SQLskills Immersion Event

IEPTO1: Performance Tuning and Optimization

Module 2: Data File Internals and Maintenance

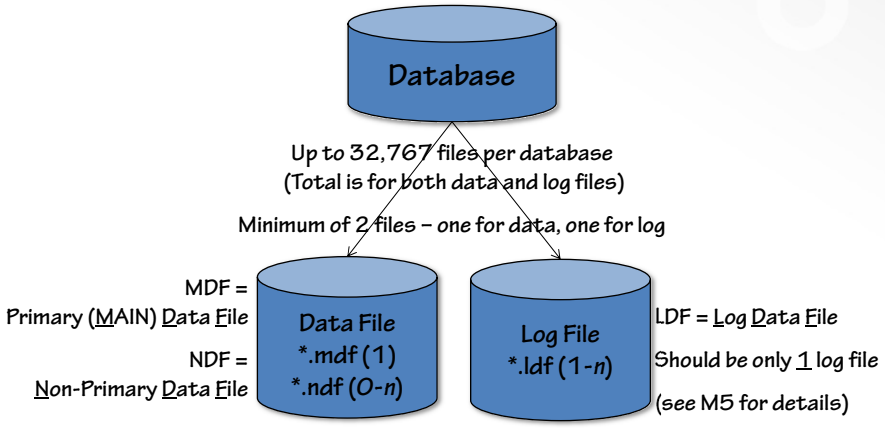
Paul S. Randal
Paul@SQLskills.com

Overview

- Physical layout considerations
- Allocation algorithms
- Instant initialization
- Auto-grow
- To shrink or not to shrink?
- Data compression
- Tempdb

Database Structure

Up to 32,767 databases per instance

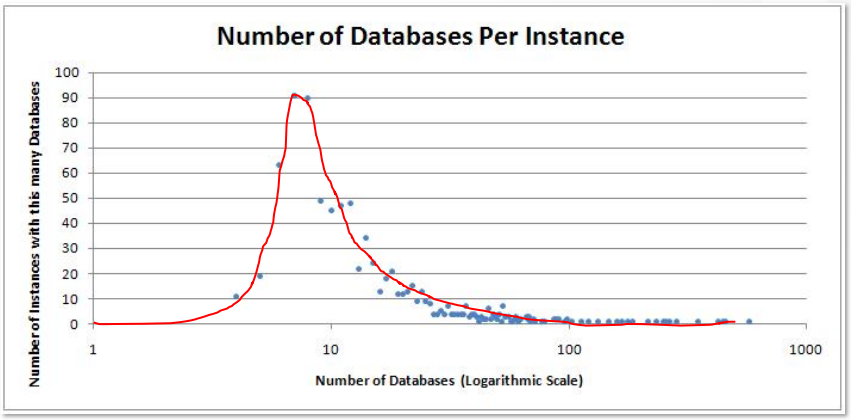


Notes for prior slide

Consolidation Issues

- **Too many databases per instance can lead to:**
  - Performance problems (lack of resources)
    - Constantly fighting for buffer pool resources
    - I/O subsystem placement difficulties
  - Maintenance problems (lack of resources)
    - Constant background I/O and CPU load from maintenance, DBCC CHECKDB, and backups
- **Too many database files can lead to long startup time**
  - Every file has to be opened and read to get the file header page

Databases Per Instance



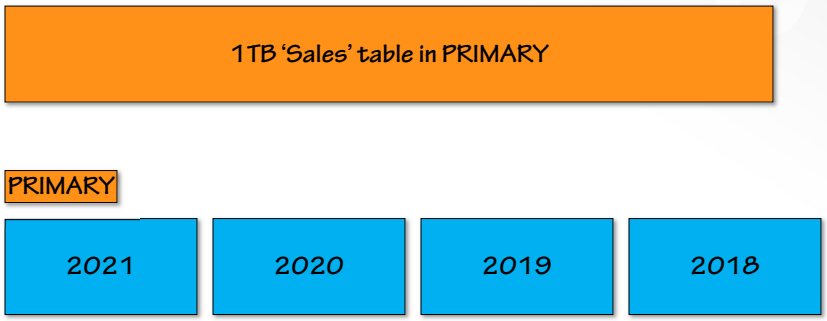
- Source: https://sqlskills.com/p/005

Files and Filegroups

- **A filegroup contains one or more files**
  - A table or index is wholly contained in a single filegroup
- **Every database has at least one filegroup – the PRIMARY filegroup**
  - Contains at least one file – the MDF
  - Keep this is small as possible
- **Multiple secondary filegroups can and should be defined**
  - Multiple good reasons for this:
    - #1 availability
    - #2 manageability
    - #3 performance
  - Multiple data files per filegroup may give better performance
    - Test and also check what guidance from I/O subsystem vendor
    - See https://sqlskills.com/p/006 for test showing better perf with more files
- **Bigger the database, more reason for multiple filegroups**

One vs. Multiple Filegroups

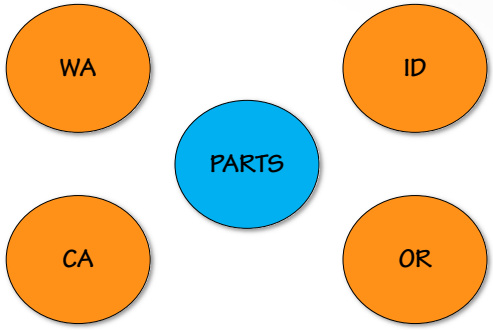
- Consider availability and maintenance of each solution



- But how to split *your* data up?

