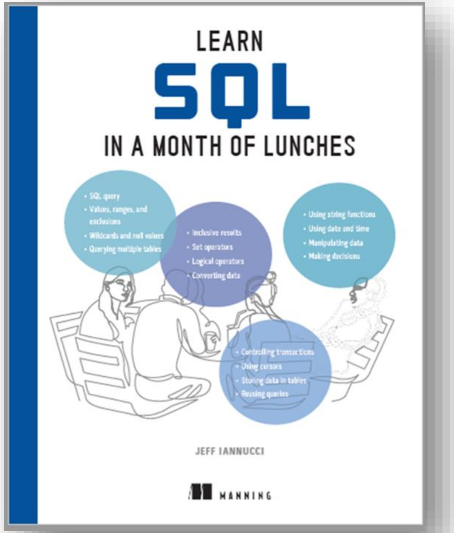
Consultant at Straight Path Solutions

PLURALSIGHT

Content Author at Pluralsight

MANNING

Author of “Learn SQL in a Month of Lunches”



github.com/desertdba

In the “Presentations” folder:

- All the slides!
- One handy T-SQL script!

A word of thanks to these folks

Conor Cunningham

Erik Darling

Grant Fritchey

Tara Kizer

Kendra Little

Brent Ozar

Paul Randal

Erin Stellato

Has this been your experience?

- Query Store made performance worse
- Query Store unexpectedly went READ_ONLY
- Forced query plans didn’t always get used
- Can we solve any REAL problems with this thing?

YES, WE CAN!!!

“Real world” solutions

- Ways to use Query Store beyond forcing plans
- Considerations for defaults and “best practices”
- Queries and free, community-supported tools
- Common problems you can solve

Setting up Query Store

What defaults to change...maybe

- MAX_STORAGE_SIZE_MB
- MAX_PLANS_PER_QUERY
- STALE_QUERY_THRESHOLD_DAYS
- QUERY_CAPTURE_MODE

What defaults to leave alone

- DATA_FLUSH_INTERVAL_SECONDS
- INTERVAL_LENGTH_MINUTES
- SIZE_BASED_CLEANUP_MODE
- WAIT_STATS_CAPTURE_MODE

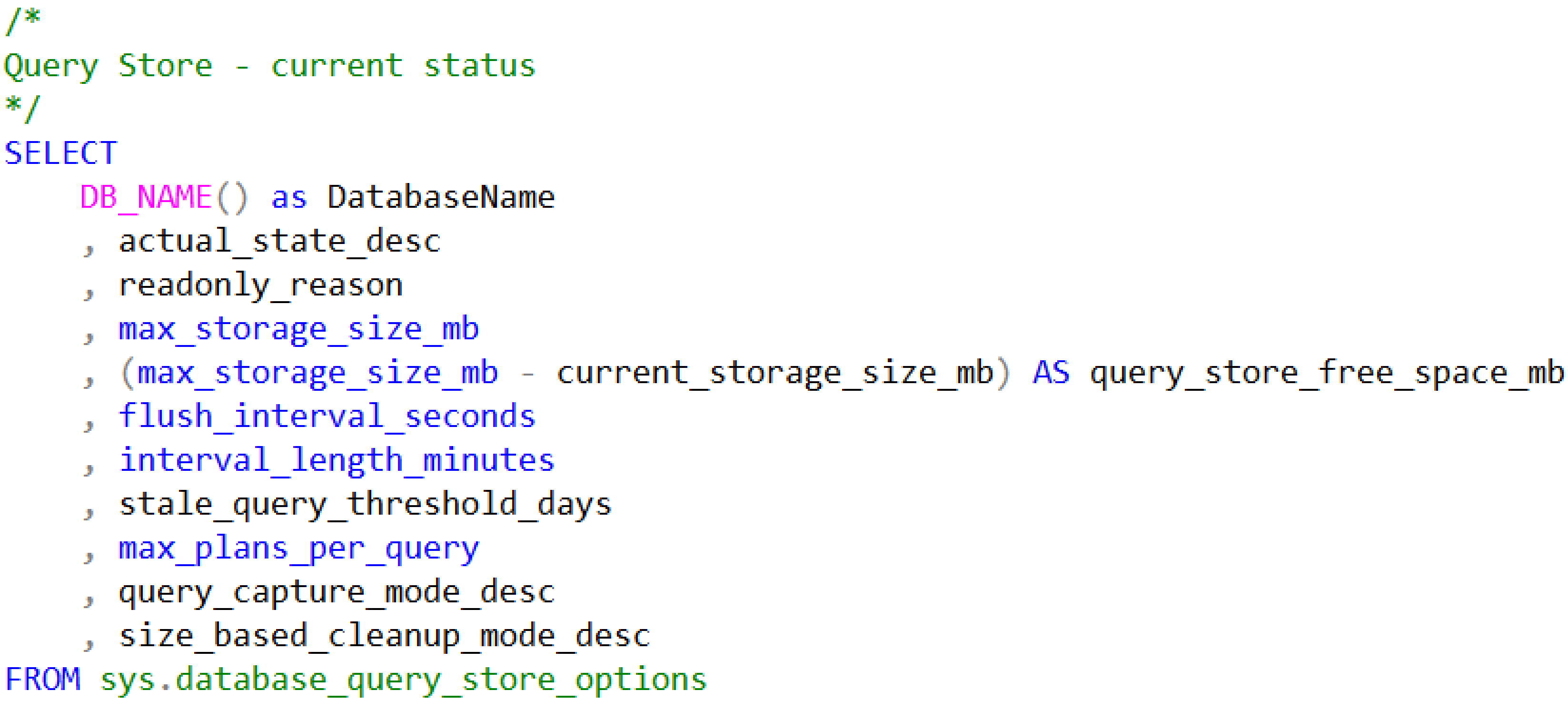
Trace flags

- Query Store uses in-memory tables
- TF 7752 – allows queries to execute while QS loads into memory (SQL Server 2016 & 2017 only)
- TF 7745 – bypasses writing to disk at shutdown, losing unflushed data (all versions)

dba-tools (PowerShell)

- Get-DbaDbQueryStoreOption
- Set-DbaDbQueryStoreOption
- Copy-DbaDbQueryStoreOption
- Test-DbaDbQueryStore

You should get a job (for monitoring)



```sql
/*
Query Store - current status
*/
SELECT
    DB_NAME() as DatabaseName
    , actual_state_desc
    , readonly_reason
    , max_storage_size_mb
    , (max_storage_size_mb - current_storage_size_mb) AS query_store_free_space_mb
    , flush_interval_seconds
    , interval_length_minutes
    , stale_query_threshold_days
    , max_plans_per_query
    , query_capture_mode_desc
    , size_based_cleanup_mode_desc
FROM sys.database_query_store_options
```

Tools you can use

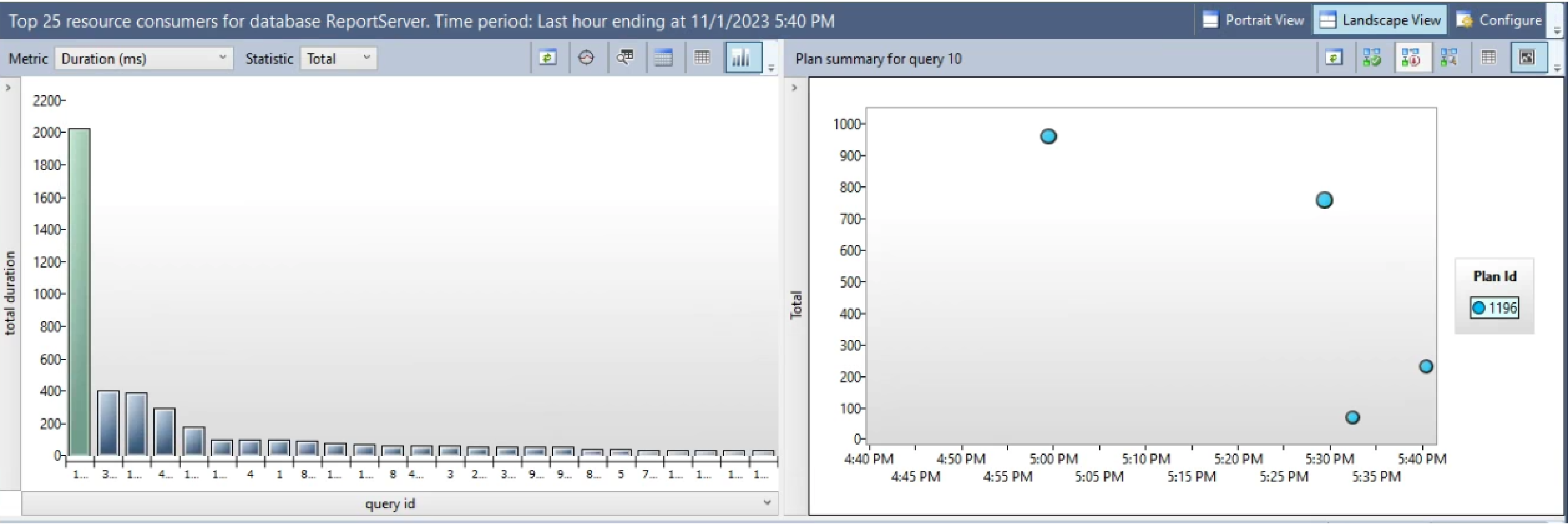
SSMS built-in reports

- Top Resource Consuming Queries
- Queries With Forced Plans
- Tracked Queries

SSMS built-in reports

- Top Resource Consuming Queries
- Queries With Forced Plans
- Tracked Queries

Top Resource Consuming Queries



Top 25 resource consumers for database ReportServer. Time period: Last hour ending at 11/1/2023 5:40 PM

Metric: Duration (ms)

Statistic: Total

Plan summary for query 10

Defaults are…

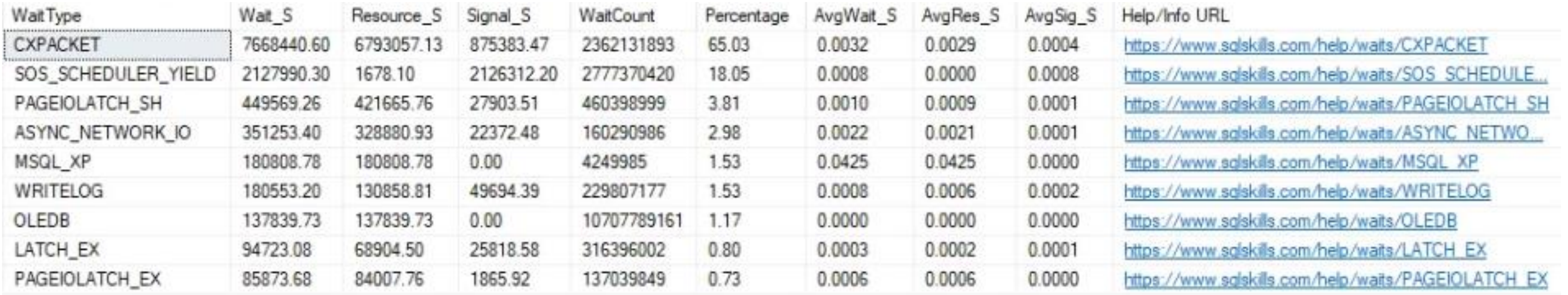
- Top 25
- Last hour
- Total and Duration (2 separate defaults)
- Cute and colorful circles (squares and triangles too)

What is the real problem?

