How to identify slow running queries in SQL Server

sqlshack.com/how-to-identify-slow-running-queries-in-sql-server

Musab
Umair

May 31,
2017

Overview

Slow running queries are one of the most common problems in every organization dealing with huge amounts of data. And the most challenging problem, in almost all the clients, I work with, is how to find the queries running slow and figuring out what is the actual cause behind the performance problem. Thankfully, the solution, in most cases, is simple.

I always suggest to spend most of the time on figuring out the actual cause behind the problem, not on thinking about the potential solutions which might exist.

Fortunately, there are some tools and techniques which a Developer or DBA should always use (at least) to have a fair idea about the queries running slow.

Before going into the details, I would like to mention here that the tools and techniques I will mention here will be for SQL Developers who do not have expert knowledge of database administration and for Database Administrators who are in the start of their career.

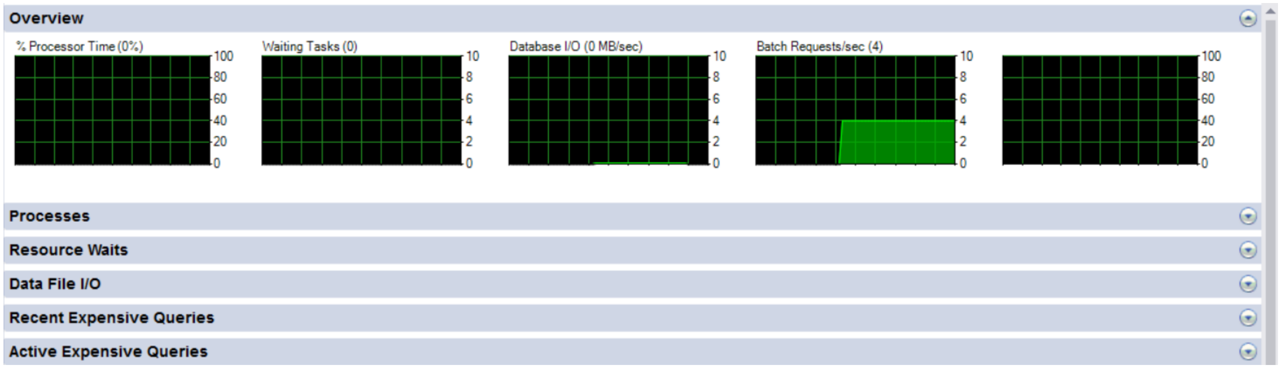
Note: I will be using SQL Server 2016 for my test cases in this article. If you have any prior version, then the Query Store is not available for you but all the other tools will still work.

Tools & techniques

Like every kind of work in the world there are special tools required. The process of identifying slow running queries is no different. In this article, I will only mention tools which are already exist on your system or you can download them free.. I would not say that commercial tools do not help, in fact, in large organizations I would strongly recommend those to save time performing deep dives into each server for highlighting slow running queries. But for the purposes of this article, we want to roll up our sleeves and learn the fundamentals using the existing tools we already have at our disposal.

The first tool which I will mention here is abuilt-in tool for SQL Server Management Studio;**"Activity Monitor".** Youcan view this by Right Clicking on Instance Name in SQL Server Management Studio and selecting "Activity Monitor".

Activity monitor tells you what the current and recent activities are in your SQL Server Instance.

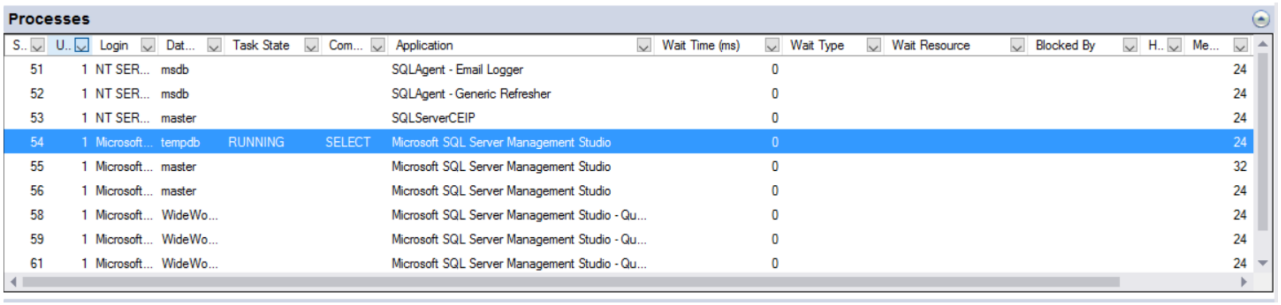


The above screenshot displays an overview window for the Activity Monitor. This screen will show you the graphs for Processor Times, Waiting Tasks, and Batch Requests. Generally, the lower the number of counts the better the performance. In large organizations with huge load there might be huge number of batch requests with high processor times but that does not necessarily indicate a performance problem.

After the overview, you need to focus on Processes which gives you the access to view all the processes running in your instance and having a deeper look for how many processes are waiting, blocking or blocked. This way you can get the idea if you have queries running slow because of any specific wait or if the queries taking time are being blocked by other processes. In this view, you can Right Click on any process and click on *Details* to view the actual TSQL running for that session.

The queries which are being *blocked* are those which are actually suspended because of any other process working on the resources the process depends upon. So, if you find queries which are being blocked by other processes then simply check for the root blocker which is causing all the blocking by looking at the *Blocked By* column. Try to consider just that query, not all the processes which are blocked.

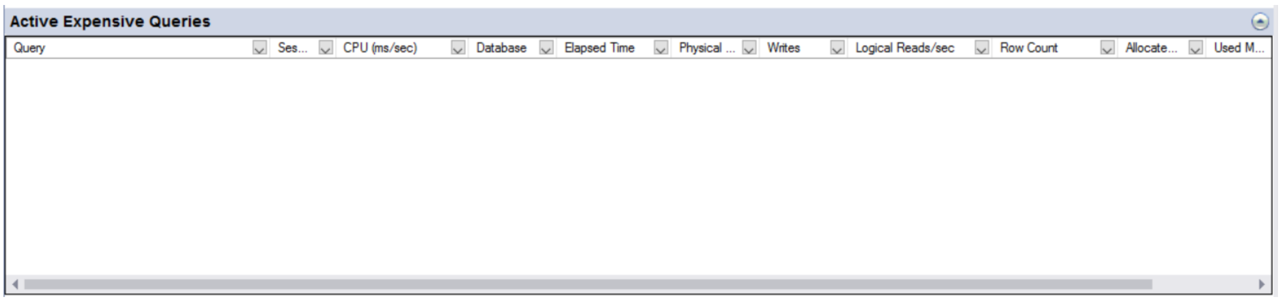
And the queries which are waiting for any specific resource, gives you information about the *Wait Resource* so you can check for the *Wait Type* and try to figure out the solution for that problem. Some of the most common wait stats are listed in a comprehensive SQL Shack's SQL Server wait types section. Go through that in details for further actions.


