DMV - Dynamic Management Views

DMV to gather the query performance information, and performance issues generally.

We have a variety of DMVs. So, what are DMVs?

Well, Dynamic Management Views are system views; that's it.

They start off with sys.dm_, so dynamic management underscore.

Then, we have a word which is the functional area like exec and db and tran, and then an underscore and what's the actual view is.

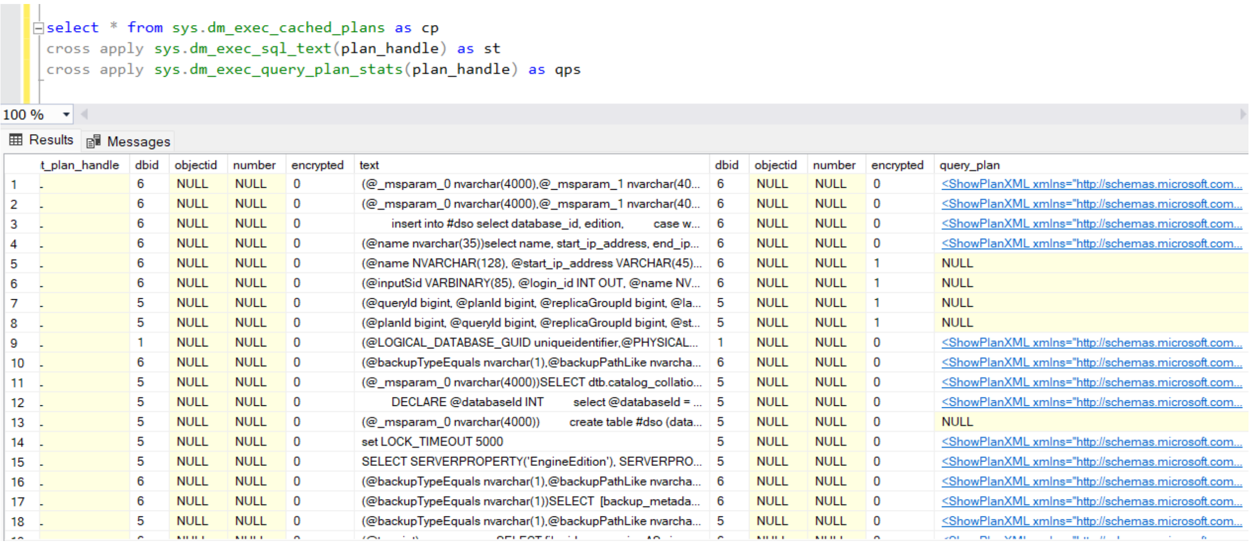
Requirement	**DMV's**
Find top N queries ranked by average CPU time The most cumulative CPU queries	**sys.dm_exec_query_stats**
Find top N stored procedure ranked by average CPU time	**sys.dm_exec_procedure_stats**
Find still running sessions (right now) Concurrent requests right now (status='RUNNABLE') **In Azure SQL Databases, it relates only to current databases and background tasks, not other databases.**	**sys.dm_exec_requests**
Curren active sessions right now	**sys.dm_exec_connections**
Identity data and log I/O usage	**sys.dm_db_resources_stats (Azure SQL Database)** **sys.resource_stats (all databases – must be in "master" database in Azure SQL database)** **sys.server_resource_stats (Managed Instances)** **sys.elastic_pool_resource_stats (elastic pool databases)**
Find long running transactions	**sys.dm_tran_active_transactions**
Retrieve cached plans	**sys.dm_exec_cached_plans** **sys.dm_exec_sql_text** **sys.dm_exec_query_plan_stats**

