

SQLskills Immersion Event

IEPTO1: Performance Tuning and Optimization

Module 1: Database Structures

Paul S. Randal
Paul@SQLskills.com



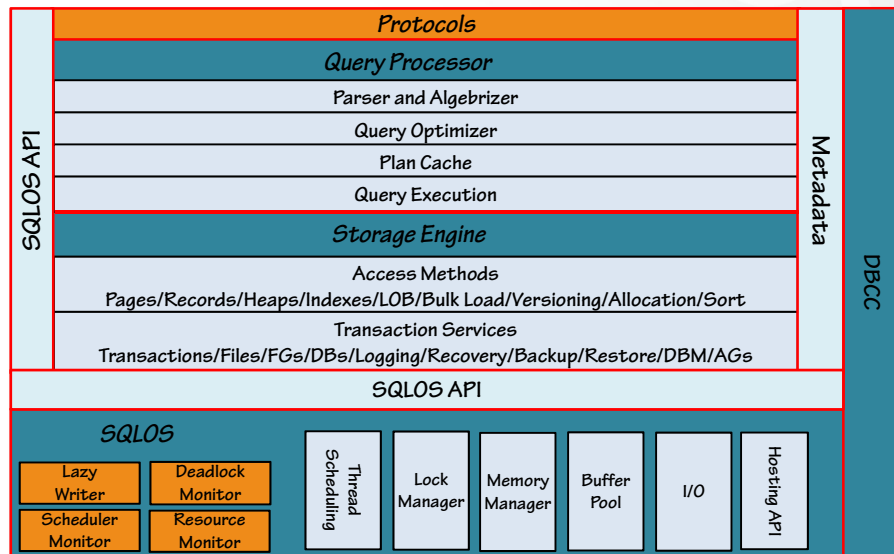
Why Cover Internals?

- Internals aren't just to geek-out on (although that's fun to do too! ☺)
- Understanding how data is stored, accessed, and optimized at all levels is key when architecting a system so that it will perform well and be more easily maintained
 - Explains why some decisions are good or bad...
 - Helps to troubleshoot what's actually happening...
 - Gives a clearer understanding in how to design appropriately for SQL Server
- These are the building blocks for understanding the class

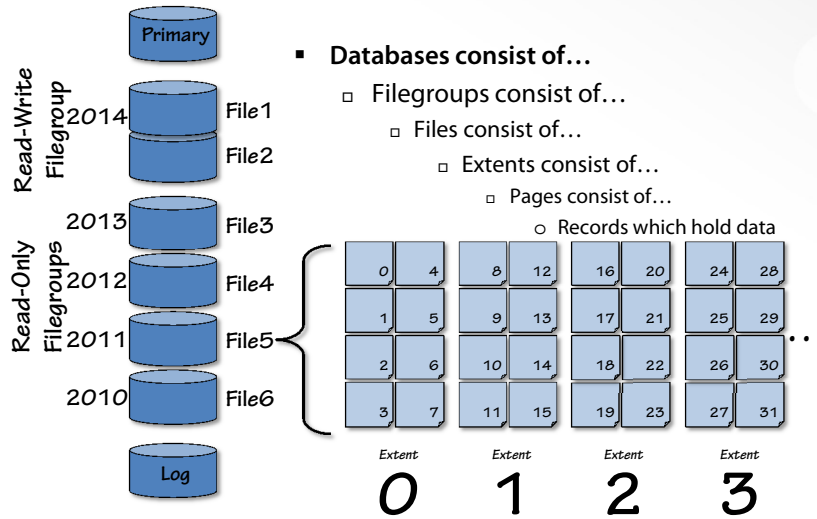
Overview

- Records
- Pages
- Extents
- Allocation bitmaps
- IAM chains and allocation units
- Appendix: Rarely used structures: sparse columns and FILESTREAM

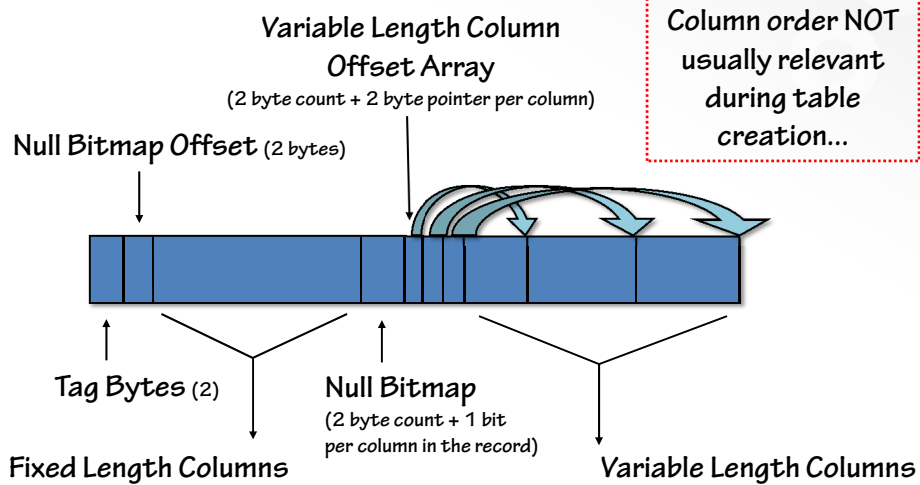
Server Architecture



Database Components



Record Structure (Non-Compressed)