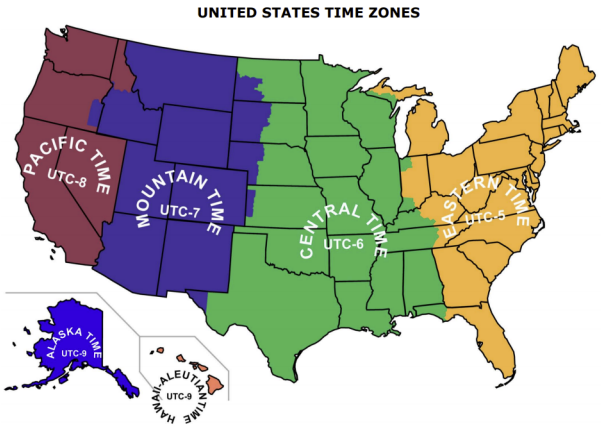
Fictitious Example

- Online auto/car parts store



Client Example

- Client example: doctor office patient check-in across the US



SSDs/Flash Storage

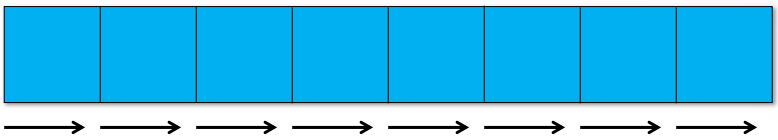
- SSD = Solid State Drive
- Extremely low I/O latency and high throughput
- Very common now, and prices very affordable
- Beware of 'best practices'
  - Don't just put tempdb or transaction logs on them!
  - Don't ignore index fragmentation when using them!
  - Use them in RAID configuration, not just a single drive!
- Consider Buffer Pool Extension for Standard Edition
  - Ability to have unchanged data stored on SSDs
  - Useful to work around max memory limitation
- Also Hybrid Buffer Pool using PMEM devices in 2019+, similar to BPE
  - Server and database configurable, all Editions, Linux and Windows
- See Paul's old SSD blog series: https://sqlskills.com/p/007
- And CSS post on SSDs: https://sqlskills.com/p/008

Physical Layout Considerations (1)

- Know your workload and your I/O subsystem!
- Much of this may be out of your control or irrelevant depending on hardware in use
- RAID array configuration
  - Choose appropriate RAID level (if you can)
    - E.g. not RAID-5 for a high volume OLTP system
  - Different kinds of data onto different storage
    - E.g. rarely used LOB data on RAID 5, OLTP data on RAID 1/10
- Possibly defrag the volumes
  - NTFS-level fragmentation very small effect on performance of large scans
  - Do not do volume defragging while SQL Server is running
    - Beware of 3rd-party "open file" defraggers
  - If heavy fragmentation, auto-growth is set incorrectly, which is a perf issue
- Capacity planning considerations?

Performance Problem with L:

- What's the I/O pattern of this volume with all these log files on?
- May or may not be a problem for your I/O subsystem



Physical Layout Considerations (2)

- **Separation of files is old advice but might still be applicable**
  - Degree of separation depends on I/O workload and I/O subsystem
    - E.g. SAN can do a better job of spreading I/O costs than a single drive
    - E.g. on all-flash storage, likely won't have any effect
  - If on older storage, consider separating:
    - Log files from data files, even separate databases on different LUNs
      - Gets separate I/O metrics and perfmon counters
      - Sequential workload separate from random workload, generally is a good thing
      - Beware of putting all log files on L:, becomes a random pattern
    - SQL Server files separate from other uses (e.g. OS files on C:)
- **Your I/O subsystem may be fast enough to avoid separation**
- **tempdb – see later in this module**
- **Old paper: *Physical Database Storage Design* (https://sqlskills.com/p/009)**
- **Read the whitepaper from your storage vendor**

Volume Configuration

- **Disk partition alignment**
  - Make sure partition start offset is divisible by RAID stripe size
  - Before Windows Server 2008, default offset is 31.5KB (WS2008+ uses 1MB)
    - Not RAID stripe aligned, so extra I/Os to read/write data
  - On WS2008+, still must check after formatting to ensure I/O subsystem did not intercept and use incorrect alignment
    - See https://sqlskills.com/p/010 for details of how to check
    - Also http://sqlpowerdoc.codeplex.com/ for PowerShell method
- **RAID stripe size: whatever your I/O vendor recommends**
- **NTFS cluster (allocation unit) size doesn't matter on modern hardware**
  - No current empirical evidence that 64KB is better than anything else
  - Do not reformat if it's set to 4KB
- **Whitepaper: *Disk Partition Alignment Best Practices For SQL Server***
  - https://sqlskills.com/p/011

Overview

- Physical layout considerations
- **Allocation algorithms**
- **Instant initialization**
- **Auto-grow**
- **To shrink or not to shrink?**
- Data compression
- Tempdb

How Does Allocation Work?

