**Monitor Microsoft Azure SQL Database performance using dynamic management views**

**In this article**

1. [Permissions](https://learn.microsoft.com/en-us/azure/azure-sql/database/monitoring-with-dmvs?view=azuresql#permissions)
2. [Identify CPU performance issues](https://learn.microsoft.com/en-us/azure/azure-sql/database/monitoring-with-dmvs?view=azuresql#identify-cpu-performance-issues)
3. [Identify I/O performance issues](https://learn.microsoft.com/en-us/azure/azure-sql/database/monitoring-with-dmvs?view=azuresql#identify-io-performance-issues)
4. [Identify tempdb performance issues](https://learn.microsoft.com/en-us/azure/azure-sql/database/monitoring-with-dmvs?view=azuresql#identify-tempdb-performance-issues)

Show 8 more

**Applies to:**  [Azure SQL Database](https://learn.microsoft.com/en-us/sql/sql-server/sql-docs-navigation-guide#applies-to)

Azure SQL Database

Microsoft Azure SQL Database enables a subset of dynamic management views to diagnose performance problems, which might be caused by blocked or long-running queries, resource bottlenecks, poor query plans, and more.

This article provides information on how to detect common performance problems by querying dynamic management views via T-SQL. You can use any query tool, such as:

* The [SQL query editor in the Azure portal](https://learn.microsoft.com/en-us/azure/azure-sql/database/query-editor?view=azuresql)
* [SQL Server Management Studio (SSMS)](https://learn.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms)
* [Azure Data Studio](https://learn.microsoft.com/en-us/azure-data-studio/what-is-azure-data-studio)

**Permissions**

In Azure SQL Database, depending on the compute size and deployment option, querying a DMV may require either VIEW DATABASE STATE or VIEW SERVER STATE permission. The latter permission may be granted via membership in the ##MS\_ServerStateReader## server role.

To grant the **VIEW DATABASE STATE** permission to a specific database user, run the following query as an example:

SQLCopy

GRANT VIEW DATABASE STATE TO database\_user;

To grant membership to the ##MS\_ServerStateReader## server role to a login for the [logical server in Azure](https://learn.microsoft.com/en-us/azure/azure-sql/database/logical-servers?view=azuresql), connect to the master database then run the following query as an example:

SQLCopy

ALTER SERVER ROLE [##MS\_ServerStateReader##] ADD MEMBER [login];

In an instance of SQL Server and in Azure SQL Managed Instance, dynamic management views return server state information. In Azure SQL Database, they return information regarding your current logical database only.

**Identify CPU performance issues**

If CPU consumption is above 80% for extended periods of time, consider the following troubleshooting steps whether [the CPU issue is occurring now](https://learn.microsoft.com/en-us/azure/azure-sql/database/monitoring-with-dmvs?view=azuresql#the-cpu-issue-is-occurring-now) or has [occurred in the past](https://learn.microsoft.com/en-us/azure/azure-sql/database/monitoring-with-dmvs?view=azuresql#the-cpu-issue-occurred-in-the-past).

**The CPU issue is occurring now**

If issue is occurring right now, there are two possible scenarios:

**Many individual queries that cumulatively consume high CPU**

* Use the following query to identify top query hashes:

SQLCopy

PRINT '-- top 10 Active CPU Consuming Queries (aggregated)--';

SELECT TOP 10 GETDATE() runtime, \*

FROM (SELECT query\_stats.query\_hash, SUM(query\_stats.cpu\_time) 'Total\_Request\_Cpu\_Time\_Ms', SUM(logical\_reads) 'Total\_Request\_Logical\_Reads', MIN(start\_time) 'Earliest\_Request\_start\_Time', COUNT(\*) 'Number\_Of\_Requests', SUBSTRING(REPLACE(REPLACE(MIN(query\_stats.statement\_text), CHAR(10), ' '), CHAR(13), ' '), 1, 256) AS "Statement\_Text"

FROM (SELECT req.\*, 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) AS statement\_text

FROM sys.dm\_exec\_requests AS req

CROSS APPLY sys.dm\_exec\_sql\_text(req.sql\_handle) AS ST ) AS query\_stats

GROUP BY query\_hash) AS t

ORDER BY Total\_Request\_Cpu\_Time\_Ms DESC;

**Long running queries that consume CPU are still running**

* Use the following query to identify these queries:

SQLCopy

PRINT '--top 10 Active CPU Consuming Queries by sessions--';

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;

GO

**The CPU issue occurred in the past**

If the issue occurred in the past and you want to do root cause analysis, use [Query Store](https://learn.microsoft.com/en-us/sql/relational-databases/performance/monitoring-performance-by-using-the-query-store?view=azuresqldb-current&preserve-view=true). Users with database access can use T-SQL to query Query Store data. Query Store default configurations use a granularity of 1 hour.

1. Use the following query to look at activity for high CPU consuming queries. This query returns the top 15 CPU consuming queries. Remember to change rsi.start\_time >= DATEADD(hour, -2, GETUTCDATE():

SQLCopy

-- Top 15 CPU consuming queries by query hash

-- Note that a query hash can have many query ids if not parameterized or not parameterized properly

WITH AggregatedCPU

AS (

SELECT q.query\_hash

,SUM(count\_executions \* avg\_cpu\_time / 1000.0) AS total\_cpu\_ms

,SUM(count\_executions \* avg\_cpu\_time / 1000.0) / SUM(count\_executions) AS avg\_cpu\_ms

,MAX(rs.max\_cpu\_time / 1000.00) AS max\_cpu\_ms

,MAX(max\_logical\_io\_reads) max\_logical\_reads

,COUNT(DISTINCT p.plan\_id) AS number\_of\_distinct\_plans

,COUNT(DISTINCT p.query\_id) AS number\_of\_distinct\_query\_ids

,SUM(CASE

WHEN rs.execution\_type\_desc = 'Aborted'

