Active Transactions

Last updated by | Amie Coleman | Nov 22, 2022 at 1:51 AM PST

Contents

Issue

The purpose of this document is to provide guidance/a walk through on how you can identify Active Transactions in SQL and how to troubleshoot them.

Background

A transaction is a single unit of work, that if successful, will be committed (all data modifications made during that transaction become permanent in the database) and if errors are encountered or the transaction is cancelled, will be rolled back (all modifications made during that transaction are erased). An Active Transaction simply describes a transaction that is still open or active (i.e., it hasn't been committed or rolled back).

Active Transactions can be the cause of several issues, including performance issues, space issues (transaction log usage), and blocking.

Investigation/Analysis

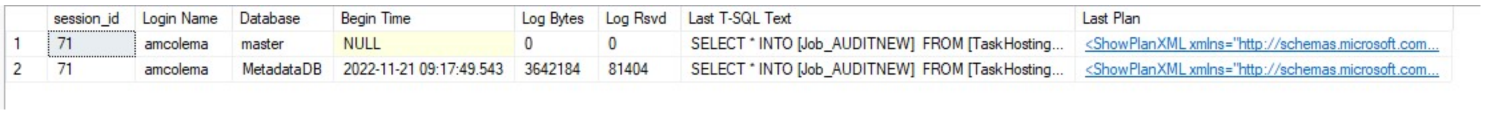
The following T-SQL script will return the below information on the active transactions and needs to be executed against the customers environment (using SQL Server Management Studio, for example).

- Session ID
- Login Name
- Database Context
- Transaction Begin Time
- Number of log records generated by the transaction
- Amount of log space consumed by those log records
- Volume of log space reserved in case of transaction roll back
- Last T-SQL executed in the context of the transaction

- Last Execution Plan that was executed (only for currently executing queries)

```
SELECT
    [s_tst].[session_id],
    [s_es].[login_name] AS [Login Name],
    DB_NAME (s_tdt.database_id) AS [Database],
    [s_tdt].[database_transaction_begin_time] AS [Begin Time],
    [s_tdt].[database_transaction_log_bytes_used] AS [Log Bytes],
    [s_tdt].[database_transaction_log_bytes_reserved] AS [Log Rsvd],
    [s_est].text AS [Last T-SQL Text],
    [s_eqp].[query_plan] AS [Last Plan]
FROM sys.dm_tran_database_transactions [s_tdt]
JOIN sys.dm_tran_session_transactions [s_tst] ON [s_tst].[transaction_id] = [s_tdt].[transaction_id]
JOIN sys.[dm_exec_sessions] [s_es] ON [s_es].[session_id] = [s_tst].[session_id]
JOIN sys.dm_exec_connections [s_ec] ON [s_ec].[session_id] = [s_tst].[session_id]
LEFT OUTER JOIN sys.dm_exec_requests [s_er] ON [s_er].[session_id] = [s_tst].[session_id]
CROSS APPLY sys.dm_exec_sql_text ([s_ec].[most_recent_sql_handle]) AS [s_est]
OUTER APPLY sys.dm_exec_query_plan ([s_er].[plan_handle]) AS [s_eqp]
ORDER BY [Begin Time] ASC;
```

Example output



	session_id	Login Name	Database	Begin Time	Log Bytes	Log Rsvd	Last T-SQL Text	Last Plan
1	71	amcolema	master	NULL	0	0	SELECT * INTO [Job_AUDITNEW] FROM [TaskHosting...	<ShowPlanXML xmlns="http://schemas.microsoft.com...
2	71	amcolema	MetadataDB	2022-11-21 09:17:49.543	3642184	81404	SELECT * INTO [Job_AUDITNEW] FROM [TaskHosting...	<ShowPlanXML xmlns="http://schemas.microsoft.com...

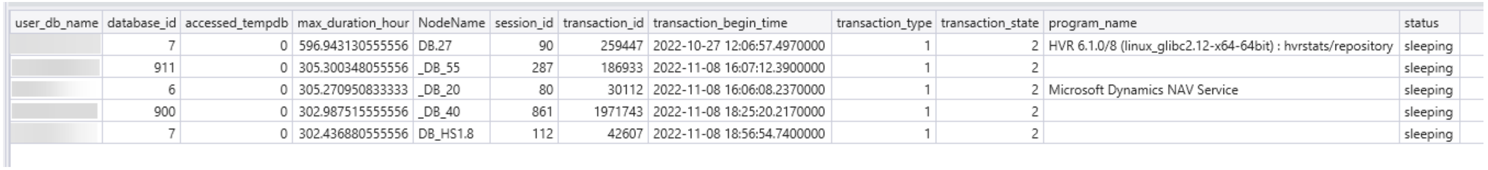
Alternatively, utilise the below T-SQL script to return any active transaction on the system and provide detailed information about the transaction, the user session, the application that submitted it, and the query that started it, plus much more.

```
SELECT
  GETDATE() as now,
  DATEDIFF(SECOND, transaction_begin_time, GETDATE()) as tran_elapsed_time_seconds,
  st.session_id,
  txt.text,
  *
FROM
  sys.dm_tran_active_transactions at
  INNER JOIN sys.dm_tran_session_transactions st ON st.transaction_id = at.transaction_id
  LEFT OUTER JOIN sys.dm_exec_sessions sess ON st.session_id = sess.session_id
  LEFT OUTER JOIN sys.dm_exec_connections conn ON conn.session_id = sess.session_id
    OUTER APPLY sys.dm_exec_sql_text(conn.most_recent_sql_handle)  AS txt
ORDER BY
  tran_elapsed_time_seconds DESC;
```