- CPU usage
- Memory consumption
- I/O (reads and/or writes)
- Total, Average, Maximum

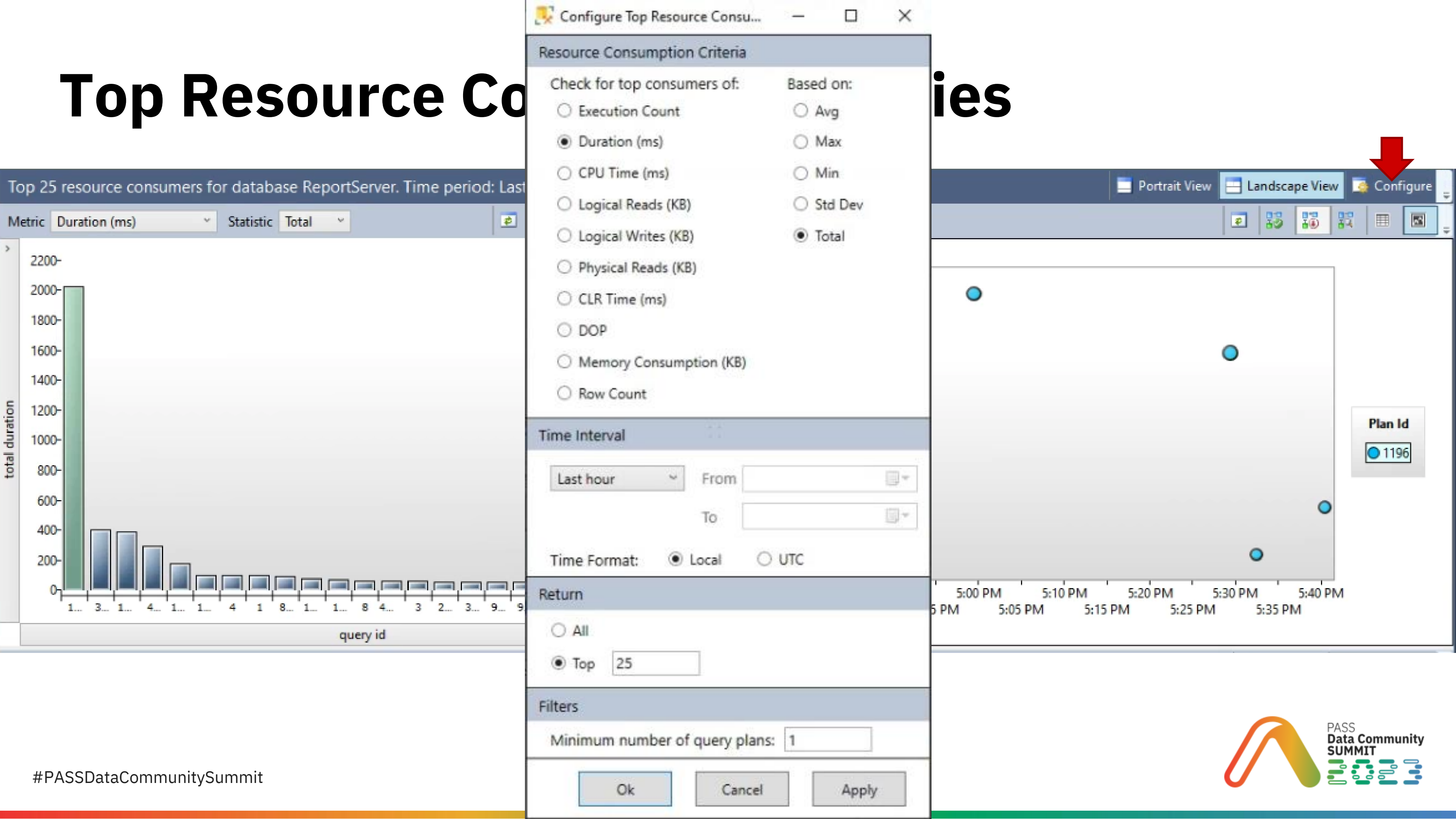
Paul Randal’s Wait Stats

- https://www.sqlskills.com/blogs/paul/wait-statistics-or-please-tell-me-where-it-hurts/



WaitType	Wait_S	Resource_S	Signal_S	WaitCount	Percentage	AvgWait_S	AvgRes_S	AvgSig_S	Help/Info URL
CXPACKET	7668440.60	6793057.13	875383.47	2362131893	65.03	0.0032	0.0029	0.0004	https://www.sqlskills.com/help/waits/CXPACKET
SOS_SCHEDULER_YIELD	2127990.30	1678.10	2126312.20	2777370420	18.05	0.0008	0.0000	0.0008	https://www.sqlskills.com/help/waits/SOS_SCHEDULE...
PAGEIOLATCH_SH	449569.26	421665.76	27903.51	460398999	3.81	0.0010	0.0009	0.0001	https://www.sqlskills.com/help/waits/PAGEIOLATCH_SH
ASYNC_NETWORK_IO	351253.40	328880.93	22372.48	160290986	2.98	0.0022	0.0021	0.0001	https://www.sqlskills.com/help/waits/ASYNC_NETWO...
MSQL_XP	180808.78	180808.78	0.00	4249985	1.53	0.0425	0.0425	0.0000	https://www.sqlskills.com/help/waits/MSQL_XP
WRITELOG	180553.20	130858.81	49694.39	229807177	1.53	0.0008	0.0006	0.0002	https://www.sqlskills.com/help/waits/WRITELOG
OLEDB	137839.73	137839.73	0.00	10707789161	1.17	0.0000	0.0000	0.0000	https://www.sqlskills.com/help/waits/OLEDB
LATCH_EX	94723.08	68904.50	25818.58	316396002	0.80	0.0003	0.0002	0.0001	https://www.sqlskills.com/help/waits/LATCH_EX
PAGEIOLATCH_EX	85873.68	84007.76	1865.92	137039849	0.73	0.0006	0.0006	0.0000	https://www.sqlskills.com/help/waits/PAGEIOLATCH_EX

Top Resource Co ies



Configure Top Resource Consu...

Resource Consumption Criteria

Check for top consumers of:

○ Execution Count
● Duration (ms)
○ CPU Time (ms)
○ Logical Reads (KB)
○ Logical Writes (KB)
○ Physical Reads (KB)
○ CLR Time (ms)
○ DOP
○ Memory Consumption (KB)
○ Row Count

Based on:

○ Avg
○ Max
○ Min
○ Std Dev
● Total

Time Interval

Last hour

From:

To:

Time Format: ● Local ○ UTC

Return

○ All
● Top 25

Filters

Minimum number of query plans: 1

Top 25 resource consumers for database ReportServer. Time period: Last

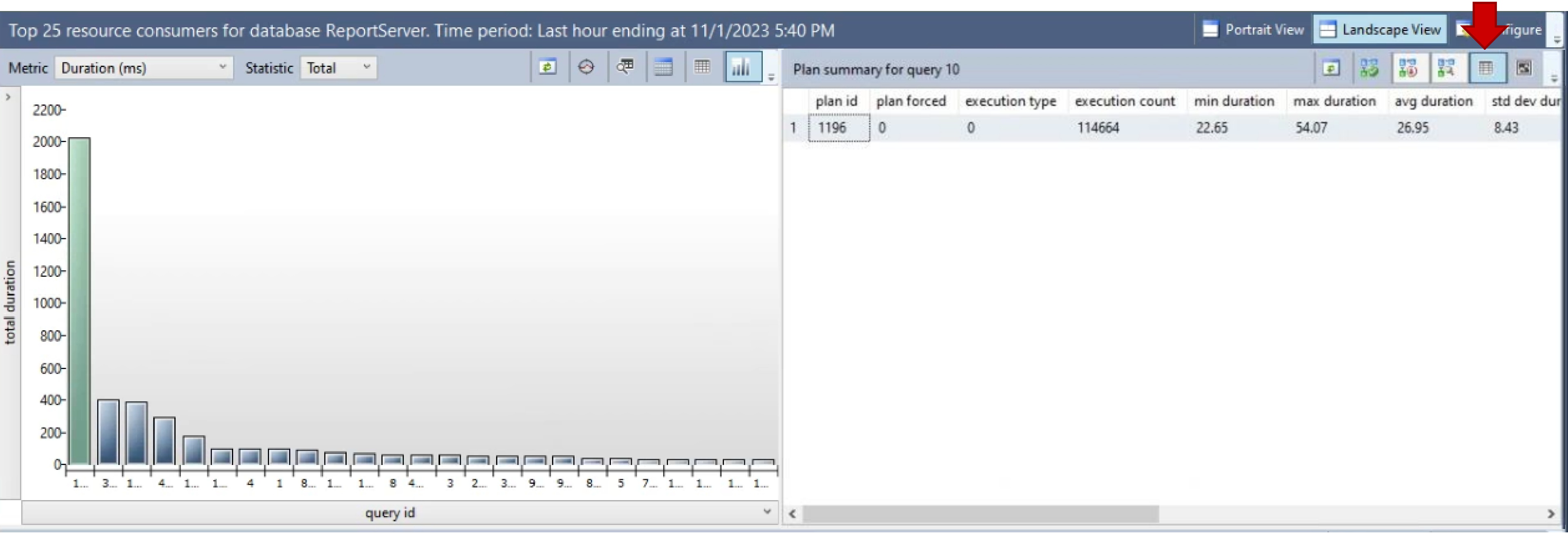
Metric: Duration (ms)

Statistic: Total

#PASSDataCommunitySummit

PASS Data Community SUMMIT 2023

Top Resource Consuming Queries



Top 25 resource consumers for database ReportServer. Time period: Last hour ending at 11/1/2023 5:40 PM

Metric: Duration (ms)

