Fast Start for Microsoft Azure - SQL Server Azure Database



Demo – Query Store in SQL Database

Table of Contents

[1 Summary 1](#_Toc422156661)

[2 Enable Query Store for Your Database 2](#_Toc422156662)

[2.1 Run the Workload 3](#_Toc422156663)

[2.2 Perform Data Analysis 3](#_Toc422156664)

[2.3 Tracking Query and Plan Execution 6](#_Toc422156665)

[2.4 Addressing Performance for Poorly Running Queries 8](#_Toc422156666)

[2.5 Examining Regressed Queries 9](#_Toc422156667)

[2.6 Examining the Query Store Using DMVs 11](#_Toc422156668)

[2.7 Plan forcing from Transact-SQL 13](#_Toc422156669)

[2.8 Overall Resource Consumption 14](#_Toc422156670)

[3 References 17](#_Toc422156671)

1. Summary

Query Store is a new feature in SQL Server 2016, and is available in preview on the latest version of Azure SQL Database V12. It’s designed to help you with query performance troubleshooting.

As your database grows in size it is not unusual to experience performance regressions. Without the ability to see all the changes in the query plans it’s difficult to figure out why regressions happen and what you can do to prevent them in the future. Troubleshooting can take hours or even days away from your precious productivity time.

Similar to an airplane’s flight data recorder, Query Store collects and presents detailed historic information about all queries, greatly simplifying performance forensics by reducing the time to diagnose and resolve issues.

Collected data is separated by time windows, allowing you to observe changes in query performance over time. If a query generates different plans, you can use Query Store to analyze the plan changes, identify possible performance degradation, and even force the query processor to use a particular plan for your query.

The Query Store helps as it collects all query texts along with the relevant properties. It stores all plan choices and performance metrics and allows you to force plans from history. Furthermore, it works across recompiles, restarts, failovers and upgrades.

The Query Store is accessible through Transact-SQL. To analyze collected data and manage the feature you must install SQL Server Management Studio or SQL Server Data Tools.

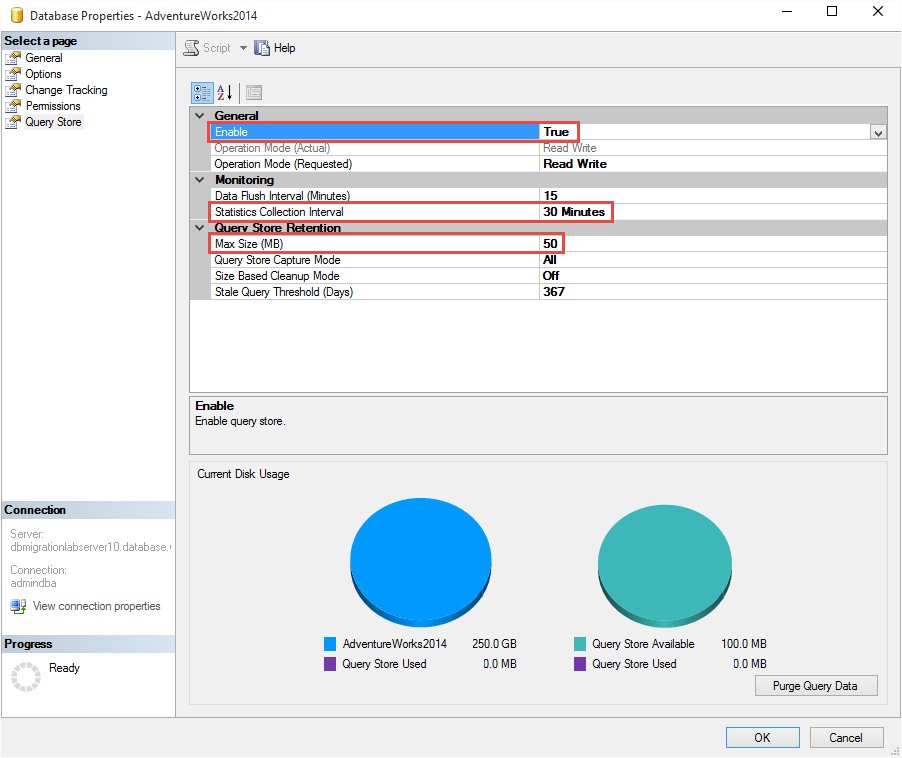
We will be using the preview portal <https://portal.azure.com/> and SQL Server Management Studio from SQL Server 2016.

1. Enable Query Store for Your Database

The Query Store is a database scoped feature so you must first enable it for the databases that you want to monitor. Once enabled, Query Store will store collected data inside the database storage space. To enable the feature for SQL Database, connect to your Azure SQL Server instance and run the following script while you are connected to the **master** database.

ALTER DATABASE [AdventureWorks2014] SET QUERY\_STORE = ON;

If you are running SQL Server Management Studio for SQL Server 2016 you can enable the feature from the GUI. By right clicking the database and clicking the Property/Query Store tab:



You may want to change defaults for the following options in order to adjust them to your requirements:

* **Max Size MB** – limit for the data space that Query Store will take inside your database. The default (100 MB) may not be sufficient if your workload generates large number of different queries or plans or if you want to keep query history for a longer period of time. Keep track of current space usage and increase configuration value to prevent Query Store from switching to “Read Only” mode when disk size exceeds the limit.
* **Statistics Collection Interval:** defines level of granularity for the runtime data (the default is 1 hour). Consider using lower value if you need finer granularity but keep in mind that it will directly affect the size of Query Store data.

For the sake of this demo we will change the **statistics collection interval** from 1 hour to 1 minute. This is the level of Query Store granularity with a very small impact on performance.  
  
