Query Store Scenarios:

1. **Parameter Sniffing :**

**P.S:- Put a Disclaimer that it has to be validated and may not be fast always.**

1. **Database Upgrade :**
2. **High CPU :**
3. **Disk IO :**
4. **Memory usage :**
5. **Blocking (Dependent : Ref VSTS # 3512507) :**
6. **Identify Top Query Plans :**
7. **Slow Query troubleshooting:**
   1. **Query Lacks Proper Index :**
   2. **Finding Long Compiles Plans**
   3. **Datatype Mismatch**
8. **Attention Queries**
9. **Time taken by CLR**

**Scenario1: Parameter Sniffing. Intermittently query takes a long time to run**

**==========**

Here are the steps to repro an issue that can be caused by Parameter Sniffing: In this example data in large\_table is not distributed evenly. For Id 1 to 9999 there is one row for each Id but for Id 10,001 there are 10,000 rows. Stored proc test\_proc retrieves data from large\_table based on the parameter @i. Let’s assume most of the time this SP will be called with @i = 10001. For that it will be better for the plan to do Table Scan instead of using Index Seek.

--Parameter Snoffing Example

drop table large\_table

create table large\_table (id int, name varchar(30), c3 binary(2000))

set nocount on

go

declare @i int

set @i = 1

while @i < 10000

begin

insert into large\_table (id, name) values (@i, 'test')

set @i = @i + 1

end

set nocount on

go

-- Insert similar records

insert into large\_table (id, name) values (100001, 'test')

go 10000

create index idx\_large\_table\_id on large\_table(id)

go

create procedure dbo.test\_proc (@i int) as

select \* from dbo.large\_table where id = @i

--Stored proc was called for the 1st time with parameter 10 and this atypical parameter

--i.e. It is not used that oftem

exec test\_proc 10 -- got the bad plan. Doing index seek

--now the plan is cached and it will use above plan irrespective of what parameters we pass

exec test\_proc 100001 -- -- got the bad plan. Doing index seek. Plan good for parameter 10

-- to get rid of the bad plan let's clear the cache

dbcc freeproccache

--after the cache was cleared stored proc is called for the 1st time with the parameter that will be used most often

exec test\_proc 100001 -- plan doing table scan

**To resolve the parameter sniffing issue using QS: (Note that this needs to be validated and may not be fast all the time)**

1. Identify the query (i.e. query id)

select query\_text\_id, \* from sys.query\_store\_query\_text

where query\_sql\_text like '%large\_table%'

--got 168

select query\_id, \* from sys.query\_store\_query

where query\_text\_id = 168

--got 255

**2. Identify the good plan id**

You can use this query to identify multiple plans

--Queries with multiple plans

WITH Query\_MultPlans

AS

(

select count(\*) as cnt, q.query\_id 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 p.query\_id = q.query\_id

group by q.query\_id

having count(distinct plan\_id) > 1

)

select q.query\_id, object\_name(object\_id) as ContainingObject,query\_sql\_text,

plan\_id, CONVERT(xml, p.query\_plan) AS query\_plan\_xml,--p.query\_plan as plan\_xml,