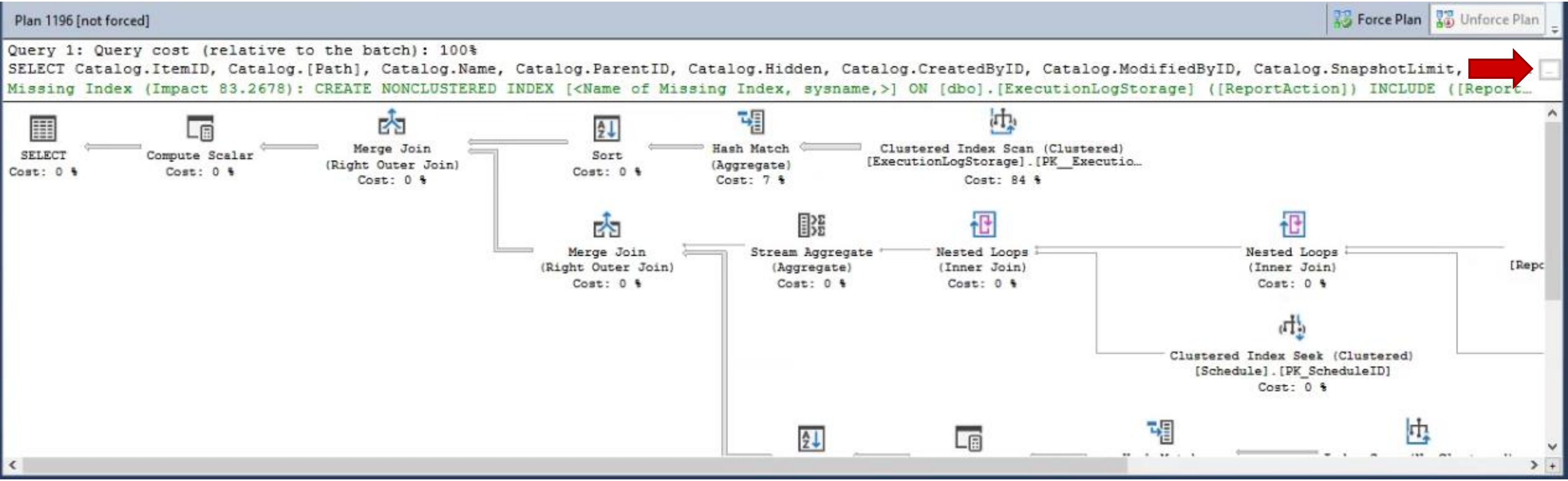
Statistic: Total

Plan summary for query 10

	plan id	plan forced	execution type	execution count	min duration	max duration	avg duration	std dev dur
1	1196	0	0	114664	22.65	54.07	26.95	8.43

#PASSDataCommunitySummit

Top Resource Consuming Queries



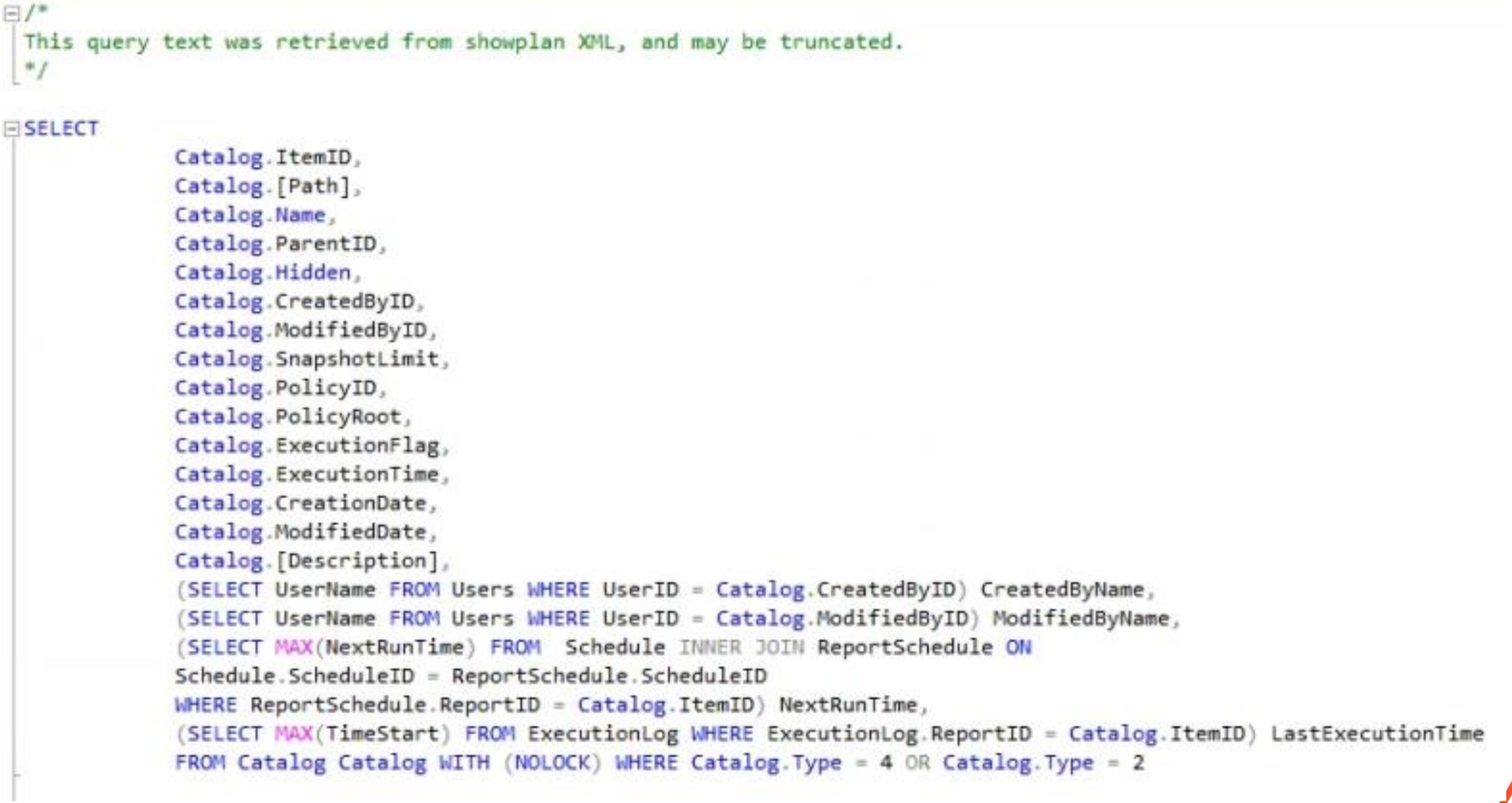
Plan 1196 [not forced]

```
Query 1: Query cost (relative to the batch): 100%
SELECT Catalog.ItemID, Catalog.[Path], Catalog.Name, Catalog.ParentID, Catalog.Hidden, Catalog.CreatedByID, Catalog.ModifiedByID, Catalog.SnapshotLimit,
Missing Index (Impact 83.2678): CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname,>] ON [dbo].[ExecutionLogStorage] ([ReportAction]) INCLUDE ([Report…
```

#PASSDataCommunitySummit

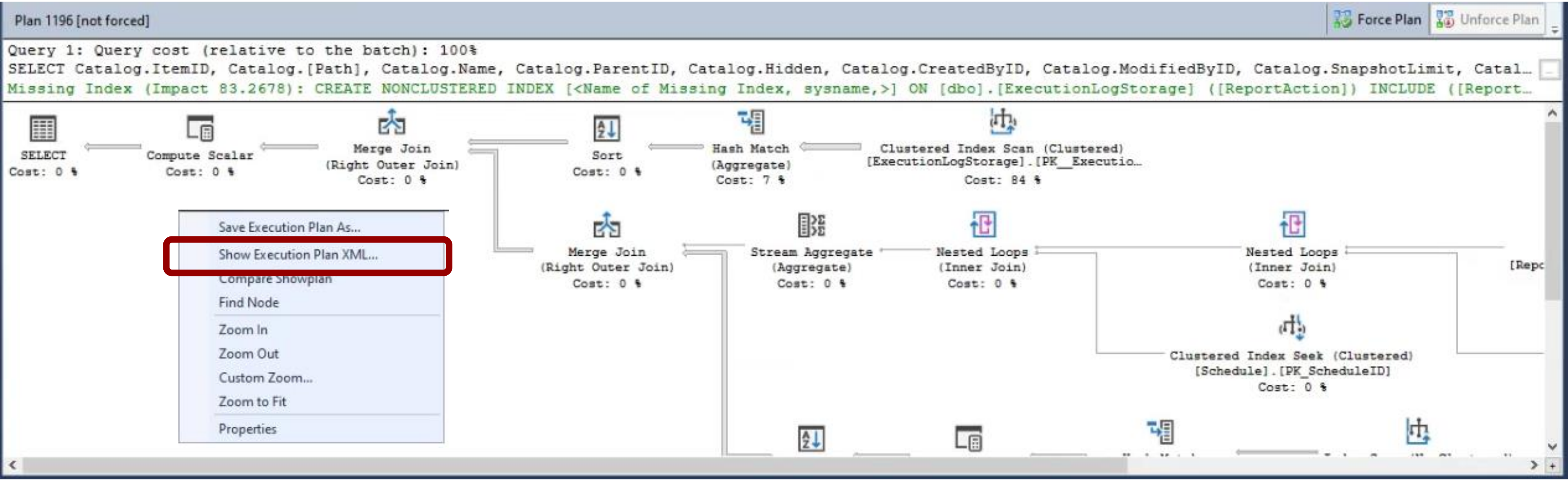
PASS Data Community SUMMIT 2023

Top Resource Consuming Queries



```
/*
This query text was retrieved from showplan XML, and may be truncated.
*/

SELECT
            Catalog.ItemID,
            Catalog.[Path],
            Catalog.Name,
            Catalog.ParentID,
            Catalog.Hidden,
            Catalog.CreatedByID,
            Catalog.ModifiedByID,
            Catalog.SnapshotLimit,
            Catalog.PolicyID,
            Catalog.PolicyRoot,
            Catalog.ExecutionFlag,
            Catalog.ExecutionTime,
            Catalog.CreationDate,
            Catalog.ModifiedDate,
            Catalog.[Description],
            (SELECT UserName FROM Users WHERE UserID = Catalog.CreatedByID) CreatedByName,
            (SELECT UserName FROM Users WHERE UserID = Catalog.ModifiedByID) ModifiedByName,
            (SELECT MAX(NextRunTime) FROM  Schedule INNER JOIN ReportSchedule ON
            Schedule.ScheduleID = ReportSchedule.ScheduleID
            WHERE ReportSchedule.ReportID = Catalog.ItemID) NextRunTime,
            (SELECT MAX(TimeStart) FROM ExecutionLog WHERE ExecutionLog.ReportID = Catalog.ItemID) LastExecutionTime
            FROM Catalog Catalog WITH (NOLOCK) WHERE Catalog.Type = 4 OR Catalog.Type = 2
```