  
  
The T-SQL for this change is:   
  
ALTER DATABASE AdventureWorks2014   
SET QUERY\_STORE (INTERVAL\_LENGTH\_MINUTES = 1)

* 1. Run the Workload

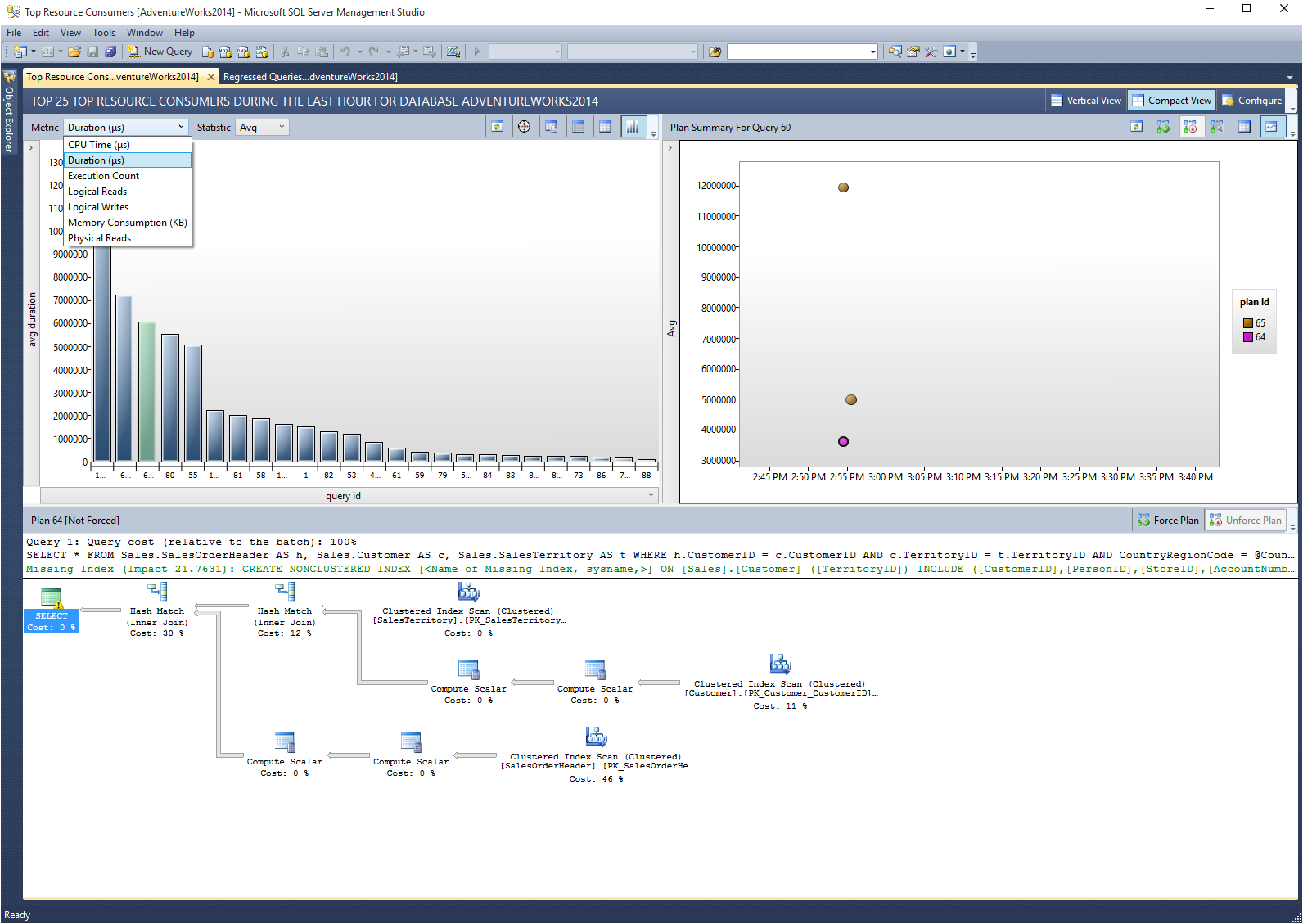
Run your own workload against your database. In this demo we are running a workload against the AdventureWorks2014.

Run the QueryStoreDemo.cmd file located in the *..\DEMOS\\_08 Query Store\QueryStoreDemo* directory. This will launch 3 batch executions in a loop causing plan caching with various impacts.

* 1. Perform Data Analysis

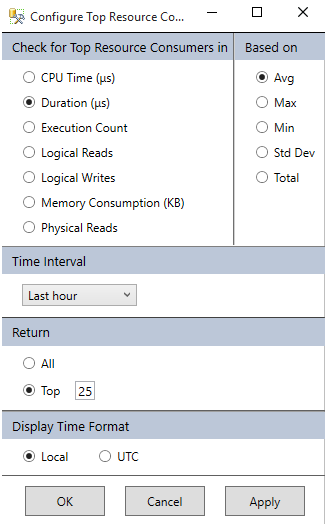
In this step, we will show you how to use Query Store to easily identify the most resource consuming queries in your workload.

If you are running SQL Server Management Studio for SQL Server 2016, double click on the **Top Resource Consuming Queries** sub-folder under the database node in Object Explorer. This will open a new window with detailed information on the most expensive queries in your system.



User interface is separated into three panes: A histogram representing top resource consuming queries (left), a plan summary for selected query (right) and visual query plan for selected plan (bottom).

Click the **Configure** button to control how many queries you want to analyze and the time interval of interest. Additionally, you can choose between different resource consumption dimensions (duration, CPU, memory, IO, number of execution) and the baseline (Average, Min, Max, Total, Standard Deviation).