-- Retrive the last execution plans --

select * from sys.dm_exec_cached_plans as cp

cross apply sys.dm_exec_sql_text(plan_handle) as st

cross apply sys.dm_exec_query_plan_stats(plan_handle) as qps



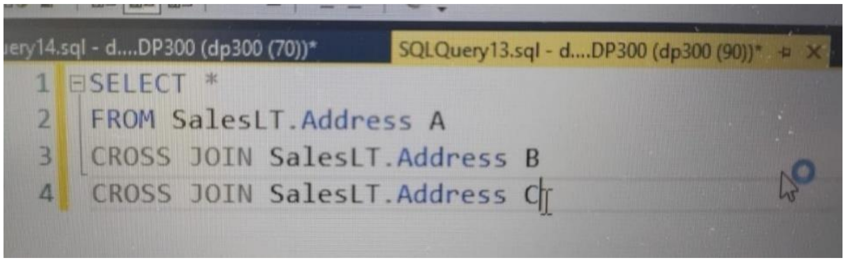
```
select * from sys.dm_exec_cached_plans as cp
cross apply sys.dm_exec_sql_text(plan_handle) as st
cross apply sys.dm_exec_query_plan_stats(plan_handle) as qps
```

	t_plan_handle	dbid	objectid	number	encrypted	text	dbid	objectid	number	encrypted	query_plan
1	.	6	NULL	NULL	0	(@_msparam_0 nvarchar(4000),@_msparam_1 nvarchar(40...	6	NULL	NULL	0	<ShowPlanXML xmlns="http://schemas.microsoft.com...
2	.	6	NULL	NULL	0	(@_msparam_0 nvarchar(4000),@_msparam_1 nvarchar(40...	6	NULL	NULL	0	<ShowPlanXML xmlns="http://schemas.microsoft.com...
3	.	6	NULL	NULL	0	insert into #dso select database_id, edition, case w...	6	NULL	NULL	0	<ShowPlanXML xmlns="http://schemas.microsoft.com...
4	.	6	NULL	NULL	0	(@name nvarchar(35))select name, start_ip_address, end_ip...	6	NULL	NULL	0	<ShowPlanXML xmlns="http://schemas.microsoft.com...
5	.	6	NULL	NULL	0	(@name NVARCHAR(128), @start_ip_address VARCHAR(45)...	6	NULL	NULL	1	NULL
6	.	6	NULL	NULL	0	(@inputSid VARBINARY(85), @login_id INT OUT, @name NV...	6	NULL	NULL	1	NULL
7	.	5	NULL	NULL	0	(@queryId bigint, @planId bigint, @replicaGroupId bigint, @la...	5	NULL	NULL	1	NULL
8	.	5	NULL	NULL	0	(@planId bigint, @queryId bigint, @replicaGroupId bigint, @st...	5	NULL	NULL	1	NULL
9	.	1	NULL	NULL	0	(@LOGICAL_DATABASE_GUID uniqueidentifier,@PHYSICAL...	1	NULL	NULL	0	<ShowPlanXML xmlns="http://schemas.microsoft.com...
10	.	6	NULL	NULL	0	(@backupTypeEquals nvarchar(1),@backupPathLike nvarcha...	6	NULL	NULL	0	<ShowPlanXML xmlns="http://schemas.microsoft.com...
11	.	5	NULL	NULL	0	(@_msparam_0 nvarchar(4000))SELECT dtb.catalog_collatio...	5	NULL	NULL	0	<ShowPlanXML xmlns="http://schemas.microsoft.com...
12	.	5	NULL	NULL	0	DECLARE @databaseId INT select @databaseId = ...	5	NULL	NULL	0	<ShowPlanXML xmlns="http://schemas.microsoft.com...
13	.	5	NULL	NULL	0	(@_msparam_0 nvarchar(4000)) create table #dso (data...	5	NULL	NULL	0	NULL
14	.	5	NULL	NULL	0	set LOCK_TIMEOUT 5000	5	NULL	NULL	0	<ShowPlanXML xmlns="http://schemas.microsoft.com...
15	.	5	NULL	NULL	0	SELECT SERVERPROPERTY('EngineEdition'), SERVERPRO...	5	NULL	NULL	0	<ShowPlanXML xmlns="http://schemas.microsoft.com...
16	.	6	NULL	NULL	0	(@backupTypeEquals nvarchar(1),@backupPathLike nvarcha...	6	NULL	NULL	0	<ShowPlanXML xmlns="http://schemas.microsoft.com...
17	.	6	NULL	NULL	0	(@backupTypeEquals nvarchar(1))SELECT [backup_metada...	6	NULL	NULL	0	<ShowPlanXML xmlns="http://schemas.microsoft.com...
18	.	5	NULL	NULL	0	(@backupTypeEquals nvarchar(1),@backupPathLike nvarcha...	5	NULL	NULL	0	<ShowPlanXML xmlns="http://schemas.microsoft.com...