Processes

S..	U..	Login	Dat...	Task State	Com...	Application	Wait Time (ms)	Wait Type	Wait Resource	Blocked By	H..	Me...
51	1	NT SER...	msdb			SQLAgent - Email Logger	0					24
52	1	NT SER...	msdb			SQLAgent - Generic Refresher	0					24
53	1	NT SER...	master			SQLServerCEIP	0					24
54	1	Microsoft...	tempdb	RUNNING	SELECT	Microsoft SQL Server Management Studio	0					24
55	1	Microsoft...	master			Microsoft SQL Server Management Studio	0					32
56	1	Microsoft...	master			Microsoft SQL Server Management Studio	0					24
58	1	Microsoft...	WideWo...			Microsoft SQL Server Management Studio - Qu...	0					24
59	1	Microsoft...	WideWo...			Microsoft SQL Server Management Studio - Qu...	0					24
61	1	Microsoft...	WideWo...			Microsoft SQL Server Management Studio - Qu...	0					24

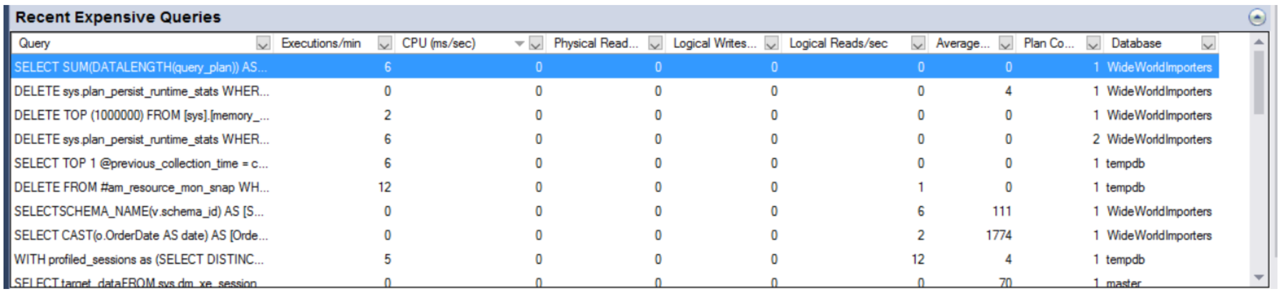
The Active Expensive and Recent Expensive queries will give you information about the queries which have high CPU, Logical Reads or High Elapsed time.

You can go to each section for Current or Recent expensive queries. Sort them by Elapsed time, Logical Read and CPU Time one by one and check the execution plan. In the execution plan you will be able to find out why these expensive queries were taking an inordinate amount of time so that you can take appropriate actions to resolve them. I will let you know how to go through the SQL Server Query Execution Plan later in this article so stay tuned.



Active Expensive Queries

Query	Ses...	CPU (ms/sec)	Database	Elapsed Time	Physical ...	Writes	Logical Reads/sec	Row Count	Allocate...	Used M...