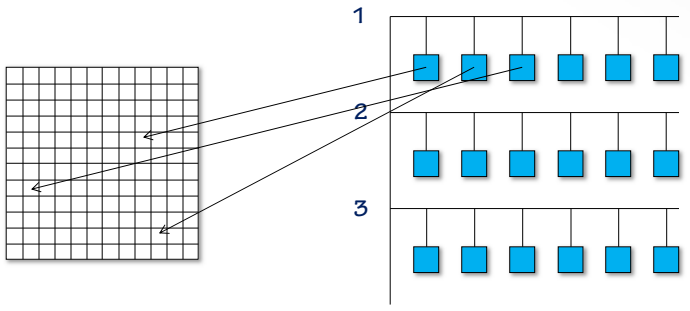
- **With a single-file database, very simple**
- **When filegroup has multiple files, two allocation algorithms are used:**
  - Round-robin allocation
    - Allocations are made from each data file in the filegroup in turn, essentially striping the data across multiple files
  - Proportional fill
    - Allocations are made from each data file during round-robin proportional to the amount of free space in each file
      - Weightings recalculated whenever a file is added/dropped, or at least 8192 extent allocations take place in the filegroup – see https://sqlskills.com/p/013
    - For best results, data files should be the same size with same free space
- **Misconception: you cannot add another data file to a filegroup and rebalance the allocations across all files**
  - You have to create a new filegroup to do that
  - Even rebuilding indexes doesn't do what you'd think

How Does the Buffer Pool Work?

- Sometimes called the 'buffer cache'
- Memory block used to hold in-memory copies of pages from data files
- Pages are read into the buffer pool when requested by other parts of the Storage Engine and not already in memory
  - When a page is already in memory, this is a logical I/O
  - When a page has to be read from disk, this is a physical I/O
  - All I/Os start as logical and may become physical
- Page lifetime in memory depends on many things
  - Two last access times are tracked and provide an LRU indication
    - Tracked as smallint, not a full datetime
  - When memory pressure is felt, the least-recently-used pages are 'tossed' out of the buffer pool to make way for needed pages
  - Also done proactively by the lazy writer background process
- Hash tables exist to quickly find page images in memory

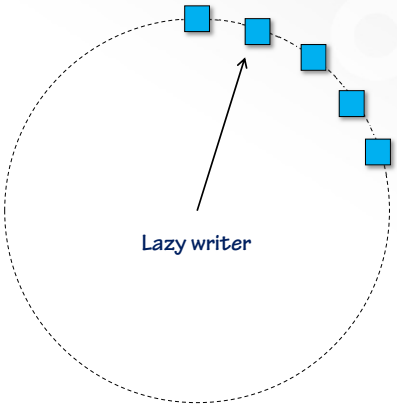
Buffer Pool Internals

- Hash list of BUF structures per database, ordered by page ID, for quick access and determining if a particular page is in memory or not



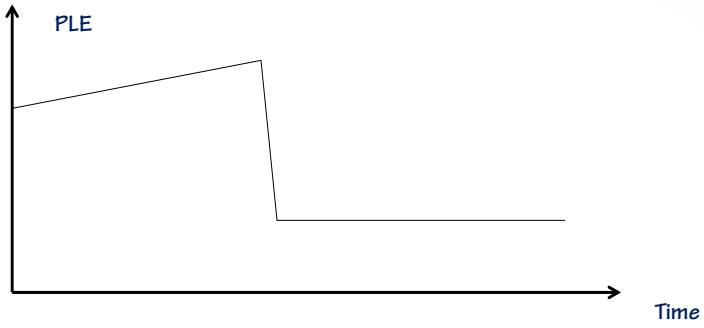
The Lazy Writer

- Wakes once per second
- If pressure on the buffer pool, sweeps the BUF structures looking for the least-recently-used pages to drop from memory
  - LRU-k, k=2
- Pressure on the buffer pool lowers the page life expectancy



Page Life Expectancy

- Ticks up 1 per second when there's no pressure on the buffer pool
- Normal for it to go up and down
- Threshold to care? When it drops below your normal and stays there

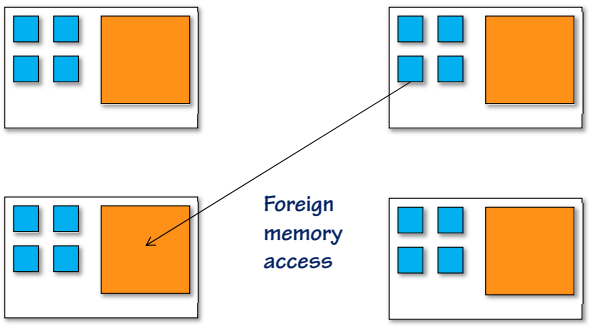


Monitoring Page Life Expectancy

- **Buffer Manager: Page Life Expectancy**
- **But if you have NUMA, buffer pool is split up (see next slide) with equal portions per NUMA node**
  - See sys.dm_os_nodes to see how many NUMA nodes
  - NUMA: non-uniform memory access
  - NUMA: https://www.sqlskills.com/blogs/jonathan/category/numa/
- **With NUMA:**
  - One lazy writer per NUMA node
  - Entirely possible for there to be memory pressure in just one NUMA node
  - Monitor each Buffer Node: Page Life Expectancy
  - Buffer Manager counter is a 'harmonic mean' of all node PLEs
  - See https://sqlskills.com/p/121

NUMA

- **Example with four NUMA nodes with four physical cores in each**
- **Buffer pool is split into four partitions, one per NUMA node using the node-local memory**
- **Foreign memory accesses are very expensive**