Internal Telemetry - Kusto/ASC

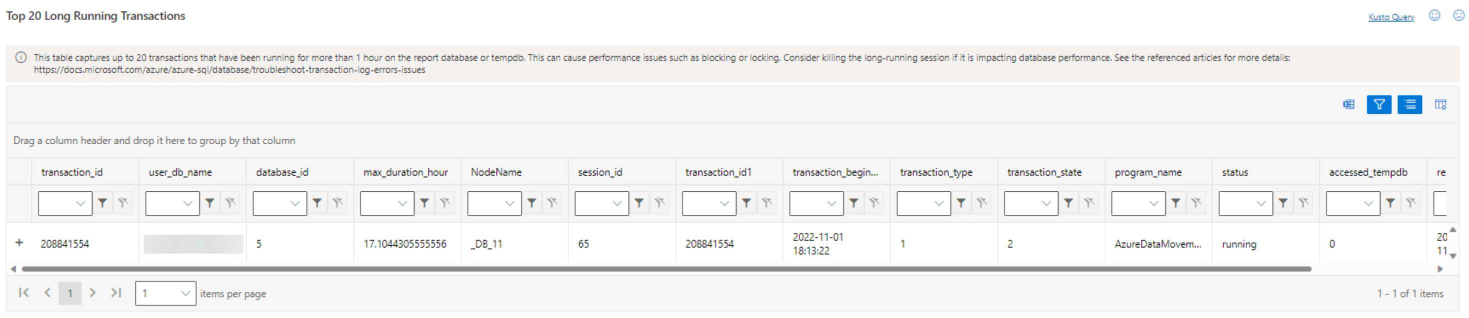
Additionally, you can utilise our internal telemetry using the following Kusto query/ASC Report to return Active Transaction information. Note that the **preferred** method would be to review the Active Transactions with the customer directly against their database during a remote session, using the aforementioned T-SQL scripts. While we are collecting data on transactions in the MonDmTranActiveTransactions table, as with all telemetry, it is not real-time.

```
let appname_name_input = '';
let startTime = datetime(2022-11-21 00:00:00);
let endTime = datetime(2022-11-21 11:00:00);
MonDmTranActiveTransactions
| where TIMESTAMP >= startTime and TIMESTAMP < endTime
| where AppName !startswith 'b-' and AppName !startswith 'v-'
| where AppName contains appname_name_input
| where session_id != -1
| extend duration_hour = (end_utc_date - transaction_begin_time) / time(1h)
| summarize max_duration_hour = arg_max(duration_hour,
            NodeName,
            session_id,
            transaction_id,
            transaction_begin_time,
            transaction_type,
            transaction_state,
            program_name,
            status
) by user_db_name,
database_id,
accessed_tempdb
| sort by max_duration_hour desc
| limit 500
```



user_db_name	database_id	accessed_tempdb	max_duration_hour	NodeName	session_id	transaction_id	transaction_begin_time	transaction_type	transaction_state	program_name	status
[redacted]	7	0	596.943130555556	DB.27	90	259447	2022-10-27 12:06:57.4970000	1	2	HVR 6.1.0/8 (linux_glibc2.12-x64-64bit) : hvrstats/repository	sleeping
[redacted]	911	0	305.300348055556	_DB_55	287	186933	2022-11-08 16:07:12.3900000	1	2		sleeping
[redacted]	6	0	305.270950833333	_DB_20	80	30112	2022-11-08 16:06:08.2370000	1	2	Microsoft Dynamics NAV Service	sleeping
[redacted]	900	0	302.987515555556	_DB_40	861	1971743	2022-11-08 18:25:20.2170000	1	2		sleeping
[redacted]	7	0	302.436880555556	DB_HS1.8	112	42607	2022-11-08 18:56:54.7400000	1	2		sleeping

In ASC, go to Performance > Blocking & Deadlocking report which will show you any "Long Running Transactions" (this queries the MonDmTranActiveTransactions table)



Top 20 Long Running Transactions

This table captures up to 20 transactions that have been running for more than 1 hour on the report database or tempdb. This can cause performance issues such as blocking or locking. Consider killing the long-running session if it is impacting database performance. See the referenced articles for more details: https://docs.microsoft.com/azure/azure-sql/database/troubleshoot-transaction-log-errors-issues

Drag a column header and drop it here to group by that column

transaction_id	user_db_name	database_id	max_duration_hour	NodeName	session_id	transaction_id1	transaction_begin...	transaction_type	transaction_state	program_name	status	accessed_tempdb	re
208841554	[redacted]	5	17.1044305555556	_DB_11	65	208841554	2022-11-01 18:13:22	1	2	AzureDataMovem...	running	0	20 11

