

# CDC Change Tables cleanup slow or not working

Last updated by | Holger Linke | Dec 7, 2022 at 9:39 AM PST

## Contents

- [Issue](#)
- [Investigation / Analysis](#)
  - [Confirm the configured retention period](#)
  - [Check the start\\_lsn of the capture instance and relate it to i...](#)
  - [Check what the earliest startlsn is on the "\\_CT" table](#)
  - [Compare the startlsn of CDC.changetables with the oldest ...](#)
  - [Check for index fragmentation on "\\_CT" table](#)
- [Mitigation](#)
  - [Possible causes](#)
  - [Mitigation 1: Change the Delete batch size and the cleanu...](#)
  - [Mitigation 2: Explicitly run cleanup for the capture instance](#)
  - [Mitigation 3: Rebuild indexes if fragmentation is present](#)
  - [Mitigation 4: Open an IcM and involve the Product Group](#)
- [More Information](#)
- [Public Doc Reference](#)

## Issue

The customer has configured Change Data Capture (CDC) on Managed Instance. On the data replication side (the CDC capture job), everything appears to be working normally. But when monitoring the change tracking system tables ( `tablename_CT` ), they notice that the size is increasing over time and never reducing. The CDC cleanup job runs on its job schedule, but is apparently unable to remove the rows from the "\_CT" system tables.

When the CSS case was opened, the "\_CT" contained 1.2 billion rows (1.250.000.000), consuming almost 1 TB, and the Managed Instance was getting close to its maximum storage capacity.


## Investigation / Analysis

You need to find out the following details to assess if there is a problem at the CDC metadata side:

- What is the actual retention period?
- What is the current `start_lsn` of the capture instance? (this equals the cutoff from the latest cleanup)
- To what datetime does this `start_lsn` relate - when was that change captured?
- Does the "\_CT" table contain rows with an LSN that is older than the capture instance's `start_lsn` ?

You also need to cross-check if the issue might be related to a plain performance issue, e.g. due to index fragmentation on the "\_CT" table.

## Confirm the configured retention period

Get the CDC configuration parameters by executing [sys.sp\\_cdc\\_help\\_jobs](#) :

```
-- display details on the capture and cleanup job - includes retention:
exec sys.sp_cdc_help_jobs
```

```
-- sample output:
```

job_id	job_type	job_name	maxtrans	maxscans	continuous	pollinginterval	retention	threshold
5C7359FB-52...	capture	cdc.dbname_capture	500	10	1	5	0	0
FEDB9B08-F9...	cleanup	cdc.dbname_cleanup	0	0	0	0	4320	5000

The retention period of the CDC metadata is set through the cleanup job. Here it is the default of 4320 minutes = 72 hours = 3 days, which you can see on the `retention` column. Also note the value that is returned on the `threshold` column; it represents the Delete Batch Size for the cleanup, with a default value of 5000.

## Check the start\_lsn of the capture instance and relate it to its entry date

The customer should know the name of the source table, capture instance, and "\_CT" table:

```
-- check start_lsn for affected table - set capture_instance = 'dbo_tablename'
declare @startlsn varbinary(50)
select @startlsn = start_lsn from CDC.change_tables where capture_instance = 'dbo_nameofsourcetable'
select @startlsn, * from CDC.change_tables where capture_instance = 'dbo_nameofsourcetable'
```

```
-- relate the start_lsn to system entry time
select * from CDC.lsn_time_mapping where start_lsn >= @startlsn
```

```
-- sample output:
```

```
-- CDC.change_tables:
(No column name)      object_id  version  source_object_id  capture_instance      start_lsn      end_l
-----
0x00271CF800002FE0001  2045966365 0        2020202247        dbo_nameofsourcetable 0x00271CF800002FE0001 NULL
```

```
-- CDC.lsn_time_mapping:
start_lsn      tran_begin_time      tran_end_time      tran_id      tran_begin_lsn
-----
0x00271CF800002FE0001  2022-04-09 08:40:03.780  2022-04-09 08:40:03.780  0x00      0x0000000000000000
0x00271D08000044A0001  2022-04-09 08:45:04.723  2022-04-09 08:45:04.723  0x00      0x0000000000000000
(...)
```

```
-- the query was executed on 2022-04-12 so it confirms the 3-day retention from above
```