More on the Buffer Pool

- **What's in the buffer pool?**
  - SELECT * FROM sys.dm_os_buffer_descriptors
  - Be aware that large amounts of index fragmentation can lead to lots of wasted buffer pool space
  - Blog posts with scripts: https://sqlskills.com/p/014
  - More on this in M7
- **Memory management**
  - Buffer pool does full-extent reads when ramping up in Enterprise Edition
    - Until Buffer Manager: Total Pages equals Buffer Manager: Target Pages (or Buffer Node: with NUMA)
    - And with TF840 in other editions: https://sqlskills.com/p/015
    - As long as the I/O subsystem can keep up, no downsides to enabling

Creating or Growing Data Files

- **Whenever a data file is created or grown, its 'high-water mark' must be set in NTFS**
  - This is the point in the file up to which NTFS knows the data is trustworthy and will allow it to be read
- **By default, this is done by zeroing out the new/additional space**
  - This is done by SQL Server, single-threaded, by issuing successive writes of blocks of zeroes to the file
  - The new /additional space cannot be used until the process completes and the allocation that triggered it will pause until it is done
- **This process can be skipped by enabling instant file initialization**
  - Only applies to data files

Instant File Initialization

- Allows SQL Server to skip the zeroing process
- Instead of doing the zeroing, SQL Server calls the Windows API SetFileValidData, which sets the file highwater mark
- Disabled by default because there is a potential, rare, security risk of enabling it:
  - If a volume is being shared by a database and a file server storing 'secure files' and if the file server deletes some files, and then the database grows, it may pick up some of the space used by the deleted filed
  - As instant file initialization skips zeroing the new space, the old raw bytes of the deleted files will be accessible to a DBA using DBCC PAGE
  - If this is not the case then no security risk, or if the DBA is a Windows Administrator then the risk is already there
- If you can, turn it on!
- Very important to have enabled for downtime savings when restoring

Instant File Initialization in Action

- Hardware configuration – Dell PowerEdge R720
  32 cores, 64GB memory, RAID 10 array with 6 x 300 GB, 15k disks
- Software configuration: SQL 2014 (similar results on all versions)
  - With zero initialization
    - CREATE DATABASE with 20GB Data file = 7:12 minutes
    - ALTER DATABASE BY 10GB = 4:20 minutes
    - RESTORE 30GB DATABASE (EMPTY Backup) = 29:13 minutes
    - RESTORE 30GB DATABASE (11GB Backup) = 30:27 minutes
  - With instant file initialization
    - CREATE DATABASE with 20GB Data file = 8 seconds
    - ALTER DATABASE BY 10GB = < 1 second
    - RESTORE 30GB DATABASE (EMPTY Backup) = 14 seconds
    - RESTORE 30GB DATABASE (11GB Backup) = 1:32 minutes
- Additional data: https://sqlskills.com/p/016

Enabling Instant File Initialization

- **Cannot be enabled from within SQL Server**
  - Option to enable it in SQL Server 2016+ installation wizard
  - Status reported in error log at instance startup in SQL Server 2016+
- **Requires:**
  - Any Edition of SQL Server
  - "Perform Volume Maintenance Tasks" security permission granted to SQL Server service account or group
- **Use Local Security Policy Editor to grant permission**
  - Administrative Tools -> Local Security Policy and then Local Policies -> User Rights Assignment (defaults to Local Administrators Group)
  - Video demo of procedure at https://sqlskills.com/p/017
- **Instant initialization happens automatically once SQL Server is restarted after granting the permission**
- **Look at sys.dm_server_services to see if enabled**

Auto-Grow

- **When a file is full, it will grow to provide more space**
  - Enable instant file initialization to skip zero initialization
  - 1MB auto-growth default often not changed leading to lots of auto-growth
    - SQL Server 2016+ is better: 8MB initial size with 64MB auto-grow
- **Best practice to manually manage file sizes**
  - Allows you to decide which files to grow, by how much, and when
  - Monitor file usage and alert when threshold reached
- **But auto-grow should still be enabled 'just in case'**
  - If it's off, and no-one monitors space, the workload could stop
  - Monitor auto-growth using SQL Trace or Extended Events
  - Always have file growth to be a fixed amount, not a percentage
- **TF 1117 to force all files in a filegroup to grow at once**
  - 2016+: ALTER DATABASE … MODIFY FILEGROUP {AUTOGROW_ALL_FILES | AUTOGROW_SINGLE_FILE}

Auto-Shrink or Regular Shrinking

- **When to use auto-shrink? NEVER!**
  - About the only thing there is NOT an 'it depends' answer for
  - This should be one of the first things that gets checked when you take over a new database
- **Why not?**
  - Data file shrink (same code as auto-shrink) is almost guaranteed to introduce index fragmentation within the data files
  - The database will most likely just auto-grow again, and then auto-shrink, auto-grow in a cycle that wastes resources
  - You can't control when it kicks in and affects performance
  - Blog post: https://sqlskills.com/p/018
- **Regular shrinking is just as bad**
  - Watch out for Maintenance Plans that include a shrink

Demo

Impact of shrink on index fragmentation

When to Use Data File Shrink?

- **Should be a very rare operation**
  - Because it causes index fragmentation
- **Three scenarios:**
  - Emptying a file before removing it
  - When large amount of data has been deleted AND space won't be reused
  - Moving a file/filegroup/database to read-only
- **Even then, shrink may not be the best method**
  - Remember – it causes fragmentation
- **So, how to perform a data file shrink?**

How to Shrink a Data File?