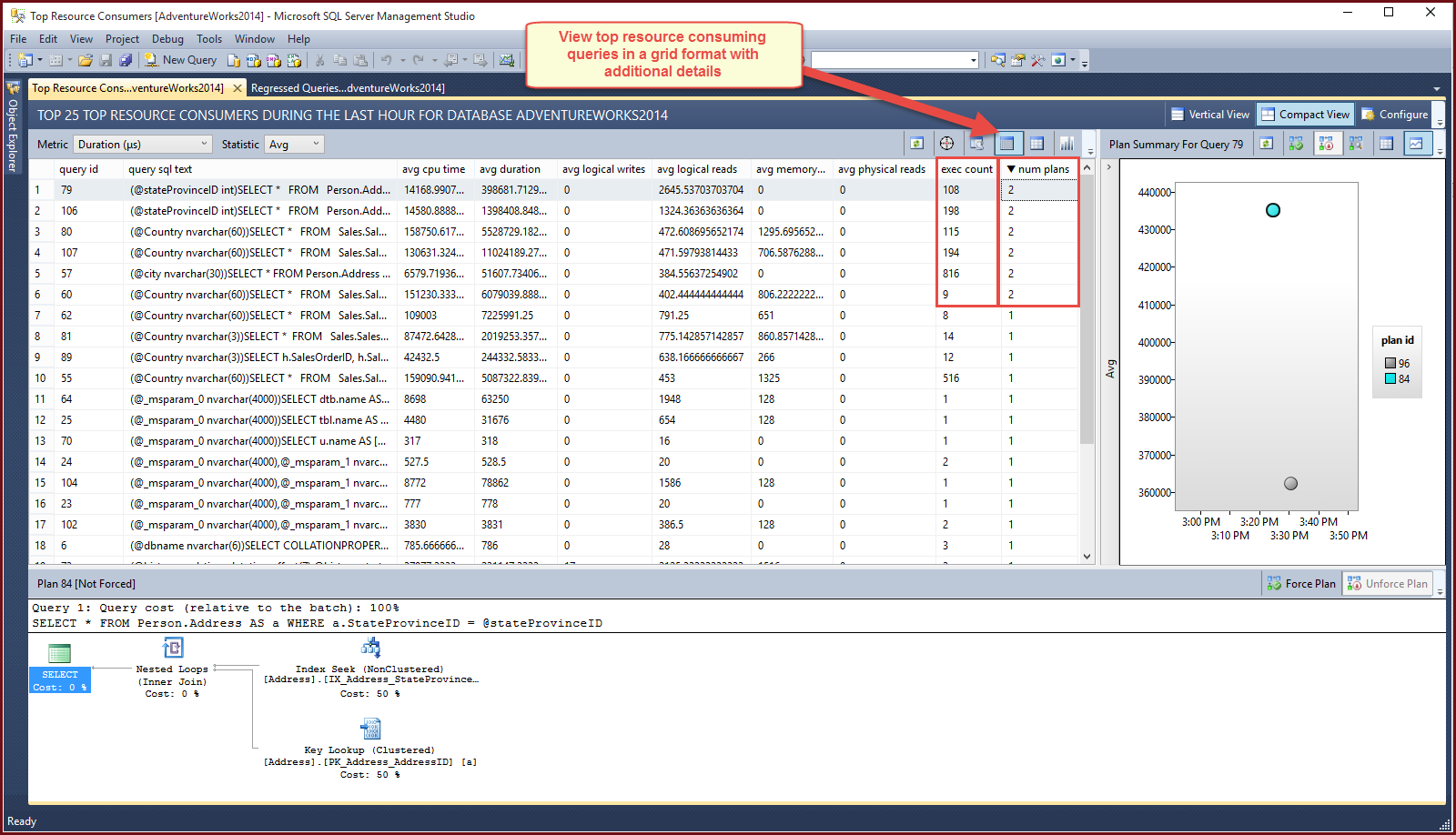
A tooltip over the query provides details on query text, query id (internal identification), average duration, execution count and the number of different plans.

Look at the plan summary on the right to analyze the execution history and learn about the different plans and their runtime statistics. Use the bottom pane to examine the different plan shapes or to compare them visually, rendered side by side (use the **Compare** button).