Now, if you wish to do the DP-300 certification, you will need to memorize a fair number of these in terms of what they can be used for. You don't necessarily need to know the exact columns or the exact output, but you need to know roughly what they are used for.

we're going to take a few of these DMV and have a look at what you can do with them, but the important thing is what is in green, the DMV.

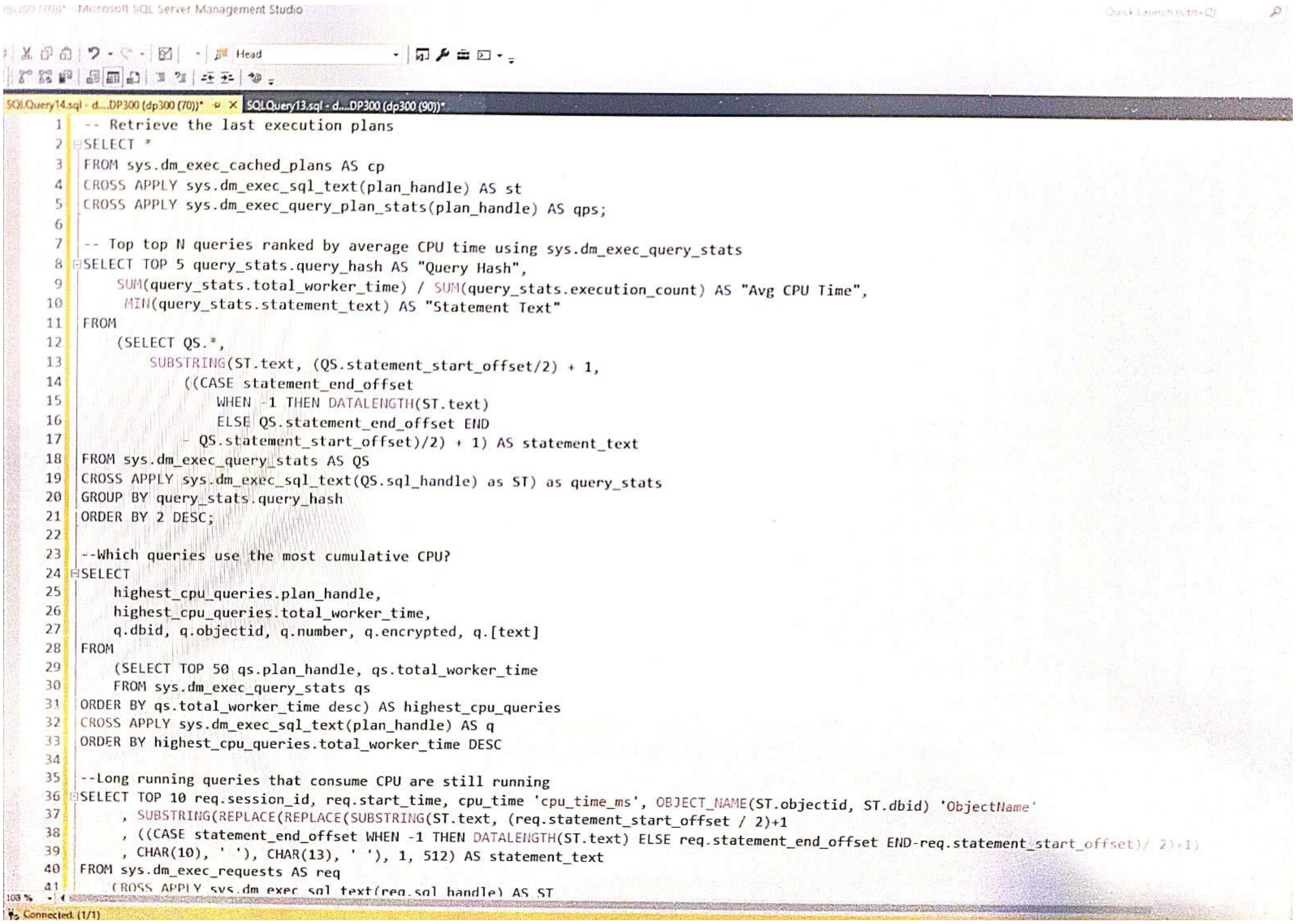
Now, before I do anything with this, I'm going to run this query, the separate window. All is this table, which has 450 roles multiplied by 450 roles multiplied by 450 roles. So, you can see it's going to take a long time, so I'm just going to leave that running.