- **Drop nonclustered indexes**
- **Create a new filegroup and move all clustered indexes into it**
  - Use CREATE INDEX… WITH (DROP_EXISTING=ON) and specify the location to be the new filegroup
    - Doesn't move LOB data, but see https://sqlskills.com/p/019
  - Can be performed online
- **If any table are heaps, use shrink to move them**
  - Heaps are unordered so shrink does not fragment them
  - Beware shrink is *very* slow for heaps and LOB data
- **Recreate nonclustered indexes on new filegroup**
- **Drop old filegroup**
- **If you have to shrink a data file, use ALTER INDEX … REORGANIZE to remove index fragmentation**
  - Or consider disabling then rebuilding them after the shrink

Overview

- Physical layout considerations
- Allocation algorithms
- Instant initialization
- Auto-grow
- To shrink or not to shrink?
- **Data compression**
- Tempdb

Data Compression: The Problem

- **Cost of storage rises with database size**
  - High-end storage is expensive
  - Multiple copies required – test, HA, backups
- **Cost of managing storage rises with database size**
  - Time taken for backups and maintenance operations (I/O bound)
  - Time taken to restore backups in a disaster
- **Migration from other platforms (Oracle or DB2) that support compression is hard without compression**
  - Reduced TCO of SQL Server can be outweighed by increased storage costs from 3-5x increase in database size
- **Compressing data on-page leads to better memory utilization**
  - Blog post: https://sqlskills.com/p/021
  - Sometimes does better than columnstore – see https://sqlskills.com/p/105

Is Data Compression Suitable?

- **Performance gain comes from having to perform fewer I/Os, tradeoff against CPU, for large scans**
  - But, data is compressed on disk AND in memory so each access of the data must decompress it, using CPU
  - Enabling can be a performance detriment for volatile data (e.g. OLTP)
- **Monetary gain comes from saving disk space**
  - Can you do that with backup compression?
  - Can you do that by removing wasted space from fragmentation?
- **Questions to ask:**
  - Is the I/O vs. CPU tradeoff worthwhile?
  - Will there be a significant space saving?
  - Are cost savings more important than potential workload impact?
- **WP: *Data Compression: Strategy, Capacity Planning and Best Practices***
  - https://sqlskills.com/p/022

What Can Be Compressed?

- **Compression can be applied to:**
  - A whole heap
  - A whole clustered index
  - A whole non-clustered index
  - A whole indexed view
  - Single partitions of partitioned tables and indexes
    - Different partitions can have different compression settings
- **Each index must be compressed separately**
  - Compressing the heap or clustered index does not compress all indexes
- **Switching from heap <-> clustered index keeps compression setting**
- **System tables cannot be compressed**
- **How do you choose what/when to compress?**

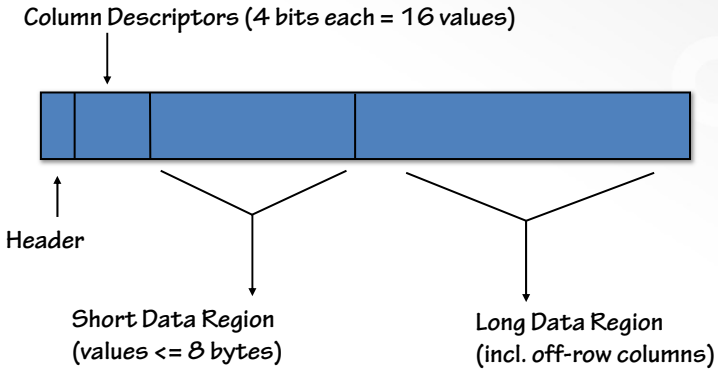
Estimating Space Savings

- **Enabling data compression on existing data can be very costly**
  - Basically the partition, index, or table is rebuilt
- **It makes sense to evaluate the potential savings before enabling compression, using the SP**
  - *sp_estimate_data_compression_savings*
- **Creates a 5% sampled subset of the data in tempdb and compresses it using the requested compression mechanism**
- **Returns the estimated size of the table/index/partition with the requested compression, plus details of the sample size**
  - Can also be used when turning OFF compression
- **Note: A storage decrease from compression is not guaranteed**

ROW Compression

- **Stores fixed-length numeric data as variable-length**
  - E.g. integer, decimal, float, datetime, money
  - E.g. a bigint holding the value 34 will only store 1 byte
- **Stores fixed-length character data as variable-length**
  - Blank characters are not stored
  - E.g. a char (50) holding 'Paul Randal' will only store 11 bytes
- **Strips out unused bytes for Unicode strings if all zeroes**
- **A new variable-length row format is used to bring the per-column metadata overhead down from 2-bytes to 4-bits**
  - Null and zero values can be stored only using the 4-bits of metadata
- **Details of row compression can be found in Books Online**
  - 'Row Compression Implementation' in BOL index

Record Structure



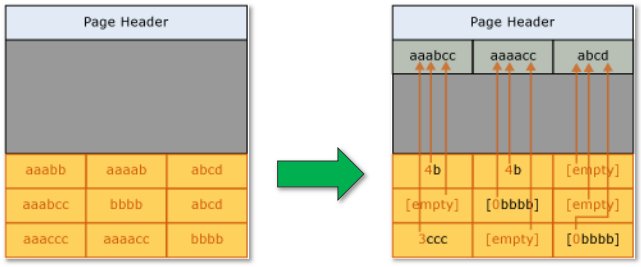
- Per-column descriptors give column length, plus special values such as zero or null
- 'Jump points' every 32 columns to speed up access

