**Automated Tuning**

# How to check if feature is enabled

Run on Primary server

SELECT name, actual\_state\_desc, reason\_desc

FROM <yourDatabase>.sys.database\_automatic\_tuning\_options;

Graphical user interface, text, application, table

Description automatically generated

If you run on Secondary, you will get wrong result.

Graphical user interface, text, application

Description automatically generated

After failover and replica becomes a Primary one, feature will be automatically enabled on new Primary and disabled on old Primary replica.

# How to Enable feature

1. Make sure that Query Store is enabled in database and in Read-Write mode
2. Run command only on Primary replica

ALTER DATABASE <yourDatabase>

SET AUTOMATIC\_TUNING ( FORCE\_LAST\_GOOD\_PLAN = ON );

1. You will get an error if you try to enable the feature on Secondary replica.

Graphical user interface, text, application

Description automatically generated

# How to check recommendations

Note, after failover existing recommendations are cleared, and system becomes gathering data from the beginning.

Execute query (assuming *PolicySales* is your target database)

SELECT

name,reason,valid\_since,last\_refresh,state,is\_executable\_action,is\_revertable\_action, score, details

from Policysales.sys.dm\_db\_tuning\_recommendations order by last\_refresh desc

A screenshot of a computer

Description automatically generated

To see currently forced plans by the feature, add condition

SELECT

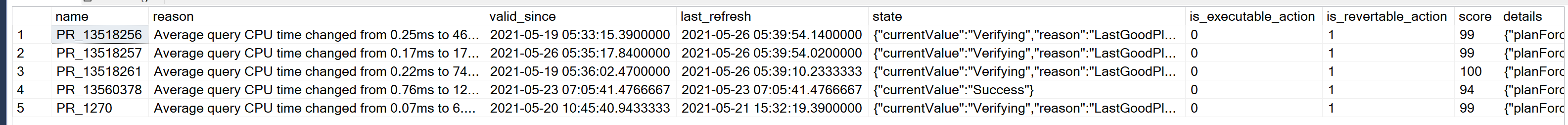
name,reason,valid\_since,last\_refresh,state,is\_executable\_action,is\_revertable\_action, score, details

from Policysales.sys.dm\_db\_tuning\_recommendations

WHERE is\_revertable\_action = 1

AND execute\_action\_initiated\_by = 'SYSTEM'

order by last\_refresh desc



You may also verify this information by checking *Queries with Forced Plans* tab in Query Store.

A screenshot of a computer

Description automatically generated

# How to check quality of recommendations (plan forcing)

You may get *query\_id* from result of query above (and remove prefix PR\_), then open tab *Tracked Queries* in Query Store, then enter required query\_id. Forced plan is checked. You can visually see the benefits of the forced plan.

Sometimes you need to expand time interval in Configure tab to see results of executed queries

**A screenshot of a computer

Description automatically generated**

# How to get details of feature recommendations

Execute query

USE PolicySales

GO

SELECT TOP 10 reason, score,

script = JSON\_VALUE(details, '$.implementationDetails.script'),

planForceDetails.\*,

estimated\_gain = (regressedPlanExecutionCount + recommendedPlanExecutionCount)

\* (regressedPlanCpuTimeAverage - recommendedPlanCpuTimeAverage)/1000000,

error\_prone = IIF(regressedPlanErrorCount > recommendedPlanErrorCount, 'YES','NO')

FROM sys.dm\_db\_tuning\_recommendations

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

ORDER BY score desc;

Results will be similar to below screen where you can see estimated gain in seconds based on number of executions of recommended plan vs regressed one.

A picture containing table

Description automatically generated

# How to undo feature recommendations

To undo plan forcing, simply unforce the plan.

You can use *Queries with Forced Plan tab* and simple click on Unforce Plan button with chosen plan highlighted

Graphical user interface, application

Description automatically generated

Another way to unforce the plan is to execute stored procedure

EXEC sp\_query\_store\_unforce\_plan @query\_id = 13560378, @plan\_id = 12213552;

Here values for query\_id and plan\_id are taken from screenshot above.

Graphical user interface, text, application

Description automatically generated

And re-executing the same SQL statement, you can see the values in *state* and *is\_revertable\_action* has changed

A picture containing calendar

Description automatically generated

# Preventing Plan from Re-Forcing

You may delete plan from query store after un-forcing it and prevent it for consideration for further recommendations by the feature in the future (if plan is still valid).