uery14.sql - d....DP300 (dp300 (70))*
SQLQuery13.sql - d....DP300 (dp300 (90))*

```
SELECT *
FROM SalesLT.Address A
CROSS JOIN SalesLT.Address B
CROSS JOIN SalesLT.Address C
```

So, let's have a look at the first of our DMV


dp300 (70))* - Microsoft SQL Server Management Studio
Head
SQLQuery14.sql - d....DP300 (dp300 (70))*
SQLQuery13.sql - d....DP300 (dp300 (90))*
103 %
Connected. (1/1)

```sql
-- Retrieve the last execution plans
SELECT *
FROM sys.dm_exec_cached_plans AS cp
CROSS APPLY sys.dm_exec_sql_text(plan_handle) AS st
CROSS APPLY sys.dm_exec_query_plan_stats(plan_handle) AS qps;

-- Top top N queries ranked by average CPU time using sys.dm_exec_query_stats
SELECT TOP 5 query_stats.query_hash AS "Query Hash",
    SUM(query_stats.total_worker_time) / SUM(query_stats.execution_count) AS "Avg CPU Time",
     MIN(query_stats.statement_text) AS "Statement Text"
FROM
    (SELECT QS.*,
        SUBSTRING(ST.text, (QS.statement_start_offset/2) + 1,
            ((CASE statement_end_offset
                WHEN -1 THEN DATALENGTH(ST.text)
                ELSE QS.statement_end_offset END
              - QS.statement_start_offset)/2) + 1) AS statement_text
FROM sys.dm_exec_query_stats AS QS
CROSS APPLY sys.dm_exec_sql_text(QS.sql_handle) as ST) as query_stats
GROUP BY query_stats.query_hash
ORDER BY 2 DESC;

--Which queries use the most cumulative CPU?
SELECT
    highest_cpu_queries.plan_handle,
    highest_cpu_queries.total_worker_time,
    q.dbid, q.objectid, q.number, q.encrypted, q.[text]
FROM
    (SELECT TOP 50 qs.plan_handle, qs.total_worker_time
    FROM sys.dm_exec_query_stats qs
ORDER BY qs.total_worker_time desc) AS highest_cpu_queries
CROSS APPLY sys.dm_exec_sql_text(plan_handle) AS q
ORDER BY highest_cpu_queries.total_worker_time DESC

--Long running queries that consume CPU are still running
SELECT TOP 10 req.session_id, req.start_time, cpu_time 'cpu_time_ms', OBJECT_NAME(ST.objectid, ST.dbid) 'ObjectName'
    , SUBSTRING(REPLACE(REPLACE(SUBSTRING(ST.text, (req.statement_start_offset / 2)+1
    , ((CASE statement_end_offset WHEN -1 THEN DATALENGTH(ST.text) ELSE req.statement_end_offset END-req.statement_start_offset)/ 2)+1)
    , CHAR(10), ' '), CHAR(13), ' '), 1, 512) AS statement_text
FROM sys.dm_exec_requests AS req
    CROSS APPLY sys.dm_exec_sql_text(req.sql_handle) AS ST
```

and we've got dm_exec_cached_plans. So, this can retrieve the last execution plans, which are in the cache.