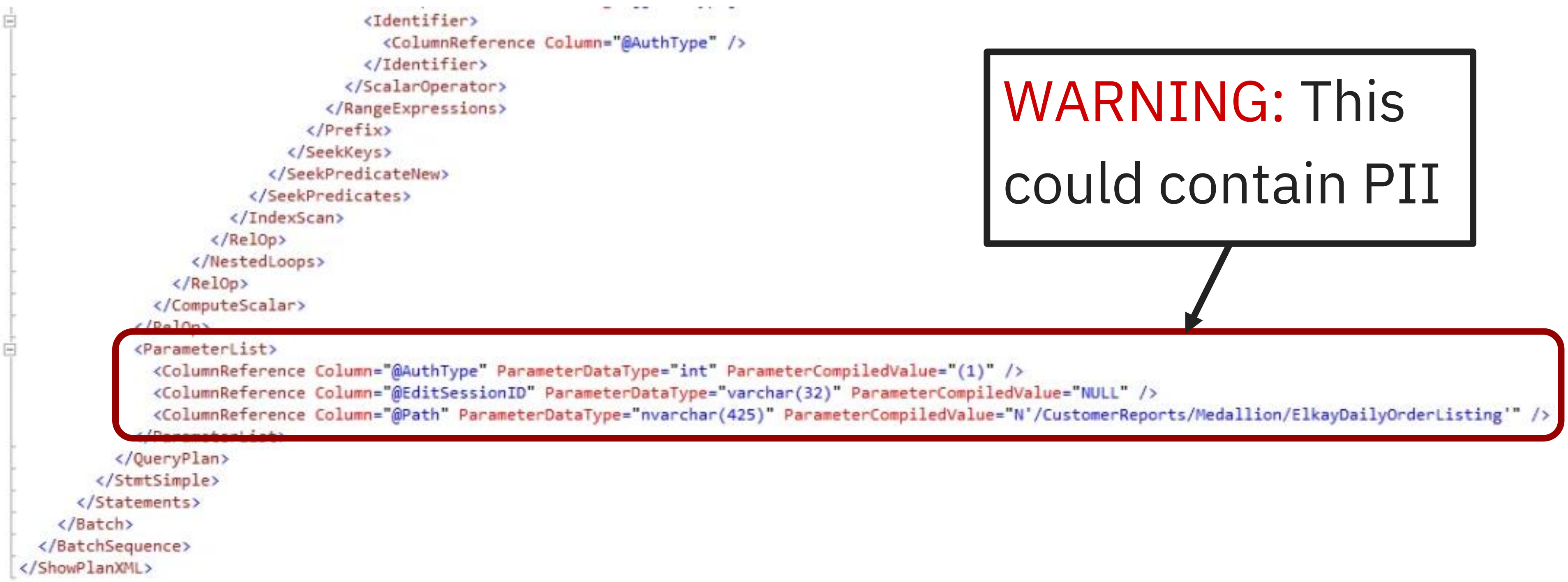
Top Resource Consuming Queries



```
Query 1: Query cost (relative to the batch): 100%
SELECT Catalog.ItemID, Catalog.[Path], Catalog.Name, Catalog.ParentID, Catalog.Hidden, Catalog.CreatedByID, Catalog.ModifiedByID, Catalog.SnapshotLimit, Catal…
Missing Index (Impact 83.2678): CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname,>] ON [dbo].[ExecutionLogStorage] ([ReportAction]) INCLUDE ([Report…
```

#PASSDataCommunitySummit

Top Resource Consuming Queries



```
                                    <Identifier>
                                      <ColumnReference Column="@AuthType" />
                                    </Identifier>
                                  </ScalarOperator>
                                </RangeExpressions>
                              </Prefix>
                            </SeekKeys>
                          </SeekPredicateNew>
                        </SeekPredicates>
                      </IndexScan>
                    </RelOp>
                  </NestedLoops>
                </RelOp>
              </ComputeScalar>
            </RelOp>
            <ParameterList>
              <ColumnReference Column="@AuthType" ParameterDataType="int" ParameterCompiledValue="(1)" />
              <ColumnReference Column="@EditSessionID" ParameterDataType="varchar(32)" ParameterCompiledValue="NULL" />
              <ColumnReference Column="@Path" ParameterDataType="nvarchar(425)" ParameterCompiledValue="N'/CustomerReports/Medallion/ElkayDailyOrderListing'" />
            </ParameterList>
          </QueryPlan>
        </StmtSimple>
      </Statements>
    </Batch>
  </BatchSequence>
</ShowPlanXML>
```

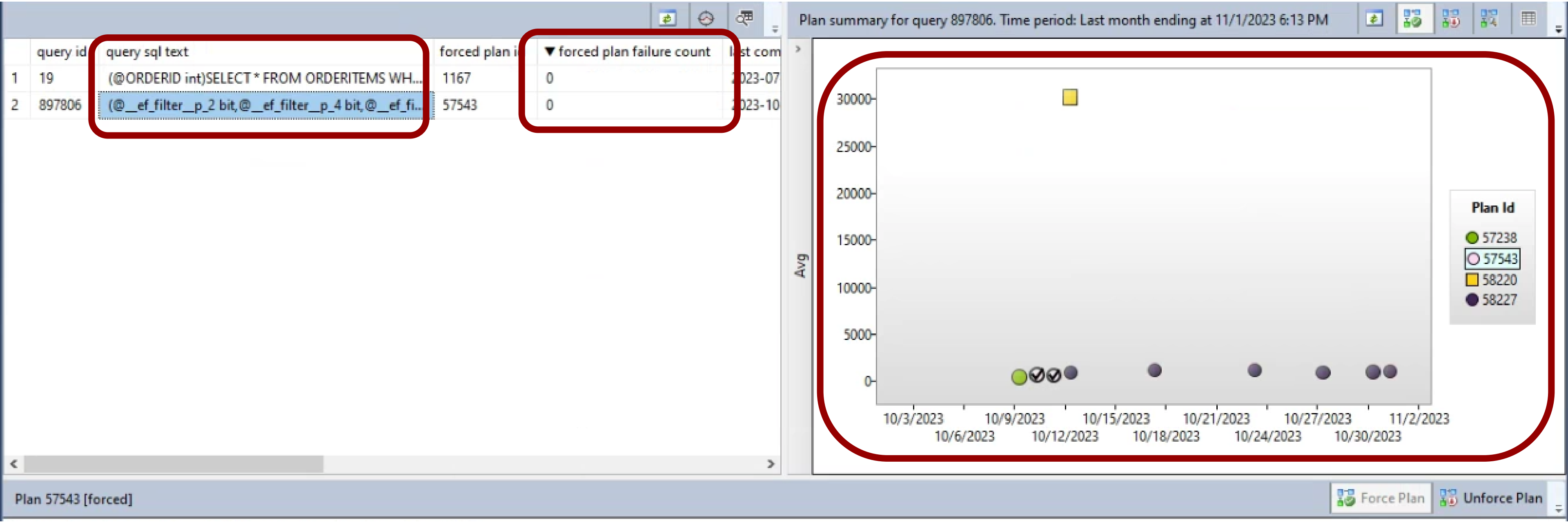
WARNING: This could contain PII

SSMS built-in reports

- Top Resource Consuming Queries
- Queries With Forced Plans
- Tracked Queries

Forced plans are rarely a long-term solution

Queries With Forced Plans

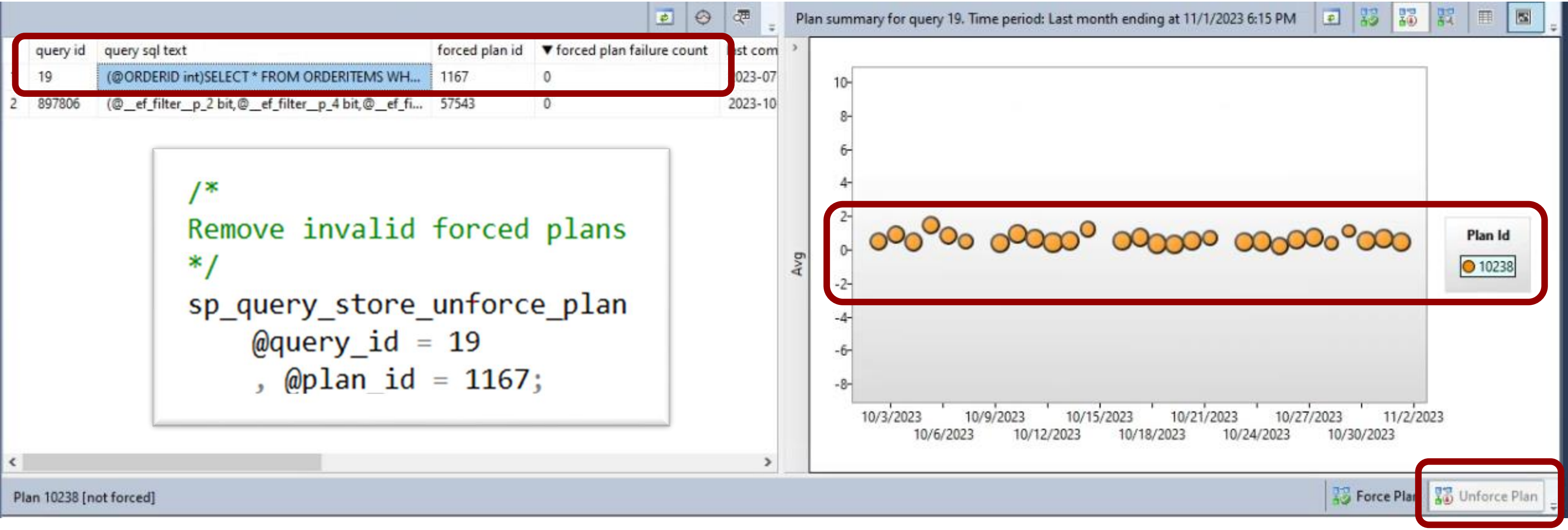


Plan summary for query 897806. Time period: Last month ending at 11/1/2023 6:13 PM

	query id	query sql text	forced plan i	▼ forced plan failure count	last com
1	19	(@ORDERID int)SELECT * FROM ORDERITEMS WH...	1167	0	2023-07
2	897806	(@__ef_filter__p_2 bit,@__ef_filter__p_4 bit,@__ef_fi...	57543	0	2023-10

Plan 57543 [forced]

Queries With Forced Plans



	query id	query sql text	forced plan id	▼ forced plan failure count	...st com
1	19	(@ORDERID int)SELECT * FROM ORDERITEMS WH...	1167	0	...023-07
2	897806	(@__ef_filter__p_2 bit,@__ef_filter__p_4 bit,@__ef_fi...	57543	0	2023-10