Mitigation

Impact considerations

Most of the time, the active transaction will lead back to the customers application or user queries (ultimately, the customers responsibility). We briefly mentioned the impact above, but to expand on this:

1. Performance issues
   Active transactions can sometimes cause performance issues if they are particularly intensive or resource consuming and run for prolonged periods of time
2. Space issues
   Consider how an active transaction will not allow log truncation. Current Log Reuse Wait can be checked

using the log_reuse_wait_desc column in sys.databases. If an active transaction is preventing log truncation, you will see ACTIVE_TRANSACTION as the wait

```
SELECT log_reuse_wait_desc
FROM sys.databases
WHERE name = 'MyDBName';
```

3. Blocking
   An active transaction may be holding locks for extended periods of time unnecessarily and causing blocking
4. TempDB Growth
   Depending on the nature of the statement/query being executed, you may see TempDB impact/Growth

If there is impact: The simplest method for mitigation is to KILL the active transaction to start the rollback process. The decision to kill the transaction will ultimately be up to the customer once they have reviewed the query being executed and weighed up the impact of letting it run (if it is a genuine query and requires more time) or killing it to prevent potential impact.

KILL command

The KILL command accepts a session_id, which you would have gathered using the earlier T-SQL scripts. Note that to run this command, it requires the KILL DATABASE CONNECTION permission (the server-level principal login has the KILL DATABASE CONNECTION).

For example, to KILL session ID 53:

```
KILL 53;
GO
```

Obtain status report for the session that was killed using WITH STATUSONLY:

```
KILL 53 WITH STATUSONLY;
GO
```

Result:

```
--This is the progress report.
spid 53: Transaction rollback in progress. Estimated rollback completion: 70% Estimated time left: 50 seconds.
```

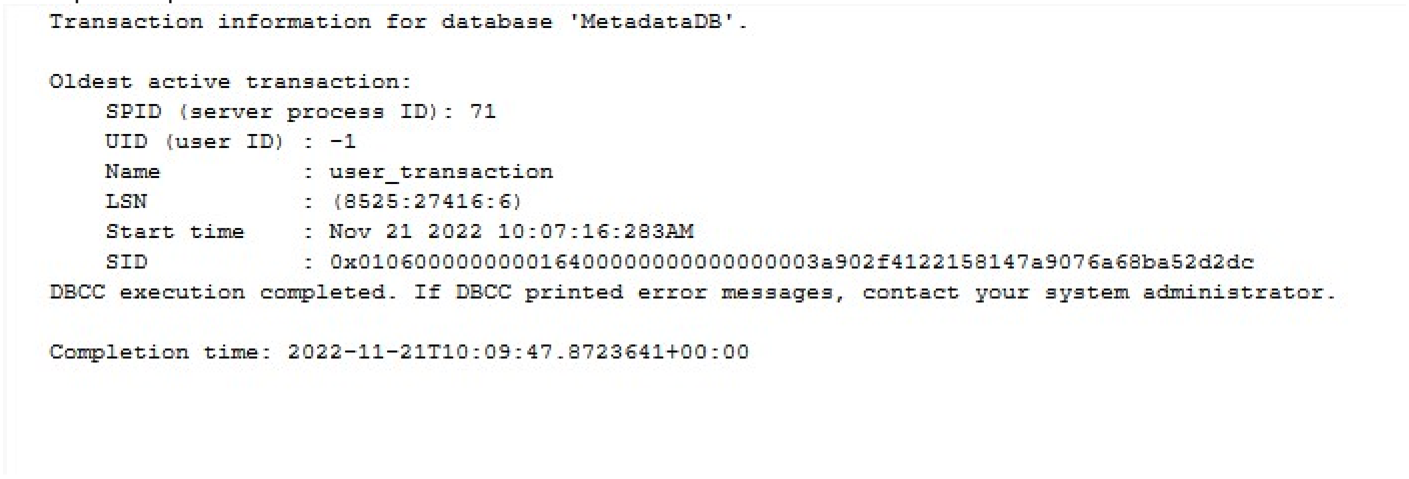
If no impact is observed, the customer may decide that they want to leave the transaction running until completion.

More Information

We reviewed any active transaction above, but if you need to narrow this down to find the **single oldest active transaction** you can use DBCC OPENTRAN

```
DBCC OPENTRAN
```

Example output



```
Transaction information for database 'MetadataDB'.

Oldest active transaction:
    SPID (server process ID): 71
    UID (user ID) : -1
    Name          : user_transaction
    LSN           : (8525:27416:6)
    Start time    : Nov 21 2022 10:07:16:283AM
    SID           : 0x01060000000001640000000000000000003a902f4122158147a9076a68ba52d2dc
DBCC execution completed. If DBCC printed error messages, contact your system administrator.

Completion time: 2022-11-21T10:09:47.8723641+00:00
```

Public Doc Reference

Transactions
sys.dm_tran_active_transactions
KILL command
KILL WITH STATUSONLY option DBCC OPENTRAN

How good have you found this content?