THEN count\_executions

ELSE 0

END) AS Aborted\_Execution\_Count

,SUM(CASE

WHEN rs.execution\_type\_desc = 'Regular'

THEN count\_executions

ELSE 0

END) AS Regular\_Execution\_Count

,SUM(CASE

WHEN rs.execution\_type\_desc = 'Exception'

THEN count\_executions

ELSE 0

END) AS Exception\_Execution\_Count

,SUM(count\_executions) AS total\_executions

,MIN(qt.query\_sql\_text) AS sampled\_query\_text

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

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

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

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

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

WHERE rs.execution\_type\_desc IN ('Regular','Aborted','Exception')

AND rsi.start\_time >= DATEADD(HOUR, - 2, GETUTCDATE())

GROUP BY q.query\_hash

)

,OrderedCPU

AS (

SELECT query\_hash

,total\_cpu\_ms

,avg\_cpu\_ms

,max\_cpu\_ms

,max\_logical\_reads

,number\_of\_distinct\_plans

,number\_of\_distinct\_query\_ids

,total\_executions

,Aborted\_Execution\_Count

,Regular\_Execution\_Count

,Exception\_Execution\_Count

,sampled\_query\_text

,ROW\_NUMBER() OVER (

ORDER BY total\_cpu\_ms DESC

,query\_hash ASC

) AS query\_hash\_row\_number

FROM AggregatedCPU

)

SELECT OD.query\_hash

,OD.total\_cpu\_ms

,OD.avg\_cpu\_ms

,OD.max\_cpu\_ms

,OD.max\_logical\_reads

,OD.number\_of\_distinct\_plans

,OD.number\_of\_distinct\_query\_ids

,OD.total\_executions

,OD.Aborted\_Execution\_Count

,OD.Regular\_Execution\_Count

,OD.Exception\_Execution\_Count

,OD.sampled\_query\_text

,OD.query\_hash\_row\_number

FROM OrderedCPU AS OD

WHERE OD.query\_hash\_row\_number <= 15 --get top 15 rows by total\_cpu\_ms

ORDER BY total\_cpu\_ms DESC;

1. Once you identify the problematic queries, it's time to tune those queries to reduce CPU utilization. If you don't have time to tune the queries, you may also choose to upgrade the SLO of the database to work around the issue.

For more information about handling CPU performance problems in Azure SQL Database, see [Diagnose and troubleshoot high CPU on Azure SQL Database](https://learn.microsoft.com/en-us/azure/azure-sql/database/high-cpu-diagnose-troubleshoot?view=azuresql).

**Identify I/O performance issues**

When identifying storage input/output (I/O) performance issues, the top wait types associated with I/O issues are:

* PAGEIOLATCH\_\*

For data file I/O issues (including PAGEIOLATCH\_SH, PAGEIOLATCH\_EX, PAGEIOLATCH\_UP). If the wait type name has **IO** in it, it points to an I/O issue. If there is no **IO** in the page latch wait name, it points to a different type of problem (for example, tempdb contention).

* WRITE\_LOG

For transaction log I/O issues.

**If the I/O issue is occurring right now**

Use the [sys.dm\_exec\_requests](https://learn.microsoft.com/en-us/sql/relational-databases/system-dynamic-management-views/sys-dm-exec-requests-transact-sql?view=azuresqldb-current&preserve-view=true) or [sys.dm\_os\_waiting\_tasks](https://learn.microsoft.com/en-us/sql/relational-databases/system-dynamic-management-views/sys-dm-os-waiting-tasks-transact-sql?view=azuresqldb-current&preserve-view=true) to see the wait\_type and wait\_time.

**Identify data and log I/O usage**

Use the following query to identify data and log I/O usage. If the data or log I/O is above 80%, it means users have used the available I/O for the Azure SQL Database service tier.

SQLCopy

SELECT

database\_name = DB\_NAME()

, UTC\_time = end\_time

, 'CPU Utilization In % of Limit' = rs.avg\_cpu\_percent

, 'Data IO In % of Limit' = rs.avg\_data\_io\_percent

, 'Log Write Utilization In % of Limit' = rs.avg\_log\_write\_percent

, 'Memory Usage In % of Limit' = rs.avg\_memory\_usage\_percent

, 'In-Memory OLTP Storage in % of Limit' = rs.xtp\_storage\_percent

, 'Concurrent Worker Threads in % of Limit' = rs.max\_worker\_percent

, 'Concurrent Sessions in % of Limit' = rs.max\_session\_percent

FROM sys.dm\_db\_resource\_stats AS rs --past hour only

ORDER BY rs.end\_time DESC;

For more examples using sys.dm\_db\_resource\_stats, see the [Monitor resource use](https://learn.microsoft.com/en-us/azure/azure-sql/database/monitoring-with-dmvs?view=azuresql#monitor-resource-use) section later in this article.

If the I/O limit has been reached, you have two options:

* Upgrade the compute size or service tier
* Identify and tune the queries consuming the most I/O.