## Check what the earliest start\_lsn is on the "\_CT" table

```

select top 100 __$start_lsn, __$operation /* add more columns if you want to see the changes themselves*/
from CDC.dbo_PRODUCT_USAGE_CT
order by __$start_lsn

-- sample output:
__$start_lsn      __$operation
-----
0x001B815E000073C8000D 3
0x001B815E000073C8000D 4
0x001B815E000073C8000D 3
0x001B815E000073C8000D 4
(...)

```

## Compare the start\_lsn of CDC.change\_tables with the oldest value from "\_CT" table

This will answer the question: have all the changes older than the retention been deleted from the "\_CT" table?

The start\_lsn is the Log Sequence Number (LSN) from the transaction log which is hosting CDC. LSNs are steadily increasing on the transaction log. The LSNs consist of three segments and are read from left to right.

In this example, the "1B815E" of the "\_CT" table is significantly lower than the "271CF8" from CDC.change\_tables :

0x001B815E000073C8000D	1B815E:73C8:D
0x00271CF800002FE00001	271CF8:2FE0:1

This means that the "\_CT" table has rows which are much older than the cutoff LSN (0x00271CF800002FE00001) set by the cleanup job on CDC.change\_tables .

## Check for index fragmentation on "\_CT" table

Use the query for identifying index fragmentation from the [Indexes](#) page.

Sample query:

```

-- run in CDC-enabled database and check for tables names ending with "_CT"
SELECT DB_NAME() AS DBName
,OBJECT_NAME(ps.object_id) AS TableName
,i.name AS IndexName
,ips.index_type_desc
,ips.avg_fragmentation_in_percent
FROM sys.dm_db_partition_stats ps
INNER JOIN sys.indexes i
ON ps.object_id = i.object_id
AND ps.index_id = i.index_id
CROSS APPLY sys.dm_db_index_physical_stats(DB_ID(), ps.object_id, ps.index_id, null, 'LIMITED') ips
ORDER BY ips.avg_fragmentation_in_percent DESC

```

Recommend an index rebuild to the customer if the fragmentation is high.

## Mitigation

From the investigation, the CDC.dbo\_tablename\_CT table has LSNs that are a lot older than the current watermark/start\_lsn of the capture instance. It looks as if the cleanup has adjusted the cutoff date for the

capture instance in `CDC.change_tables` , but then failed to delete the outdated entries from the "\_CT" table.

## Possible causes

One possible reason is the Delete Batch size for the cleanup. From the `sp_cdc_help_jobs` output above, you can see that the default is 5000. If there are literally billions of rows in the "\_CT" table, it requires a lot of iterations on the Delete loop to remove all the outdated entries.

Another possible reason is that the cleanup job runs in a round-robin fashion, removing rows from all "\_CT" tables on the database. If there are a lot of CDC-enabled tables, it might take a while to come around to work on the large tables.

## Mitigation 1: Change the Delete batch size and the cleanup job schedule

### Delete Batch size:

Use [sys.sp\\_cdc\\_change\\_job](#)  to modify the `threshold` of the CDC cleanup job:

```
-- run in CDC-enabled database
EXECUTE sys.sp_cdc_change_job
    @job_type = N'cleanup',
    @threshold = 4999;
GO

-- confirm the change - check "threshold" value for cleanup job:
exec sys.sp_cdc_help_jobs
GO
```

### Recommendations:


- The `@threshold` is the Delete Batch size; the Delete will run in a loop, removing `@threshold` number of rows in each iteration.
- In a first attempt, try using a value smaller than 5000 to avoid lock escalation on the "\_CT" table (see [Log Reader and Distribution Cleanup blocking each other](#) for further details).
- If this is too slow, and if there are no new incoming changes: then you might try running with a much higher `@threshold` value 50000 or 100000. Be aware that this might block the \_CT table - but it might be your only chance to cleanup a very large "\_CT" table.