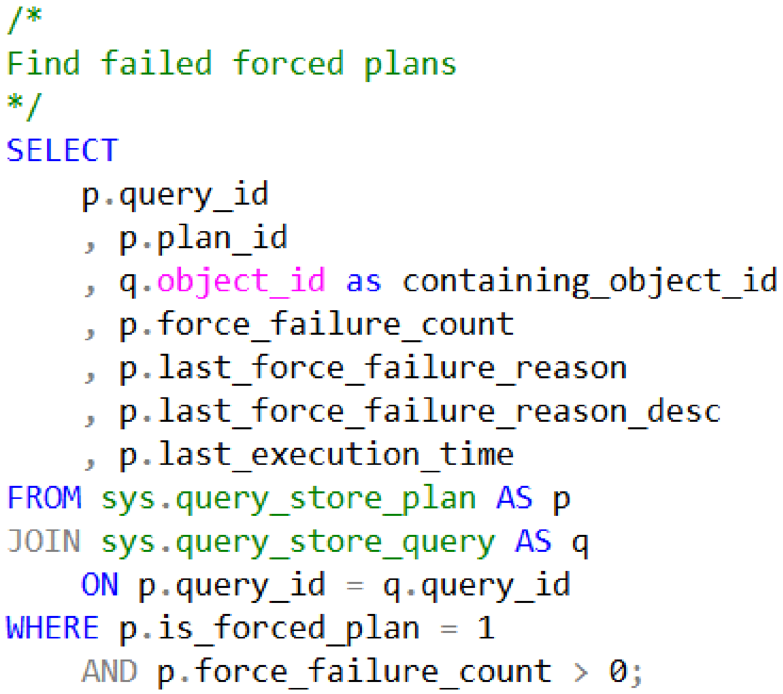
```
/*
Remove invalid forced plans
*/
sp_query_store_unforce_plan
    @query_id = 19
  , @plan_id = 1167;
```

Plan summary for query 19. Time period: Last month ending at 11/1/2023 6:15 PM

Plan 10238 [not forced]

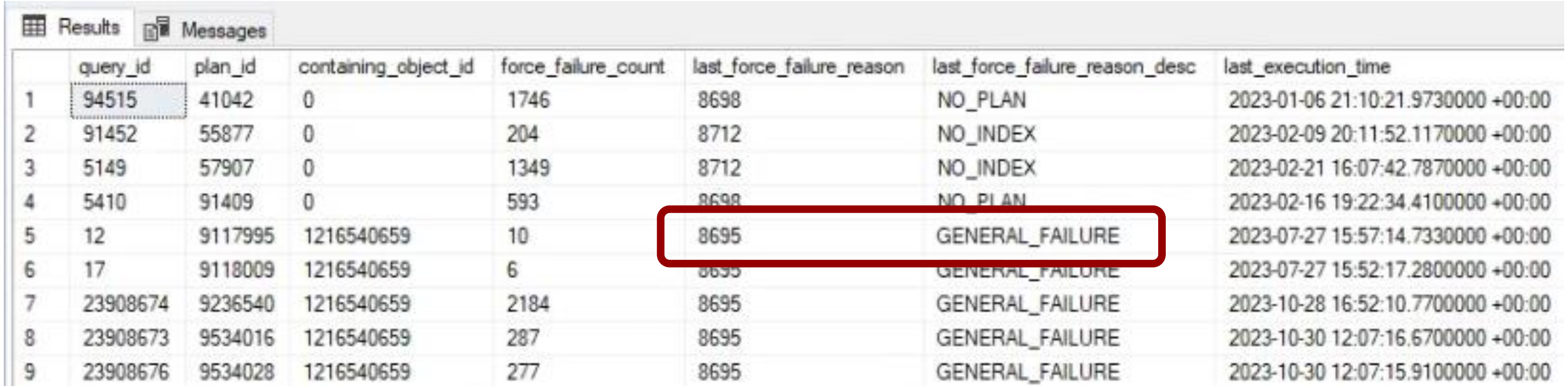
#PASSDataCommunitySummit

Failed forced plans



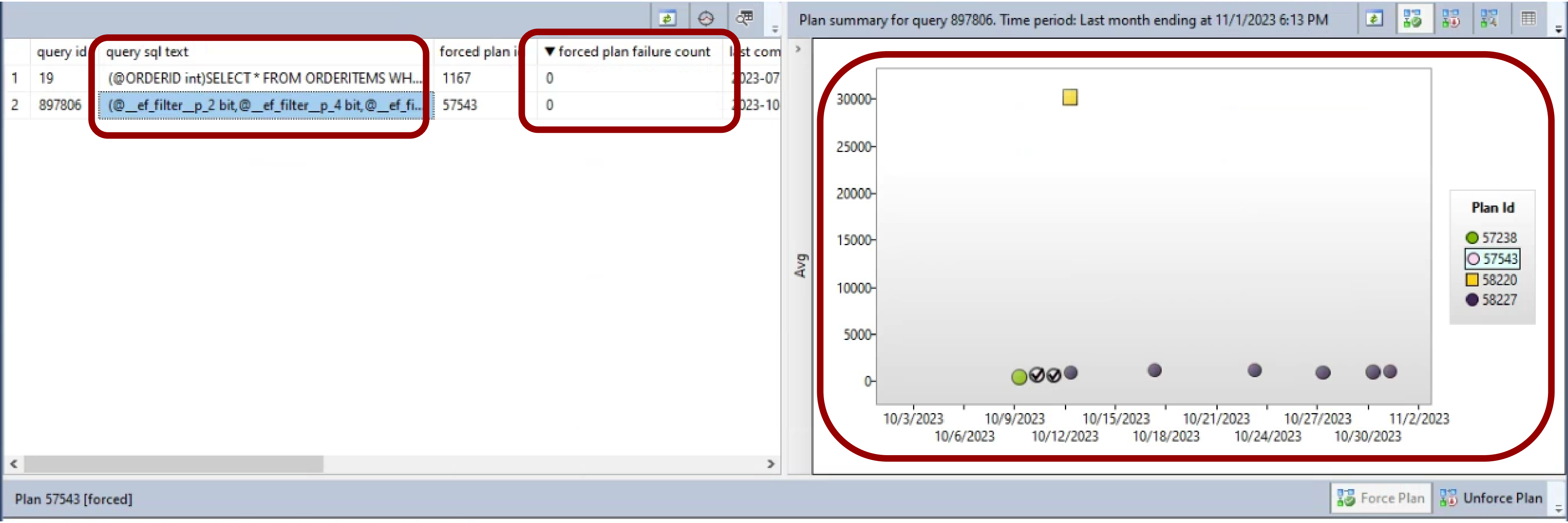
```
/*
Find failed forced plans
*/
SELECT
    p.query_id
    , p.plan_id
    , q.object_id as containing_object_id
    , p.force_failure_count
    , p.last_force_failure_reason
    , p.last_force_failure_reason_desc
    , p.last_execution_time
FROM sys.query_store_plan AS p
JOIN sys.query_store_query AS q
    ON p.query_id = q.query_id
WHERE p.is_forced_plan = 1
    AND p.force_failure_count > 0;
```

Failed forced plans



	query_id	plan_id	containing_object_id	force_failure_count	last_force_failure_reason	last_force_failure_reason_desc	last_execution_time
1	94515	41042	0	1746	8698	NO_PLAN	2023-01-06 21:10:21.9730000 +00:00
2	91452	55877	0	204	8712	NO_INDEX	2023-02-09 20:11:52.1170000 +00:00
3	5149	57907	0	1349	8712	NO_INDEX	2023-02-21 16:07:42.7870000 +00:00
4	5410	91409	0	593	8698	NO_PLAN	2023-02-16 19:22:34.4100000 +00:00
5	12	9117995	1216540659	10	8695	GENERAL_FAILURE	2023-07-27 15:57:14.7330000 +00:00
6	17	9118009	1216540659	6	8695	GENERAL_FAILURE	2023-07-27 15:52:17.2800000 +00:00
7	23908674	9236540	1216540659	2184	8695	GENERAL_FAILURE	2023-10-28 16:52:10.7700000 +00:00
8	23908673	9534016	1216540659	287	8695	GENERAL_FAILURE	2023-10-30 12:07:16.6700000 +00:00
9	23908676	9534028	1216540659	277	8695	GENERAL_FAILURE	2023-10-30 12:07:15.9100000 +00:00

But…what’s the deal here?



	query id	query sql text	forced plan i	▼ forced plan failure count	l st com
1	19	(@ORDERID int)SELECT * FROM ORDERITEMS WH...	1167	0	2023-07
2	897806	(@__ef_filter__p_2 bit,@__ef_filter__p_4 bit,@__ef_fi...	57543	0	2023-10

Plan summary for query 897806. Time period: Last month ending at 11/1/2023 6:13 PM

Plan 57543 [forced]

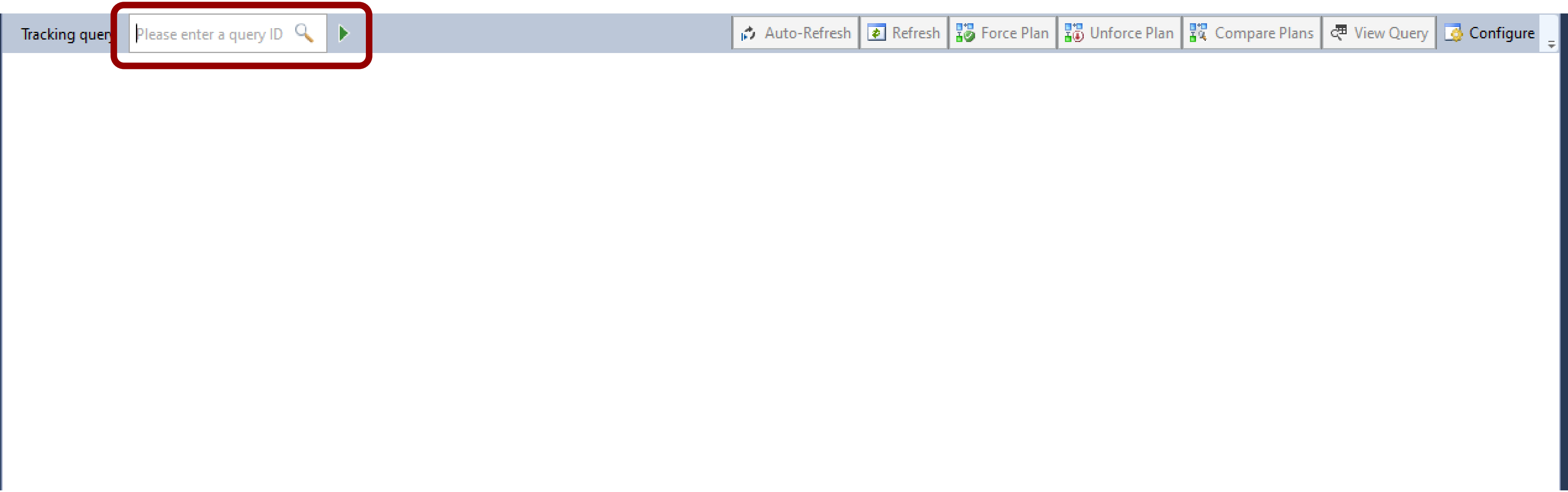
“Morally equivalent plans”

- Query Store has some flexibility
- Not a bug, but a feature
- Don’t show as failures
- Forced plans are rarely a long-term solution

