

# Shrink Database Best Practices

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

---

## Contents

- [Introduction](#)
- [Best practices](#)
- [IcM](#)
- [Public Doc Reference](#)
- [Root Cause Classification](#)

## Introduction


This guide covers the best practices when shrinking a SQL database.

As a general rule, database shrinks should not be considered as routine maintenance operations. Most databases require some available free space for regular day-to-day operations. If the shrink process reclaims unused allocated space that will again be shortly reused as part of the customers regular workload then there is no need to shrink the database.

Due to the potential effect on database performance, SQL Database doesn't automatically shrink data files to reclaim unused allocated space. However, you can manually shrink the data files to reclaim the space. Unlike data files, SQL Database automatically shrinks transaction log files to avoid excessive space usage that can lead to out-of-space errors. It is usually not necessary to shrink the transaction log file.

A database shrink operation to reclaim unused allocated space is a CPU and IO resource-intensive process that can take significant time to complete. If the space is hundreds of gigabytes or multi-terabytes, this process can run for many hours or even days. However, there are a number of best practices that can be applied to reduce the time taken.

## Best practices

- A shrink operation is effective after an operation that creates unused space, such as a truncate table or a drop table operation.
- As database performance is impacted, the shrink process should be ideally run at a quiet time of day.
- Before starting the shrink rebuild all indexes, or at least the indexes on the largest tables. The maintain Azure SQL Indexes and Statistics script can be used for this purpose:  
[How to maintain Azure SQL indexes and statistics](#) 
- If the CPU, Data IO, or Log IO utilization is (or near) 100%, scale the database up to obtain more CPU cores and increased IO throughput. The service objective of the SQL database should also be taken into consideration. Business critical or Premium service tiers will perform better than General purpose or Standard service tiers as they use local SSD storage for the data files and provide higher IOPS, throughput

and low I/O latency. Therefore consider using a Business critical or Premium service tiers if shrinking is taking longer than desired.

- To manually shrink the database files use DBCC SHRINKFILE or DBCC SHRINKDATABASE commands. DBCC SHRINKFILE shrinks the current database's specified data or log file, whereas DBCC SHRINKDATABASE shrinks all data and log files for the database.

To list the used, allocated and max storage space for each file:

```
SELECT
    file_id,
    name,
    FILEPROPERTY (name, 'SpaceUsed')/128.0 as Used_Space_MB,
    size/128.0 as Allocated_Space_MB,
    max_size/128/1024 as Max_Storage_Size_GB
FROM sys.database_files ;
```

Initially consider running the DBCC SHRINKFILE command using the TRUNCATEONLY parameter. If any allocated but unused space is at the end of the file, it is removed and without movement of data:

```
-- Shrinks database data file named 'data_0' by removing all unused at the end of the file (if any)
DBCC SHRINKFILE ('data_0', TRUNCATEONLY);
```

- If the data file size after executing the DBCC SHRINKFILE command using the TRUNCATEONLY parameter remains larger than wanted, shrink the file to a target size:

```
-- Shrinks database data file named 'data_0' to 500Mb
DBCC SHRINKFILE ('data_0', 500);
```

Repeat this process to perform incremental shrinks, for example, shrink the file to 500 MB and then repeat the operation to shrink down to 250 MB.

In addition, the DBCC SHRINKFILE command can run parallel to other DBCC SHRINKFILE commands to shrink multiple files at the same time and reduce the total time of shrink. This is at the expense of higher resource usage and a higher chance of blocking user queries, if they are executing during shrink.

- If selecting DBCC SHRINKDATABASE as the shrink option the command to execute is:

```
-- Shrink data space allocated for database 'db1'
DBCC SHRINKDATABASE (N'db1');
```

- Unless there is a specific requirement, don't set the AUTO\_SHRINK database option to ON.
- A shrink operation doesn't preserve the fragmentation state of indexes in the database, and generally increases fragmentation to a degree. Therefore after data files are shrunk, the indexes can become fragmented. If performance issues occur, consider rebuilding the indexes once more.

## IcM

If an lCM is needed because the shrink process is failing or continues to have issues after following the best practices the lCM should be created with the Azure SQL DB -> Storage Engine (PVS, CTR, ATM, Logging, Recovery, XFCB) team.

Since telemetry data is only available for a limited time, ensure the customer has recently retried the shrink so that the necessary telemetry data is available before opening an lCM.

## Public Doc Reference

[Shrink a database](#) 

[DBCC SHRINKDATABASE](#) 

[DBCC SHRINKFILE](#) 

## Root Cause Classification

Cases resolved by this TSG should be coded to the following root cause:

Azure SQL v3/Performance/Space Management/User DB

## How good have you found this content?



-