**View buffer-related I/O using the Query Store**

For option 2, you can use the following query against Query Store for buffer-related I/O to view the last two hours of tracked activity:

SQLCopy

-- Top queries that waited on buffer

-- Note these are finished queries

WITH Aggregated AS (SELECT q.query\_hash, SUM(total\_query\_wait\_time\_ms) total\_wait\_time\_ms, SUM(total\_query\_wait\_time\_ms / avg\_query\_wait\_time\_ms) AS total\_executions, MIN(qt.query\_sql\_text) AS sampled\_query\_text, MIN(wait\_category\_desc) AS wait\_category\_desc

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

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

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

INNER JOIN sys.query\_store\_wait\_stats AS waits ON waits.plan\_id=p.plan\_id

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

WHERE wait\_category\_desc='Buffer IO' AND rsi.start\_time>=DATEADD(HOUR, -2, GETUTCDATE())

GROUP BY q.query\_hash), Ordered AS (SELECT query\_hash, total\_executions, total\_wait\_time\_ms, sampled\_query\_text, wait\_category\_desc, ROW\_NUMBER() OVER (ORDER BY total\_wait\_time\_ms DESC, query\_hash ASC) AS query\_hash\_row\_number

FROM Aggregated)

SELECT OD.query\_hash, OD.total\_executions, OD.total\_wait\_time\_ms, OD.sampled\_query\_text, OD.wait\_category\_desc, OD.query\_hash\_row\_number

FROM Ordered AS OD

WHERE OD.query\_hash\_row\_number <= 15 -- get top 15 rows by total\_wait\_time\_ms

ORDER BY total\_wait\_time\_ms DESC;

GO

**View total log I/O for WRITELOG waits**

If the wait type is WRITELOG, use the following query to view total log I/O by statement:

SQLCopy

-- Top transaction log consumers

-- Adjust the time window by changing

-- rsi.start\_time >= DATEADD(hour, -2, GETUTCDATE())

WITH AggregatedLogUsed

AS (SELECT q.query\_hash,

SUM(count\_executions \* avg\_cpu\_time / 1000.0) AS total\_cpu\_ms,

SUM(count\_executions \* avg\_cpu\_time / 1000.0) / SUM(count\_executions) AS avg\_cpu\_ms,

SUM(count\_executions \* avg\_log\_bytes\_used) AS total\_log\_bytes\_used,

MAX(rs.max\_cpu\_time / 1000.00) AS max\_cpu\_ms,

MAX(max\_logical\_io\_reads) max\_logical\_reads,

COUNT(DISTINCT p.plan\_id) AS number\_of\_distinct\_plans,

COUNT(DISTINCT p.query\_id) AS number\_of\_distinct\_query\_ids,

SUM( CASE

WHEN rs.execution\_type\_desc = 'Aborted' THEN

count\_executions

ELSE 0

END

) AS Aborted\_Execution\_Count,

SUM( CASE

WHEN rs.execution\_type\_desc = 'Regular' THEN

count\_executions

ELSE 0

END

) AS Regular\_Execution\_Count,

SUM( CASE

WHEN rs.execution\_type\_desc = 'Exception' THEN

count\_executions

ELSE 0

END

) AS Exception\_Execution\_Count,

SUM(count\_executions) AS total\_executions,

MIN(qt.query\_sql\_text) AS sampled\_query\_text

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

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

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

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

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

WHERE rs.execution\_type\_desc IN ( 'Regular', 'Aborted', 'Exception' )

AND rsi.start\_time >= DATEADD(HOUR, -2, GETUTCDATE())

GROUP BY q.query\_hash),

OrderedLogUsed

AS (SELECT query\_hash,

total\_log\_bytes\_used,

number\_of\_distinct\_plans,

number\_of\_distinct\_query\_ids,

total\_executions,

Aborted\_Execution\_Count,

Regular\_Execution\_Count,

Exception\_Execution\_Count,

sampled\_query\_text,

ROW\_NUMBER() OVER (ORDER BY total\_log\_bytes\_used DESC, query\_hash ASC) AS query\_hash\_row\_number

FROM AggregatedLogUsed)

SELECT OD.total\_log\_bytes\_used,

OD.number\_of\_distinct\_plans,

OD.number\_of\_distinct\_query\_ids,

OD.total\_executions,

OD.Aborted\_Execution\_Count,

OD.Regular\_Execution\_Count,

OD.Exception\_Execution\_Count,

OD.sampled\_query\_text,

OD.query\_hash\_row\_number

FROM OrderedLogUsed AS OD

WHERE OD.query\_hash\_row\_number <= 15 -- get top 15 rows by total\_log\_bytes\_used

ORDER BY total\_log\_bytes\_used DESC;

GO

**Identify tempdb performance issues**

When identifying I/O performance issues, the top wait types associated with tempdb issues is PAGELATCH\_\* (not PAGEIOLATCH\_\*). However, PAGELATCH\_\* waits do not always mean you have tempdb contention. This wait may also mean that you have user-object data page contention due to concurrent requests targeting the same data page. To further confirm tempdb contention, use [sys.dm\_exec\_requests](https://learn.microsoft.com/en-us/sql/relational-databases/system-dynamic-management-views/sys-dm-exec-requests-transact-sql?view=azuresqldb-current&preserve-view=true) to confirm that the wait\_resource value begins with 2:x:y where 2 is tempdb is the database ID, x is the file ID, and y is the page ID.

For tempdb contention, a common method is to reduce or rewrite application code that relies on tempdb. Common tempdb usage areas include:

* Temp tables
* Table variables
* Table-valued parameters
* Version store usage (associated with long running transactions)
* Queries that have query plans that use sorts, hash joins, and spools

For more information, see [tempdb in Azure SQL](https://learn.microsoft.com/en-us/sql/relational-databases/databases/tempdb-database?view=azuresqldb-current&preserve-view=true" \l "tempdb-in-azure-sql).

