

Azure SQL Database or SQL Managed Instance Database used data space is much larger than expected

 techcommunity.microsoft.com/t5/azure-database-support-blog/azure-sql-database-or-sql-managed-instance-database-used-data/ba-p/2162130

February 25, 2021

[Skip to footer content](#)

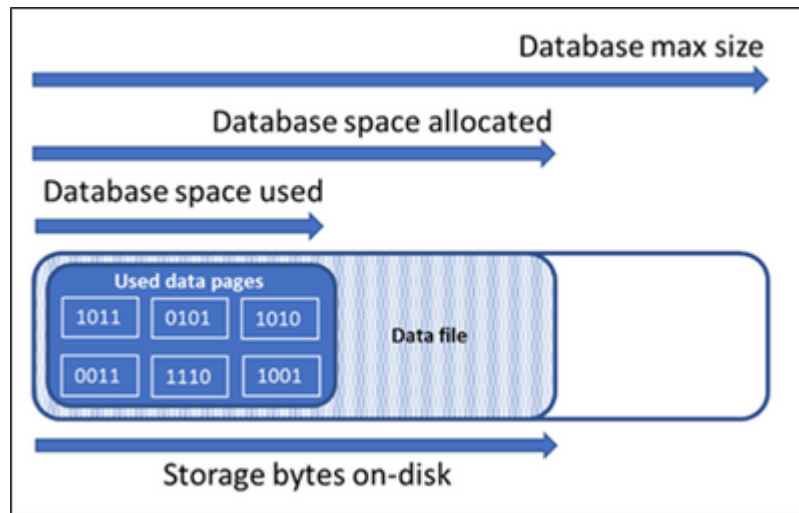
The data space used in an Azure SQL database or SQL Managed Instance database can be larger than expected - and on occasions significantly larger than expected – when compared with the actual number of records in the individual tables. This can lead to the impression of a problem with the database storage itself. However, this is almost certainly never the case and the issue can be resolved by carrying out a few maintenance procedures.

Storage space types for a database

To understand why this happens we should first review the different types of storage space used to manage the file space of a database:

- **Used Data space** - This is the amount of space used to store the database data, which is stored in 8 KB pages. Generally, the space used increases and decreases due to inserts and deletes respectively. In some cases, the space used does not change on inserts or deletes depending on the amount and pattern of data involved in the operation and any fragmentation. For example, deleting one row from every data page does not necessarily decrease the space used.
- **Allocated Data space** - The amount of formatted file space made available for storing database data. The amount of space allocated grows automatically, but never decreases after deletes. This behavior ensures that future inserts are faster since space does not need to be reformatted.
- **Data space allocated but unused** - The difference between the amount of data space allocated and data space used. This quantity represents the maximum amount of free space that can be reclaimed by shrinking database data files.
- **Data max size** - The maximum amount of space that can be used for storing database data.

The following diagram illustrates the relationship between the different types of storage space for a database:



Maintenance plan

Now we know that the used data space does not always change when inserts and deletes are performed and therefore can be greater than what could be expected when considering the number of records in the tables. How do we resolve this? This can be achieved by following the maintenance steps below to reduce index fragmentation, cleaning up any ghost records and then cleaning the Persisted Version Store:

1) Index fragmentation

Fragmentation exists when indexes have pages in which the logical ordering within the index, based on the key value of the index, does not match the physical ordering inside the index pages. The following example finds the average fragmentation percentage of all indexes in the Sales.SalesOrderDetail table in the AdventureWorks2012 database:

```
SELECT a.index_id, name, avg_fragmentation_in_percent, fragment_count,
avg_fragment_size_in_pages FROM sys.dm_db_index_physical_stats
(DB_ID('AdventureWorks2012'), object_id('Sales.SalesOrderDetail'), NULL, NULL,
NULL) AS a
JOIN sys.indexes AS b ON a.object_id = b.object_id AND a.index_id = b.index_id
```

The following links detail how to rebuild indexes to reduce the fragmentation (the second link includes an index and statistics maintenance script you can download):

2) Ghost Records

Ghost records are records that are deleted from a leaf level of an index page but aren't physically removed from the page. Instead, the record is marked as ghosted meaning to be deleted. This means that the row stays on the page, but the row header is modified to indicate the row is a confirmed ghost record. The reason behind this is to optimize performance during a delete operation. Ghosts are necessary for row-level locking, but also necessary for snapshot isolation where we need to maintain the older versions of rows. The number of ghost records can build up in a database until they are cleaned. The database engine runs a ghost cleanup process in the background that sometime after the delete transaction is committed, physically removes ghosted records from pages.