**Examining the Grid Format**

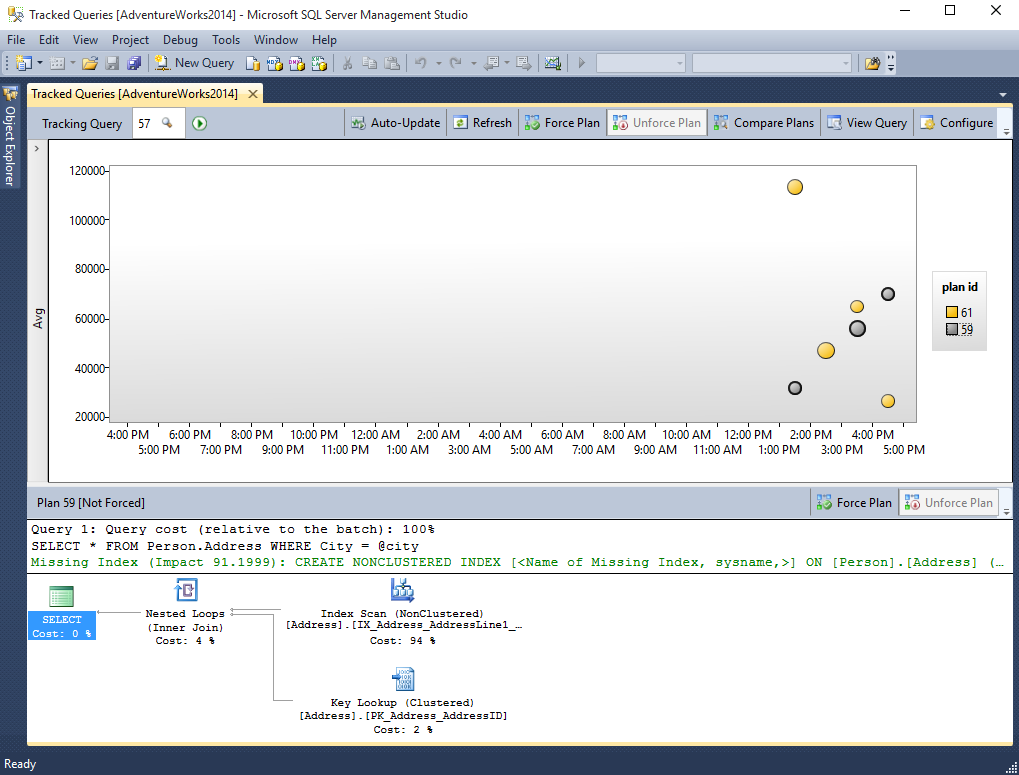
The Top Resource Consuming Queries dashboard is very robust. You can make several other adjustments to be able to analyze the query store in different ways. For example, click the **‘View top resource consuming queries in a grid format with additional details’** box . The third icon from the right on the left most pane. You can then sort by the different columns such as execution count, plan generation, etc.

Examine the report by looking at the queries listed and adjusting the middle divider which lets you look at Avg, Min, Max, and StdDiv. Also adjust the **Metric** and the **Statistic** which allows you to adjust the view and level of details.



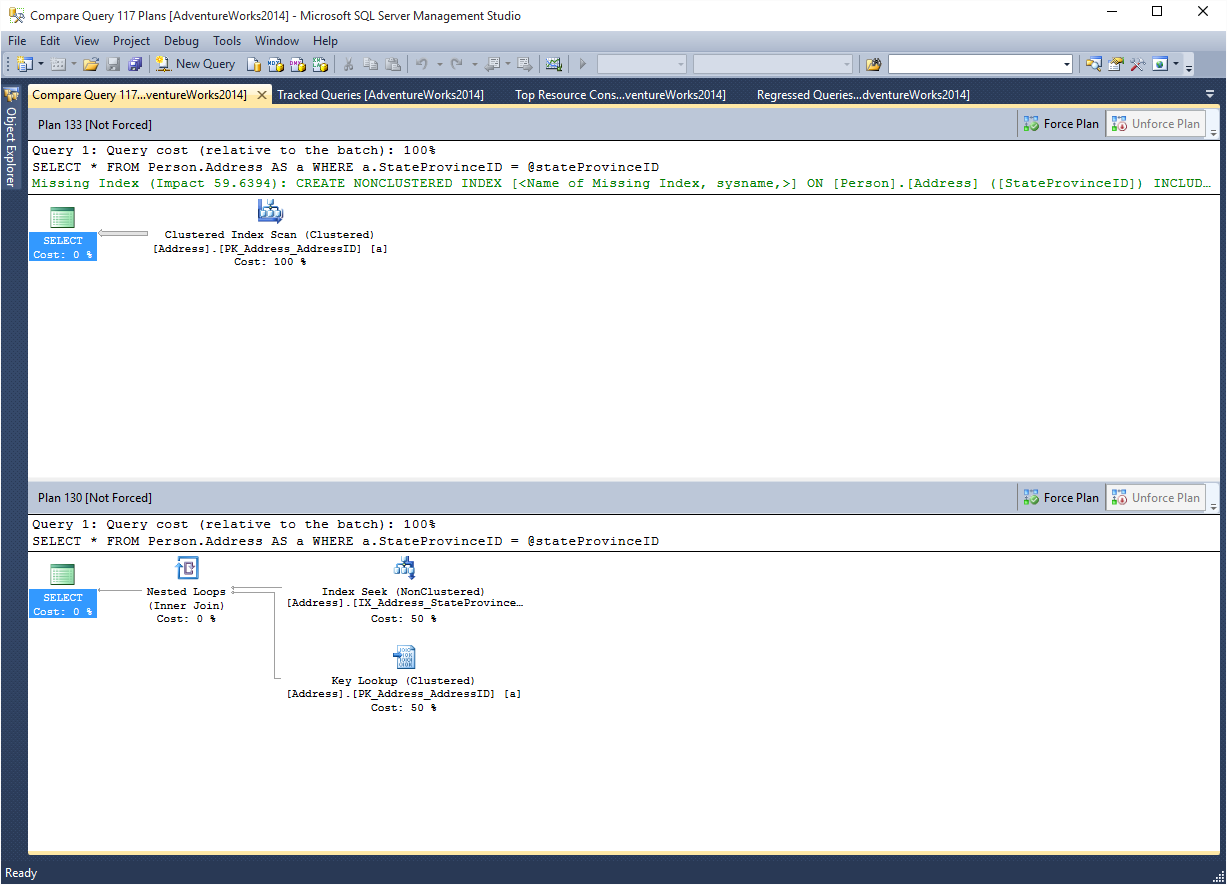
* 1. Tracking Query and Plan Execution

You can also track a query and add it to the Tracked Queries view by clicking the target  the second icon over to the left in the let most pane. This will track this particular query in the Tracked Query view.