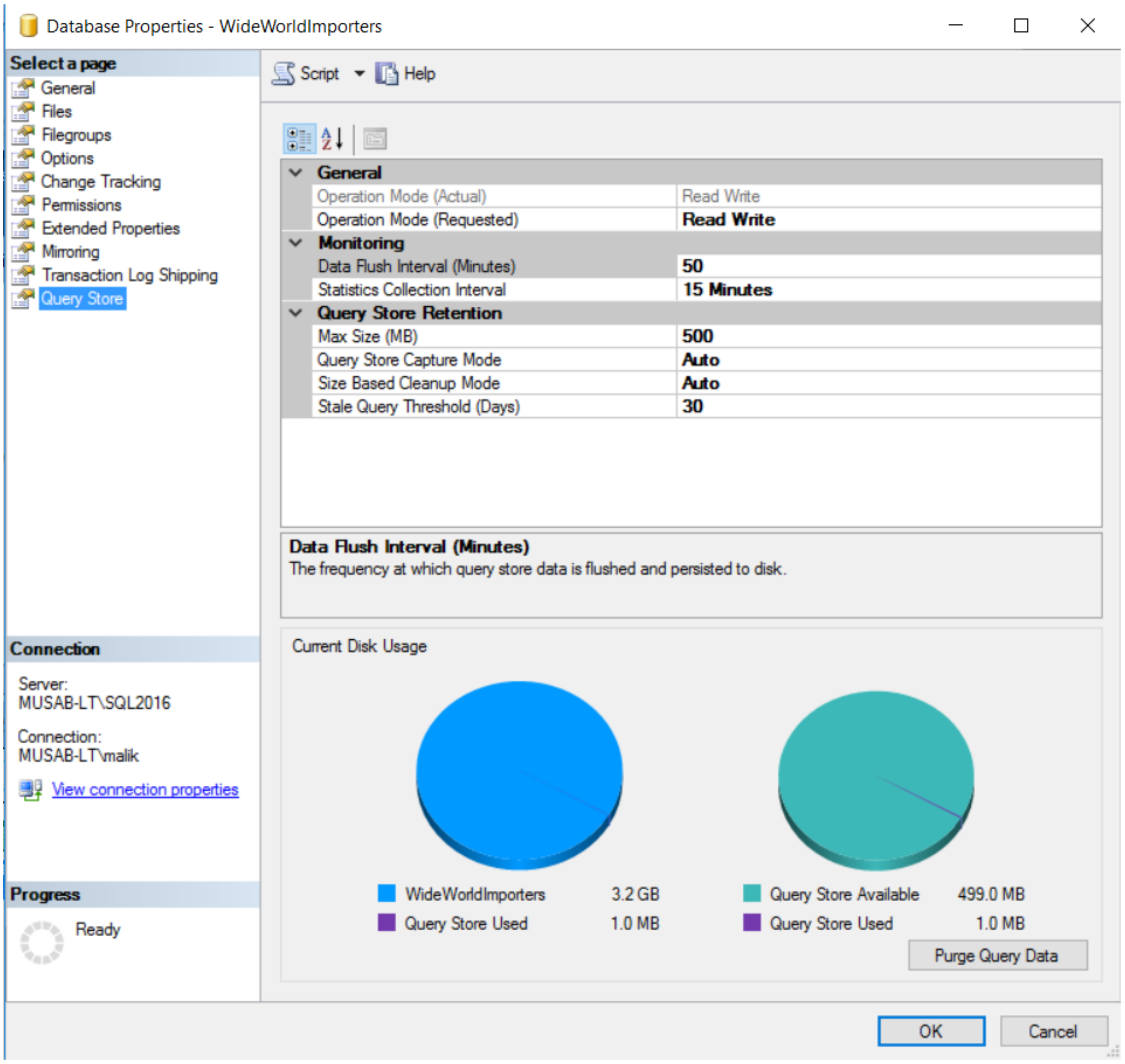
Recent Expensive Queries

Query	Executions/min	CPU (ms/sec)	Physical Read...	Logical Writes...	Logical Reads/sec	Average...	Plan Co...	Database
SELECT SUM(DATALENGTH(query_plan)) AS...	6	0	0	0	0	0	1	WideWorldImporters
DELETE sys.plan_persist_runtime_stats WHER...	0	0	0	0	0	4	1	WideWorldImporters
DELETE TOP (1000000) FROM [sys].[memory_...	2	0	0	0	0	0	1	WideWorldImporters
DELETE sys.plan_persist_runtime_stats WHER...	6	0	0	0	0	0	2	WideWorldImporters
SELECT TOP 1 @previous_collection_time = c...	6	0	0	0	0	0	1	tempdb
DELETE FROM #am_resource_mon_snap WH...	12	0	0	0	1	0	1	tempdb
SELECTSCHEMA_NAME(v.schema_id) AS [S...	0	0	0	0	6	111	1	WideWorldImporters
SELECT CAST(o.OrderDate AS date) AS [Orde...	0	0	0	0	2	1774	1	WideWorldImporters
WITH profiled_sessions as (SELECT DISTINC...	5	0	0	0	12	4	1	tempdb
SELECT target_dataFROM sys.dm_xe_session	0	0	0	0	0	70	1	master

The next tool is the *"Query Store"*. This is helpful and could save your life in situation where you were called in the middle of the night to check why SQL Server was slow 1 hour earlier.

Generally, prior to SQL Server 2016, without any third-party application or custom solutions, you are not able to look at the history of query execution. So, the Query Store provides a great deal of value added functionality in this regard. *Ed Pollack* wrote about Query Store here so do check this article as it's a great resource to deep dive into query store.

If you have SQL Server 2016 or higher so first you need to enable it in your database properties. After enabling the *Query Store,* you will have the properties of your database as shown in the screenshot below:


Database Properties - WideWorldImporters

General	
Operation Mode (Actual)	Read Write
Operation Mode (Requested)	Read Write
Monitoring	
Data Flush Interval (Minutes)	50
Statistics Collection Interval	15 Minutes
Query Store Retention	
Max Size (MB)	500
Query Store Capture Mode	Auto
Size Based Cleanup Mode	Auto
Stale Query Threshold (Days)	30

Data Flush Interval (Minutes)
The frequency at which query store data is flushed and persisted to disk.

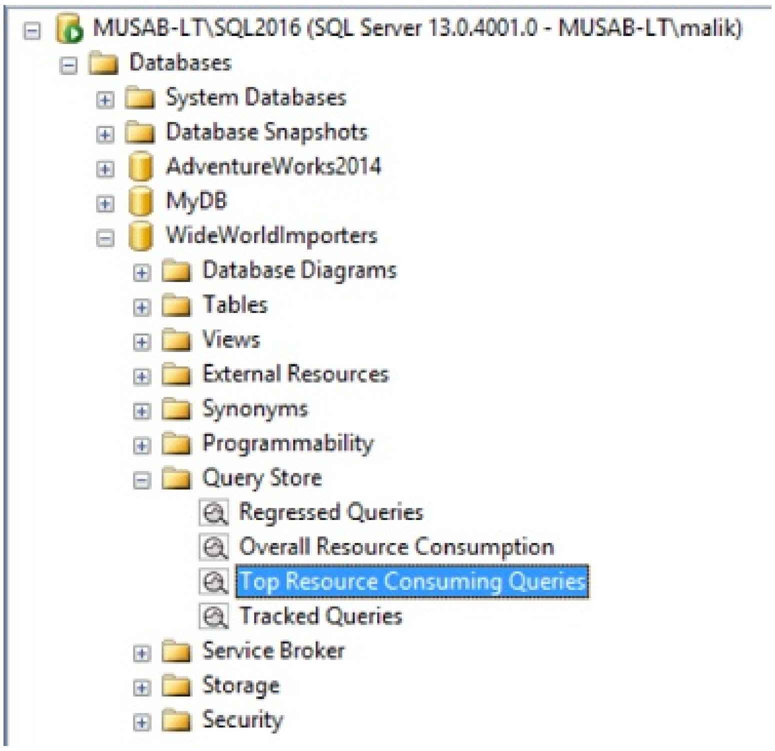
Current Disk Usage

WideWorldImporters	3.2 GB	Query Store Available	499.0 MB
Query Store Used	1.0 MB	Query Store Used	1.0 MB