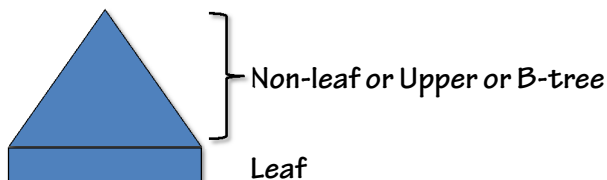


Record Structure Details

- **One bit in the null bitmap for each column in the record**
 - Performance optimization
 - Added columns without default values are not added to records until the record is next updated
 - Same goes for columns with default values from SQL 2012 onwards
- **Null bitmap always exists in data records**
 - Except when table ONLY has SPARSE columns
- **Null bitmap may exist in index records if nullable columns are part of the index record**
- **Variable length column offset array stores offsets of ends of columns**
 - To allow easy calculation of the column size without storing it, saves 2 bytes
- **No need to store row length, saves 2 bytes**
- **Cluster keys will become first columns in data record structure**

Record Types (1)

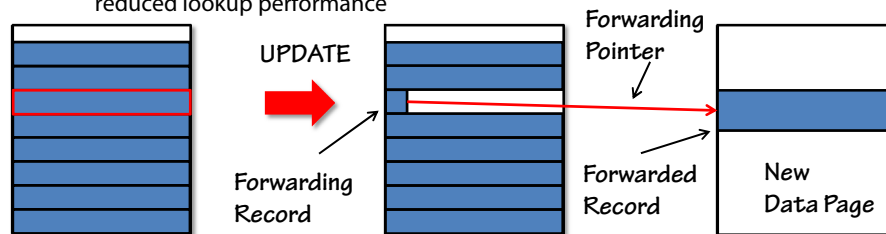
- **Data records**
 - Occur in heaps (tables without clustered indexes) and at the 'leaf-level' of clustered indexes
 - Clustered indexes are stored as B-trees, with the lowest level being data records in data pages (technically B+ trees that are NOT balanced in real-time)
 - Non-unique clustered indexes will contain a hidden 'uniquifier' column
 - Data records store all the columns of the table row
 - Note: 'row' == 'record' == 'slot'



Record Types (2)

- **Forwarding/forwarded records**

- Only occur in heaps
- If a data record is updated to be larger and there is no space on the page, it is moved to a new page, and the old location has a pointer to the new location (and the new record has back-link to the old)
- The record in the new location is the 'forwarded' record, and the pointer to it in the old location is the 'forwarding' record
- This avoids nonclustered indexes having to be updated, but can lead to reduced lookup performance



Record Types (3)

- **Index records**

- Index records come in two types: leaf and non-leaf

- **Leaf-level index records**

- Occur in nonclustered indexes only, at the leaf-level
- Store all nonclustered index key columns, plus:
 - A link to the matching row in the table (heap or clustered index)
 - Any INCLUDED columns

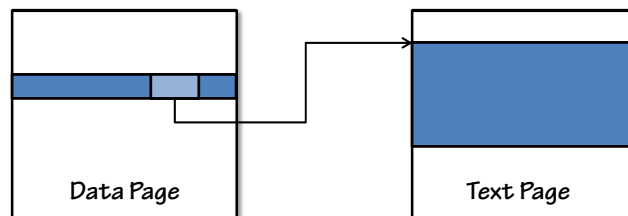
- **Non-leaf-level index records**

- Occur in all index types in the levels above the leaf level
- Contain information to assist the Storage Engine in navigating to the correct point at the leaf level

- **Much more on these with Kimberly**

Record Types (4)

- **Text records**
 - Used to store 'off-row' LOB (Large Object) and all row-overflow data
- **'Off-row' means the data/index record stores a pointer to the root of a loose tree structure that holds the LOB data in text records**
 - Pointer is 16 or 24 bytes, possibly up to 72 bytes in increments of 12 bytes
 - Text tree is not a b-tree like an index



LOB Data Storage Settings

- **Regular and legacy types differ for default on/off-row storage**
 - Legacy types (n/text, image) off-row by default
 - Regular types (n/varchar(max), varbinary(max), XML) on-row by default as long as there is space, and up to 8,000 bytes only
- **For legacy LOB data types:**
 - Use the 'text in row' table option (defaults to OFF)
 - `sp_tableoption N'ProdTable','text in row','ON'`
 - Defaults the in-row limit to 256 bytes
 - `sp_tableoption N'ProdTable','text in row','1000'`
 - Beware! Turning the option off is an immediate size-of-data operation
- **For regular LOB data types:**
 - Use the 'large value types out of row' option (defaults to OFF)
 - `sp_tableoption N'MyTable','large value types out of row','ON'`
 - `sp_tableoption N'MyTable','large value types out of row','OFF'`
 - Existing values are migrated the next time the column is changed