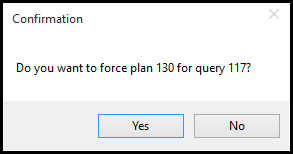
Here you can automatically update query store statistics, view different plan executions, compare plans, and force a preferred plan (or unforce).

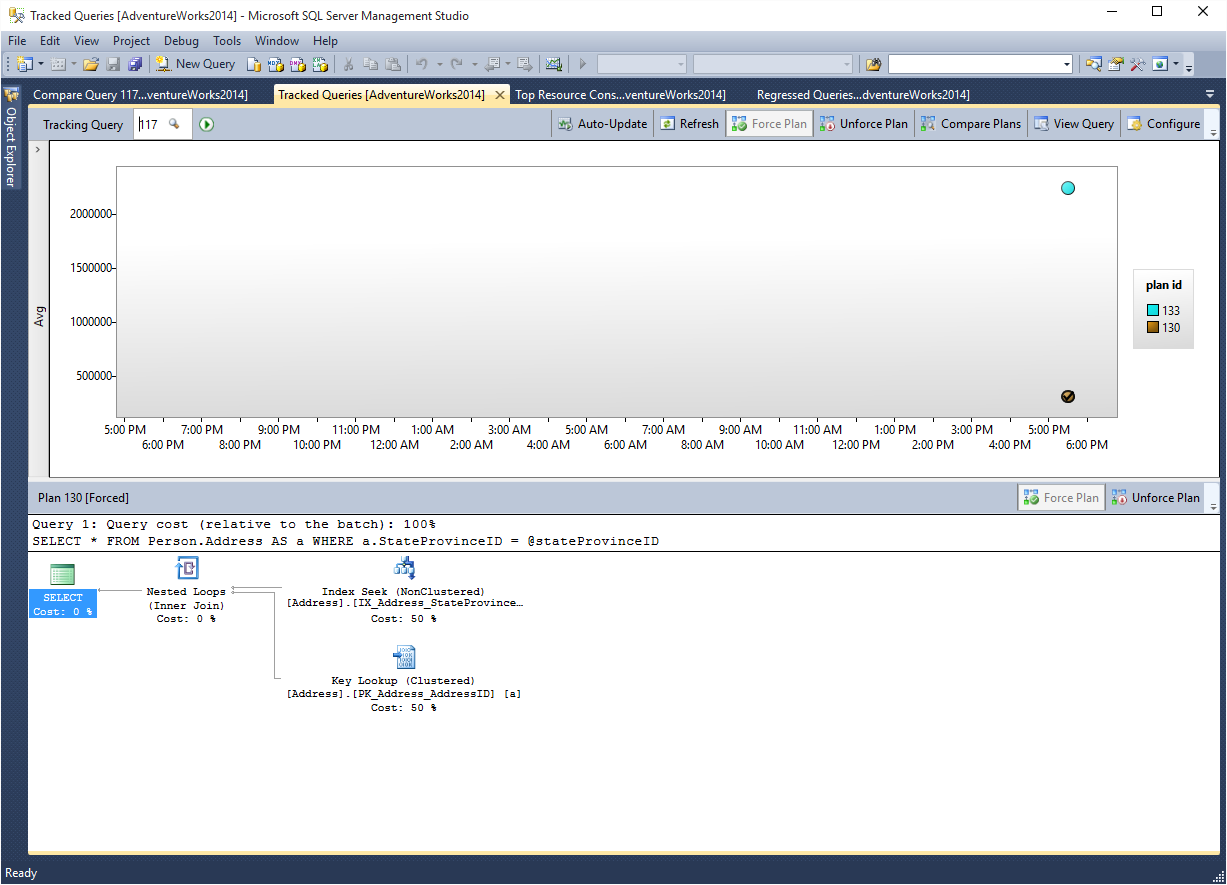
When you Click **Compare** plans, you will be able to see the statement and plans side by side.



* 1. Addressing Performance for Poorly Running Queries

If your workload contains a query that is being executed with different plans and variable performance, you can use Query Store to force SQL Server to always use the optimal plan in future executions.

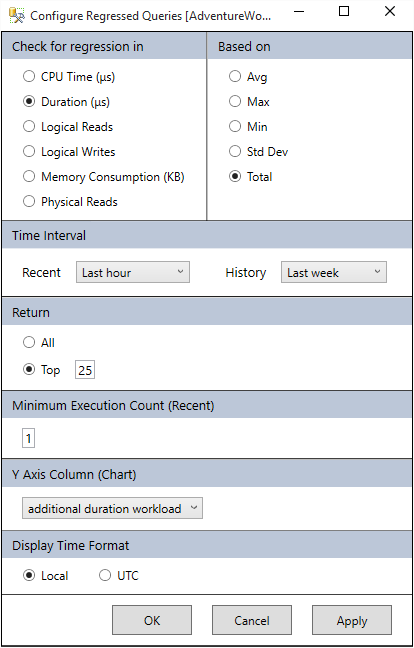
  
  
By clicking on the preferred, better performing SQL Server plan and clicking ‘Force Plan’, further executions would be leverage the chosen plan. Clicking **Force Plan** will make sure that SQL Server always uses that plan. The Forced plan now appears with a check mark and Plan Summary shows that it’s being used in subsequent executions