For example, to delete unforced earlier plan 12213552, execute command

EXEC sp\_query\_store\_remove\_plan @plan\_id = 12213552

After that in Tracked Queries tab of Query Store for the same query\_id, we will see only remaining plans, and Automatic Tuning will not be able to use deleted plan in the further analysis (until new executions will generate similar plan, but with another plan\_id)

Graphical user interface, text, application

Description automatically generated

# How to Disable feature

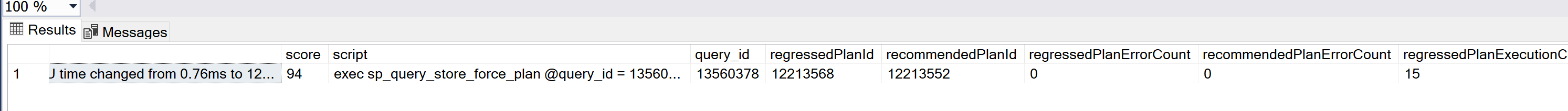
1. Make sure that Query Store is enabled in database and in Read-Write mode
2. Run command only on Primary replica

ALTER DATABASE <yourDatabase>

SET AUTOMATIC\_TUNING ( FORCE\_LAST\_GOOD\_PLAN = OFF );

A picture containing diagram

Description automatically generated



**TempDB Metadata Memory Optimized**

/\*

Because it�s so easy to create & use temp tables, some applications grow to depend on �em.

If your workload creates, loads, and drops thousands of temp tables per second,

you can hit a SQL Server bottleneck involving how SQL Server tracks where objects live in TempDB.

It�s called latch contention, also known as PFS/GAM/SGAM contention. We�ll demo it,

and then talk about a few ways to resolve it.

\*/

/\* If you're on SQL Server 2019, check to make

sure this returns a 0: \*/

SELECT SERVERPROPERTY('IsTempDBMetadataMemoryOptimized');

GO

/\* If it returns a 1, let's turn this feature off

for now in order to demo SQL Server's traditional

problem in TempDB: \*/

ALTER SERVER CONFIGURATION SET MEMORY\_OPTIMIZED TEMPDB\_METADATA=OFF;

/\* And you'll then need to restart SQL Server. \*/

/\* Before we start, review the number of files

we've got in TempDB: \*/

SELECT type\_desc, name, physical\_name,

    size \* 8.0 / 1024 AS size\_mb

    FROM tempdb.sys.database\_files

    ORDER BY type\_desc DESC;

USE StackOverflow2010;

GO

CREATE OR ALTER PROCEDURE dbo.TempTable AS

    SELECT TOP 1000 Id, AboutMe

    INTO #t1

    FROM dbo.Users WITH (NOLOCK)

    OPTION (MAXDOP 1);

GO

/\* When you run just one, it's quick: \*/

SET STATISTICS TIME, IO ON;

GO

EXEC TempTable;

/\* But run that in SQLQueryStress on a lot of

threads while we measure: \*/

EXEC sp\_BlitzFirst @ExpertMode = 1, @Seconds = 60;

EXEC sp\_WhoIsActive;

GO

--1518.2  PAGELATCH\_UP

/\* What if we try table variables? \*/

CREATE OR ALTER PROCEDURE dbo.TableVariable AS

BEGIN

    DECLARE @TableVariable TABLE (Id INT, AboutMe NVARCHAR(MAX));

    INSERT INTO @TableVariable (Id, AboutMe)

    SELECT TOP 1000 Id, AboutMe

    FROM dbo.Users WITH (NOLOCK)

    OPTION (MAXDOP 1);

END

GO

/\* Try just one: \*/

SET STATISTICS TIME, IO ON;

GO

EXEC TableVariable;

/\* Then run it in SQLQueryStress on a lot of threads. \*/

EXEC sp\_BlitzFirst @ExpertMode = 1, @Seconds = 60;

EXEC sp\_WhoIsActive;

GO 20

/\* SQL Server 2019 brings a new system-level

feature to help solve this: \*/

ALTER SERVER CONFIGURATION SET MEMORY\_OPTIMIZED TEMPDB\_METADATA=ON;

/\* Restart the SQL Server instance for it to take

effect, then check it: \*/

SELECT SERVERPROPERTY('IsTempDBMetadataMemoryOptimized');

GO

/\* Then run both the TempTable and TableVariable

load tests again while watching page latch waits. \*/

EXEC sp\_BlitzFirst @ExpertMode = 1, @Seconds = 60;

EXEC sp\_WhoIsActive;

GO 20

/\* Turn this back off for the lab: \*/

ALTER SERVER CONFIGURATION SET MEMORY\_OPTIMIZED TEMPDB\_METADATA=OFF;

/\* Restart the SQL Server instance for it to take

effect, then check it: \*/

SELECT SERVERPROPERTY('IsTempDBMetadataMemoryOptimized');

GO

/\* And look for PAGELATCH waits amongst your top 10,

indicating that you may need more files (or a

code change): \*/

sp\_BlitzFirst @OutputType = 'Top10'

/\* Not PAGEIOLATCH% or LATCH% - we're specifically

looking for PAGELATCH%. \*/