Connection
Server:
MUSAB-LT\SQL2016
Connection:
MUSAB-LT\malik

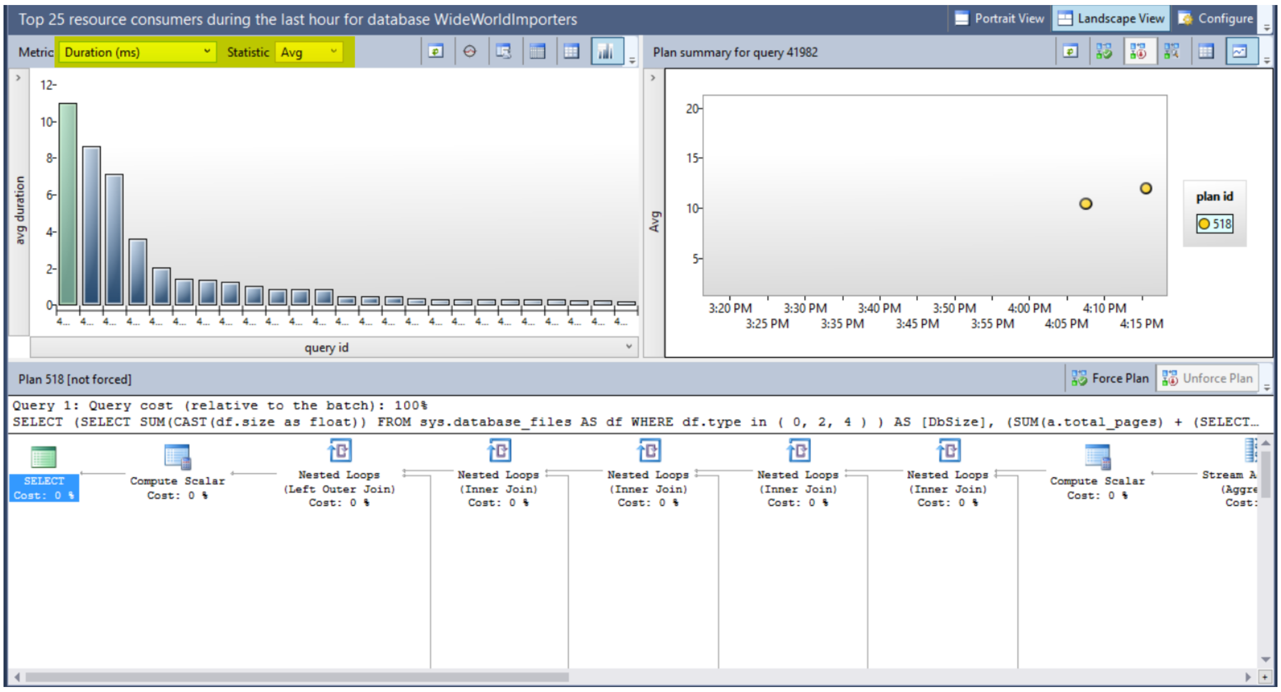
After you have enabled the Query Store you can expand the database objects and go to the “Top Resource Consuming Queries” as shown in the below screenshot:

Note: Give the Query Store a day or two to capture the production load so that you can easily work on it with real load.



```
MUSAB-LT\SQL2016 (SQL Server 13.0.4001.0 - MUSAB-LT\malik)
  Databases
    System Databases
    Database Snapshots
    AdventureWorks2014
    MyDB
    WideWorldImporters
      Database Diagrams
      Tables
      Views
      External Resources
      Synonyms
      Programmability
      Query Store
        Regressed Queries
        Overall Resource Consumption
        Top Resource Consuming Queries
        Tracked Queries
      Service Broker
      Storage
      Security
```

Right Click on *Top Resource Consuming Queries* and select “View Top Resource Consuming Queries”, you will be taken to the window showing these high resource consuming queries. You can customize the view by selecting an appropriate “metric” like Duration, CPU Time, Logical Read or Memory Consumption. The second thing you need to change is “Statistic”. You can change it to Min, Max or Avg. I would recommend to use Average Statistic with all metrics mentioned above to get queries.



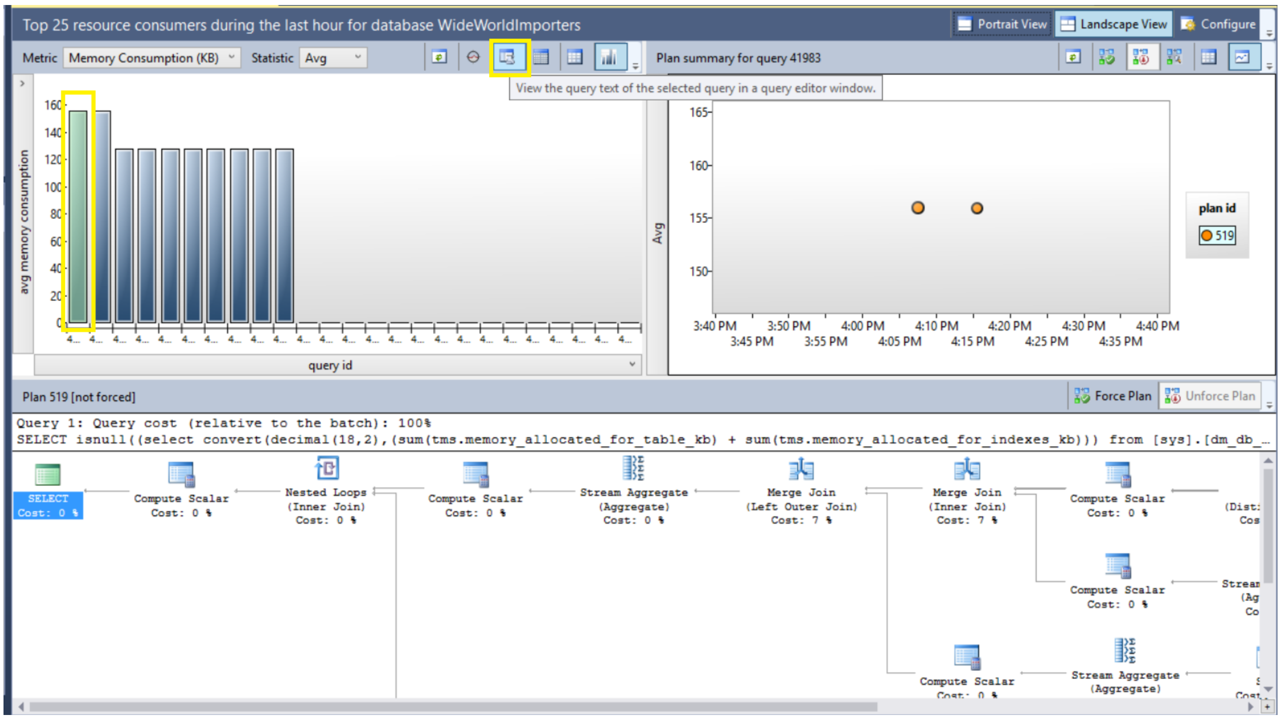
Top 25 resource consumers during the last hour for database WideWorldImporters
Portrait View
Landscape View
Configure
Metric Duration (ms)
Statistic Avg
Plan summary for query 41982
avg duration
query id
Avg
plan id
518
Plan 518 [not forced]
Force Plan
Unforce Plan

```
Query 1: Query cost (relative to the batch): 100%
SELECT (SELECT SUM(CAST(df.size as float)) FROM sys.database_files AS df WHERE df.type in ( 0, 2, 4 ) ) AS [DbSize], (SUM(a.total_pages) + (SELECT…
```

