1. Use instant file initialization for data files (does not have to zero out the file). Helps with reducing time taken to auto-grow. Reduces time to restore from backups by a lot. Zeroing-out of file cannot be skipped for log files.
2. Use multiple files of equal size per file group (between 4 to 8 files per file group. 4 is a decent default) and equal auto-growth.
3. Although writes to log are sequential (written to serially. If concurrent transactions are running at the same time, they are interleaved), but if you place multiple logs files (from different databases) on the same volume (shouldn’t it be filegroup?), then the performance might come down (because trying to write to multiple of these log files at the same time would result in random IO). But it should not matter for SSD drives as they are efficient for random accesses?? Even though SSDs solve the random-access performance, we will have contention issues (IO throughput) on that drive if log file from multiple databases are stored on the same drive and being written to concurrently.
4. Data compression maybe is not suitable for OLTP workloads. CPU vs IO trade-off.
5. De-fragmentation can also reduce space used. Worthwhile to enable backup compression.
6. Use sp\_estimate\_data\_compression\_savings to estimate compression savings before turning on data compression.
7. Row compression makes everything variable length where possible (e.g. char to varchar). Page compression, loosely explained, extracts common substrings and stores them once and references them where needed.
8. Number of tempdb data files should be = no. of cores of cores<8 and =8 if no. of cores >8 (and keep increasing by 4 if still seeing contention). But PFS latch contention is fixed in SQL 2019. So only 1 data file for tempdb should work?? But SQL Server still created 8 data files by default on my 8 core machine. Should the extra tempdb data files be created on separate filegroups?
9. Versioning isolations (RCSI and Snapshot Isolation) will remove blocking for readers (not for writers which will still use locking). We can also have replication to have subscribers handle read queries. In that case, the queries served by subscribers can block the replication and vice-versa. Having versioning isolation on subscribers can also reduce blocking.
10. Use RCSI as default for new on-prem dbs (it is the default in Azure). Just make sure tempdb is configured right (on SSD and with right number of files and size of files). The issue with this could be that it is mostly write workload, then even though those writes would be versioned, no one would be reading the versioned data. Hence, we would do extra work for versioning (tempdb overhead) that no one uses. Better to start with RCSI only instead of RCSI in conjunctions with SnapshotIsolation.
11. A long running transaction would require an equal amount of free space in log file to be able to undo it (and if that amount of free space is not available, the log file would autogrow. So there is always = amount of free space available unless the file cannot grow for e.g. auto-growth not set). An insert into a table which inserts millions of rows is a long running transaction. So, is it better run batch the large insert into chunks and then every so often between these batch inserts, we take a backup to truncate the log or do we do checkpoint??? What type of backups we take to mitigate this? Log backups or database backups?? Checkpoint does clear out the log file by flushing the changes to data files. If the recovery model is simple, checkpoint will clear/truncate the log. If it is bulk-logged/full, then transaction log backup will clear/truncate the log. Log clear/truncate is misnomer as it neither clears the log nor truncates the log (dbcc shrinkfile is used for log file truncation. Transaction log shrinking is different from data file shrinking as no fragmentation is caused in log file by shrinking). It only marks log space not being used by open transactions to be available for future use. Log size would remain same. So make sure to either checkpoint or take log backups regularly to prevent auto-growing log scenario.
12. For long running transactions, which take long time to rollback and cause huge log file growth, one can also use ‘accelerated database recovery’ from sqlserv2019+. It allows instantaneous rollback of long transactions and aggressive log truncation/clearing. Cons include slower inserts(10%slower)/ deletes (10%slower) /update (2 to 3 times slower).
13. Along with bulk insert and BCP/openrowset, Select \* into is also a bulk-logged (minimally logged) operation.
14. There are some intricacies when switching from Full to Simple and back. Avoid if possible.
15. Lazywriter vs checkpoint. Checkpoint flushes changed data pages to data files on disk (does it also free up log file space). But lazywriter could also write the dirty page to disk when there is memory pressure.
16. When you create a database even in Full recovery model, that is in pseudo-simple recovery model until you take a Full backup. So, take a full data backup and put it in Full-recovery mode.
17. Fragmentation in heaps is caused by forwarding pointers(when updates are done, if no updates, no fragmentation), not page splits which is the case for clustered indexes. Because the NCI on heap would have to be adjusted if the row moved in a heap. That is not required for NCI on CI as the book mark in that case is the index key (so no dependency on the page on which a row resides). Just like CI, you an do rebuild to remove fragmentation (‘forwarding pointer’) in heaps
18. Heaps have better insert performance than CI but less efficient updates. Also, if you are doing forward only inserts in a heap after a lot of deletes, in that case it would inefficient (inefficient than a CI) as in heap it has to use PFS to see where free space is on pages to use the free space for new inserts. If the inserts are not forward only, then in CI as well we could have pages splits and thus bad performance.
19. If you have CI tables prone to fragmentation, use a good fillfactor.
20. For pagelatch\_ex waits, in SQL2019 you can use optimize\_for\_sequential\_key.
21. If you create a CI on a 100 MB heap, 100 MB would be reserved for the CI where it will be created, around 25 MB would be the sorting space. Total you need on the data file is 100 + 25 MB in addition to the 100 MB already being used by heap. Heap and sort space would be freed after CI is created. It would also require space in log file. If full logging, then 100 MB in log file. So total extra 225 MB over data and log file needed while it is being created and after it is done you 125 Mb would be freed.
22. If you have a column that is storing something like a blob or varchar that is not read very often, then that can be stored off the main data page into an overflow page that well help with ProcessingRate/ReadRate coeff or effective throughput.
23. [Run this script](https://www.sqlskills.com/blogs/paul/performance-issues-from-wasted-buffer-pool-memory/) to analyse how the buffer pool looks like and it pool memory is being wasted by fragmentated pages (internal). If more 30% is free, then you might want to look at rebuild/reorganizing the indexes.
24. It is a good idea to restore a backup of the db on a test server and on that try you out defrag methods to get a rough idea around what needs to be done.
25. Think about cost from these perspectives: modifications (insert/deletes), maintenance (index rebuilds, etc), wasted buffer pool space (cache), wasted space on disks (log, backups, etc.)
26. Adding filtered indexes require some session settings change on both the client app and db server. If you cannot changes session settings change on the client, then it will error out when tying to modify data. This is a gotcha. Try to use the session settings of SSMS everywhere (Kimberly slide with options. Even the SQLCMD and SQL srv agent defaults are different from SSMS).
27. Sp can be recompiled at 3 different levels: a. With recompile when sp is called, b. in the definition of the sp and at the stamen level inside sp.
28. Bulk load into a heap (will it be parallelized, or do we have to parallelize it?), then build the clustered index on it (the index build will also be parallelized in this case). It will allow faster ingest, faster index build, less fragmentation and will have uptodate stats. This can also be done in a staging db and then either you switch in the partition into production or you alter the partitioned view to include the new table or just build index on it.
29. When query tuning, also use DTA to see what indexes and stats it recommends. Do it in test environment.
30. Hypothetical indexes are one that only have stats but not the (b+tree structure). They can be created (using DBCC AutoPilot option) very fast on large tables to help you compare and contrast competing indexes during query optimization phase instead of having to wait for index creation.
31. Parameter sniffing tells you that parameter values were taken into account when the procedure wa compiled to a plan. But variables cannot be sniffed as SQL Ser would have to execute the procedure to see what the variable value is (it could assigned a computed value instead of declaration) and therefore optimizer uses a ‘unknown’ value for the variable to optimize.