http://www.sqlservercentral.com/articles/SHRINKFILE/71414/
Printed 2011/12/16 06:57AM

Why SHRINKFILE is a very bad thing, and what to do about it.

By Simon Facer, 2010/10/15

The Problem

Shrinking a database file (either using SSMS or T-SQL code) should be avoided if at all possible. In an ideal situation, it is much better to size the file appropriately in the first place, and then grow it as necessary. Of course, in the real world there are always going to be situations where files are over-allocated, bloated, or data is deleted or compressed, and we need to recover the space from the file. When that happens, we are limited to the SHRINKFILE command.

In the SHRINKFILE command, SQL Server isn't especially careful about where it puts the pages being moved from the end of the file to open pages towards the beginning of the file. This causes two problems:

1. the data becomes fragmented, potentially up to 100% fragmentation, this is a performance killer for your database;
2. the operation is slow - all pointers to / from the page / rows being moved have to be fixed up, and the SHRINKFILE operation is single-threaded, so it can be *really* slow (the single-threaded nature of SHRINKFILE is not going to change any time soon).

SHRINKDATABASE vs SHRINKFILE vs AutoShrink

The SHRINKDATABASE command effectively issues sequential SHRINKFILE commands, one for each file in the database, so the rest of this article just references the SHRINKFILE command, but everything can be applied to SHRINKDATABASE as well. Look in BOL for the differences / usage for the two DBCC commands.

The AutoShrink setting isn't a magical alternative to SHRINKDATABASE or SHRINKFILE - this does a SHRINKDATABASE command.

Recommendation (1) - Try a TRUNCATEONLY first.

Try a DBCC SHRINKFILE TRUNCATEONLY first; this just removes empty space from the end of the file, without re-organizing the used data pages first. It's probably not going to work to your satisfaction (i.e. it's not going to free enough space), but it's worth a try, especially as it's much faster than a SHRINKFILE without the TRUNCATEONLY option.

Recommendation (2) - REORGANIZE / REBUILD indexes after the SHRINKFILE.

If a SHRINKFILE must be done, allocate enough downtime for the process to complete, but be aware that you can stop the process at any time without incurring a roll-back. The process is logged, but any work done up to the point of canceling the process is kept, so you can pick up again later and all the time you just spent isn't totally wasted, the downside is that you just fragmented a lot of your data.

Once the SHRINKFILE has completed, identify the Indexes that need to be rebuilt / reorganized and run that process. This is where it gets tricky to know how much time to allow for the downtime - not only do you have to shrink the file, but you have to estimate the time for the REBUILD / REORGANIZE operations after the SHRINKFILE has completed. This will also have the negative impact of growing your file again, though hopefully not as much as you just shrank it by.

The included proc pr_RebuildIndexes (and helper Functions) will selectively Rebuild or Reorganize indexes based on threshold settings, and also includes an option to just Log the commands that would have been run (@LogCommandsOnly = 1). You can use the LogCommandsOnly output to manually run your Index operations if you would rather do it that way. If you don't have a Maintenance Plan for rebuilding indexes, or don't want to just rebuild everything, please use this code; we use it and it works really well.

NOTE - SQL 2005 + only, this doesn't work in SQL 2000 or earlier.

We compile the proc (and all our other routines) into a DBA database on each Instance; you will have to decide where to put the code in your environment. The comment block at the top of the proc describes how to use it, though I'll be happy to answer any questions.

Recommendation (3) - Move user data to a new file.

Add a new file (s) to the database into a new FileGroup, rebuild all the indexes / data into the new FileGroup, then shrink the original file(s). There are several advantages to this method:

1. avoids the Fragmentation problem on the SHRINKFILE operation for the original file - there's no user data to move around and fragment;
2. makes the SHRINKFILE operation for the original file go much faster - there's no user data to move around;
3. you don't need to re-index tables in the original file, so you don't end up re-growing it again;
4. you can do it in smaller, more manageable chunks without impacting the data left in the original file

Of course, the new FileGroup should be set to be the Default; otherwise you're just going to have new tables / objects added to the old FileGroup.

This may not be possible, for a wide variety of reasons (not enough disk space for the new file(s), no indexes on Heap tables, management reluctance, etc, etc). However, this option does have the distinct advantage of implementing a best practice of separating out the User data from the System data in the database.

Conclusion

This isn't meant to be an exhaustive look into the SHRINKFILE command, just enough to bring to your attention that SHRINKFILE, SHRINKDATABASE and AutoShrink are potentially very bad things to do to a database, and some options if you have to use one of them.

References and Acknowledgments

Thanks to Anita, our Microsoft SQL Server DSE, for her work on researching / confirming this for me.
Paul Randal's Blog: http://www.sqlskills.com/BLOGS/PAUL/post/A-SQL-Server-DBA-myth-a-day-(930)-data-file-shrink-does-not-affect-performance.aspx
MSDN : http://msdn.microsoft.com/en-us/library/ms190488.aspx