SELECT Cost: 0 %
Compute Scalar Cost: 0 %
Nested Loops (Left Outer Join) Cost: 0 %
Nested Loops (Inner Join) Cost: 0 %
Nested Loops (Inner Join) Cost: 0 %
Nested Loops (Inner Join) Cost: 0 %
Nested Loops (Inner Join) Cost: 0 %
Compute Scalar Cost: 0 %
Stream A (Aggre Cost:

The next step is to highlight the queries which are consuming high resources. After highlighting the graph value in the left-hand side window (as highlighted in the screenshot below) you will get the query execution plan in the bottom window.

You can click on the mentioned below highlighted button in the Query Store window to get the actual Query Text for further analysis.


Top 25 resource consumers during the last hour for database WideWorldImporters
Portrait View
Landscape View
Configure
Metric
Memory Consumption (KB)
Statistic
Avg
View the query text of the selected query in a query editor window.
Plan summary for query 41983
avg memory consumption
query id
Avg
plan id
519
Plan 519 [not forced]
Force Plan
Unforce Plan

```
Query 1: Query cost (relative to the batch): 100%
SELECT isnull((select convert(decimal(18,2),(sum(tms.memory_allocated_for_table_kb) + sum(tms.memory_allocated_for_indexes_kb))) from [sys].[dm_db_...
```

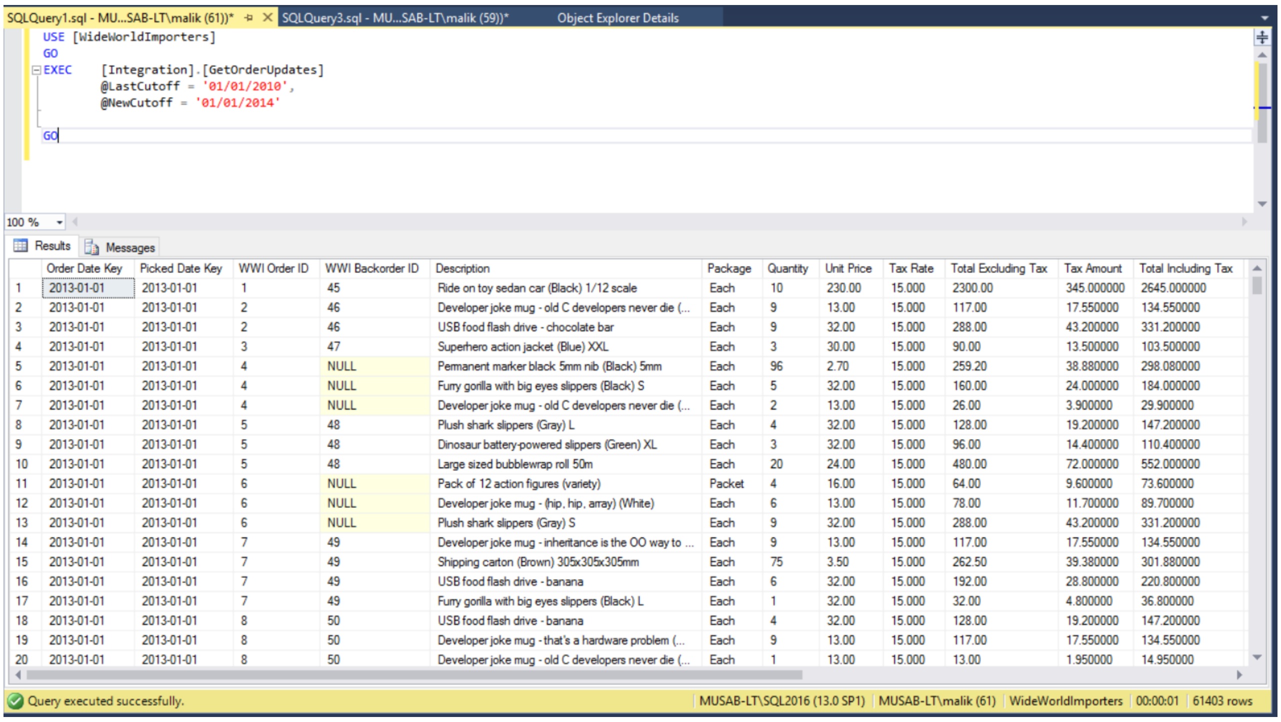
SELECT Cost: 0 %
Compute Scalar Cost: 0 %
Nested Loops (Inner Join) Cost: 0 %
Compute Scalar Cost: 0 %
Stream Aggregate (Aggregate) Cost: 0 %
Merge Join (Left Outer Join) Cost: 7 %
Merge Join (Inner Join) Cost: 7 %
Compute Scalar Cost: 0 %
Compute Scalar Cost: 0 %
Compute Scalar
Stream Aggregate (Aggregate)

So, as of now, you have multiple ways to get High Resource usage queries. Now we will see how we can check why the queries are running slow and which part of the query needs to be fixed (if required).

So, here I will take the example of a query used in Microsoft sample database **"WideWorldImporters"**. The TSQL executes a stored procedure **"[Integration].[GetOrderUpdates]"**.

The call of this Stored Procedure takes around a second and I will not be optimizing it. This is just to give you an example that how may know, how this *second* was spent. We also want to know which part of the query is taking most time as well as on which table we must focus.