So, you can see, we have got things like the plan handle and other information. Now this plan handle is used in two separate DMVs, dm_exec_sql_text and dm_exec_query_plan stats.

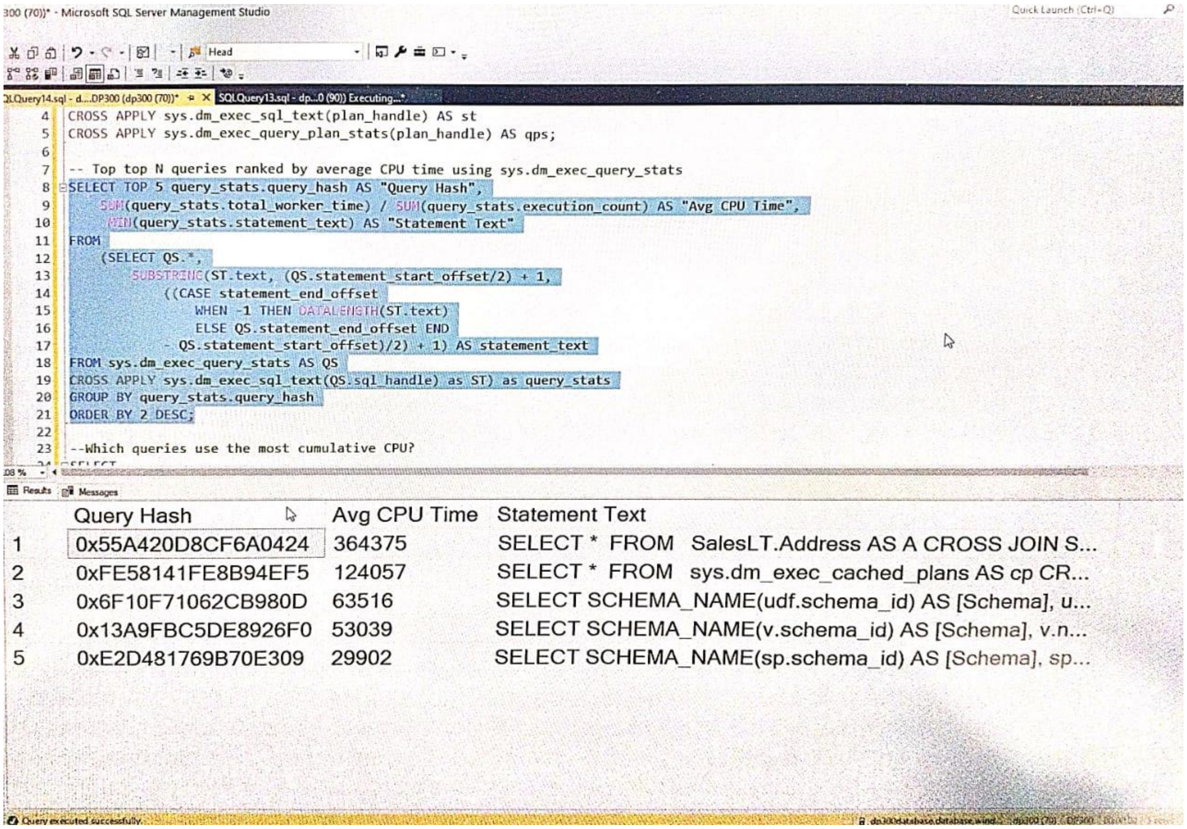
So, if I use this using a CROSS APPLY, you don't need to worry why it's CROSS APPLY. The reason for that is because we have a different plan handle for each one of these roles.

So, it's not a LEFT JOIN or RIGHT JOIN or INNER JOIN, it’s an appy. What we have is the text from the query and the plan in XML format. Now you'll see that it's underlined, so if I click on any of these plans, you can see that we've got a similar sort of execution plan that we have seen many a time.

www.linkedin.com/in/ragasudha-m

So, these three work hand in hand together with each other. So, we have the actual plans, and then we extract the SQL text and the query plan stats. This next one is about having a look at the top end, so the top five queries in this case, right by average CPU, computer time.