**Top queries that use table variables and temporary tables**

Use the following query to identify top queries that use table variables and temporary tables:

SQLCopy

SELECT plan\_handle, execution\_count, query\_plan

INTO #tmpPlan

FROM sys.dm\_exec\_query\_stats

CROSS APPLY sys.dm\_exec\_query\_plan(plan\_handle);

GO

WITH XMLNAMESPACES('http://schemas.microsoft.com/sqlserver/2004/07/showplan' AS sp)

SELECT plan\_handle, stmt.stmt\_details.value('@Database', 'varchar(max)') AS 'Database'

, stmt.stmt\_details.value('@Schema', 'varchar(max)') AS 'Schema'

, stmt.stmt\_details.value('@Table', 'varchar(max)') AS 'table'

INTO #tmp2

FROM

(SELECT CAST(query\_plan AS XML) sqlplan, plan\_handle FROM #tmpPlan) AS p

CROSS APPLY sqlplan.nodes('//sp:Object') AS stmt(stmt\_details);

GO

SELECT t.plan\_handle, [Database], [Schema], [table], execution\_count

FROM

(SELECT DISTINCT plan\_handle, [Database], [Schema], [table]

FROM #tmp2

WHERE [table] LIKE '%@%' OR [table] LIKE '%#%') AS t

INNER JOIN #tmpPlan AS t2 ON t.plan\_handle=t2.plan\_handle;

GO

DROP TABLE #tmpPlan

DROP TABLE #tmp2

**Identify long running transactions**

Use the following query to identify long running transactions. Long running transactions prevent persistent version store (PVS) cleanup. For more information, see [Troubleshoot accelerated database recovery](https://learn.microsoft.com/en-us/sql/relational-databases/accelerated-database-recovery-troubleshoot?view=azuresqldb-current&preserve-view=true).

SQLCopy

SELECT DB\_NAME(dtr.database\_id) 'database\_name',

sess.session\_id,

atr.name AS 'tran\_name',

atr.transaction\_id,

transaction\_type,

transaction\_begin\_time,

database\_transaction\_begin\_time,

transaction\_state,

is\_user\_transaction,

sess.open\_transaction\_count,

TRIM(REPLACE(

REPLACE(

SUBSTRING(

SUBSTRING(

txt.text,

(req.statement\_start\_offset / 2) + 1,

((CASE req.statement\_end\_offset

WHEN -1 THEN

DATALENGTH(txt.text)

ELSE

req.statement\_end\_offset

END - req.statement\_start\_offset

) / 2

) + 1

),

1,

1000

),

CHAR(10),

' '

),

CHAR(13),

' '

)

) Running\_stmt\_text,

recenttxt.text 'MostRecentSQLText'

FROM sys.dm\_tran\_active\_transactions AS atr

INNER JOIN sys.dm\_tran\_database\_transactions AS dtr

ON dtr.transaction\_id = atr.transaction\_id

LEFT JOIN sys.dm\_tran\_session\_transactions AS sess

ON sess.transaction\_id = atr.transaction\_id

LEFT JOIN sys.dm\_exec\_requests AS req

ON req.session\_id = sess.session\_id

AND req.transaction\_id = sess.transaction\_id

LEFT JOIN sys.dm\_exec\_connections AS conn

ON sess.session\_id = conn.session\_id

OUTER APPLY sys.dm\_exec\_sql\_text(req.sql\_handle) AS txt

OUTER APPLY sys.dm\_exec\_sql\_text(conn.most\_recent\_sql\_handle) AS recenttxt

WHERE atr.transaction\_type != 2

AND sess.session\_id != @@spid

ORDER BY start\_time ASC;

**Identify memory grant wait performance issues**

If your top wait type is RESOURCE\_SEMAPHORE and you don't have a high CPU usage issue, you may have a memory grant waiting issue.

**Determine if a RESOURCE\_SEMAPHORE wait is a top wait**

Use the following query to determine if a RESOURCE\_SEMAPHORE wait is a top wait. Also indicative would be a rising wait time rank of RESOURCE\_SEMAPHORE in recent history. For more information on troubleshooting memory grant waiting issues, see [Troubleshoot slow performance or low memory issues caused by memory grants in SQL Server](https://learn.microsoft.com/en-us/troubleshoot/sql/database-engine/performance/troubleshoot-memory-grant-issues).

SQLCopy

SELECT wait\_type,

SUM(wait\_time) AS total\_wait\_time\_ms

FROM sys.dm\_exec\_requests AS req

INNER JOIN sys.dm\_exec\_sessions AS sess

ON req.session\_id = sess.session\_id

WHERE is\_user\_process = 1

GROUP BY wait\_type

ORDER BY SUM(wait\_time) DESC;

**Identify high memory-consuming statements**

If you encounter out of memory errors in Azure SQL Database, review [sys.dm\_os\_out\_of\_memory\_events](https://learn.microsoft.com/en-us/sql/relational-databases/system-dynamic-management-views/sys-dm-os-out-of-memory-events?view=azuresqldb-current&preserve-view=true). For more information, see [Troubleshoot out of memory errors with Azure SQL Database](https://learn.microsoft.com/en-us/azure/azure-sql/database/troubleshoot-memory-errors-issues?view=azuresql).

First, modify the below script to update relevant values of start\_time and end\_time. Then, run the following query to identify high memory-consuming statements:

SQLCopy

SELECT IDENTITY(INT, 1, 1) rowId,

CAST(query\_plan AS XML) query\_plan,

p.query\_id

INTO #tmp

FROM sys.query\_store\_plan AS p

INNER JOIN sys.query\_store\_runtime\_stats AS r

ON p.plan\_id = r.plan\_id

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

ON r.runtime\_stats\_interval\_id = i.runtime\_stats\_interval\_id

WHERE start\_time > '2018-10-11 14:00:00.0000000'

AND end\_time < '2018-10-17 20:00:00.0000000';

GO

;WITH cte

AS (SELECT query\_id,

query\_plan,

m.c.value('@SerialDesiredMemory', 'INT') AS SerialDesiredMemory

FROM #tmp AS t

CROSS APPLY t.query\_plan.nodes('//\*:MemoryGrantInfo[@SerialDesiredMemory[. > 0]]') AS m(c) )