PAGE Compression

- Page compression does three things:
  - ROW compression
  - Per-column prefix compression
  - Per-page dictionary compression
- When compressing a page, these three operations are done in the order listed above
- Page compression adds a record at the start of each page called a compression information structure, or CI
- Details of page compression can be found in Books Online
  - 'Page Compression Implementation' in BOL index

PAGE Compression: Prefix Compression

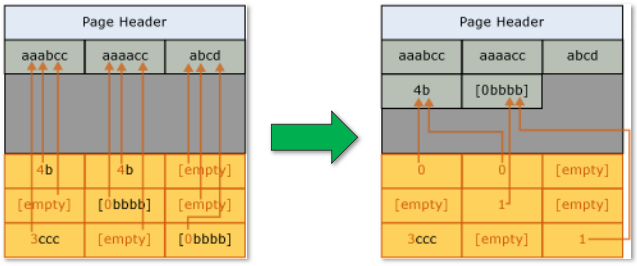
- For each column, a value is chosen that allows space reduction and is stored in the compression information structure (CI)
- The in-row values are replaced with indicators of full or partial matches with the value in the CI



- Note that the largest value for each column is stored in the CI
- The process uses byte-level comparisons across all data types

PAGE Compression: Dictionary Compression

- Dictionary compression is done AFTER prefix compression
- The whole page is scanned looking for common values, which are stored in the CI area on the page
- The in-row values are replaced with pointers to the CI area



When is Data Compressed?

- **After PAGE compression is enabled, new pages are only ROW compressed until they fill up, then the next row insertion triggers page compression**
- **The page is PAGE compressed, and if there's 20% space savings after compression has occurred (and the CI structure added), the next row is compressed and inserted**
  - If there's not enough space saving, the page is not compressed and the row will be inserted on a new page
- **When a table or index is rebuilt with PAGE compression on, PAGE compression occurs as part of the rebuild, but follows the same process**
  - I.e., there may be some uncompressed pages if PAGE compression was not worthwhile
- **Per-page modification counter triggers re-compression of page**

Limitations

- **Data compression was Enterprise Edition only until 2016 SP1**
  - Restoring a backup with compressed data in would fail on a lower edition (same as happened with partitioning in SQL Server 2005)
  - Check the *sys.dm_db_persisted_sku_features* DMV
- **Off-row data is not compressed**
- **Enabling or disabling are index rebuilds, so not a trivial process**
- **You might be burning a lot of CPU attempting compression on non-compressible data**
  - Check in sys.dm_db_index_operational_stats
    - *page_compression_attempt_count* should not be a lot higher than *page_compression_success_count*

Customer Experiences with Data Compression

Customer	Space Savings	Throughput impact	Notes
Customer #1	40%	5%	PAGE compression. OLTP web application. Large volume of transactions.
Customer #2	62%	40%-60%	PAGE compression. DW application. Large sequential range queries .
Customer #3	38%	-1%	PAGE compression. OLTP with some reporting. 500 users, 1,500 trans/sec.
Customer #4	80%	-11%	PAGE compression. OLTP application. A lot of insert, update and delete activity.
Customer #5	52%	2% - 3%	PAGE compression. OLTP Application.
Customer #6	81%	3%	PAGE compression. ERP application – small transactions.

Your Mileage Will Vary!

More Info on Data Compression

- WP: *Data Compression: Strategy, Capacity Planning and Best Practices*
  - https://sqlskills.com/p/022
- Blog post series:
  - Walking through data compression analysis of a large database, plus investigation of CPU usage, wait stats, backup timings, and much more
  - https://sqlskills.com/p/023 is for final part, with links to previous parts

Overview

- Physical layout considerations
- Allocation algorithms
- Instant initialization
- Auto-grow
- To shrink or not to shrink?
- Data compression
- **Tempdb**

Traditional tempdb Uses

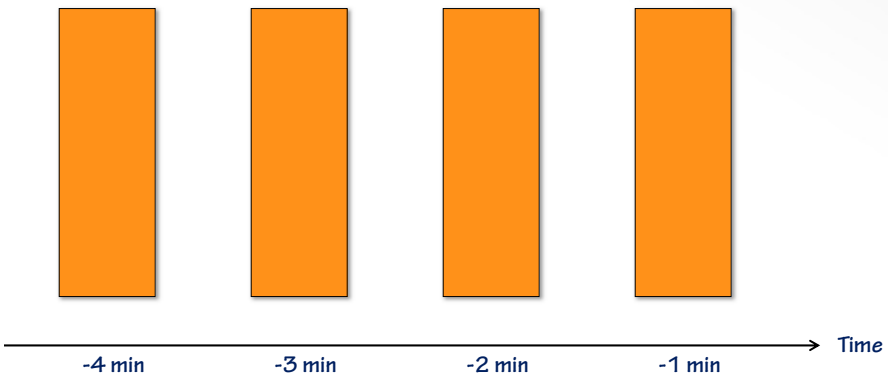
- **User objects:**
  - Table variables (@)
  - #temp and global ##temp tables/indexes
  - Regular tables
- **Internal objects**
  - Worktables (e.g. sort, intermediate results, …) from memory spills to disk
  - Workfiles (only for hash joins and hash aggregates) from spills to disk
  - Intermediate sort results from index create/rebuilds
    - Only if SORT_IN_TEMPDB is specified
  - Intermediate DBCC CHECKDB results (in a worktable)
  - Version store (main one + online index operations one)
- **Full list and explanations in BOL: Capacity Planning for Tempdb**