p.last\_compile\_start\_time, p.last\_execution\_time

from Query\_MultPlans qm

join sys.query\_store\_query q

on qm.query\_id = q.query\_id

join sys.query\_store\_plan p

on q.query\_id = p.query\_id

join sys.query\_store\_query\_text qt on qt.query\_text\_id = q.query\_text\_id

and q.query\_text\_id = 168 -- query\_text\_id we got from above

order by query\_id, plan\_id

**Got this sample output:**

query\_id ContainingObject query\_sql\_text plan\_id

255 test\_proc (@i int)select \* from dbo.large\_table where id = @i 271

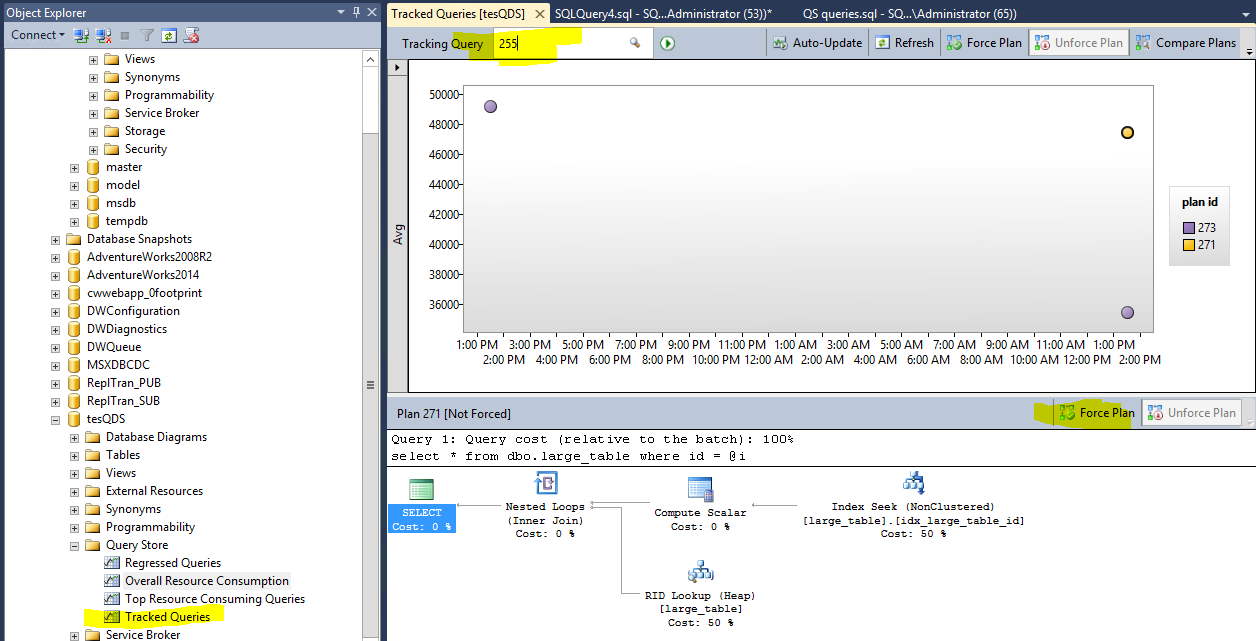
255 test\_proc (@i int)select \* from dbo.large\_table where id = @i 273

**3. Force the query plan that you want the query to use**

exec sp\_query\_store\_force\_plan /\*query\_id\*/255, /\*plan\_id\*/271

To do the same thing from GUI , you can follow this steps:

Go to Query Store 🡪Tracked queries folder. Enter 255 in the tracked query. Select the plan you want to force and click on Force Plan:



**Scenario 2: Database Upgrade**

**====================**

Query Store can be used to find out if the query performance regressed after the Databases are Upgrade say for e.g. SQL 14 to SQL 16 or if you apply any Service pack or CU. To do the same use the following steps:

* Enable QS before the database is upgraded
* Run your workload. Get the baseline.
* Upgrade the database
* Run your workload
* Check the Regressed Queries folder.

**Scenario 3: High CPU / Disk IO / Memory usage**

================================

If most of the queries are run on the SQL Server against the user database on which QS is enabled, we can use the query below to identify top queries that used the most CPU

--Query aggregates CPU for all finished queries executed in the last hour window and then returns top 25:

WITH AggregatedCPULastHour

AS

(

SELECT q.query\_id, SUM(count\_executions \* avg\_cpu\_time) as total\_CPU\_time,

COUNT (distinct p.plan\_id) as number\_of\_plans

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 rs.plan\_id = p.plan\_id

JOIN sys.query\_store\_runtime\_stats\_interval rsi ON rsi.runtime\_stats\_interval\_id = rs.runtime\_stats\_interval\_id

WHERE rsi.start\_time >= DATEADD(hour, -1, GETUTCDATE())AND rs.execution\_type\_desc = 'Regular'

GROUP BY q.query\_id

)

,OrderedCPU

AS

(

SELECT query\_id, total\_CPU\_time, number\_of\_plans, ROW\_NUMBER () OVER (ORDER BY total\_CPU\_time DESC, query\_id) AS RN

FROM AggregatedCPULastHour

)

SELECT qt.query\_sql\_text, object\_name(q.object\_id) AS containing\_object,

total\_cpu\_time AS total\_cpu, number\_of\_plans,

CONVERT(xml, p.query\_plan) AS query\_plan\_xml, p.is\_forced\_plan, p.last\_compile\_start\_time,q.last\_execution\_time