SSMS built-in reports

- Top Resource Consuming Queries
- Queries With Forced Plans
- Tracked Queries

Tracked Queries



Tracked Queries

- Shows performance of one specific query_id

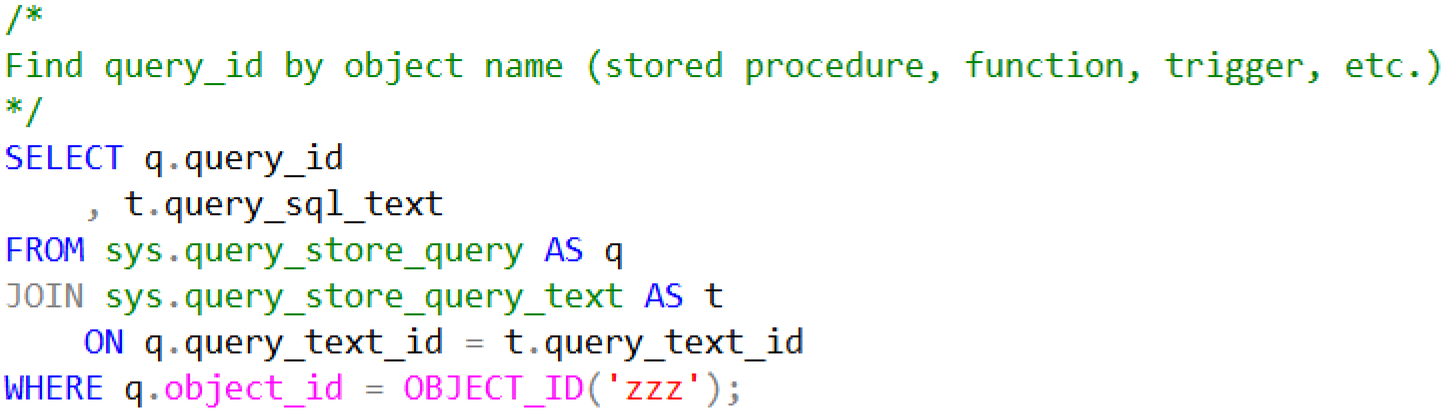
- Can compare plans

- Use “Auto-Update” to track in real time

Find queries (query_id) by...

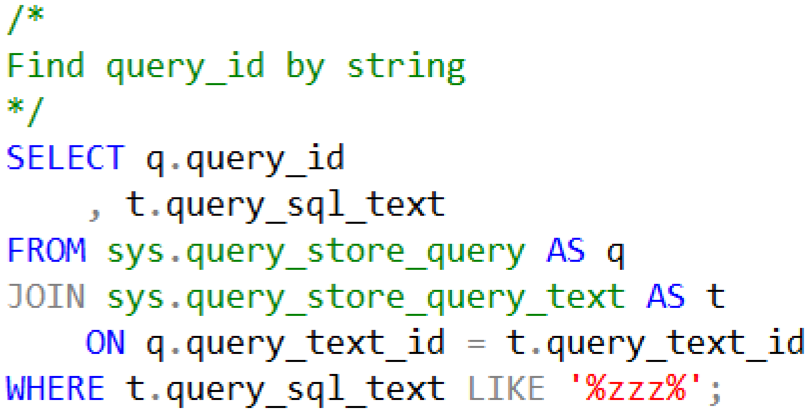
- Object name (stored procedure/function/trigger)
- String of characters
- Time of execution

...by object name



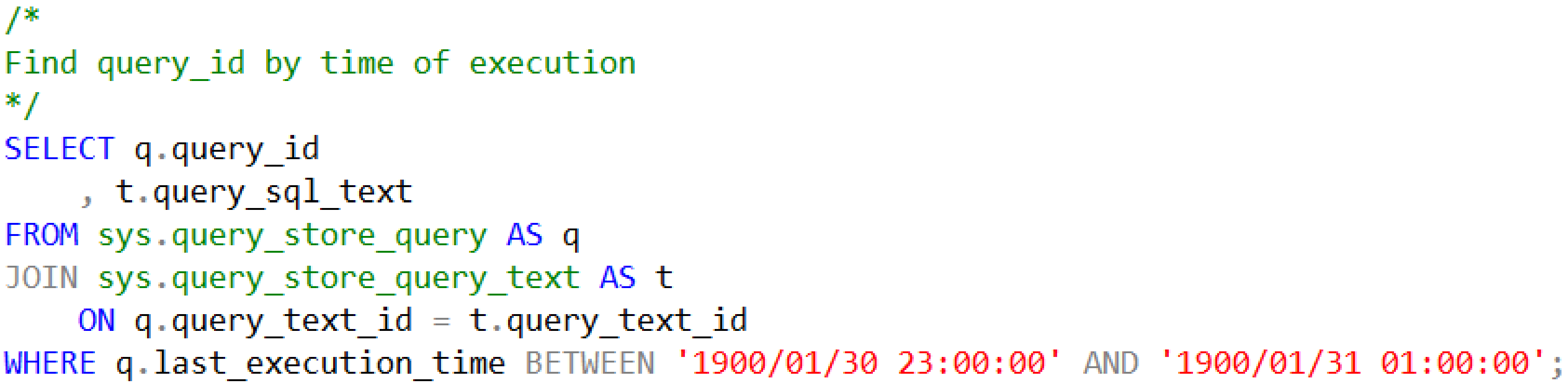
```
/*
Find query_id by object name (stored procedure, function, trigger, etc.)
*/
SELECT q.query_id
     , t.query_sql_text
FROM sys.query_store_query AS q
JOIN sys.query_store_query_text AS t
    ON q.query_text_id = t.query_text_id
WHERE q.object_id = OBJECT_ID('zzz');
```

...by string



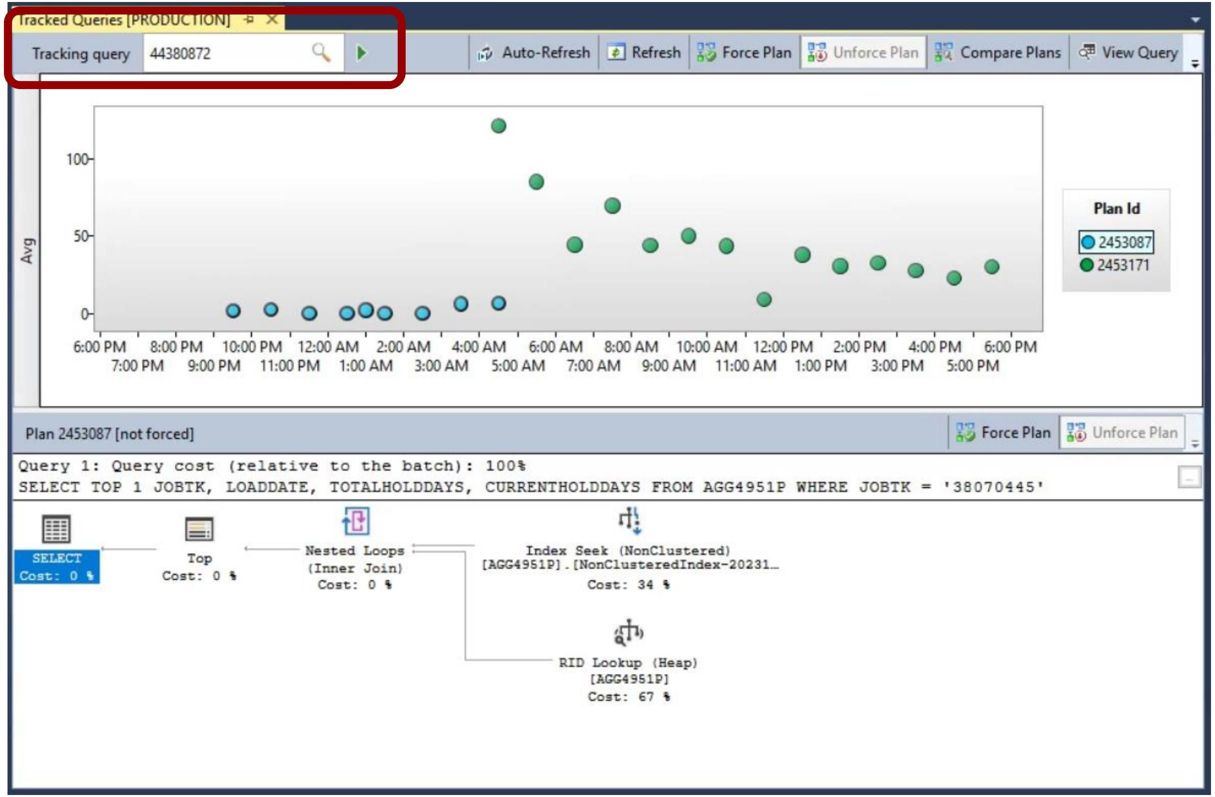
```
/*
Find query_id by string
*/
SELECT q.query_id
    , t.query_sql_text
FROM sys.query_store_query AS q
JOIN sys.query_store_query_text AS t
    ON q.query_text_id = t.query_text_id
WHERE t.query_sql_text LIKE '%zzz%';
```

...by time of execution



```
/*
Find query_id by time of execution
*/
SELECT q.query_id
    , t.query_sql_text
FROM sys.query_store_query AS q
JOIN sys.query_store_query_text AS t
    ON q.query_text_id = t.query_text_id
WHERE q.last_execution_time BETWEEN '1900/01/30 23:00:00' AND '1900/01/31 01:00:00';
```