SELECT TOP 50

cte.query\_id,

t.query\_sql\_text,

cte.query\_plan,

CAST(SerialDesiredMemory / 1024. AS DECIMAL(10, 2)) SerialDesiredMemory\_MB

FROM cte

INNER JOIN sys.query\_store\_query AS q

ON cte.query\_id = q.query\_id

INNER JOIN sys.query\_store\_query\_text AS t

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

ORDER BY SerialDesiredMemory DESC;

**Identify the top 10 active memory grants**

Use the following query to identify the top 10 active memory grants:

SQLCopy

SELECT TOP 10

CONVERT(VARCHAR(30), GETDATE(), 121) AS runtime,

r.session\_id,

r.blocking\_session\_id,

r.cpu\_time,

r.total\_elapsed\_time,

r.reads,

r.writes,

r.logical\_reads,

r.row\_count,

wait\_time,

wait\_type,

r.command,

OBJECT\_NAME(txt.objectid, txt.dbid) 'Object\_Name',

TRIM(REPLACE(REPLACE(SUBSTRING(SUBSTRING(TEXT, (r.statement\_start\_offset / 2) + 1,

( (

CASE r.statement\_end\_offset

WHEN - 1

THEN DATALENGTH(TEXT)

ELSE r.statement\_end\_offset

END - r.statement\_start\_offset

) / 2

) + 1), 1, 1000), CHAR(10), ' '), CHAR(13), ' ')) AS stmt\_text,

mg.dop, --Degree of parallelism

mg.request\_time, --Date and time when this query requested the memory grant.

mg.grant\_time, --NULL means memory has not been granted

mg.requested\_memory\_kb / 1024.0 requested\_memory\_mb, --Total requested amount of memory in megabytes

mg.granted\_memory\_kb / 1024.0 AS granted\_memory\_mb, --Total amount of memory actually granted in megabytes. NULL if not granted

mg.required\_memory\_kb / 1024.0 AS required\_memory\_mb, --Minimum memory required to run this query in megabytes.

max\_used\_memory\_kb / 1024.0 AS max\_used\_memory\_mb,

mg.query\_cost, --Estimated query cost.

mg.timeout\_sec, --Time-out in seconds before this query gives up the memory grant request.

mg.resource\_semaphore\_id, --Non-unique ID of the resource semaphore on which this query is waiting.

mg.wait\_time\_ms, --Wait time in milliseconds. NULL if the memory is already granted.

CASE mg.is\_next\_candidate --Is this process the next candidate for a memory grant

WHEN 1 THEN

'Yes'

WHEN 0 THEN

'No'

ELSE

'Memory has been granted'

END AS 'Next Candidate for Memory Grant',

qp.query\_plan

FROM sys.dm\_exec\_requests AS r

INNER JOIN sys.dm\_exec\_query\_memory\_grants AS mg

ON r.session\_id = mg.session\_id

AND r.request\_id = mg.request\_id

CROSS APPLY sys.dm\_exec\_sql\_text(mg.sql\_handle) AS txt

CROSS APPLY sys.dm\_exec\_query\_plan(r.plan\_handle) AS qp

ORDER BY mg.granted\_memory\_kb DESC;

**Monitor connections**

You can use the [sys.dm\_exec\_connections](https://learn.microsoft.com/en-us/sql/relational-databases/system-dynamic-management-views/sys-dm-exec-connections-transact-sql?view=azuresqldb-current&preserve-view=true) view to retrieve information about the connections established to a specific database or elastic pool and the details of each connection. In addition, the [sys.dm\_exec\_sessions](https://learn.microsoft.com/en-us/sql/relational-databases/system-dynamic-management-views/sys-dm-exec-sessions-transact-sql?view=azuresqldb-current&preserve-view=true) view is helpful when retrieving information about all active user connections and internal tasks.

**View current sessions**

The following query retrieves information on the current connection. To view all sessions, remove the WHERE clause.

You see all executing sessions on the database only if you have **VIEW DATABASE STATE** permission on the database when executing the sys.dm\_exec\_requests and sys.dm\_exec\_sessions views. Otherwise, you see only the current session.

SQLCopy

SELECT

c.session\_id, c.net\_transport, c.encrypt\_option,

c.auth\_scheme, s.host\_name, s.program\_name,

s.client\_interface\_name, s.login\_name, s.nt\_domain,

s.nt\_user\_name, s.original\_login\_name, c.connect\_time,

s.login\_time

FROM sys.dm\_exec\_connections AS c

INNER JOIN sys.dm\_exec\_sessions AS s

ON c.session\_id = s.session\_id

WHERE c.session\_id = @@SPID; --Remove to view all sessions, if permissions allow

**Monitor resource use**

You can monitor Azure SQL Database resource usage at the query level by using [SQL Database Query Performance Insight](https://learn.microsoft.com/en-us/azure/azure-sql/database/query-performance-insight-use?view=azuresql) in the Azure portal or the [Query Store](https://learn.microsoft.com/en-us/sql/relational-databases/performance/monitoring-performance-by-using-the-query-store?view=azuresqldb-current&preserve-view=true).

You can also monitor usage using these views:

* [sys.dm\_db\_resource\_stats](https://learn.microsoft.com/en-us/sql/relational-databases/system-dynamic-management-views/sys-dm-db-resource-stats-azure-sql-database?view=azuresqldb-current&preserve-view=true)
* [sys.resource\_stats](https://learn.microsoft.com/en-us/sql/relational-databases/system-catalog-views/sys-resource-stats-azure-sql-database?view=azuresqldb-current&preserve-view=true)

**sys.dm\_db\_resource\_stats**

You can use the [sys.dm\_db\_resource\_stats](https://learn.microsoft.com/en-us/sql/relational-databases/system-dynamic-management-views/sys-dm-db-resource-stats-azure-sql-database?view=azuresqldb-current&preserve-view=true) view in every database. The sys.dm\_db\_resource\_stats view shows recent resource use data relative to the service tier. Average percentages for CPU, data I/O, log writes, and memory are recorded every 15 seconds and are maintained for 1 hour.