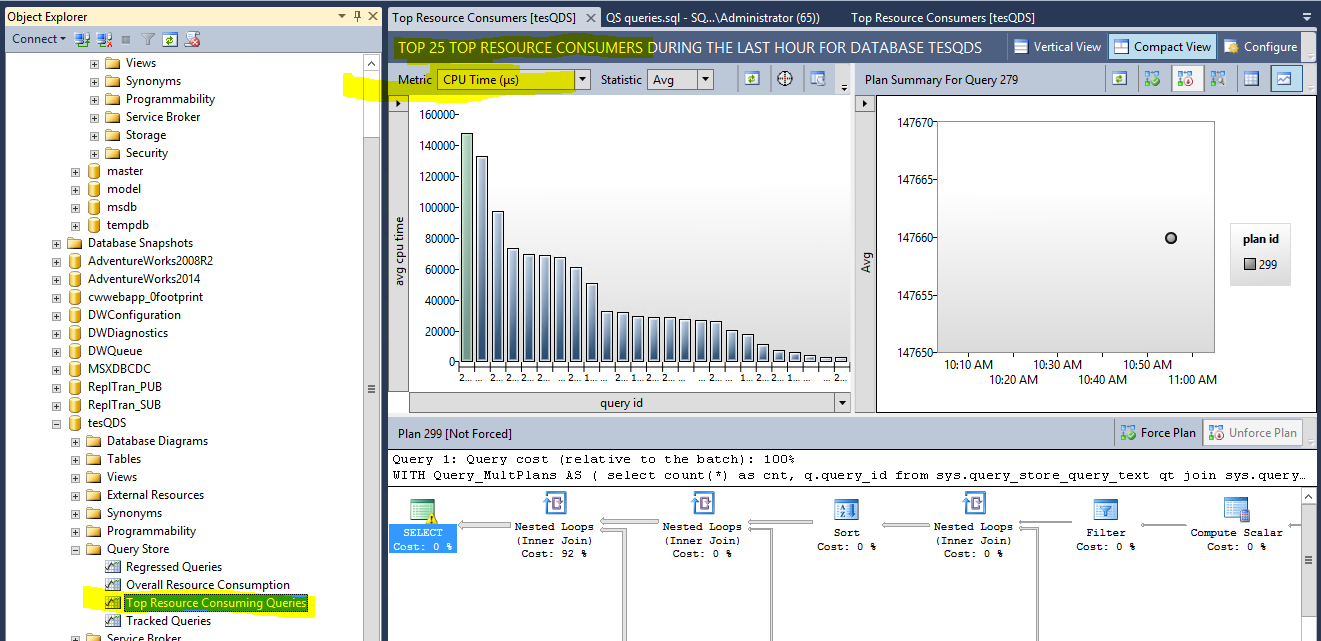
FROM OrderedCPU od JOIN sys.query\_store\_query q ON q.query\_id = od.query\_id

JOIN sys.query\_store\_query\_text qt ON q.query\_text\_id = qt.query\_text\_id

JOIN sys.query\_store\_plan p ON q.query\_id = p.query\_id

WHERE OD.RN <=25 ORDER BY total\_CPU\_time DESC

**You can do the same thing using the GUI**



**8. Slow Query Performance troubleshooting:**

a. **Query Lacks Proper Index:** You can use this query to see what query plans have the missing index warning and its suggested impact for all the queries executed in the last 1 hour ordered by the average duration:

select qt.query\_sql\_text,

CONVERT(xml, p.query\_plan).value('declare default element namespace "http://schemas.microsoft.com/sqlserver/2004/07/showplan"; (/ShowPlanXML/BatchSequence/Batch/Statements/StmtSimple/QueryPlan/MissingIndexes/MissingIndexGroup/@Impact)[1]' , 'decimal(18,4)') as Impact,

CONVERT(xml, p.query\_plan) AS query\_plan\_xml,

q.query\_id, qt.query\_text\_id, p.plan\_id, getutcdate() as CurrentUTCTime, rs.last\_execution\_time, rs.avg\_duration

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

where rs.last\_execution\_time > dateadd(hour, -1, getutcdate())

and CONVERT(xml,p.query\_plan).exist('declare default element namespace "http://schemas.microsoft.com/sqlserver/2004/07/showplan";/ShowPlanXML/BatchSequence/Batch/Statements/StmtSimple/QueryPlan/MissingIndexes/MissingIndexGroup/MissingIndex[@Database!="m"]') = 1

order by rs.avg\_duration desc

**b. Queries with the longest compile time executed in the last one hour**

-- N queries with the longest compile time executed within last hour

select top 10 qt.query\_sql\_text, p.avg\_compile\_duration, q.query\_id, qt.query\_text\_id, p.plan\_id, getutcdate() as CurrentUTCTime, rs.last\_execution\_time, rs.avg\_duration

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

where rs.last\_execution\_time > dateadd(hour, -1, getutcdate())

order by p.avg\_compile\_duration desc

9.

**10. Time taken by CLR queries**

Let’s say we have an Update statement that uses a UDF function and that in turn calls a CLR assembly.

For e.g.

update Account\_tbl\_ENC set AcctNumber\_RNDL = dbo.TryEncrypt(AcctNumber, 'None')

go

In this case TryEncrypt is a UDF function and that calls a CLR assembly. For this if we want to see how much time was spent on the CLR, you can use this query:

--Last N queries that were executed on the database

SELECT TOP 100 qt.query\_sql\_text, q.query\_id, qt.query\_text\_id, p.plan\_id, rs.last\_execution\_time, rs.avg\_clr\_time, rs.max\_clr\_time,rs.avg\_cpu\_time, rs.max\_cpu\_time, rs.max\_dop, rs.avg\_duration, rs.max\_duration, rs.max\_rowcount, CONVERT(xml, p.query\_plan) AS query\_plan\_xml

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

ORDER BY rs.last\_execution\_time DESC