Below is the stored procedure call and results.



```sql
USE [WideWorldImporters]
GO
EXEC    [Integration].[GetOrderUpdates]
        @LastCutoff = '01/01/2010',
        @NewCutoff = '01/01/2014'

GO
```

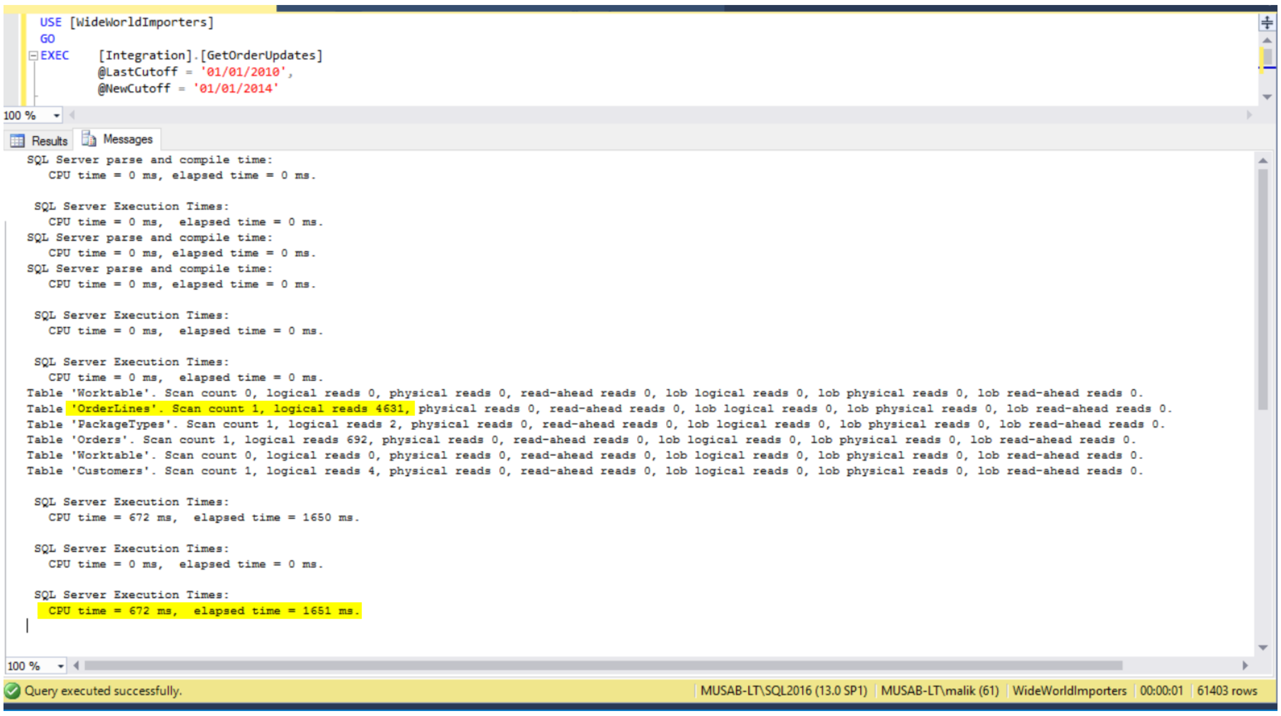
	Order Date Key	Picked Date Key	WWI Order ID	WWI Backorder ID	Description	Package	Quantity	Unit Price	Tax Rate	Total Excluding Tax	Tax Amount	Total Including Tax
1	2013-01-01	2013-01-01	1	45	Ride on toy sedan car (Black) 1/12 scale	Each	10	230.00	15.000	2300.00	345.000000	2645.000000
2	2013-01-01	2013-01-01	2	46	Developer joke mug - old C developers never die (...	Each	9	13.00	15.000	117.00	17.550000	134.550000
3	2013-01-01	2013-01-01	2	46	USB food flash drive - chocolate bar	Each	9	32.00	15.000	288.00	43.200000	331.200000
4	2013-01-01	2013-01-01	3	47	Superhero action jacket (Blue) XXL	Each	3	30.00	15.000	90.00	13.500000	103.500000
5	2013-01-01	2013-01-01	4	NULL	Permanent marker black 5mm nib (Black) 5mm	Each	96	2.70	15.000	259.20	38.880000	298.080000
6	2013-01-01	2013-01-01	4	NULL	Furry gorilla with big eyes slippers (Black) S	Each	5	32.00	15.000	160.00	24.000000	184.000000
7	2013-01-01	2013-01-01	4	NULL	Developer joke mug - old C developers never die (...	Each	2	13.00	15.000	26.00	3.900000	29.900000
8	2013-01-01	2013-01-01	5	48	Plush shark slippers (Gray) L	Each	4	32.00	15.000	128.00	19.200000	147.200000
9	2013-01-01	2013-01-01	5	48	Dinosaur battery-powered slippers (Green) XL	Each	3	32.00	15.000	96.00	14.400000	110.400000
10	2013-01-01	2013-01-01	5	48	Large sized bubblewrap roll 50m	Each	20	24.00	15.000	480.00	72.000000	552.000000
11	2013-01-01	2013-01-01	6	NULL	Pack of 12 action figures (variety)	Packet	4	16.00	15.000	64.00	9.600000	73.600000
12	2013-01-01	2013-01-01	6	NULL	Developer joke mug - (hip, hip, array) (White)	Each	6	13.00	15.000	78.00	11.700000	89.700000
13	2013-01-01	2013-01-01	6	NULL	Plush shark slippers (Gray) S	Each	9	32.00	15.000	288.00	43.200000	331.200000
14	2013-01-01	2013-01-01	7	49	Developer joke mug - inheritance is the OO way to ...	Each	9	13.00	15.000	117.00	17.550000	134.550000
15	2013-01-01	2013-01-01	7	49	Shipping carton (Brown) 305x305x305mm	Each	75	3.50	15.000	262.50	39.380000	301.880000
16	2013-01-01	2013-01-01	7	49	USB food flash drive - banana	Each	6	32.00	15.000	192.00	28.800000	220.800000
17	2013-01-01	2013-01-01	7	49	Furry gorilla with big eyes slippers (Black) L	Each	1	32.00	15.000	32.00	4.800000	36.800000
18	2013-01-01	2013-01-01	8	50	USB food flash drive - banana	Each	4	32.00	15.000	128.00	19.200000	147.200000
19	2013-01-01	2013-01-01	8	50	Developer joke mug - that's a hardware problem (...	Each	9	13.00	15.000	117.00	17.550000	134.550000
20	2013-01-01	2013-01-01	8	50	Developer joke mug - old C developers never die (...	Each	1	13.00	15.000	13.00	1.950000	14.950000

Query executed successfully. | MUSAB-LT\SQL2016 (13.0 SP1) | MUSAB-LT\malik (61) | WideWorldImporters | 00:00:01 | 61403 rows

So now we have the call and we will dig deeper into this.