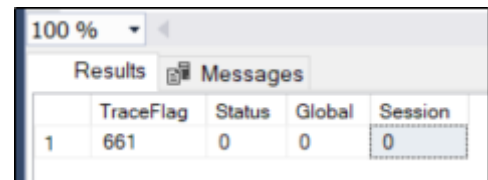
It is also possible the ghost cleanup process is disabled (not generally recommended). Disabling the ghost cleanup process can cause your database to grow unnecessarily large and can lead to performance issues. You can check if the ghost cleanup process is disabled by running the following command:

```
DBCC Tracestatus (661)
```

If the “Status” flag is set to 0, then this indicates that the ghost clean-up is enabled. If “Status” flag is set 1, then the process has been disabled.

To confirm if there are ghost records on your database execute this T-SQL:

```
SELECT sum(ghost_record_count)
total_ghost_records, db_name(database_id)
FROM sys.dm_db_index_physical_stats (NULL, NULL,
NULL, NULL, 'SAMPLED')
GROUP BY database_id
ORDER BY total_ghost_records DESC
```



	TraceFlag	Status	Global	Session
1	661	0	0	0

If there are ghost records, you can delete the ghost records manually from the database by executing an index rebuild. This process reclaims disk space by compacting the pages based on the specified or existing fill factor setting and reorders the index rows in adjoining pages.

Another option is to use `sp_clean_db_file_free_space` to clean all pages in all files of the database. For example, this T-SQL will clean the ghost records from the AdventureWorks2012 database:

```
USE master;
GO
EXEC sp_clean_db_free_space @dbname = N'AdventureWorks2012';
```

For more details about the Ghost clean process refer to the following guide: [Ghost cleanup process guide - SQL Server | Microsoft Docs](#)

3) Persisted Version Store (PVS)

PVS is a database engine mechanism for persisting the row versions generated in the database itself instead of the traditional tempdb version store. PVS enables resource isolation and improves availability of readable secondaries. The accelerated database recovery (ADR) feature uses PVS.

You can check the database PVS size by running the following T-SQL:

```
SELECT DB_Name(database_id), persistent_version_store_size_kb
FROM sys.dm_tran_persistent_version_store_stats
WHERE database_id = add your database ID
```

If PVS size is large you can enforce the PVS cleanup by executing the following T-SQL:

```
EXEC sys.sp_persistent_version_cleanup [database_name]
```

The links below contains more information about PVS and ADR:

- [Accelerated Database Recovery in Azure SQL](#)
- [Manage accelerated database recovery](#)

Database shrink process

If required, the DBCC SHRINKFILE command can be executed after the above maintenance procedures to release allocated space.

The following example shrinks the size of a data file named DataFile1 in the UserDB user database to 7 MB.

```
USE UserDB;  
GO  
DBCC SHRINKFILE (DataFile1, 7);  
GO
```

However, there are several best practices to be aware of when considering using DBCC SHRINKFILE:

- A shrink operation is most effective after an operation that creates a large amount of unused space, such as a truncate table or a drop table operation.
- Most databases require some available free space for regular day-to-day operations. If you shrink a database repeatedly and its size grows again, then it's likely that regular operations require the shrunk space. In these cases, repeatedly shrinking the database is a wasted operation.
- A shrink operation doesn't preserve the fragmentation state of indexes in the database, and generally increases fragmentation to a certain degree. This is another reason not to repeatedly shrink the database.
- Shrink multiple files in the same database sequentially instead of concurrently. Contention on system tables can cause blocking and lead to delays.

More details about DBCC SHRINKFILE are contained in this link: [DBCC SHRINKFILE \(Transact-SQL\) - SQL Server | Microsoft Docs](#)

Conclusion

In this article we have considered the scenario where the used size of an Azure SQL Database or SQL Managed Instance Database is much larger than expected when compared with the actual number of records in the tables.

This can be resolved by carrying out a few maintenance procedures such as rebuilding indexes to reduce index fragmentation, cleaning up ghost records and cleaning the Persisted Version Store. If required, the [DBCC SHRINKFILE](#) command can also be executed afterwards to release allocated space.

I hope this article was helpful for you, please feel free to share your feedback in the comments section.

Sabrin Alsahsah

4 Likes

Like