LOB Data Storage Survey

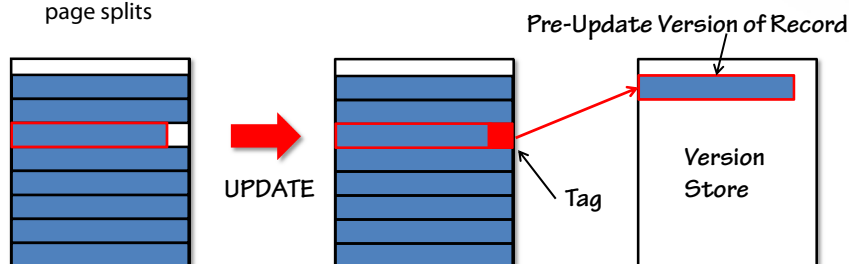
How should you store large-value character data in your schema in SQL 2005+?

As a N/CHAR column		0%	0
As a N/VARCHAR (1-8000) column		1%	1
As a N/VARCHAR (MAX) column in-row		20%	16
As a N/VARCHAR (MAX) column out-of-row		10%	8
As a N/TEXT column in-row		4%	3
As a N/TEXT column out-of-row		4%	3
In a separate table and JOIN to it when required		17%	14
It depends...		40%	32
Other? Enter here...		5%	4
Total: 81 responses			

- Source: <http://www.sqlskills.com/blogs/paul/importance-of-choosing-the-right-lob-storage-technique/> (<http://bit.ly/LRheH>)

Record Types (5)

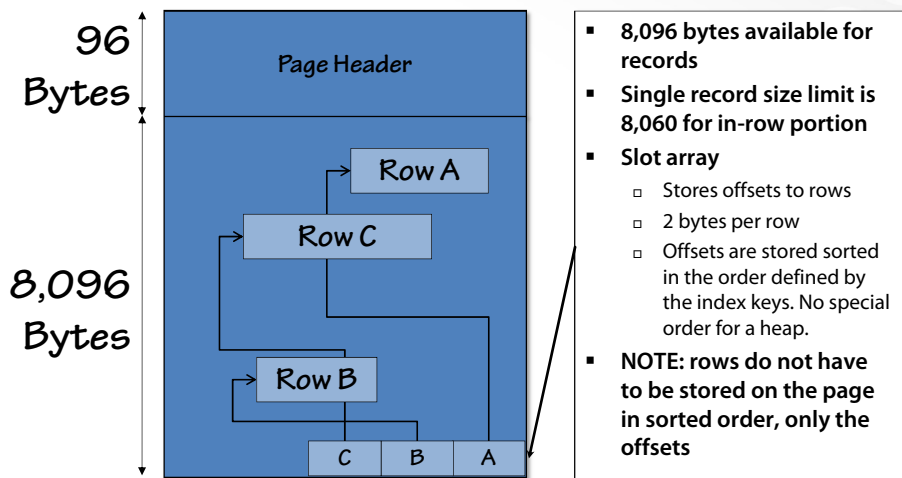
- Versioned records (data, index, text)
 - Used by features that use the versioning system
 - E.g. online index operations, snapshot isolation, DML triggers
 - E.g. allowing Availability Group readable secondaries in SQL 2012+
 - Latest version of record on a page has 14-byte tag on the end
 - Tag contains a timestamp and a pointer into the version store in tempdb
 - Record expansion can cause forwarded records, or index fragmentation from page splits



Record Types (6)

- **Ghost records (data, index, text)**
 - Deleting a record just marks it as 'ghosted' (i.e. logically deleted)
 - Ghosting occurs in indexes (and in heaps when versioning is enabled)
 - Ghosting removes need for key-range locks to protect deleted record
 - No keys in heaps, so no need for ghosting
 - Ghost record removal occurs after transaction commit
 - Performed by 'ghost cleanup' background task
 - Records are not physically overwritten, just the space they occupied on a page is no longer marked as being used, and becomes free space
- **Possible for ghost cleanup to never catch-up...**
 - Ghost cleanup takes page locks, can cause blocking (2012+ is aggressive)
 - Ghost cleanup can be disabled using TF 661, watched using TF 662
 - Ghost cleanup can be forced using:
 - Force an index scan using (INDEX=(x)) syntax, or FORCESCAN hint in 2008 R2 SP1
 - Index rebuild/reorganize, DBCC FORCEGHOSTCLEANUP (be careful!)

Page Structure



Page Types (1)

- **Data pages**
 - Store data records
 - In a heap, or leaf-level of a clustered index
- **Index pages**
 - Store index records
 - At the leaf-level of nonclustered indexes, and non-leaf levels of all index types
- **Text pages**
 - Store text records
 - Actually two types, to support the loose tree structure
 - Text tree pages
 - Used when values are larger than 8KB
 - Text mix pages
 - Used to store multiple values when they are less than 8KB (i.e. shared)

Page Types (2)

- **Boot page**
 - One per database, page (1:9) [page ID = (file:page-in-file)]
 - Stores base metadata about the database as a whole
 - Partially mirrored in log file header pages
 - Contains pointer to starting point for crash recovery
 - More on this in logging module
 - Contains information about most recent backups
 - Corruption = restore of at least file ID 1, or possible hex editor c&p from older restored copy of the same database
 - Dump using DBCC PAGE or DBCC DBINFO