First, we need to enable the Query Statistics for this session. We will enable CPU and IO statistics for this query session by issuing the TSQL "*SET STATISTICS TIME, IO ON*".

After executing the mentioned above TSQL for enabling *Statistics*, we will get IO for each table and Total CPU cost for the queries running inside the stored procedure in *Messages* tab as shown in below screenshot.



```sql
USE [WideWorldImporters]
GO
EXEC    [Integration].[GetOrderUpdates]
        @LastCutoff = '01/01/2010',
        @NewCutoff = '01/01/2014'
```

```
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 0 ms.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 0 ms.

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 0 ms.
Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.
Table 'OrderLines'. Scan count 1, logical reads 4631, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.
Table 'PackageTypes'. Scan count 1, logical reads 2, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.
Table 'Orders'. Scan count 1, logical reads 692, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.
Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.
Table 'Customers'. Scan count 1, logical reads 4, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

 SQL Server Execution Times:
   CPU time = 672 ms,  elapsed time = 1650 ms.

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 0 ms.

 SQL Server Execution Times:
   CPU time = 672 ms,  elapsed time = 1651 ms.
```

Query executed successfully. | MUSAB-LT\SQL2016 (13.0 SP1) | MUSAB-LT\malik (61) | WideWorldImporters | 00:00:01 | 61403 rows

In the above screenshot, we can see that the most IO is taken by *OrderLines* table and there is only one query which is executing inside the Stored Procedure which is taking *672 ms CPU time (1650 ms Elapsed time).*

Note: There might be multiple queries running inside a stored procedure so keep in mind that Statistics will give you time for each query as well as the total for all the queries at the end. So, in case of Stored Procedures for total CPU time only consider the last CPU time and for each query consider its CPU time only and exclude the last CPU time as it's only a total for all.

As of now, we know that the *OrderLines* table is taking most of the Logical Reads.

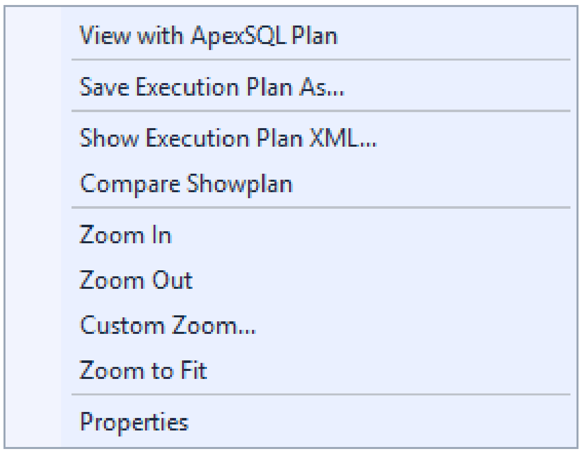
Next, we will enable the *Actual Execution Plan* for the query by clicking the icon (Ctrl +M) in SQL Server Management Studio and will try to answer the question of why this table was taking this IO and which component of the execution plan is taking most of the time.

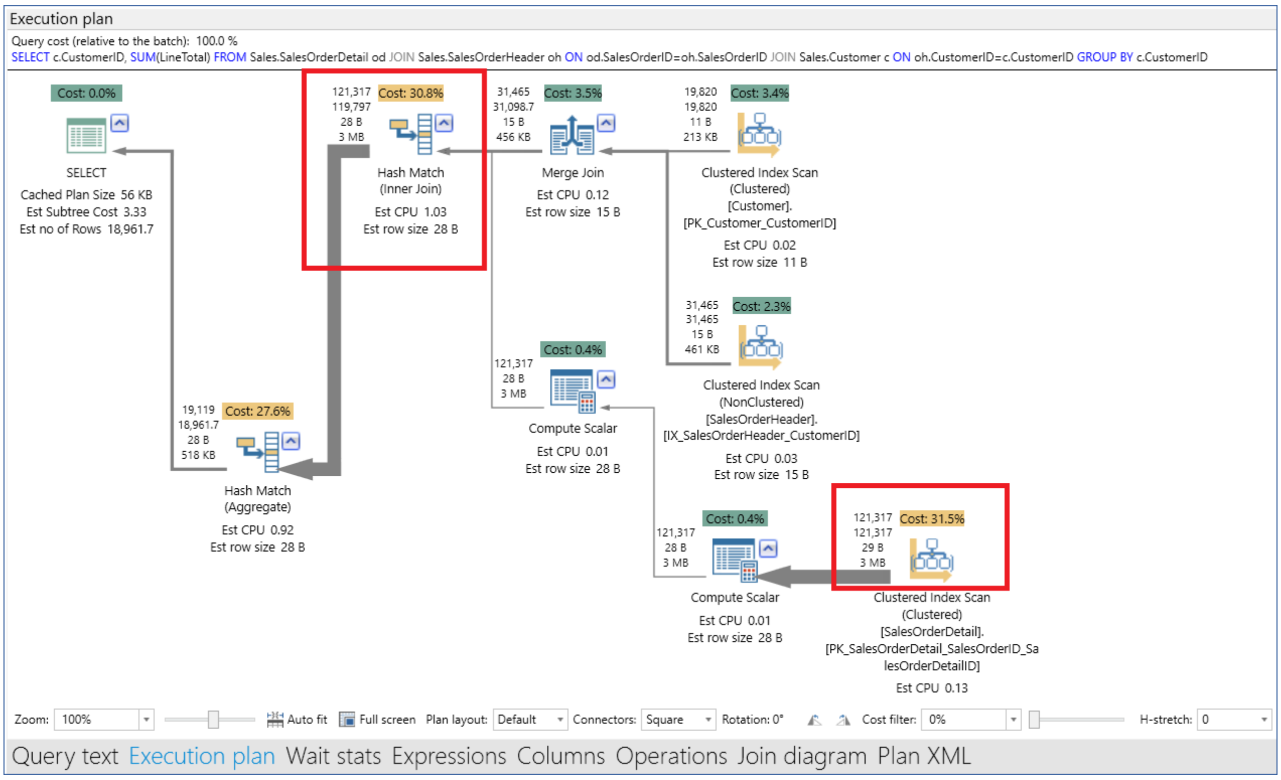
After including the Actual Execution Plan, we will re-execute the query and view the execution plan.

