**Shrinking a Database**

The primary purpose of shrinking a database (or individual files) is to reclaim the SPACE by removing unused pages when a file no longer needs to be as large as it once was; shrinking the file may then become necessary, but as we will demonstrate, this process is highly discouraged and is an extremely poor practice in the production database environment.

1000mb mdf files

100mb mdf files

100mb

**Things to note**

* Both data and transaction log files can be shrunk individually or collectively
* When using the DBCC SHRINKDATABASE statement, you cannot shrink a whole database to be smaller than its original size. However, you can shrink the individual database files to a smaller size than their initial size by using the DBCC SHRINKFILE statement.
* The size of the virtual log files within the log determines the possible reductions in size.
* Shrinking causes massive fragmentation and will just result in the data file growing again next time data gets added. When that happens, the entire system will slow down as the file is expanded.

**Automatic Database Shrinking**

When the AUTO\_SHRINK database option has been set to ON, the Database Engine automatically shrinks databases that have free space. By default, it is set to OFF. Leave it as is.

**Database Shrinking Commands**

* DBCC SHRINKDATABASE (ShrinkDB, 10)
* DBCC SHRINKFILE (ShrinkDB, 10)

**Best Practices**

* Size the database accordingly so as to prevent shrinking and expanding the database
* When unused space is created due to dropping of tables, shrinking of the database may be needed - but rarely
* Don't constantly shrink the database as it will grow again
* When you shrink a database, the indexes on tables are not preserved and as such will causes massive fragmentation and will have to be rebuilt
* Shrinking the log file may be necessary if your log has grown out of control however, shrinking the log should be rare also
* Regular maintenance plans should never have shrink database job
* Other issues that shrinking causes are lot of I/O, of CPU usage, and transaction log gets larger – as everything it does is fully logged.

/\*the following example will illustrate that shrinking a database will

1. Causes pages being moved from the front of file to the end

2. It will cause massive fragmentation

3. It will not reduce the size of the data file, but actually increase it

4. That shrinking a database will cause unnecessary I/O hits

5. That shrinking a database was a futile endeavor because of poor planning regarding sizing of database

6. And that shrinking a database should be very rarely done

\*/

USE [master];

GO

CREATE DATABASE ShrinkDB;

GO

USE [ShrinkDB];

GO

-- Create an initial table at the 'front' of the data file

CREATE TABLE Initial (

[col1] INT IDENTITY,

[col2] CHAR (8000) DEFAULT 'Front');

GO

-- Insert data into Initial table

INSERT INTO Initial DEFAULT VALUES;

GO 1500

select \* from Initial

--check the size of the database

sp\_helpdb [ShrinkDB]

--14.83 MB

-- Create the second table, which will be created 'after' the initial table in the data file

CREATE TABLE second (

[col1] INT IDENTITY,

[col2] CHAR (8000) DEFAULT 'after');

--create a clusterd index on the second table

CREATE CLUSTERED INDEX [coll] ON second ([col1]);

GO

-- Insert data into second table

INSERT INTO second DEFAULT VALUES;

GO 1500

select \* from second

--check db size

sp\_helpdb [ShrinkDB]

--database expanded due to insert of data in the second table (26.83 MB)

-- Check the fragmentation of the second table

SELECT

[avg\_fragmentation\_in\_percent]

FROM sys.dm\_db\_index\_physical\_stats (

DB\_ID (N'ShrinkDB'), OBJECT\_ID (N'second'), 1, NULL, 'LIMITED');

GO

--notice that the fragmentation of the clustered index for the second table is almost zero before the shrink

--We will now drop the initial table we created and execute the shrink command to reclaim the SPACE at the front of the data file

-- then see what happens to the fragmentaion.

DROP TABLE Initial;

GO

sp\_helpdb [ShrinkDB]

-- 26.83 MB the data file has not shrunk due to the deletion of the initial table

-- Shrink the database

DBCC SHRINKDATABASE ([ShrinkDB]);

GO

--notice that the SPACE after the shrink went down from 26.83 to 15.02 mb

sp\_helpdb [ShrinkDB]

--15.02 MB

-- But notice what happened to the fragmentation of the data file because of the shrinking of the database???

--when Checking the index fragmentation again, we notice that the fragmentation has drastically increased to almost 100%!!!

--this is because we have shuffled all the data pages and the index is not in a sorted position

SELECT

[avg\_fragmentation\_in\_percent]

FROM sys.dm\_db\_index\_physical\_stats (

DB\_ID (N'ShrinkDB'), OBJECT\_ID (N'second'), 1, NULL, 'LIMITED');

GO

--99.6

--while the database has shrunk, and we have reclaimed space from the data file, we MUST now the fix the fragmented index of the table by rebuilding the index!!!

-- Rebuild the Clustered Index

ALTER INDEX [coll] ON second REBUILD

GO

-- Checking the index fragmentation again indicates that the fragmentaion of the index has been restored, but notice the size of the data

--file when we run the sp\_helpdb [ShrinkDB] - it has actually GROWN even more than it started from!!!

SELECT

[avg\_fragmentation\_in\_percent]

FROM sys.dm\_db\_index\_physical\_stats (

DB\_ID (N'ShrinkDB'), OBJECT\_ID (N'second'), 1, NULL, 'LIMITED');

GO

--0.2%

sp\_helpdb [ShrinkDB]

--41.81 MB

--the database file has grown because of the rebuilding of the index and the logging of the index

--use master

--go

--select \* from sys.databases

--sp\_helpdb ShrinkDB

--transaction log = 13888 KB

--individual file shrink

USE ShrinkDB

GO

DBCC SHRINKFILE (N'ShrinkDB\_Log' , 0, TRUNCATEONLY)

GO

sp\_helpdb ShrinkDB

--784 KB

--USE ShrinkDB

--GO

--DBCC SHRINKFILE (N'ShrinkDB\_Data' , 0, TRUNCATEONLY)

--GO

--USE ShrinkDB

--GO

--DBCC SHRINKDATABASE(N'ShrinkDB' )

--GO