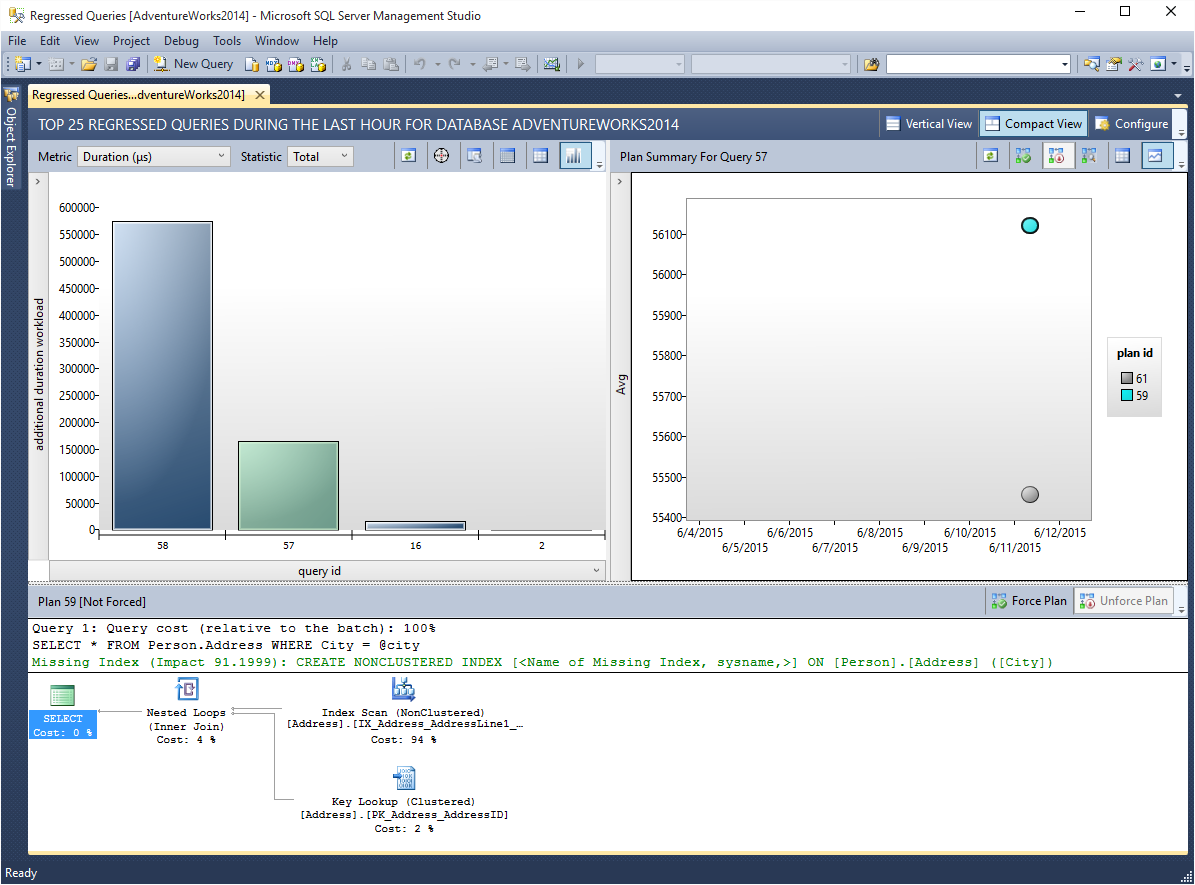
**Note:** We will cover forcing plans via T-SQL in a subsequent step.

* 1. Examining Regressed Queries

Click the ‘Regressed Queries’ from the Query Store folder in SQL Server Management Studio. This will show you any queries that have regressed based on the configuration. You can change the regression metric, what the impact is based on, the time interval, and the number of queries being returned in the regression view.



You can click on each of the regression queries in the bar chart, change the view to query grid view to examine the impact at the query level.



* 1. Examining the Query Store Using DMVs

You can identify the most resource consuming queries using a Transact-SQL script. This is especially useful if you do not have access to the SQL Server Management Studio for SQL Server 2016. The Query Store exposes the following views for your analysis.

|  |  |
| --- | --- |
| View | Description |
| sys.query\_store\_query\_text | Information about captured query texts. |
| sys.query\_context\_settings | Different runtime combinations of semantics-affecting context settings (SET options that influence plan shape, language ID, …) |
| sys.query\_store\_query | Unique combination of query text and context settings |
| sys.query\_store\_plan | Information about plans SQL Server uses to execute queries in the system. |
| sys.query\_store\_runtime\_stats\_interval | Aggregation intervals (time windows) created in Query Store. |
| sys.query\_store\_runtime\_stats | Runtime statistics for executed query plans, aggregated on per-interval basis |

You can run the following script to get *25 queries based on total duration within last hour of execution*:

WITH AggregatedDurationLastHour

AS

(

SELECT q.query\_id, SUM(count\_executions \* avg\_duration) AS total\_duration,

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

FROM sys.query\_store\_query\_text AS qt JOIN sys.query\_store\_query AS q

ON qt.query\_text\_id = q.query\_text\_id

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

JOIN sys.query\_store\_runtime\_stats AS rs ON rs.plan\_id = p.plan\_id

JOIN sys.query\_store\_runtime\_stats\_interval AS 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

)

,OrderedDuration

AS

(

SELECT query\_id, total\_duration, number\_of\_plans,

ROW\_NUMBER () OVER (ORDER BY total\_duration DESC, query\_id) AS RN

FROM AggregatedDurationLastHour

)

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

total\_duration AS total\_duration\_microseconds, 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 OrderedDuration od JOIN sys.query\_store\_query AS 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\_duration DESC

* 1. Plan forcing from Transact-SQL

You can identify queries with plan regressions by directly querying Query Store views. The script from above is slightly modified to return queries with multiple plans and to include average duration per plan for every measurement interval for the purpose of comparison:

WITH AggregatedDurationLastHour

AS

(

SELECT q.query\_id, SUM(count\_executions \* avg\_duration) AS total\_duration,

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

FROM sys.query\_store\_query\_text AS qt JOIN sys.query\_store\_query AS q

ON qt.query\_text\_id = q.query\_text\_id

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

JOIN sys.query\_store\_runtime\_stats AS rs ON rs.plan\_id = p.plan\_id

JOIN sys.query\_store\_runtime\_stats\_interval AS 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

)

,OrderedDuration

AS

( SELECT query\_id, total\_duration, number\_of\_plans,

ROW\_NUMBER () OVER (ORDER BY total\_duration DESC, query\_id) AS RN

FROM AggregatedDurationLastHour )

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

p.plan\_id,rsi.start\_time as interval\_start, rs.avg\_duration,

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

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

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

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

JOIN sys.query\_store\_runtime\_stats AS rs ON rs.plan\_id = p.plan\_id

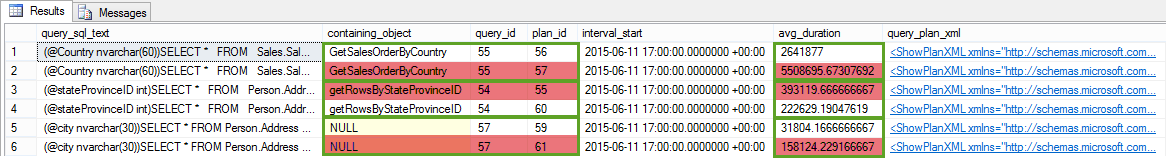
JOIN sys.query\_store\_runtime\_stats\_interval AS rsi ON rsi.runtime\_stats\_interval\_id = rs.runtime\_stats\_interval\_id

WHERE rsi.start\_time >= DATEADD(hour, -1, GETUTCDATE())

AND OD.RN <=25 AND number\_of\_plans > 1

ORDER BY total\_duration DESC, query\_id, rsi.runtime\_stats\_interval\_id, p.plan\_id

This is the part of the obtained data set:



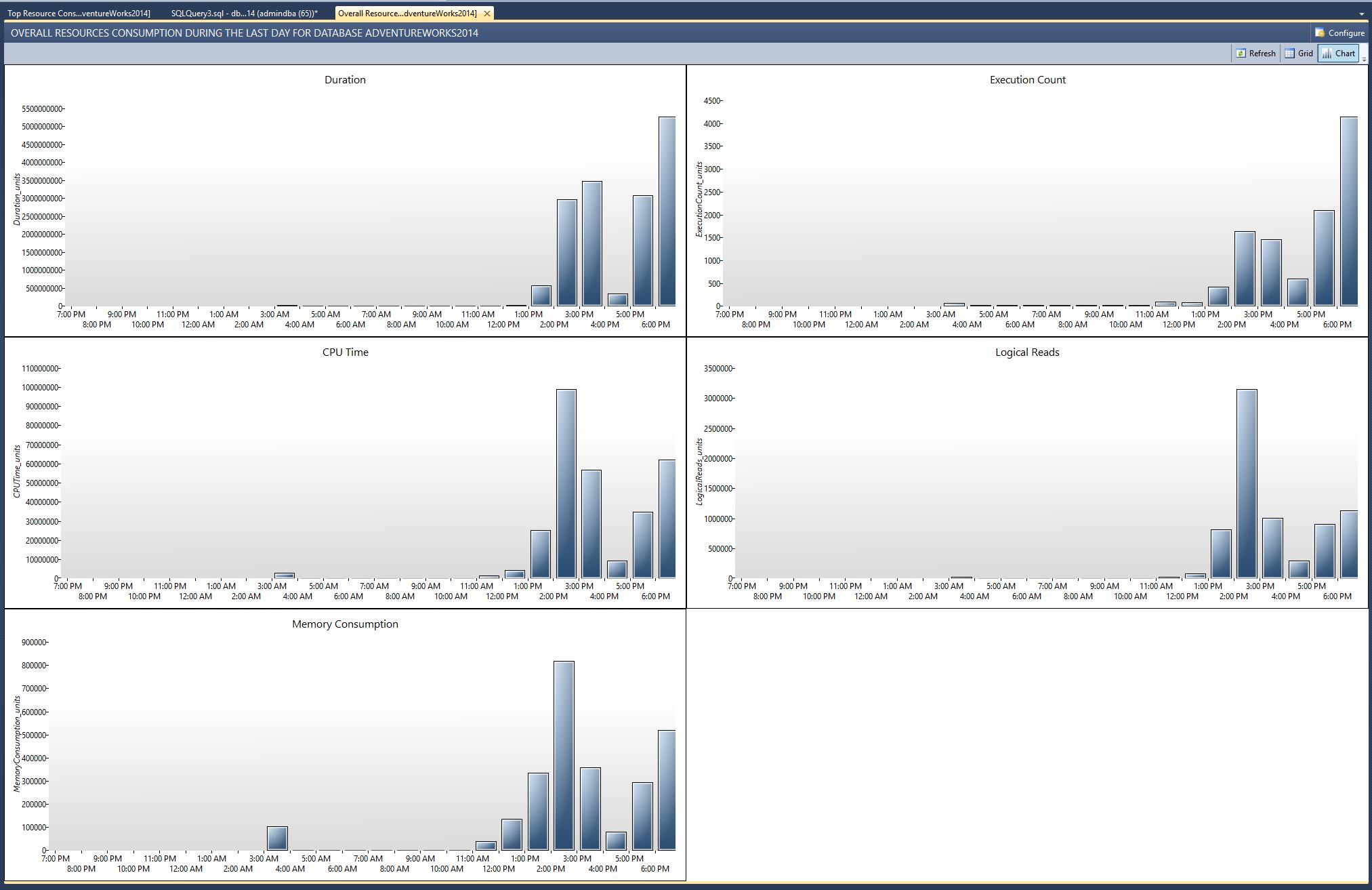
From the data set you can observe that query id = 55, 54, and 57 were executed with two plans each that had different performance (additionally, you can click on query\_plan\_xml column in the result grid to visually inspect the difference). For example, the plan with plan id = 56 (green rectangles) shows on average 2x better performance (i.e. less avg\_duration), than plan id = 57 (red rectangles) for every measurement interval.