Because this view provides a more granular look at resource use, use sys.dm\_db\_resource\_stats first for any current-state analysis or troubleshooting. For example, this query shows the average and maximum resource use for the current database over the past hour:

SQLCopy

SELECT

Database\_Name = DB\_NAME(),

tier\_limit = COALESCE(rs.dtu\_limit, cpu\_limit), --DTU or vCore limit

AVG(avg\_cpu\_percent) AS 'Average CPU use in percent',

MAX(avg\_cpu\_percent) AS 'Maximum CPU use in percent',

AVG(avg\_data\_io\_percent) AS 'Average data IO in percent',

MAX(avg\_data\_io\_percent) AS 'Maximum data IO in percent',

AVG(avg\_log\_write\_percent) AS 'Average log write use in percent',

MAX(avg\_log\_write\_percent) AS 'Maximum log write use in percent',

AVG(avg\_memory\_usage\_percent) AS 'Average memory use in percent',

MAX(avg\_memory\_usage\_percent) AS 'Maximum memory use in percent'

FROM sys.dm\_db\_resource\_stats AS rs --past hour only

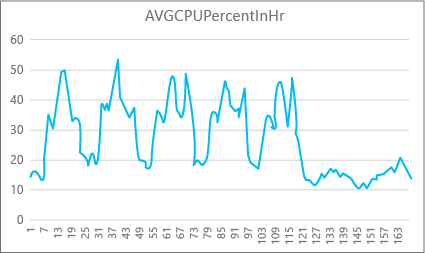
GROUP BY rs.dtu\_limit, rs.cpu\_limit;

For other queries, see the examples in [sys.dm\_db\_resource\_stats](https://learn.microsoft.com/en-us/sql/relational-databases/system-dynamic-management-views/sys-dm-db-resource-stats-azure-sql-database?view=azuresqldb-current&preserve-view=true).

**sys.resource\_stats**

The [sys.resource\_stats](https://learn.microsoft.com/en-us/sql/relational-databases/system-catalog-views/sys-resource-stats-azure-sql-database?view=azuresqldb-current&preserve-view=true) view in the master database has additional information that can help you monitor the performance of your database at its specific service tier and compute size. The data is collected every 5 minutes and is maintained for approximately 14 days. This view is useful for a longer-term historical analysis of how your database uses resources.

The following graph shows the CPU resource use for a Premium database with the P2 compute size for each hour in a week. This graph starts on a Monday, shows five work days, and then shows a weekend, when much less happens on the application.



From the data, this database currently has a peak CPU load of just over 50 percent CPU use relative to the P2 compute size (midday on Tuesday). If CPU is the dominant factor in the application's resource profile, then you might decide that P2 is the right compute size to guarantee that the workload always fits. If you expect an application to grow over time, it's a good idea to have an extra resource buffer so that the application doesn't ever reach the performance-level limit. If you increase the compute size, you can help avoid customer-visible errors that might occur when a database doesn't have enough power to process requests effectively, especially in latency-sensitive environments. An example is a database that supports an application that paints webpages based on the results of database calls.

Other application types might interpret the same graph differently. For example, if an application tries to process payroll data each day and has the same chart, this kind of "batch job" model might do fine at a P1 compute size. The P1 compute size has 100 DTUs compared to 200 DTUs at the P2 compute size. The P1 compute size provides half the performance of the P2 compute size. So, 50 percent of CPU use in P2 equals 100 percent CPU use in P1. If the application does not have timeouts, it might not matter if a job takes 2 hours or 2.5 hours to finish, if it gets done today. An application in this category probably can use a P1 compute size. You can take advantage of the fact that there are periods of time during the day when resource use is lower, so that any "big peak" might spill over into one of the troughs later in the day. The P1 compute size might be good for that kind of application (and save money), as long as the jobs can finish on time each day.

The database engine exposes consumed resource information for each active database in the sys.resource\_stats view of the master database in each server. The data in the table is aggregated for 5-minute intervals. With the Basic, Standard, and Premium service tiers, the data can take more than 5 minutes to appear in the table, so this data is more useful for historical analysis rather than near-real-time analysis. Query the sys.resource\_stats view to see the recent history of a database and to validate whether the reservation you chose delivered the performance you want when needed.

**Note**

On Azure SQL Database, you must be connected to the master database to query sys.resource\_stats in the following examples.

This example shows you how the data in this view is exposed:

SQLCopy

SELECT TOP 10 \*

FROM sys.resource\_stats

WHERE database\_name = 'userdb1'

ORDER BY start\_time DESC;

The next example shows you different ways that you can use the sys.resource\_stats catalog view to get information about how your database uses resources:

1. To look at the past week's resource use for the user database userdb1, you can run this query, substituting your own database name:

SQLCopy

SELECT \*

FROM sys.resource\_stats

WHERE database\_name = 'userdb1'

AND start\_time > DATEADD(day, -7, GETDATE())

ORDER BY start\_time DESC;

1. To evaluate how well your workload fits the compute size, you need to drill down into each aspect of the resource metrics: CPU, reads, writes, number of workers, and number of sessions. Here's a revised query using sys.resource\_stats to report the average and maximum values of these resource metrics, for each service tier the database has been provisioned for:

SQLCopy

SELECT rs.database\_name

, rs.sku

, storage\_mb = MAX(rs.Storage\_in\_megabytes)