Page Types (3)

- **File header pages**

- One per data and log file, always first page (i.e. page 0)
- Log file header page partially mirrors the boot page
 - Which is what allows a tail-log backup if data files are damaged/destroyed
- Stores metadata about that file
- Corruption = restore of at least that file , or possible hex editor c&p from older restored copy of the same database
 - More tricky if log file header or file ID=1 header
- Dump using DBCC PAGE or DBCC FILEHEADER

- **Allocation bitmaps**

- PFS, GAM, SGAM, IAM, DIFF_MAP, ML_MAP
- More on these later

Demo

Examining pages and records

Using DBCC PAGE and DBCC IND

- **DBCC IND dumps a list of pages**
 - `dbcc ind ( { 'dbname' | dbid }, { 'objname' | objid }, { nonclustered indid | 1 | 0 | -1 | -2 } [, partition_number] )`
- **DBCC PAGE dumps an individual page**
 - `dbcc page ( { 'dbname' | dbid }, filenum, pagenum [, printopt={0|1|2|3} ])`
 - Requires TF 3604 to get results
 - Use WITH TABLERESULTS to get tabular output
- **Also new DMV from SQL Server 2012+**
 - `sys.dm_db_database_page_allocations`

Extents

- **An extent is a group of 8 contiguous pages, starting on 8-page boundaries, with the first starting at page 0 in the data file**
 - Tracked in IAM, GAM, SGAM pages
- **Extents exist to make the allocation system more efficient**
 - First 8 pages allocated to a table/index are one-page-at-a-time from anywhere in the filegroup (i.e. mixed extents)
 - Carry-over from 7.0 when storage was expensive
 - Doesn't happen if TF 1118 is enabled to help with tempdb contention
 - Once 8 pages have been allocated, then switch to dedicated extents
 - When a dedicated extent is allocated to a table/index, only the first page is actually allocated and used
 - The whole extent is reserved for exclusive use by the index/object
 - Pages from that extent are subsequently allocated as needed
 - Allocation bitmaps track extents
 - Backup/restore are at the extent level (except page-level restore)

PFS Pages and Intervals

- **PFS = Page Free Space**
- **A PFS page tracks (among other things):**
 - page allocation state
 - free space for heap data and text pages
- **One byte per page**
- **PFS interval is 64MB, one PFS page per PFS interval**
 - 64MB = 8,088 database pages (8,088 bytes used in the PFS page)
- **Each data file is conceptually split into PFS intervals, starting with page zero in the file**
- **At the start of every PFS interval is a PFS extent from which the PFS page is allocated**
 - Only 7 pages are available for allocation in that extent
 - Except when the PFS interval boundary matches a GAM interval boundary when the whole extent is used for allocation bitmaps

PFS Byte-Map (1)

- **The byte-map contains the following info:**
 - bits 0-2: how much free space is on the page
 - 0x00: empty
 - 0x01: 1 to 50% full
 - 0x02: 51 to 80% full
 - 0x03: 81 to 95% full
 - 0x04: 96 to 100% full
 - bit 3 (0x08): is there one or more ghost records on the page?
 - bit 4 (0x10): is the page an IAM page?
 - bit 5 (0x20): is the page a mixed-page?
 - bit 6 (0x40): is the page allocated?
- **For example, an allocation IAM page will have a PFS value of 0x70 (IAM + mixed + allocated)**

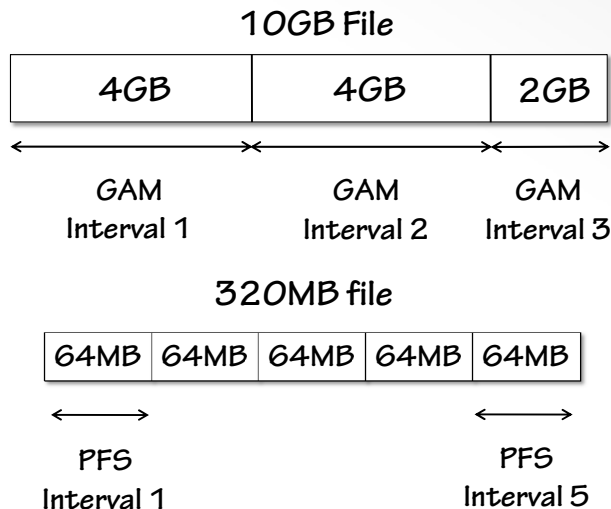
PFS Byte-Map (2)

- **Free space is only tracked on PFS pages for heap data pages and text pages, to help find insertion points**
 - If there are no suitable pages in the in-memory free space cache, PFS pages will be scanned
 - No choice about the insertion point for indexes so no point tracking the free space
 - However, all pages themselves track how much free space they have
- **PFS byte values are not reset on deallocation of a page**
 - The only bit that is changed is the 'allocated' bit
 - All other bits are reset on allocation
 - Done because PFS changes are non-transactional and cannot be rolled back
 - This is just how the algorithm works

