

Blocking

Last updated by | Vitor Tomaz | Feb 24, 2023 at 3:27 AM PST

Contents

- Issue:
- Investigation - Collect blocking details
 - Step 1 - Identify head blocker and blocked sessions
 - Step 2 - Long-running and uncommitted transactions
 - Step 3 - Status information about open transactions
 - Step 4 - Locks and the affected tables
 - Step 5 - Verify waitstats on the telemetry
- Mitigation - Troubleshoot and resolve blocking scenarios
 - Blocking caused by a normally running query with a long e...
 - Blocking caused by a sleeping SPID that has an uncommitt...
 - Blocking caused by a SPID whose corresponding client ap...
 - Blocking caused by an orphaned connection
- More Information - Additional DMV queries
 - Find Blocking (simplest way)
 - Show Blocker and Blocked Sessions
- Public Doc reference
- Internal TSG reference

Resolve blocking issues for Azure SQL Database

This content is in large parts copied over from the self-help article that is displayed to customers on the Azure portal. In addition, it has variations of the DMV queries that may provide some additional information including the lock details, and a link to related Kusto queries.


As an initial step, check the "Performance" section in ASC which might already have the relevant details.

Issue:


Blocking is an unavoidable characteristic of any relational database management system with lock-based concurrency. On SQL Server, blocking occurs when one session/spid holds a lock on a specific resource and a second session/spid attempts to acquire a conflicting lock type on the same resource. The duration and transaction context of a query determines how long its locks are held and, thereby, their impact on other queries:

- If the query is not executed within a transaction and no lock hints are used, the locks for `SELECT` statements will only be held on a resource at the time it is actually being read, not for the duration of the query. For `INSERT`, `UPDATE`, and `DELETE` statements, the locks are held for the duration of the query, to maintain data consistency and to allow the query to be rolled back if necessary.

- For queries executed within a transaction, the duration for which the locks are held are determined by the type of query, the transaction isolation level, and whether or not lock hints are used in the query.

Blocked queries can cause poor performance, including slow-running or long-running queries that contribute to excessive resource consumption. While the concepts of blocking are the same for SQL Server and Azure SQL Database, the default isolation level is different. For Azure SQL Databases, [READ COMMITTED SNAPSHOT](#)  is enabled by default, which avoids most of the blocking scenarios you would see with on-premise SQL Server.

See the sections below for initial steps and details that will help resolve your blocking issue.

In addition, review and work through [Understand and resolve Azure SQL Database blocking problems](#) . This complete troubleshooting article describes blocking in full detail and demonstrates step-by-step instructions on how to troubleshoot and resolve the most common blocking scenarios.

Investigation - Collect blocking details

The first troubleshooting task is to identify the main blocking session (head blocker). The second task is to then find the query and transaction that is causing the blocking and holding the blocking locks.

The following SQL queries will help you with these tasks. Run the queries in the affected database where you see the blocking; you can also copy the queries into the same script file and run them in one batch to capture a snapshot of the blocking situation.

Step 1 - Identify head blocker and blocked sessions

While the blocking issue is still occurring, run the following query to capture details about the head blocker and the blocked sessions.

```

SELECT
[HeadBlocker] =
CASE
    -- session has an active request, is blocked, but is blocking others
    -- or session is idle but has an open transaction and is blocking others
    WHEN r2.session_id IS NOT NULL AND (r.blocking_session_id = 0 OR r.session_id IS NULL) THEN '1'
    -- session is either not blocking someone, or is blocking someone but is blocked by another party
    ELSE ''
END,
[SessionID] = s.session_id,
[Login] = s.login_name,
[Database] = db_name(p.dbid),
[BlockedBy] = w.blocking_session_id,
[OpenTransactions] = r.open_transaction_count,
[Status] = s.status,
[WaitType] = w.wait_type,
[WaitTime_ms] = w.wait_duration_ms,
[WaitResource] = r.wait_resource,
[WaitResourceDesc] = w.resource_description,
[Command] = r.command,
[Application] = s.program_name,
[TotalCPU_ms] = s.cpu_time,
[TotalPhysicalIO_MB] = (s.reads + s.writes) * 8 / 1024,
[MemoryUse_KB] = s.memory_usage * 8192 / 1024,
[LoginTime] = s.login_time,
[LastRequestStartTime] = s.last_request_start_time,
[HostName] = s.host_name,
[QueryHash] = r.query_hash,
[BlockerQuery_or_MostRecentQuery] = txt.text
FROM sys.dm_exec_sessions s
LEFT OUTER JOIN sys.dm_exec_connections c ON (s.session_id = c.session_id)
LEFT OUTER JOIN sys.dm_exec_requests r ON (s.session_id = r.session_id)
LEFT OUTER JOIN sys.dm_os_tasks t ON (r.session_id = t.session_id AND r.request_id = t.request_id)
LEFT OUTER JOIN
(
    SELECT *, ROW_NUMBER() OVER (PARTITION BY waiting_task_address ORDER BY wait_duration_ms DESC) AS row_num
    FROM sys.dm_os_waiting_tasks
) w ON (t.task_address = w.waiting_task_address) AND w.row_num = 1
LEFT OUTER JOIN sys.dm_exec_requests r2 ON (s.session_id = r2.blocking_session_id)
LEFT OUTER JOIN sys.sysprocesses p ON (s.session_id = p.spid)
OUTER APPLY sys.dm_exec_sql_text (ISNULL(r.[sql_handle], c.most_recent_sql_handle)) AS txt
WHERE s.is_user_process = 1
AND (r2.session_id IS NOT NULL AND (r.blocking_session_id = 0 OR r.session_id IS NULL))
OR blocked > 0
ORDER BY [HeadBlocker] desc, s.session_id;