, 'Average CPU Utilization In %' = AVG(rs.avg\_cpu\_percent)

, 'Maximum CPU Utilization In %' = MAX(rs.avg\_cpu\_percent)

, 'Average Data IO In %' = AVG(rs.avg\_data\_io\_percent)

, 'Maximum Data IO In %' = MAX(rs.avg\_data\_io\_percent)

, 'Average Log Write Utilization In %' = AVG(rs.avg\_log\_write\_percent)

, 'Maximum Log Write Utilization In %' = MAX(rs.avg\_log\_write\_percent)

, 'Average Requests In %' = AVG(rs.max\_worker\_percent)

, 'Maximum Requests In %' = MAX(rs.max\_worker\_percent)

, 'Average Sessions In %' = AVG(rs.max\_session\_percent)

, 'Maximum Sessions In %' = MAX(rs.max\_session\_percent)

FROM sys.resource\_stats AS rs

WHERE rs.database\_name = 'userdb1'

AND rs.start\_time > DATEADD(day, -7, GETDATE())

GROUP BY rs.database\_name, rs.sku;

1. With this information about the average and maximum values of each resource metric, you can assess how well your workload fits into the compute size you chose. Usually, average values from sys.resource\_stats give you a good baseline to use against the target size. It should be your primary measurement stick.
   * For **DTU purchasing model** databases:

For an example, you might be using the Standard service tier with S2 compute size. The average use percentages for CPU and I/O reads and writes are below 40 percent, the average number of workers is below 50, and the average number of sessions is below 200. Your workload might fit into the S1 compute size. It's easy to see whether your database fits in the worker and session limits. To see whether a database fits into a lower compute size, divide the DTU number of the lower compute size by the DTU number of your current compute size, and then multiply the result by 100:

S1 DTU / S2 DTU \* 100 = 20 / 50 \* 100 = 40

The result is the relative performance difference between the two compute sizes in percentage. If your resource use doesn't exceed this amount, your workload might fit into the lower compute size. However, you need to look at all ranges of resource use values, and determine, by percentage, how often your database workload would fit into the lower compute size. The following query outputs the fit percentage per resource dimension, based on the threshold of 40 percent that we calculated in this example:

SQLCopy

SELECT database\_name,

100\*((COUNT(database\_name) - SUM(CASE WHEN avg\_cpu\_percent >= 40 THEN 1 ELSE 0 END) \* 1.0) / COUNT(database\_name)) AS 'CPU Fit Percent',

100\*((COUNT(database\_name) - SUM(CASE WHEN avg\_log\_write\_percent >= 40 THEN 1 ELSE 0 END) \* 1.0) / COUNT(database\_name)) AS 'Log Write Fit Percent',

100\*((COUNT(database\_name) - SUM(CASE WHEN avg\_data\_io\_percent >= 40 THEN 1 ELSE 0 END) \* 1.0) / COUNT(database\_name)) AS 'Physical Data IO Fit Percent'

FROM sys.resource\_stats

WHERE start\_time > DATEADD(day, -7, GETDATE())

AND database\_name = 'sample' --remove to see all databases

GROUP BY database\_name;

Based on your database service tier, you can decide whether your workload fits into the lower compute size. If your database workload objective is 99.9 percent and the preceding query returns values greater than 99.9 percent for all three resource dimensions, your workload likely fits into the lower compute size.

Looking at the fit percentage also gives you insight into whether you should move to the next higher compute size to meet your objective. For example, the CPU usage for a sample database over the past week:

Expand table

| **Average CPU percent** | **Maximum CPU percent** |
| --- | --- |
| 24.5 | 100.00 |

The average CPU is about a quarter of the limit of the compute size, which would fit well into the compute size of the database.

* + For **DTU purchasing model** and **vCore purchasing model** databases:

The maximum value shows that the database reaches the limit of the compute size. Do you need to move to the next higher compute size? Look at how many times your workload reaches 100 percent, and then compare it to your database workload objective.

SQLCopy

SELECT database\_name,

100\*((COUNT(database\_name) - SUM(CASE WHEN avg\_cpu\_percent >= 100 THEN 1 ELSE 0 END) \* 1.0) / COUNT(database\_name)) AS 'CPU Fit Percent',

100\*((COUNT(database\_name) - SUM(CASE WHEN avg\_log\_write\_percent >= 100 THEN 1 ELSE 0 END) \* 1.0) / COUNT(database\_name)) AS 'Log Write Fit Percent',

100\*((COUNT(database\_name) - SUM(CASE WHEN avg\_data\_io\_percent >= 100 THEN 1 ELSE 0 END) \* 1.0) / COUNT(database\_name)) AS 'Physical Data IO Fit Percent'

FROM sys.resource\_stats

WHERE start\_time > DATEADD(day, -7, GETDATE())

AND database\_name = 'sample' --remove to see all databases

GROUP BY database\_name;

These percentages are the number of samples your workload fit *under* the current compute size. If this query returns a value less than 99.9 percent for any of the three resource dimensions, your sampled average workload exceeded the limits. Consider either moving to the next higher compute size or use application-tuning techniques to reduce the load on the database.

1. **Note**
2. For elastic pools, you can monitor individual databases in the pool with the techniques described in this section. You can also monitor the pool as a whole. For information, see [**Monitor and manage an elastic pool**](https://learn.microsoft.com/en-us/azure/azure-sql/database/elastic-pool-overview?view=azuresql).