Tracked Queries



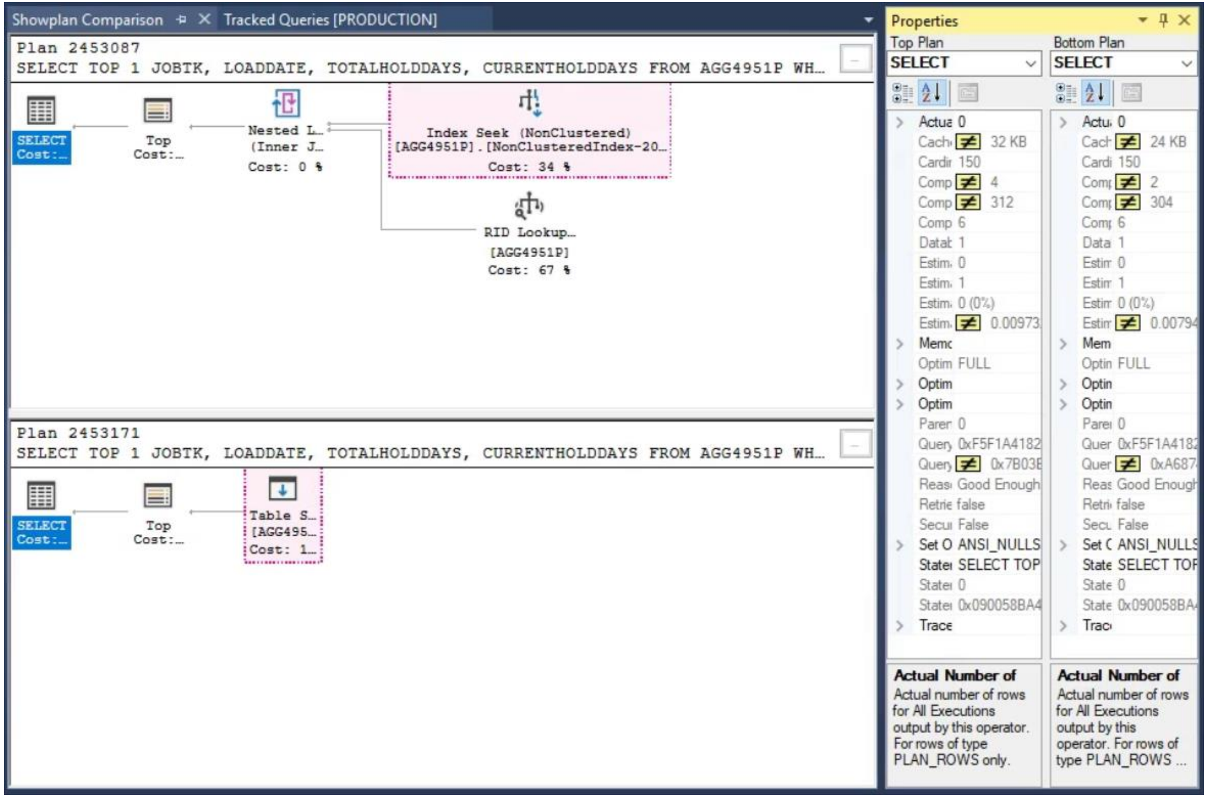
Tracked Queries [PRODUCTION]

Tracking query: 44380872

Plan 2453087 [not forced]

```
Query 1: Query cost (relative to the batch): 100%
SELECT TOP 1 JOBTK, LOADDATE, TOTALHOLDDAYS, CURRENTHOLDDAYS FROM AGG4951P WHERE JOBTK = '38070445'
```

Tracked Queries - Comparison



```
Plan 2453087
SELECT TOP 1 JOBTK, LOADDATE, TOTALHOLDDAYS, CURRENTHOLDDAYS FROM AGG4951P WH…
```

```
Plan 2453171
SELECT TOP 1 JOBTK, LOADDATE, TOTALHOLDDAYS, CURRENTHOLDDAYS FROM AGG4951P WH…
```

Properties

Top Plan		Bottom Plan	
SELECT		SELECT	
Actua	0	Actu	0
Cach	≠ 32 KB	Cach	≠ 24 KB
Cardi	150	Cardi	150
Comp	≠ 4	Com	≠ 2
Comp	≠ 312	Com	≠ 304
Comp	6	Com	6
Datat	1	Data	1
Estim	0	Estir	0
Estim	1	Estir	1
Estim	0 (0%)	Estir	0 (0%)
Estim	≠ 0.00973	Estir	≠ 0.00794
Memc		Mem	
Optim	FULL	Optin	FULL
Optim		Optin	
Optim		Optin	
Parer	0	Pare	0
Query	0xF5F1A4182	Quer	0xF5F1A4182
Query	≠ 0x7B03E	Quer	≠ 0xA687
Reas	Good Enough	Reas	Good Enough
Retrie	false	Retri	false
Secu	False	Secu	False
Set O	ANSI_NULLS	Set C	ANSI_NULLS
State	SELECT TOP	State	SELECT TOP
State	0	State	0
State	0x090058BA4	State	0x090058BA
Trace		Trac	

Actual Number of
Actual number of rows for All Executions output by this operator. For rows of type PLAN_ROWS only.

Actual Number of
Actual number of rows for All Executions output by this operator. For rows of type PLAN_ROWS ...

#PASSDataCommunitySummit

sp_BlitzQueryStore

- Prioritized listing of potential issues
- Grouped by findings (“Worst Avg CPU”, etc..)
- Better if you don’t know where to start
- https://github.com/BrentOzarULTD/SQL-Server-First-Responder-Kit

sp_QuickieStore

- Choose a metric (CPU, reads, writes, memory…)
- Returns query text by default
- Faster if you know what you’re looking for
- https://github.com/erikdarlingdata/DarlingData/tree/main/sp_QuickieStore

Common problems you can solve

Testing index changes

- Missing index requests shown in plans
- Review existing indexes before adding
- Unforce plans if adding/modifying indexes

Reducing upgrade pains

- Upgrade to higher version of SQL Server
- …but don’t raise the Compatibility Level
- If performance is stable, raise the compatibility level
- Use Query Store for analysis and regression fixes

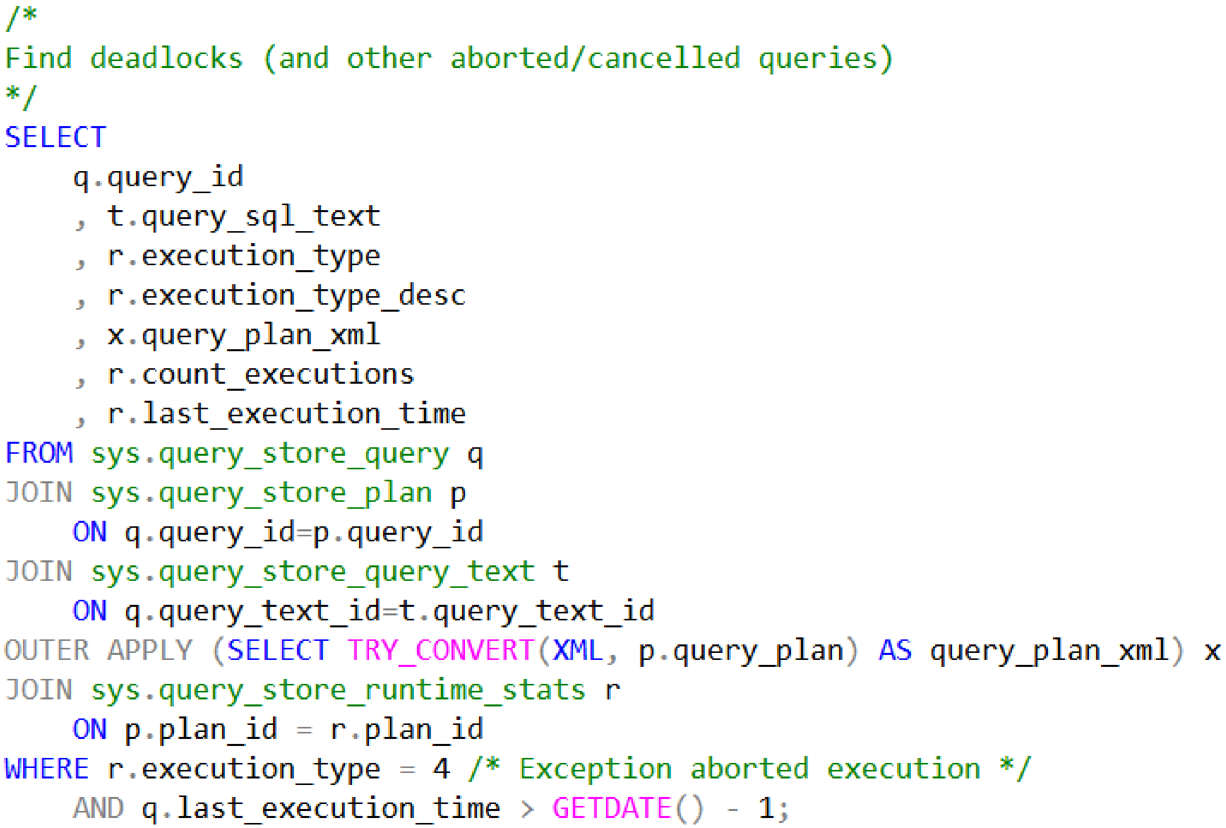
Deadlocks

- You can capture deadlocks with extended events
- Query Store can give you more info
- Use sys.query_store_runtime_stats
- WHERE execution_type = 4

What do those shapes mean?

- ● Successful query execution (0)
- ■ Aborted query (3)
- ▲ Error during execution (4)

Deadlocks



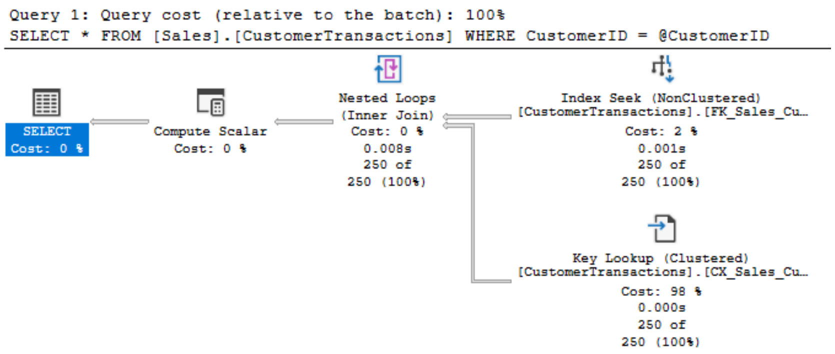
```
/*
Find deadlocks (and other aborted/cancelled queries)
*/
SELECT
    q.query_id
    , t.query_sql_text
    , r.execution_type
    , r.execution_type_desc
    , x.query_plan_xml
    , r.count_executions
    , r.last_execution_time
FROM sys.query_store_query q
JOIN sys.query_store_plan p
    ON q.query_id=p.query_id
JOIN sys.query_store_query_text t
    ON q.query_text_id=t.query_text_id
OUTER APPLY (SELECT TRY_CONVERT(XML, p.query_plan) AS query_plan_xml) x
JOIN sys.query_store_runtime_stats r
    ON p.plan_id = r.plan_id
WHERE r.execution_type = 4 /* Exception aborted execution */
    AND q.last_execution_time > GETDATE() - 1;
```

Deadlocks

- WARNING: this data is still aggregated
- Compare deadlocks using **sql_text**
- Review execution plans for objects locked in deadlocks
- Use **Tracked Queries** report

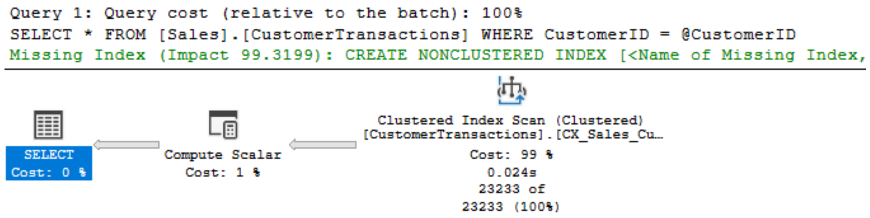
Parameter Sniffing

```
EXEC GetTransactionByCustomer
@CustomerID = 401
```



```
Query 1: Query cost (relative to the batch): 100%
SELECT * FROM [Sales].[CustomerTransactions] WHERE CustomerID = @CustomerID
```

```
EXEC GetTransactionByCustomer
@CustomerID = 976
```



```
Query 1: Query cost (relative to the batch): 100%
SELECT * FROM [Sales].[CustomerTransactions] WHERE CustomerID = @CustomerID
Missing Index (Impact 99.3199): CREATE NONCLUSTERED INDEX [<Name of Missing Index,
```