```

Sample output:

| HeadBlocker | SessionID | Login | Database | BlockedBy | OpenTransactions | Status | WaitType | WaitTime_ms | WaitResource | WaitResourceDesc | Command | Application |
|-------------|-----------|---------|----------------|-----------|------------------|----------|----------|-------------|-----------------|---|---------|--|
| 1 | 71 | holgerf | AdventureWorks | NULL | NULL | sleeping | NULL | NULL | NULL | NULL | NULL | Microsoft SQL Server Management Studio - Query |
| 2 | 76 | holgerf | AdventureWorks | 71 | 0 | running | LCK_M_UI | 31912 | PAGE: 9:1:28632 | pagelock fileid=1 pageid=28632 dbid=9 subresourc... | SELECT | Microsoft SQL Server Management Studio - Query |

| TotalCPU_ms | TotalPhysicalIO_MB | MemoryUse_KB | LoginTime | LastRequestStartTime | HostName | QueryHash | BlockerQuery_or_MostRecentQuery |
|-------------|--------------------|--------------|-------------------------|-------------------------|----------|--------------------|--|
| 2375 | 63 | 32 | 2022-09-27 14:23:40.577 | 2022-09-27 14:26:23.700 | holgerf | NULL | begin tran update person.person set FirstName = 'Arnold' where FirstName = 'Kevin' |
| 64 | 0 | 32 | 2022-09-27 14:25:16.453 | 2022-09-27 14:31:17.330 | holgerf | 0x43CD4686DA9886CB | select LastName from person.person with (updlock) where FirstName = 'Kevin' |

Step 2 - Long-running and uncommitted transactions

To catch long-running or uncommitted transactions, use the following query for identifying current open transactions. It will also indicate which databases are involved in each transaction.

```

SELECT
    [s_tst].[session_id],
    [database_name] = DB_NAME (s_tdt.database_id),
    [s_tdt].[database_transaction_begin_time],
    [sql_text] = [s_est].[text]
FROM sys.dm_tran_database_transactions [s_tdt]
INNER JOIN sys.dm_tran_session_transactions [s_tst] ON [s_tst].[transaction_id] = [s_tdt].[transaction_id]
INNER JOIN sys.dm_exec_connections [s_ec] ON [s_ec].[session_id] = [s_tst].[session_id]
CROSS APPLY sys.dm_exec_sql_text ([s_ec].[most_recent_sql_handle]) AS [s_est];

```

Sample output:

| | session_id | database_name | database_transaction_begin_time | sql_text |
|---|------------|----------------|---------------------------------|--|
| 1 | 71 | AdventureWorks | 2022-09-27 14:26:23.710 | begin tran update person.person set FirstName = 'Arnold' where FirstName = 'Kevin' |
| 2 | 71 | tempdb | NULL | begin tran update person.person set FirstName = 'Arnold' where FirstName = 'Kevin' |