Version Store

- Stores all version records (as described in M1) created in all databases
- One new allocation unit created every minute
  - Variable size depending on how many versions created in that minute
- Cleanup task runs every minute or so
  - If all version records in allocation unit no longer required, it can be deleted
- Completely non-logged, even the allocations/deallocations
- Long-running transactions using snapshot isolation can interrupt cleanup and lead to tempdb growth
- DMV added in 2016 SP2+ to see which database is causing versions
  - *sys.dm_tran_version_store_space_usage*
- See Books Online: Row Versioning Resource Usage
  - https://sqlskills.com/p/024

Version Store Internals

- For example, if a transaction needs a version from 4 minutes ago, nothing can be cleaned up



tempdb Sizing

- **No easy way to figure out initial size as so many operations can use tempdb space**
  - General practice is run production workload, see how big tempdb gets
    - General query workload
    - Index maintenance operations
- **Set tempdb size to resulting size**
- **Enable appropriate auto-growth, equal on all files**
  - 2016+ sets default autogrow to 64MB, all tempdb files grow at same time
  - 2016+ setup has a decent tempdb configuration wizard
    - 2017+ allows initial size of each file to be up to 256GB and warns if autogrowth size more than 1GB without Instant File Initialization enabled for the instance
- **We'll talk about the number of files to create in a few slides…**
- **Books Online: Capacity Planning for Tempdb**
  - https://sqlskills.com/p/025

tempdb Sizing and Restart

- **Tempdb reverts to the last specifically set size after a server restart**
- **If tempdb grows during operations, that is not the same as specifically setting the size**
- **Avoid excessive auto-growth after a server restart by manually setting the size**

tempdb and I/O Subsystems

- **Best practice has been to separate tempdb from other databases due to I/O contention and put on high-performance I/O subsystem**
  - Entirely depends on I/O subsystem and workload involved
- **Favorite use of SSDs**
  - SSDs are best suited to random I/O, but generally will give a performance boost to an I/O subsystem that's overwhelmed
- **Page checksums on tempdb weren't available until SQL 2008**
  - On by default for new instances of SQL 2008 onwards
  - Make sure they're enabled for tempdb for upgraded instances
- **In 2014+, data not flushed to disk for bulk operations in tempdb**
- **Books Online – Optimizing Tempdb Performance**
  - https://sqlskills.com/p/026

tempdb Allocation Bitmap Contention

- **Some query workloads cause multiple concurrent threads to repeatedly create/drop small temp tables and/or worktables**
  - Can also be from repeated population/truncation of temp tables
- **Easy to cause PAGELATCH_UP contention on allocation bitmaps prior to SQL Server 2019, especially PFS**
  - Use sys.dm_os_waiting_tasks to see waits on PAGELATCH_UP
  - SGAM page to manipulate mixed extents (resource 2:1:3)
  - PFS page to allocate/deallocate pages (resource 2:1:1 and then any page ID that's a multiple of 8088)
  - Small number of tempdb data files means limited in-memory PFS
- **Contention can occur in all versions, but much reduced in 2019+**
- **This can sometimes (rarely) happen in user databases with VERY high-end allocation workloads**

tempdb System Table Contention

- **Some query workloads cause multiple concurrent threads to repeatedly create/drop small temp tables and/or worktables**
  - Can cause PAGELATCH_SH/EX contention on sysobjvalues and sysseobjvalues tables ('insert hotspot')
  - Use sys.dm_os_waiting_tasks to see waits on PAGELATCH_SH/EX in tempdb
  - Check whether the page is in a system table using DBCC PAGE
- **Fixed somewhat in SQL Server 2016 builds**
  - See https://sqlskills.com/p/107 and 108
- **Can remove completely in 2019 by setting system tables in-memory**
  - ALTER SERVER CONFIGURATION SET MEMORY_OPTIMIZED TEMPDB_METADATA = ON;
  - Restart instance
- **Also a new trace flag 3427 in latest 2016 builds that speeds up small transactions using tempdb (see https://sqlskills.com/p/108)**
  - Removes overhead from Common Criteria Compliance auditing

tempdb Temp Table Caching

- **There is a cache of worktables and temp tables**
  - Worktable cache is very small
  - Temp table cache is (relatively) unbounded
- **Mechanism:**
  - When a temp table/worktable is dropped, a single IAM page, a data page/extent, and its metadata entry remain
  - As long as temp table is less than 8MB when dropped
  - Only for temp tables NOT created by ad hoc statements
    - **And in 2014+, no further DDL in SP that creates the temp table**
  - The next temp table/worktable creation pulls one out of the cache
  - All kinds of reasons a temp table may not be cached
- **See blog posts: https://sqlskills.com/p/027 and https://sqlskills.com/p/028**