So, you can see, these are our biggest users of memory and surprise, surprise if I just copy and paste that, you can see that our biggest user of memory is one that we've used from a previous query, which is a CROSS JOIN.


300 (70))* - Microsoft SQL Server Management Studio
Quick Launch (Ctrl+Q)
Head
QLQuery14.sql - d....DP300 (dp300 (70))*
SQLQuery13.sql - dp...0 (90)) Executing...*
08 %
Results
Messages
dp300database.database.wind... dp300 (70) DP300

```sql
CROSS APPLY sys.dm_exec_sql_text(plan_handle) AS st
CROSS APPLY sys.dm_exec_query_plan_stats(plan_handle) AS qps;

-- Top top N queries ranked by average CPU time using sys.dm_exec_query_stats
SELECT TOP 5 query_stats.query_hash AS "Query Hash",
    SUM(query_stats.total_worker_time) / SUM(query_stats.execution_count) AS "Avg CPU Time",
     MIN(query_stats.statement_text) AS "Statement Text"
FROM
    (SELECT QS.*,
        SUBSTRING(ST.text, (QS.statement_start_offset/2) + 1,
            ((CASE statement_end_offset
                WHEN -1 THEN DATALENGTH(ST.text)
                ELSE QS.statement_end_offset END
            - QS.statement_start_offset)/2) + 1) AS statement_text
FROM sys.dm_exec_query_stats AS QS
CROSS APPLY sys.dm_exec_sql_text(QS.sql_handle) as ST) as query_stats
GROUP BY query_stats.query_hash
ORDER BY 2 DESC;

--Which queries use the most cumulative CPU?
```

	Query Hash	Avg CPU Time	Statement Text
1	0x55A420D8CF6A0424	364375	SELECT * FROM SalesLT.Address AS A CROSS JOIN S...
2	0xFE58141FE8B94EF5	124057	SELECT * FROM sys.dm_exec_cached_plans AS cp CR...
3	0x6F10F71062CB980D	63516	SELECT SCHEMA_NAME(udf.schema_id) AS [Schema], u...
4	0x13A9FBC5DE8926F0	53039	SELECT SCHEMA_NAME(v.schema_id) AS [Schema], v.n...
5	0xE2D481769B70E309	29902	SELECT SCHEMA_NAME(sp.schema_id) AS [Schema], sp...

Query executed successfully.

Now it's very similar to the one we're using here, except this is even more extreme and no doubt we'll be number one there. So, you can see which queries are running the longest.