Step 3 - Status information about open transactions

Use the following query to return status information about open transactions. Consider a transaction's current state, its transaction_begin_time, and other situational data to evaluate how it could contribute to the blocking situation.

```

SELECT tst.session_id, [database_name] = db_name(s.database_id)
, tat.transaction_begin_time
, transaction_duration_s = datediff(s, tat.transaction_begin_time, sysdatetime())
, transaction_type = CASE tat.transaction_type
    WHEN 1 THEN 'Read/write transaction'
    WHEN 2 THEN 'Read-only transaction'
    WHEN 3 THEN 'System transaction'
    WHEN 4 THEN 'Distributed transaction' END
, input_buffer = ib.event_info, tat.transaction_uow
, transaction_state = CASE tat.transaction_state
    WHEN 0 THEN 'The transaction has not been completely initialized yet.'
    WHEN 1 THEN 'The transaction has been initialized but has not started.'
    WHEN 2 THEN 'The transaction is active - has not been committed or rolled back.'
    WHEN 3 THEN 'The transaction has ended. This is used for read-only transactions.'
    WHEN 4 THEN 'The commit process has been initiated on the distributed transaction.'
    WHEN 5 THEN 'The transaction is in a prepared state and waiting resolution.'
    WHEN 6 THEN 'The transaction has been committed.'
    WHEN 7 THEN 'The transaction is being rolled back.'
    WHEN 8 THEN 'The transaction has been rolled back.' END
, transaction_name = tat.name, request_status = r.status
, azure_dtc_state = CASE tat.dtc_state
    WHEN 1 THEN 'ACTIVE'
    WHEN 2 THEN 'PREPARED'
    WHEN 3 THEN 'COMMITTED'
    WHEN 4 THEN 'ABORTED'
    WHEN 5 THEN 'RECOVERED' END
, tst.is_user_transaction, tst.is_local
, session_open_transaction_count = tst.open_transaction_count
, s.host_name, s.program_name, s.client_interface_name, s.login_name, s.is_user_process
FROM sys.dm_tran_active_transactions tat
INNER JOIN sys.dm_tran_session_transactions tst on tat.transaction_id = tst.transaction_id
INNER JOIN sys.dm_exec_sessions s on s.session_id = tst.session_id
LEFT OUTER JOIN sys.dm_exec_requests r on r.session_id = s.session_id
CROSS APPLY sys.dm_exec_input_buffer(s.session_id, null) AS ib;

```

Sample output:

Results

Messages

| session_id | database_name | transaction_begin_time | transaction_duration_s | transaction_type | input_buffer | transaction_uow | transaction_state | |
|------------|---------------|------------------------|-------------------------|------------------|------------------------|--|-------------------|---|
| 1 | 71 | AdventureWorks | 2022-09-27 14:26:23.700 | 1160 | Read/write transaction | begin tran update person.person set FirstName... | NULL | The transaction is active - has not been committ... |

| transaction_name | request_status | azure_dtc_state | is_user_transaction | is_local | session_open_transaction_count | host_name | program_name | client_interface_name | login_name | is_user_process |
|------------------|----------------|-----------------|---------------------|----------|--------------------------------|-----------|--|------------------------------|------------|-----------------|
| user_transaction | NULL | NULL | 1 | 1 | 1 | holgerl | Microsoft SQL Server Management Studio - Query | .Net SqlClient Data Provider | holgerl | 1 |

Step 4 - Locks and the affected tables

This query looks at the issue from a different angle. It starts with the locks that are involved in the blocking, then shows the affected table, the involved session IDs, and the wait status. You can filter on the table name to reduce the result further.

```
SELECT
    table_name = schema_name(o.schema_id) + '.' + o.name,
    wt.wait_duration_ms, wt.wait_type, wt.blocking_session_id, wt.resource_description,
    tm.resource_type, tm.request_status, tm.request_mode, tm.request_session_id
FROM sys.dm_tran_locks AS tm
INNER JOIN sys.dm_os_waiting_tasks as wt ON tm.lock_owner_address = wt.resource_address
LEFT OUTER JOIN sys.partitions AS p ON p.hobt_id = tm.resource_associated_entity_id
LEFT OUTER JOIN sys.objects o ON o.object_id = p.object_id OR tm.resource_associated_entity_id = o.object_id
WHERE resource_database_id = DB_ID()
--AND object_name(p.object_id) = '<table_name>';
```

Sample output:

| Results | | Messages | | | | | | | |
|---------|---------------|------------------|-----------|---------------------|---|---------------|----------------|--------------|--------------------|
| | table_name | wait_duration_ms | wait_type | blocking_session_id | resource_description | resource_type | request_status | request_mode | request_session_id |
| 1 | Person.Person | 1074348 | LCK_M_IU | 71 | pagelock fileid=1 pageid=28632 dbid=9 subresourc... | PAGE | WAIT | IU | 76 |

Step 5 - Verify waitstats on the telemetry

Use the Kusto queries in [Verifying waitstats](#) to confirm the findings from the previous steps. The telemetry may help to narrow down on timings, e.g. when a blocking situation has started and completed, and to check on patterns within past workloads.

Mitigation - Troubleshoot and resolve blocking scenarios

Blocking caused by a normally running query with a long execution time

- The solution to this type of blocking problem is to look for ways to optimize the head blocker, the blocking query. The issue is usually a performance problem. If you've confirmed this to be the cause, troubleshoot the specific slow-running query. The performance tools in the Azure portal for Azure SQL Database will be a good starting point, especially the Query Performance Insight tool.
- If you have a long-running query that is blocking other users and can't be optimized, consider moving it from an OLTP environment to a dedicated reporting system.