### Job Schedule:

Another issue might be coming from the default configuration of the cleanup job schedule. It usually runs once per day at 02:00 in the morning, and finishes once all "\_CT" tables have been worked on. But depending on the workload pattern and when rows pass the retention, this might waste up to a day for the cleanup process to progress its work.

You could reschedule the cleanup job to execute more often, e.g. every 4 or 6 hours, or at other times that do not interfere with OLTP workload on the database.

## Mitigation 2: Explicitly run cleanup for the capture instance

To speed up the cleanup for a specific table, you can try to explicitly remove the outdated \_CT entries by calling the stored procedure [sp\\_cdc\\_cleanup\\_change\\_table](#) . This procedure will take the start\_lsn from the capture instance in `CDC.change_tables` and then try to delete everything that has a lower LSN.

-- Note that the sample script on the [sp\\_cdc\\_cleanup\\_change\\_table](#) public article is outdated, syntax has changed  
 -- The following sample command applies to Managed Instance:

```
declare @retcode int;
EXEC @retcode = sys.sp_cdc_cleanup_change_table
    @capture_instance = 'dbo_tablename',
    @low_water_mark = NULL,
    @threshold = 4000
SELECT @retcode; -- 0 (success) or 1 (failure)
```

## Recommendations:

- The `@threshold` is the Delete Batch size; the Delete will run in a loop, removing `@threshold` number of rows in each iteration.
- In a first attempt, try using a value smaller than 5000 to avoid lock escalation on the "\_CT" table (see [Log Reader and Distribution Cleanup blocking each other](#) for further details).
- If this is too slow, and if there are no new incoming changes: then you might try running with a much higher `@threshold` value 50000 or 100000. Be aware that this might block the \_CT table - but it might be your only chance to cleanup a very large "\_CT" table.

If this requires not just a one-time mitigation, but is a recurring issue, then the [CDC Change Tables cleanup custom procedure](#) is the better mitigation option.

## Mitigation 3: Rebuild indexes if fragmentation is present

Run an index rebuild on the affected "\_CT" tables if the fragmentation was found to be high. This can significantly improve the Delete performance, because the cleanup runs in a loop executing "DELETE TOP @threshold" commands. If the execution plan for this Delete operation is non-optimal because of index fragmentation, the cleanup could waste a lot of time on each loop iteration.

Note though that the index rebuild impacts the performance of the capture job or might block its execution. Please make sure to run the index maintenance outside of peak hours.

## Mitigation 4: Open an IcM and involve the Product Group

If Mitigation #1 is too slow or if you are stuck on the investigation, you should involve the PG to get assistance quickly. If the "\_CT" table continues to grow, the Managed Instance might run out of storage space eventually, thus creating a much bigger issue.

In one case, the PG had assessed that there were no recent changes on the affected "\_CT" table, and executed a Truncate Table on it from the backend. This is a very dangerous thing to do though; if done inappropriately, it would lead to data loss on the CDC topology.

## More Information

See article [CDC cleanup in Azure SQL Database \(Preview\)](#). [🔗](#)

In Azure SQL Database, a change data capture scheduler takes the place of the SQL Server Agent that invokes stored procedures to start periodic capture and cleanup of the change data capture tables. The scheduler runs capture and cleanup automatically within SQL Database, without any external dependency for reliability or performance.

Users still have the option to run capture and cleanup manually on demand using the [sp\\_cdc\\_scan](#) and [sp\\_cdc\\_cleanup\\_change\\_tables](#) procedures.

Azure SQL Database includes two dynamic management views to help you monitor change data capture: [sys.dm\\_cdc\\_log\\_scan\\_sessions](#) and [sys.dm\\_cdc\\_errors](#).

## Public Doc Reference

[What is change data capture \(CDC\)](#)

[sys.sp\\_cdc\\_cleanup\\_change\\_table \(Transact-SQL\)](#)

**How good have you found this content?**