Allocation Bitmaps

- All other allocation bitmaps have 1 bit per extent over a roughly 4GB interval, called a GAM interval
- A GAM interval covers 511,232 pages in a data file
- GAM – Global Allocation Map
- SGAM – Shared Global Allocation Map
- DIFF Map – Differential Bitmap
- ML Map – Minimally Logged Bitmap
- IAM page – Index Allocation Map

GAM and PFS Intervals



GAM Pages and Intervals

- PFS pages track the allocation state of pages
- GAM pages track the allocation state of extents
- GAM = Global Allocation Map
 - Is an extent allocated or not (doesn't matter what to)
 - Contains two records – GAM header and GAM bitmap
 - GAM header basically shows the interval covered
 - Simple bitmap covering 64,000 extents (the number of bits that can fit in the bitmap on a page)
 - If the bit is one, it's available for allocation (i.e. it is currently unused)
- The next extent to allocate is found from a pointer in the FCB (File Control Block) instead of searching through GAM pages
- GAM page searches are only done when allocations have reached the end of the file and there is free space

SGAM pages

- **SGAM = Shared GAM**
 - “Shared” is what Books Online uses – pronounce it as “es-gam”
- **Used to expedite finding a mixed extent to allocate from**
- **Exactly the same format as the GAM page but the bitmap semantics are slightly different**
- **Bitmap bit is one**
 - The extent is a mixed extent and *may have* at least one unallocated page available for use (optimistic algorithm)
- **Bitmap bit is zero**
 - The extent is either dedicated or is a mixed extent with no unallocated pages (essentially the same situation given that the SGAM is used to find mixed extents with unallocated pages)

DIFF and ML Map Pages

- **DIFF MAP = Differential Map**
 - Also called the DCM or Differential Change Map
 - All extents that have changed in any way since last full backup
 - Any operation that changes an extent marks it as changed in the differential bitmap for that GAM interval
 - Differential backups scan these to know what to back up
 - Only reset by a full backup
- **ML Map = Minimally-Logged Map**
 - Also called the BCM or Bulk Changed Map
 - Any minimally-logged operation in the BULK_LOGGED recovery model that changes an extent marks it as changed in the minimally-logged bitmap for the GAM interval
 - The next log backup scans these to know which extents to include, and then resets the bitmaps
- **Both have the same format as GAM pages**

Demo

Examining allocation bitmaps

IAM Pages

- IAM = Index Allocation Map
- Tracks all extent allocations for a table/index/partition in a GAM interval in a data file
- Uses the same bitmap format as GAM pages but has different headers
- If the bitmap bit is one, the extent is allocated to whatever grouping of allocations the IAM page belongs to
- IAM page header contains
 - Which GAM interval does the IAM page track extents for?
 - IAM pages do not have to come from the file they map
 - The sequence number and linkages in the IAM chain
 - More on this in a few slides
 - The single-page slot array
 - The first 8 allocations to any object/index are mixed pages and are tracked in this array in the first IAM page for the object/index

Combining Allocation Bitmaps

- The interplay of bits in the various bitmaps follow rules (remembering that IAM bitmaps only track dedicated extents):

GAM	SGAM	IAM	Comments
0	0	0	Mixed extent with all pages allocated
0	0	1	Dedicated extent (must be allocated to only a single IAM page)
0	1	0	Mixed extent with ≥ 1 unallocated page
0	1	1	Invalid state
1	0	0	Unallocated extent
1	0	1	Invalid state
1	1	0	Invalid state
1	1	1	Invalid state

- DBCC CHECKALLOC validates these relationships

Allocating a Page...

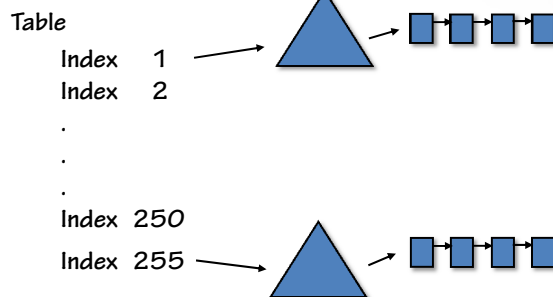
- Allocating the first page in a table is complex...
- Find an extent to allocate from
 - Maybe an existing mixed extent, or allocate a new extent to be mixed
- Allocate the data page
 - Mark it allocated + mixed in the PFS
 - Mark the extent as available in the SGAM
- Allocate the IAM page
 - Mark it allocated + mixed + IAM in the PFS
 - Mark the extent as available in the SGAM
- Track the data page ID in the IAM page
- Track the IAM page ID in the table's metadata
- Track the data page ID in the table's metadata

IAM Chains

- Each IAM page maps a 4GB GAM interval of a file
- If the allocations for a particular table/index/partition are from multiple GAM intervals (in one or more files), multiple IAM pages are needed to track them
- IAM pages are linked together in an IAM chain
- IAM chains are unordered, except by the time order in which an IAM page was added to the chain
 - But there is a doubly-linked list, with a sequence number, that DBCC CHECKDB validates and some operations make use of
- In SQL Server 2000 there was one IAM chain per index, but from SQL Server 2005 onwards it's way more complicated...