Though, we can obtain much detailed information about Query Execution Plan inside SQL Server Management Studio but there is another great free tool out on web which can be used to explore Query Execution Plan in much more intuitive way that is ApexSQL Plan.

After installing this tool, you might need to restart the SQL Server Management Studio so install it and then re-execute the query to get an Execution Plan. A screenshot tour for this tool is provided here. After executing the query, Right Click on the Execution Plan and you will have the option of "*View with ApexSQL Plan*".

After viewing the Execution Plan in ApexSQL Plan, you can see the highlighted items in mentioned below screenshot which will be opened in the ApexSQL Plan.




Execution plan

Query cost (relative to the batch): 100.0 %

```
SELECT c.CustomerID, SUM(LineTotal) FROM Sales.SalesOrderDetail od JOIN Sales.SalesOrderHeader oh ON od.SalesOrderID=oh.SalesOrderID JOIN Sales.Customer c ON oh.CustomerID=c.CustomerID GROUP BY c.CustomerID
```

Zoom: 100% Plan layout: Default Connectors: Square Rotation: 0° Cost filter: 0% H-stretch: 0

Couple of points and tips here to look at

- If the table is using Key Lookup, then try to remove that lookup by adding those columns in the index which is used by the table.
- If number of rows returned by the query is way off as compared to number of rows returned by table operators (as highlighted in the lower section of the query plan), then try to re-write the query which filters the data using more columns to reduce the number of rows.
- If estimated rows and actual rows have huge difference, then try to update the Statistics of under lying tables.
- If there are missing indexes indicated by the query, then try to evaluate the index and if that index is helping you query then add it to the corresponding table.

Finally, if your queries are performing well in single execution and only have problem while running in production load so you can easily fake the production load for a single query by using Adam Machanic's SQL Stress Tool which is available to download from the link here. Then you can capture that slow running query from the techniques mentioned above and tune it up accordingly.

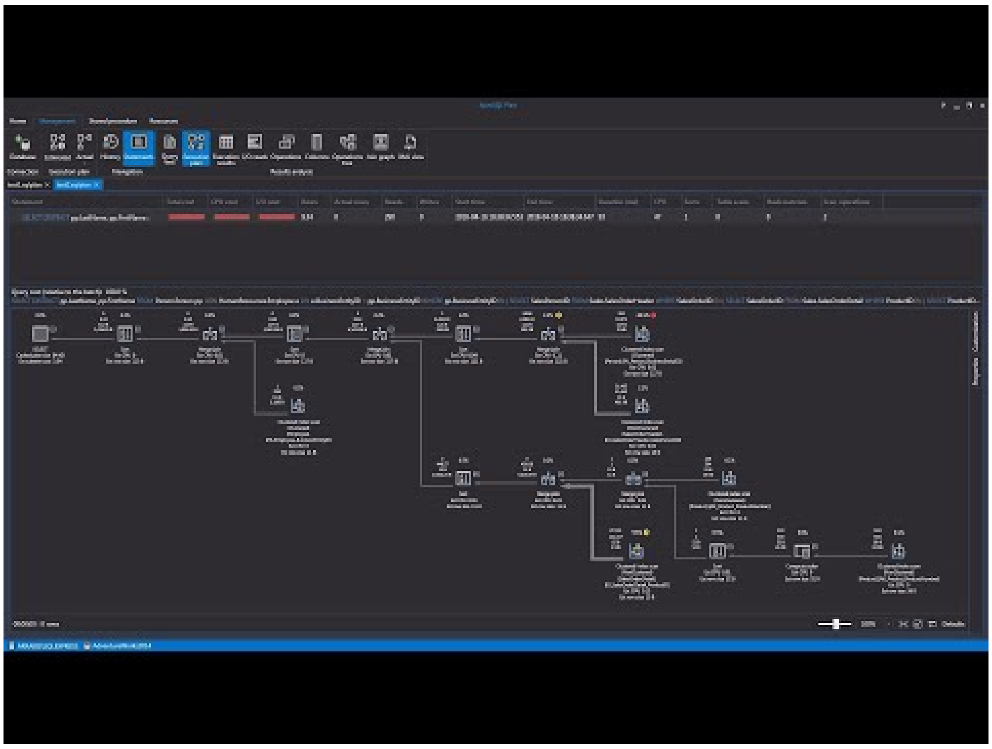
Summary

Mentioned above tools, techniques and tips are the most common solutions for troubleshooting slow running queries. These are common, generic and mostly queries do benefit from them.

Keep in mind there is no pet rule for performance optimization and troubleshooting the slow running queries and this topic is way too big to cover in some articles so I have tried to sum up in a single article some of the most important and common tools and techniques so that the basic trouble shooting should be made easy.

See more

Check out ApexSQL Plan, to view SQL execution plans for free, including comparing plans, stored procedure performance profiling, missing index details, lazy profiling, wait times, plan execution history and more



https://youtu.be/P01_Zd11HX0

FREE SQL query plan analysis and optimization ApexSQL

Musab Umair

Musab is one of the two SQL Server Featured Speakers of Professional Association for SQL Server in SQLPASS Pakistan General Conference.

He has 9+ Years of Database Development & Administration experience with Medical Billing, Startup & Financial Companies. He is a Microsoft Certified Expert for Data Platform (SQL Server 2012/2014).

Currently, performing duties as Senior SQL Server Consultant at multiple Saudi Private and Governmental Organizations.

View all posts by Musab Umair