**Maximum concurrent requests**

To see the current number of concurrent requests, run this query on your user database:

SQLCopy

SELECT COUNT(\*) AS [Concurrent\_Requests]

FROM sys.dm\_exec\_requests AS R;

To analyze the workload of a database, modify this query to filter on the specific database you want to analyze. First, update the name of the database from MyDatabase to your desired database, then run the following query to find the count of concurrent requests in that database:

SQLCopy

SELECT COUNT(\*) AS [Concurrent\_Requests]

FROM sys.dm\_exec\_requests AS R

INNER JOIN sys.databases AS D

ON D.database\_id = R.database\_id

AND D.name = 'MyDatabase';

This is just a snapshot at a single point in time. To get a better understanding of your workload and concurrent request requirements, you'll need to collect many samples over time.

**Maximum concurrent login events**

You can analyze your user and application patterns to get an idea of the frequency of login events. You also can run real-world loads in a test environment to make sure that you're not hitting this or other limits we discuss in this article. There isn't a single query or dynamic management view (DMV) that can show you concurrent login counts or history.

If multiple clients use the same connection string, the service authenticates each login. If 10 users simultaneously connect to a database by using the same username and password, there would be 10 concurrent logins. This limit applies only to the duration of the login and authentication. If the same 10 users connect to the database sequentially, the number of concurrent logins would never be greater than 1.

**Note**

Currently, this limit does not apply to databases in elastic pools.

**Maximum sessions**

To see the number of current active sessions, run this query on your database:

SQLCopy

SELECT COUNT(\*) AS [Sessions]

FROM sys.dm\_exec\_connections;

If you're analyzing a SQL Server workload, modify the query to focus on a specific database. This query helps you determine possible session needs for the database if you're considering moving it to Azure. First, update the name of the database from MyDatabase to your desired database, then run the following query:

SQLCopy

SELECT COUNT(\*) AS [Sessions]

FROM sys.dm\_exec\_connections AS C

INNER JOIN sys.dm\_exec\_sessions AS S

ON (S.session\_id = C.session\_id)

INNER JOIN sys.databases AS D

ON (D.database\_id = S.database\_id)

WHERE D.name = 'MyDatabase';

Again, these queries return a point-in-time count. If you collect multiple samples over time, you'll have the best understanding of your session use.

You can get historical statistics on sessions by querying the [sys.resource\_stats](https://learn.microsoft.com/en-us/sql/relational-databases/system-catalog-views/sys-resource-stats-azure-sql-database?view=azuresqldb-current&preserve-view=true) catalog view and reviewing the active\_session\_count column.

**Calculate database and objects sizes**

The following query returns the size of your database (in megabytes):

SQLCopy

-- Calculates the size of the database.

SELECT SUM(CAST(FILEPROPERTY(name, 'SpaceUsed') AS bigint) \* 8192.) / 1024 / 1024 AS size\_mb

FROM sys.database\_files

WHERE type\_desc = 'ROWS';

The following query returns the size of individual objects (in megabytes) in your database:

SQLCopy

-- Calculates the size of individual database objects.

SELECT o.name, SUM(ps.reserved\_page\_count) \* 8.0 / 1024 AS size\_mb

FROM sys.dm\_db\_partition\_stats AS ps

INNER JOIN sys.objects AS o

ON ps.object\_id = o.object\_id

GROUP BY o.name

ORDER BY size\_mb DESC;

**Monitor query performance**

Slow or long running queries can consume significant system resources. This section demonstrates how to use dynamic management views to detect a few common query performance problems using the [sys.dm\_exec\_query\_stats](https://learn.microsoft.com/en-us/sql/relational-databases/system-dynamic-management-views/sys-dm-exec-query-stats-transact-sql?view=azuresqldb-current&preserve-view=true) dynamic management view. The view contains one row per query statement within the cached plan, and the lifetime of the rows are tied to the plan itself. When a plan is removed from the cache, the corresponding rows are eliminated from this view.

**Find top queries by CPU time**

The following example returns information about the top 15 queries ranked by average CPU time per execution. This example aggregates the queries according to their query hash, so that logically equivalent queries are grouped by their cumulative resource consumption.

SQLCopy

SELECT TOP 15 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;

**Monitor query plans for cumulative CPU time**

An inefficient query plan also may increase CPU consumption. The following example determines which query uses the most cumulative CPU in recent history.

SQLCopy

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 15

qs.plan\_handle,

qs.total\_worker\_time

FROM

sys.dm\_exec\_query\_stats AS 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;

**Monitor blocked queries**

Slow or long-running queries can contribute to excessive resource consumption and be the consequence of blocked queries. The cause of the blocking can be poor application design, bad query plans, the lack of useful indexes, and so on.

You can use the sys.dm\_tran\_locks view to get information about the current locking activity in database. For example code, see [sys.dm\_tran\_locks](https://learn.microsoft.com/en-us/sql/relational-databases/system-dynamic-management-views/sys-dm-tran-locks-transact-sql?view=azuresqldb-current&preserve-view=true). For more information on troubleshooting blocking, see [Understand and resolve Azure SQL blocking problems](https://learn.microsoft.com/en-us/azure/azure-sql/database/understand-resolve-blocking?view=azuresql).

**Monitor deadlocks**

In some cases, two or more queries may mutually block one another, resulting in a deadlock.

You can create an Extended Events trace a database in Azure SQL Database to capture deadlock events, then find related queries and their execution plans in Query Store. Learn more in [Analyze and prevent deadlocks in Azure SQL Database](https://learn.microsoft.com/en-us/azure/azure-sql/database/analyze-prevent-deadlocks?view=azuresql), including a lab to [Cause a deadlock in AdventureWorksLT](https://learn.microsoft.com/en-us/azure/azure-sql/database/analyze-prevent-deadlocks?view=azuresql#cause-a-deadlock-in-adventureworkslt). Learn more about the types of [resources that can deadlock](https://learn.microsoft.com/en-us/sql/relational-databases/sql-server-deadlocks-guide#deadlock_resources).