IAM Chains in SQL 2000

SQL 2000



Total possible IAM chains = 251

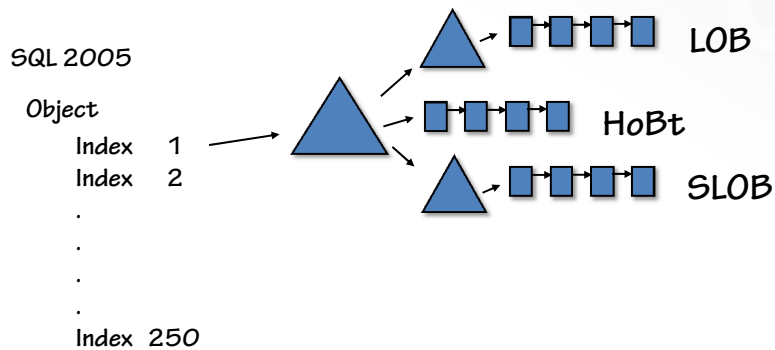
Allocation Changes in SQL 2005

- Allocation metadata rewritten for SQL 2005
- Needed to support 3 new features:
 - Row-overflow (rows larger than 8,060 bytes)
 - One or more variable-length columns pushed off-row
 - INCLUDED columns
 - Ability to INCLUDE non-key columns in a nonclustered index
 - Partitioning
 - Ability to horizontally partition a table or index
- Change from per-table/index IAM chain to multiple IAM chains per-table/index
- Name changed to allocation unit although nothing else about IAM pages and IAM chains changed
- Index Allocation Map is now a bit of a misnomer

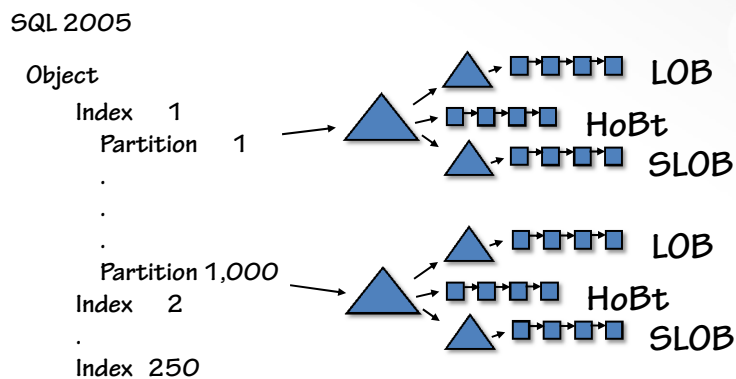
Allocation Unit Names

- Three types of allocation unit:
 - IN_ROW_DATA allocation unit
 - Data and index records
 - LOB_DATA allocation unit
 - Text records for actual LOB columns
 - ROW_OVERFLOW_DATA allocation unit
 - Text records for variable-length columns stored off-row
- The internal names you might see in some tools are, respectively:
 - HoBt – Heap-or-B-tree (pronounced ‘hobbit’ – yes, Lord of The Rings)
 - LOB – Large Object
 - SLOB – Small-LOB

Allocation Units in SQL Server 2005



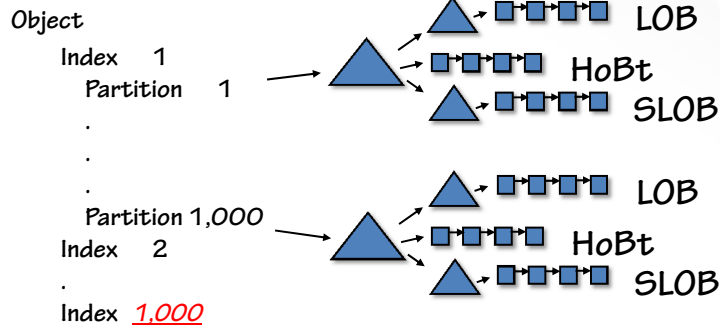
And with Partitioning...



Total possible IAM chains = 750,000 !!!
(plus XML indexes, indexed views)

And from SQL Server 2008...

SQL 2008



Total possible IAM chains = 3 million !!!
(plus XML indexes, indexed views)

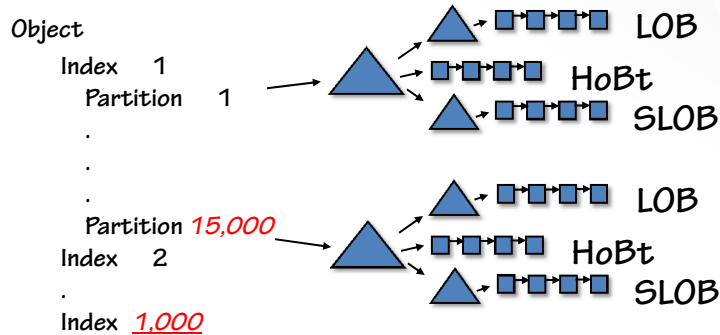


41

© SQLskills, All rights reserved.
<http://www.sqlskills.com>

And from SQL Server 2008 SP2...