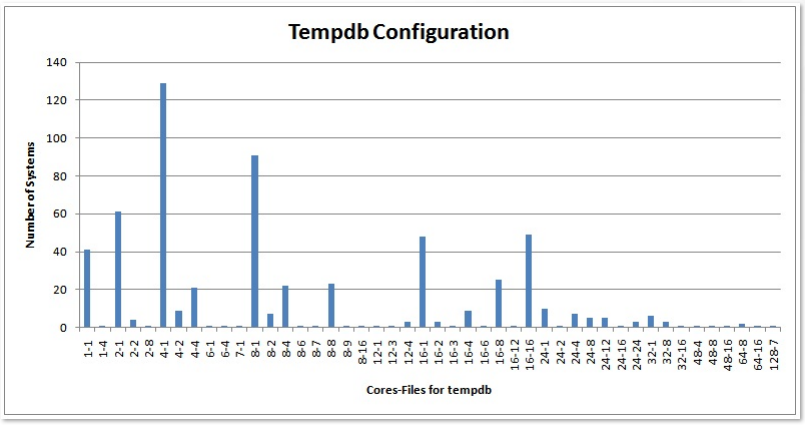
Alleviating tempdb Bitmap Contention

- **TF 1118 (KB 328551) removes mixed extents (SGAM contention)**
  - **All instances across the world should have this enabled (and T3226)**
  - **1118 behavior on by default in SQL Server 2016+**
- **Use multiple data files to reduce contention (KB 2154845)**
  - <= 8 cores: #files = #cores; > 8 cores, #files=8, then increase by 4 at a time
  - 2016+ install automatically configures multiple tempdb data files
  - Adding just one file may not work (see https://sqlskills.com/p/029)
  - Investigation article on Simple Talk: https://sqlskills.com/p/030
- **Alleviated a bit in latest 2016/2017 builds by spreading allocations over multiple PFS intervals (see https://sqlskills.com/p/106 and 108)**
- **2019 enhancements**
  - No latch for PFS updates, using special CPU instructions
  - Temp table cache optimizations to reduce spinlock contention when adding/removing entries

Demo

tempdb allocation bitmap contention: 2017 vs. 2019

tempdb Configuration Survey



- Source: https://sqlskills.com/p/031

More Info on Tempdb

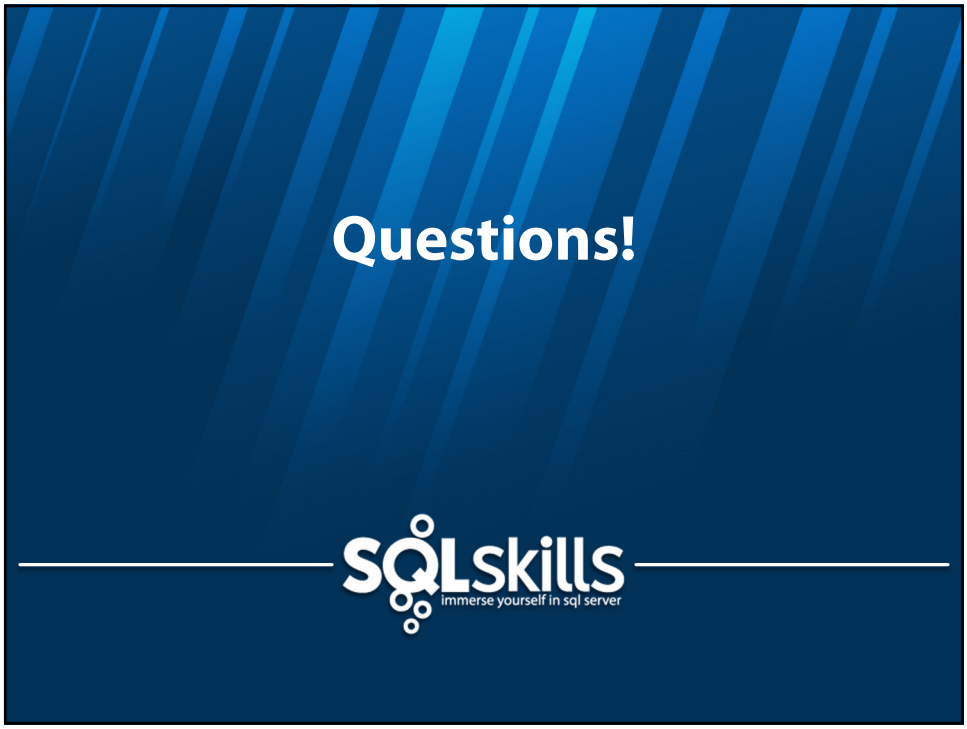
- Moving tempdb – see KB 224071
- WP: *Working with tempdb in SQL Server 2005*
  - https://sqlskills.com/p/032
  - Still very applicable today
- Blog post: Misconceptions Around TF 1118
  - https://sqlskills.com/p/033
- Blog post series on tempdb
  - https://sqlskills.com/p/034
- Books Online:
  - Troubleshooting Insufficient Disk Space in Tempdb
    - https://sqlskills.com/p/035
  - Capacity Planning for Tempdb
    - https://sqlskills.com/p/025

Some Linux Specifics…

- Instant file initialization always on for data files
- Setup does not create multiple tempdb data files
  - Microsoft recommendation is 1:1 with logical processor cores
- Having more data files definitely helps on Linux
  - According to Microsoft testing

Review

- Physical layout considerations
- Allocation algorithms
- Instant initialization
- Auto-grow
- To shrink or not to shrink?
- Data compression
- Tempdb


Questions!
SQLskills
immerse yourself in sql server