Blocking caused by a sleeping SPID that has an uncommitted transaction

- This type of blocking can often be identified by a session that's sleeping or awaiting a command, yet whose transaction nesting level (see `SELECT @@TRANSCOUNT, column open_transaction_count` in

`sys.dm_exec_requests`) is greater than zero. This can occur if the application experiences a query time-out or issues a cancel without also issuing the required number of ROLLBACK and/or COMMIT statements. A time-out or cancellation doesn't automatically roll back or commit the transaction. The application is responsible for this, as Azure SQL Database can't assume that an entire transaction must be rolled back due to a single query being canceled.

- See [Detailed blocking scenarios](#)  for further details about resolving this type of issue.

Blocking caused by a SPID whose corresponding client application didn't fetch all result rows to completion

- After sending a query to the server, all applications must immediately fetch all result rows to completion. If an application doesn't fetch all result rows, locks can be left on the tables, blocking other users. If the application can't be configured to do so, you may be unable to resolve the blocking problem.
- The impact of this scenario is reduced when read committed snapshot is enabled on the database, which is the default configuration in Azure SQL Database.
- To avoid the problem, you can restrict poorly behaved applications to a reporting or a decision-support database, separate from the main OLTP database.

Blocking caused by an orphaned connection

- If the client application crashes, disconnects from the network, or if the client workstation is restarted, the network session to the server may not be immediately canceled under some conditions. From the Azure SQL Database perspective, the client still appears to be present, and any locks acquired may still be retained.
- **Resolution:** If the client application has disconnected without appropriately cleaning up its resources, you can terminate the session by using the Transact-SQL `KILL` command. The `KILL` command takes the session ID value as input. For example, to kill session ID 99, issue the SQL command: `KILL 99`.

More Information - Additional DMV queries

The same information can be collected through the DMV queries in the [Investigation](#) section above. But some of these here might be easier to copy/paste through a Teams chat.

Find Blocking (simplest way)

Can be used if the issue is happening while you are with the customer on a live session). These options above will show you the blocking session ID (SPID). Focus on why that blocking spid is taking so long to run, hogging the resources, and blocking the other spid(s)

```
-- head blocker indicated in column: BlkBy
exec sp_who2

-- or: head blocker indicated in column: blocking_session_id
select * from sys.dm_exec_requests

-- or: active requests and their SQL text
SELECT session_id, blocking_session_id, wait_type, text
FROM sys.dm_exec_requests
CROSS APPLY sys.dm_exec_sql_text(sql_handle)
WHERE session_id > 50
```

Show Blocker and Blocked Sessions

This is a variation of the query from Step 4 above, but with additional details including the SQL text:

```
select
  t2.blocking_session_id as [blocker sid] -- spid of blocker
  , t1.request_session_id as [blocked sid] -- spid of waiter
  , t1.resource_type as [lock type]
  , db_name(resource_database_id) as [database]
  , t1.resource_associated_entity_id as [blocked object]
  , t1.request_mode as [lock req] -- lock requested
  , t2.wait_duration_ms as [wait time]
  , (select text
    from sys.dm_exec_requests as r --- get sql for waiter
    cross apply sys.dm_exec_sql_text(r.sql_handle)
    where r.session_id = t1.request_session_id) as blocked_batch
  , (select substring(qt.text,r.statement_start_offset/2,
    (case when r.statement_end_offset = -1
    then len(convert(nvarchar(max), qt.text)) * 2
    else r.statement_end_offset end - r.statement_start_offset)/2)
    from sys.dm_exec_requests as r
    cross apply sys.dm_exec_sql_text(r.sql_handle) as qt
    where r.session_id = t1.request_session_id) as blocked_stmt --- this is the statement executing right
  , (select text from sys.sysprocesses as p --- get sql for blocker
    cross apply sys.dm_exec_sql_text(p.sql_handle)
    where p.spid = t2.blocking_session_id) as blocker_stmt
from
  sys.dm_tran_locks as t1,
  sys.dm_os_waiting_tasks as t2
where
  t1.lock_owner_address = t2.resource_address
ORDER BY [blocker sid]
```

Sample output:

| Results | Messages | | | | | | | | | |
|-------------|-------------|-----------|----------------|-------------------|----------|-----------|-------------------------------------|--------------------------------------|--|--|
| blocker sid | blocked sid | lock type | database | blocked object | lock req | wait time | blocked_batch | blocked_stmt | blocker_stmt | |
| 1 | 71 | PAGE | AdventureWorks | 72057594062635008 | IU | 1341721 | select LastName from person.pers... | select LastName from person.perso... | begin tran update person.person set FirstName... | |

Public Doc reference

- [Understand and resolve Azure SQL Database blocking problems](#) 
- [READ COMMITTED SNAPSHOT](#) 

Internal TSG reference

- [Troubleshooting Blocking](#)

How good have you found this content?