SQL 2008 SP2+



Total possible IAM chains = 45 million !!!
(plus XML indexes, indexed views)



42

© SQLskills, All rights reserved.
<http://www.sqlskills.com>

Table Metadata

- Used to be sysindexes, sysobjects, syscolumns in SQL Server 7.0/2000
- From SQL Server 2005 onwards these are catalog views
- Real system tables are now:
 - sys.sysallocunits
 - sys.sysrowsets
 - sys.sysrscols
 - sys.sysschobjs
 - sys.syscolpars
 - sys.sysidxstats
- Hidden unless you connect using the Dedicated Admin Connection

Demo

Examining IAM chains and table metadata

Database Physical Version Number

- All databases have a physical version number
- Physical version number is increased during upgrade
 - And sometimes by SP features...
 - E.g. 2000 = 539, 2005 = 611/612, 2014 = 782
- All SQL Server instances have a maximum physical version number they can understand
 - This is because newer versions introduce new database structures, log records, etc.
- Database compatibility mode/level is irrelevant!
 - Only controls behavior of old query syntax
- SQL Server is NOT up-level compatible
 - You cannot restore or attach a database with a higher physical version to a SQL Server that will not understand it

Resources

- Inside the Storage Engine blog post category
 - <http://www.sqlskills.com/blogs/paul/category/inside-the-storage-engine/>
(<http://bit.ly/hQ3fGF>)
 - Anatomy of a record
 - Anatomy of a page
 - Anatomy of an extent
 - GAM, SGAM, PFS, and Other Allocation Maps
 - IAM pages, IAM chains, and allocation units
 - Ghost cleanup in depth
 - Boot pages, and boot page corruption
 - File header pages, and file header corruption
 - And much more...

Review

- Records
- Pages
- Extents
- Allocation bitmaps
- IAM chains and allocation units

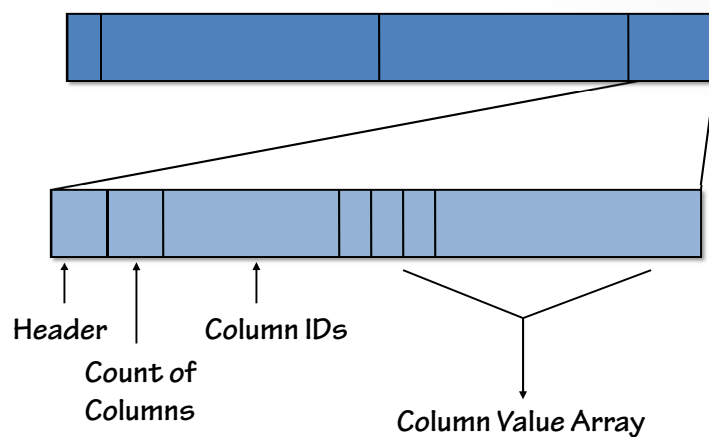
Questions!

Appendix

- Some other structures exist but are very rarely used so it's not worth spending everyone's class time on them
- Let Paul or Kimberly know if you'd like to discuss in more depth

Sparse Column Structure

Regular Record Structure



Sparse Column Details

- **Row with NULL sparse columns has no storage for them**
 - No entry in the NULL bitmap or fixed length portion
- **Row with sparse values**
 - Adds 6 bytes to the row size + 4 bytes per non-null sparse column + the column values
 - Non-null sparse values are stored in the sparse column array, which looks like a variable-length column array, and is stored as a variable-length column in the record
 - 2 bytes for the extra offset in the variable length column offset array (with 0x8000 bit set to indicate a sparse column array)
 - 2 bytes for format version and supported features (0x0005 for 2008+)
 - 2 bytes for count of sparse values
 - List of column IDs with values (2 bytes per ID)
 - List of column value offsets within sparse column array (2 bytes per column)

FILESTREAM Storage

- **Designed to allow:**
 - Fast access to LOB data
 - Greater LOB size than can be stored inside the database
- **FILESTREAM attribute on varbinary(max) columns**
- **Data is stored in the file system but rooted in the database**
- **LOB data kept transactionally consistent with structured data**
- **Accessible through T-SQL AND Win32 streaming APIs**
- **LOB size only limited by NTFS volume size**
- **Manageability integrated into SQL Server**
- **Access to the FILESTREAM data is controlled with a file system filter driver**

Preparing for FILESTREAM

- **Make sure the data is suited to FILESTREAM**
 - Design point is for data value 1MB and higher
- **Choose appropriate RAID level**
- **Separate data containers from each other and other database files, according to usage patterns**
- **Prepare NTFS for performance**
 - Defrag the volume
 - Disable 8.3 names and last access time tracking using fsutil
 - Set a large cluster size
 - (See the whitepaper reference at end of deck)

Enabling FILESTREAM

- **FILESTREAM is enabled in two stages**
 - Windows admin will enable FILESTREAM for an instance during SQL Server setup or using the SQL Server Configuration Manager
 - This allows the file system filter driver to be installed
 - SQL admin will enable/disable support through sp_configure
- **When enabling for the first time, computer or instance restart may be required to hook in the file-system filter driver**
- **Access options are:**
 - T-SQL access only
 - T-SQL and local file-system access
 - T-SQL, local, and remote file-system access (through a share)