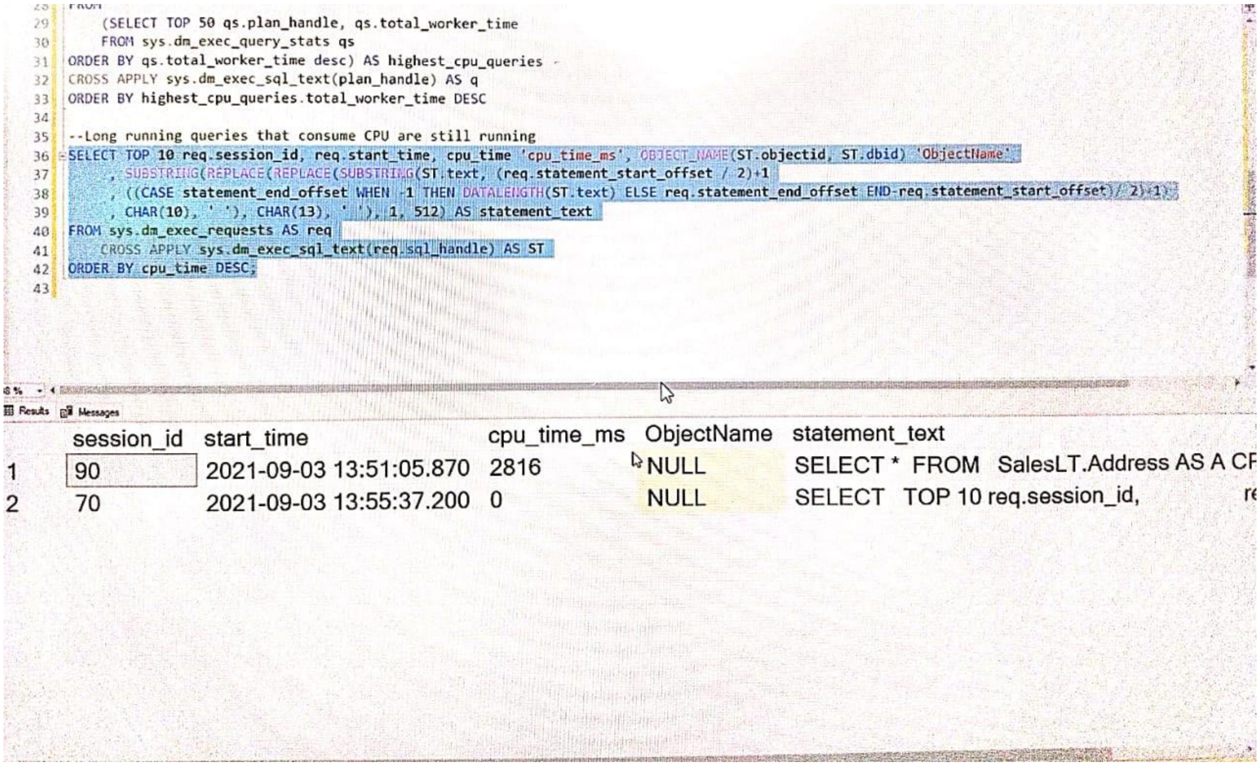
And from that, you'll be able to do well, is there a reason for it? Could the query be rewritten? Could I add some indexes? In addition, which is used the most cumulative CPU?

So, this statement may take the highest CPU for each individual query, but maybe I'll run that once, and I'll run this 100 times. Well, that would take longer in

total. So, let's run this one, and you can see the various plan handles, and here is the text over here.

You can see the total workout time in cumulative, so I might have run one thing several times, or it could be that the server did so as well.

So that users dm_exec_query_stats and dm_exec_sql_text. So, we've seen that dm_exec_sql_text earlier when we were looking at the cached plans, so it can be used quite frequently. So, this uses the most cumulative CPU, and then finally the longest running queries that consumed CPU, that are still running.



```
FROM
    (SELECT TOP 50 qs.plan_handle, qs.total_worker_time
    FROM sys.dm_exec_query_stats qs
ORDER BY qs.total_worker_time desc) AS highest_cpu_queries
CROSS APPLY sys.dm_exec_sql_text(plan_handle) AS q
ORDER BY highest_cpu_queries.total_worker_time DESC

--Long running queries that consume CPU are still running
SELECT TOP 10 req.session_id, req.start_time, cpu_time 'cpu_time_ms', OBJECT_NAME(ST.objectid, ST.dbid) 'ObjectName'
    , SUBSTRING(REPLACE(REPLACE(SUBSTRING(ST.text, (req.statement_start_offset / 2)+1
    , ((CASE statement_end_offset WHEN -1 THEN DATALENGTH(ST.text) ELSE req.statement_end_offset END-req.statement_start_offset)/ 2)+1)
    , CHAR(10), ' '), CHAR(13), ' '), 1, 512) AS statement_text
FROM sys.dm_exec_requests AS req
    CROSS APPLY sys.dm_exec_sql_text(req.sql_handle) AS ST
ORDER BY cpu_time DESC;
```

	session_id	start_time	cpu_time_ms	ObjectName	statement_text
1	90	2021-09-03 13:51:05.870	2816	NULL	SELECT * FROM SalesLT.Address AS A CF
2	70	2021-09-03 13:55:37.200	0	NULL	SELECT TOP 10 req.session_id, re

We have the statement text, so I could copy that and retrieve the text. We have the CPU time and milliseconds, it's taken a lot longer now, but this is probably the less time it's updated. The session_id now you notice the session_id is in brackets here and the start time. So, if your computer is slowing down right now,

then have a look at the dm_exec_request,

and again, this ties in with the dm_exec_sql_text.