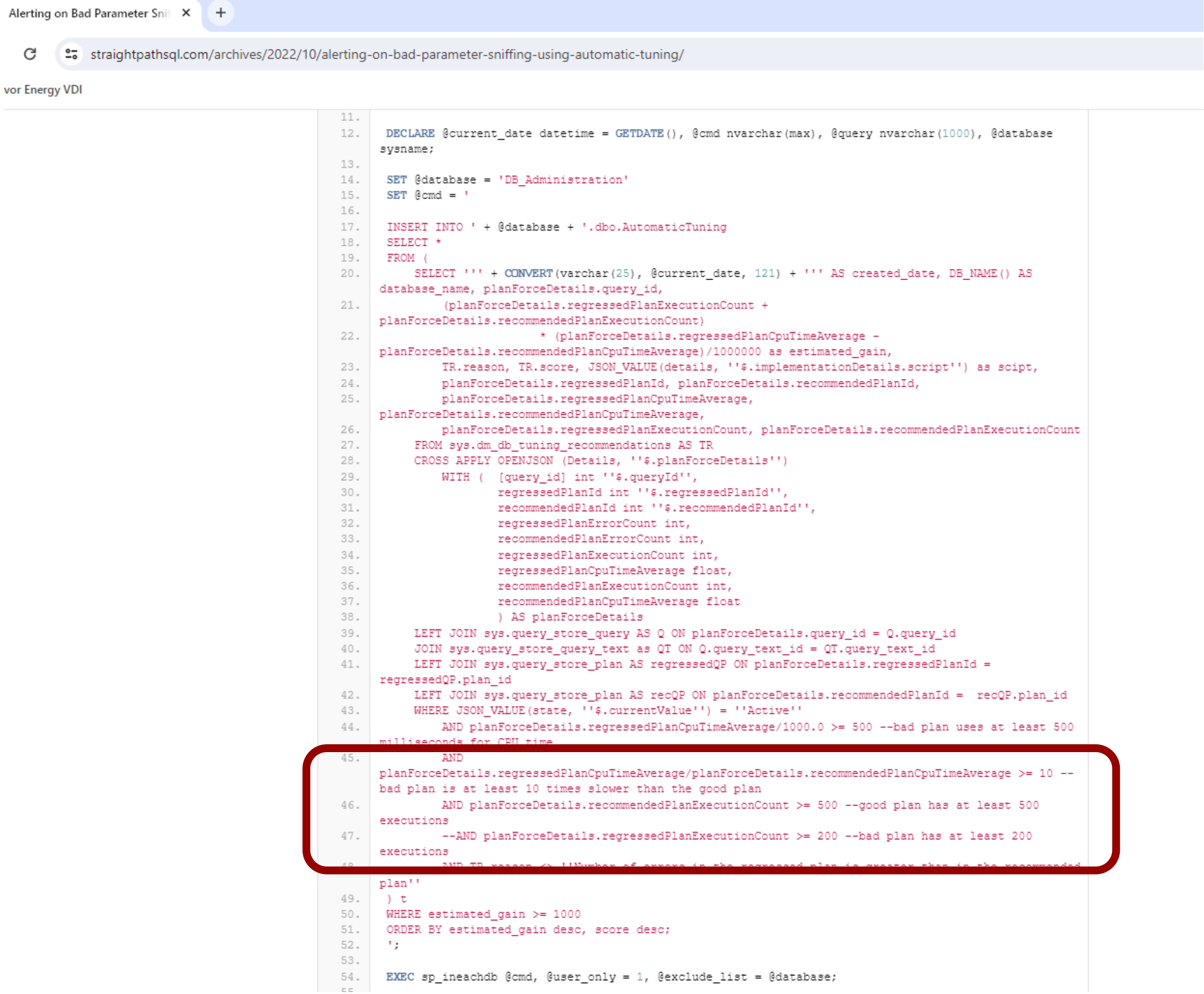
- Different plans, different performance

What about FORCE_LAST_GOOD_PLAN?

- Automatically force a better plan when found
- Can still choose a “bad” plan
- Can take hours to automatically revert

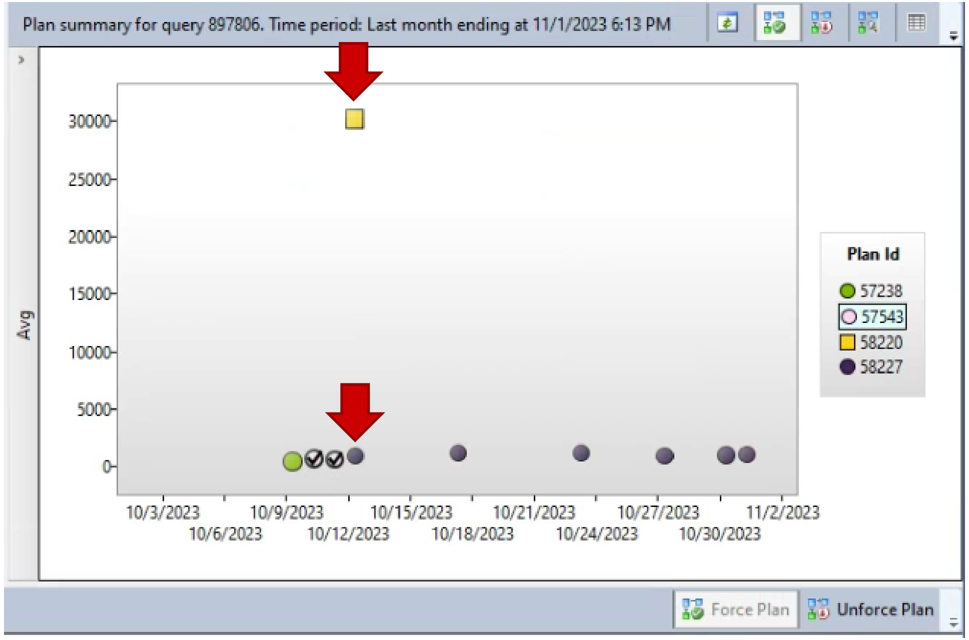
Parameter sniffing

- Query sys.dm_db_tuning_recommendations
- Insert JSON data from Details column into a table
- Create a job that queries the table
- Set job to “fail” when regressions occur


Alerting on Bad Parameter Sni
straightpathsql.com/archives/2022/10/alerting-on-bad-parameter-sniffing-using-automatic-tuning/
vor Energy VDI

```sql

 DECLARE @current_date datetime = GETDATE(), @cmd nvarchar(max), @query nvarchar(1000), @database
sysname;

 SET @database = 'DB_Administration'
 SET @cmd = '

 INSERT INTO ' + @database + '.dbo.AutomaticTuning
 SELECT *
 FROM (
     SELECT ''' + CONVERT(varchar(25), @current_date, 121) + ''' AS created_date, DB_NAME() AS
database_name, planForceDetails.query_id,
         (planForceDetails.regressedPlanExecutionCount +
planForceDetails.recommendedPlanExecutionCount)
                       * (planForceDetails.regressedPlanCpuTimeAverage -
planForceDetails.recommendedPlanCpuTimeAverage)/1000000 as estimated_gain,
         TR.reason, TR.score, JSON_VALUE(details, ''$.implementationDetails.script'') as scipt,
         planForceDetails.regressedPlanId, planForceDetails.recommendedPlanId,
         planForceDetails.regressedPlanCpuTimeAverage,
planForceDetails.recommendedPlanCpuTimeAverage,
         planForceDetails.regressedPlanExecutionCount, planForceDetails.recommendedPlanExecutionCount
     FROM sys.dm_db_tuning_recommendations AS TR
     CROSS APPLY OPENJSON (Details, ''$.planForceDetails'')
         WITH (  [query_id] int ''$.queryId'',
                 regressedPlanId int ''$.regressedPlanId'',
                 recommendedPlanId int ''$.recommendedPlanId'',
                 regressedPlanErrorCount int,
                 recommendedPlanErrorCount int,
                 regressedPlanExecutionCount int,
                 regressedPlanCpuTimeAverage float,
                 recommendedPlanExecutionCount int,
                 recommendedPlanCpuTimeAverage float
                 ) AS planForceDetails
     LEFT JOIN sys.query_store_query AS Q ON planForceDetails.query_id = Q.query_id
     JOIN sys.query_store_query_text as QT ON Q.query_text_id = QT.query_text_id
     LEFT JOIN sys.query_store_plan AS regressedQP ON planForceDetails.regressedPlanId =
regressedQP.plan_id
     LEFT JOIN sys.query_store_plan AS recQP ON planForceDetails.recommendedPlanId =  recQP.plan_id
     WHERE JSON_VALUE(state, ''$.currentValue'') = ''Active''
         AND planForceDetails.regressedPlanCpuTimeAverage/1000.0 >= 500 --bad plan uses at least 500
milliseconds for CPU time
         AND
planForceDetails.regressedPlanCpuTimeAverage/planForceDetails.recommendedPlanCpuTimeAverage >= 10 --
bad plan is at least 10 times slower than the good plan
         AND planForceDetails.recommendedPlanExecutionCount >= 500 --good plan has at least 500
executions
         --AND planForceDetails.regressedPlanExecutionCount >= 200 --bad plan has at least 200
executions
         AND TR.reason <> ''Number of errors in the regressed plan is greater than in the recommended
plan''
 ) t
 WHERE estimated_gain >= 1000
 ORDER BY estimated_gain desc, score desc;
 ';

 EXEC sp_ineachdb @cmd, @user_only = 1, @exclude_list = @database;
```

What happens next?



Plan summary for query 897806. Time period: Last month ending at 11/1/2023 6:13 PM

- https://rb.gy/p1bme2

Let’s review what we have learned

PASS Data Community SUMMIT 2023

Summary – Setting up Query Store

- Which defaults to change (maybe)
- …and which to leave alone
- Trace flags
- Set up monitoring job

Summary – Tools you can use

- SSMS Built-in Reports
- Forced plans are rarely a long-term solution
- T-SQL queries
- sp_BlitzQueryStore, sp_QuickieStore

Summary – Problems you can solve

- Testing index changes
- Reducing pain from SQL Server upgrades
- Deadlocks
- Parameter sniffing

That’s the end.
Thank you!

desertdba.com

✉ jeff@desertdba.com

github.com\desertdba