Therefore, you will likely want to enforce plan id = 55 for query id = 56:

EXEC sys.sp\_query\_store\_force\_plan @query\_id = 55, @plan\_id = 56;

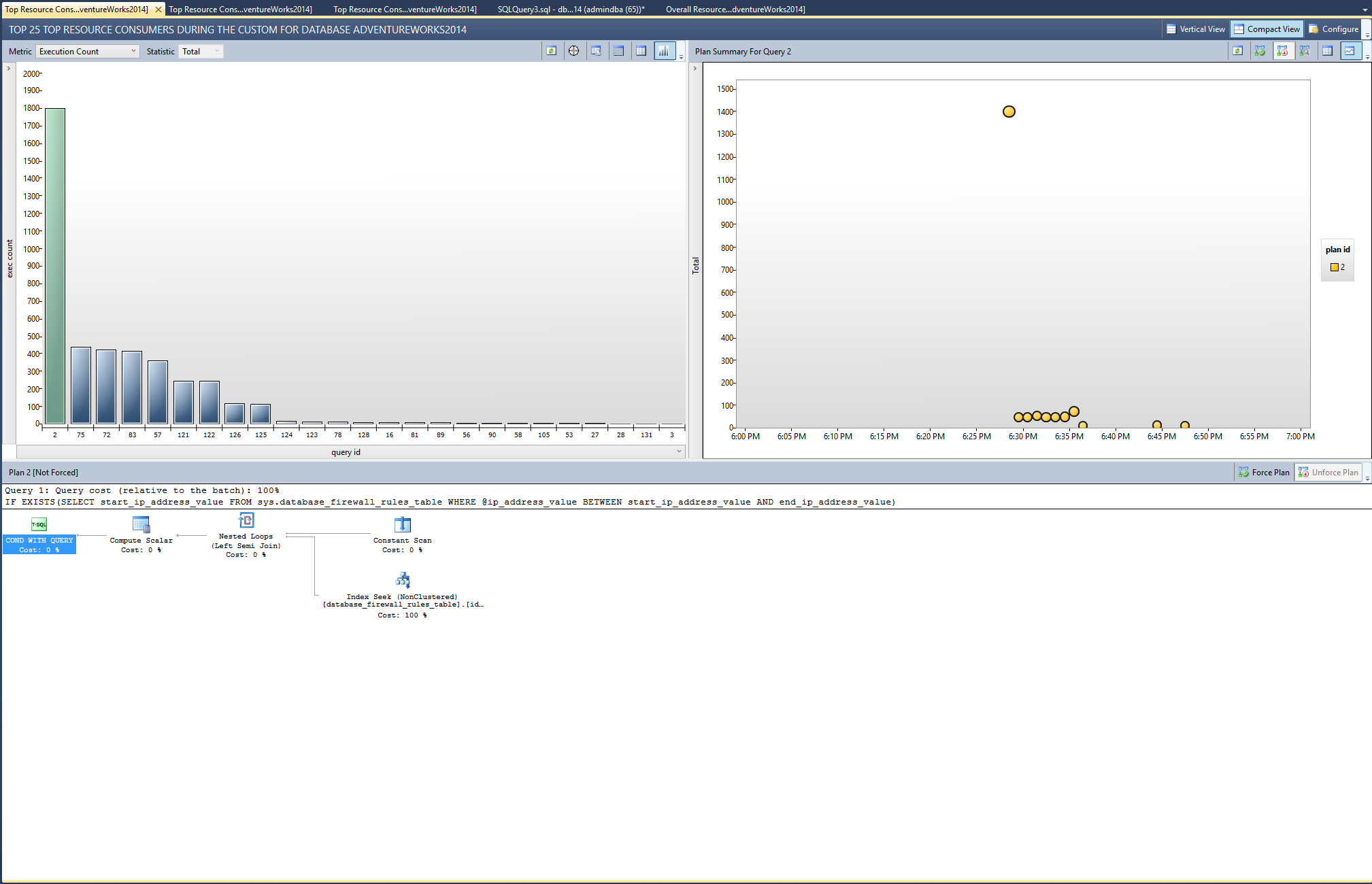
* 1. Overall Resource Consumption

The ‘Overall Resource Consumption’ dashboard from SQL Server Management Studio allows you to view duration, execution count, CPU, Logical Reads, and Memory over a configurable time frame.



You can also click into any of the time slices and drill down into the Overall Resource Consumption dashboard for a particular resource. Below we can see the execution count particular query where the Query Stats is collecting very frequently as we changed the statistics interval to 1 minute from the beginning of the demo.

Frequent collect is desirable when trying to drill down into the performance of a specific workload.



1. References

* **Query Store in SQL Server 2016**<http://channel9.msdn.com/Shows/Data-Exposed/Query-Store-in-SQL-Server-2016>
* **Query Store: A flight data recorder for your database** <http://azure.microsoft.com/blog/2015/06/08/query-store-a-flight-data-recorder-for-your-database/>
* **Preview: Azure SQL Database Query Store**

<http://azure.microsoft.com/en-us/updates/preview-azure-sql-database-query-store/>