Creating a FILESTREAM Filegroup

- **FILESTREAM data for a table stored in “data container”, which maps to a FILESTREAM filegroup**
 - Data containers stored as NTFS directories
 - Can be on compressed volumes if performance allows
- **Create a FILESTREAM filegroup at database creation time or add later**
- **Syntax**
 - ... FILEGROUP FileStreamGroup CONTAINS FILESTREAM (NAME = FileStreamDocuments,
 - FILENAME = N'C:\SQLskills\FileStreamDocuments')
- **Notes:**
 - CONTAINS FILESTREAM clause required
 - FILENAME has to exist up to but NOT including the last directory
 - Can specify maximum size from SQL 2012+

FILESTREAM Structures



- **Data container**
 - One directory per table (or partition of table)
 - One sub-directory per FILESTREAM column
 - Opaque GUID names can be found from system tables using Dedicated Admin Connection (as in demo)
- **FILESTREAM file names are transaction log LSNs (more in the logging module)**
- **FILESTREAM files are rooted in data records**

Security

- **FILESTREAM directories have restricted access while the database is open**
 - Users have permissions to open files based on SQL Server table/column security settings
 - Privileged users *can* mess with the files though...
- **When the database is closed, FILESTREAM directory access is gated by Windows security**
- **As the demo showed, the FILESTREAM directories and files are not file locked like open databases and can be altered/deleted when not actively in use**
- **FILESTREAM data cannot be natively encrypted by SQL Server**

Transactional Consistency

- **FILESTREAM has its own log, that is kept in sync with the regular database transaction log**
- **Point-in-time recovery is fully supported**
- **No partial updates, all updates create a whole new file**
- **When a FILESTREAM value is updated (by creating a whole new file), the old file is not deleted immediately**
 - The link on the data page is updated to point to the new file
 - After the transaction has committed, the OLD file is made available for garbage collection
 - If the transaction rolls back, the change to the link in the data page is rolled back and the NEW file is made available for garbage collection
- **Garbage collection is the process of removing unneeded FILESTREAM files when a checkpoint occurs**

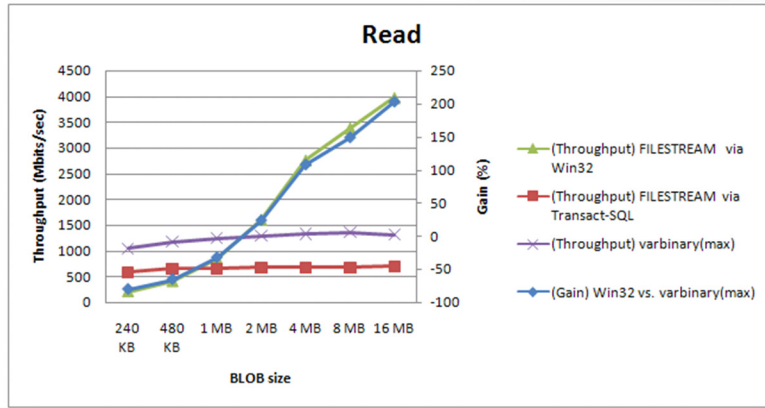
Limitations

- **Only works with NTFS**
- **FILESTREAM files cannot be properly deleted or renamed through Windows**
 - Operations have to be done transactionally otherwise they cause corruption
- **FILESTREAM columns cannot be INCLUDED columns**
- **In-place partial updates not supported**
 - New file created and old file only available for deleting after commit
- **Cannot monitor FILESTREAM data size from within SQL Server**

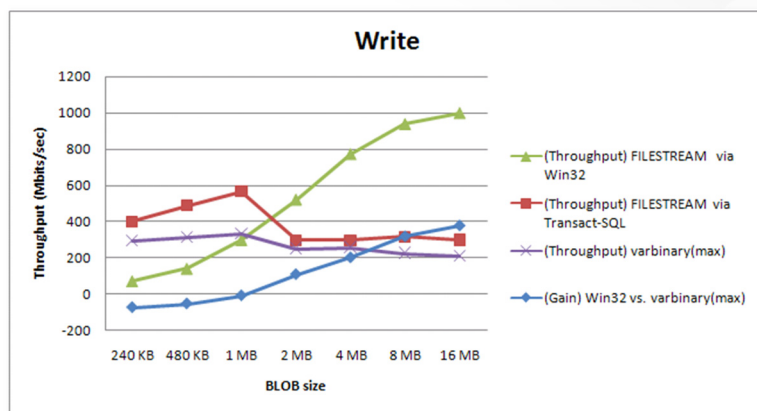
Unsupported Feature Combinations

- **Database mirroring**
 - Database mirroring cannot be configured if the database contains FILESTREAM (i.e. FILESTREAM data CANNOT be mirrored)
 - This has been a MAJOR barrier to adoption
 - (But DOES work with Availability Groups in SQL 2012+)
- **Database snapshots**
 - Snapshot can be created on non-FILESTREAM filegroups only
 - Revert to snapshot is not permitted
- **SQL Server encryption**
 - FILESTREAM data cannot be encrypted using either column level encryption OR Transparent Data Encryption

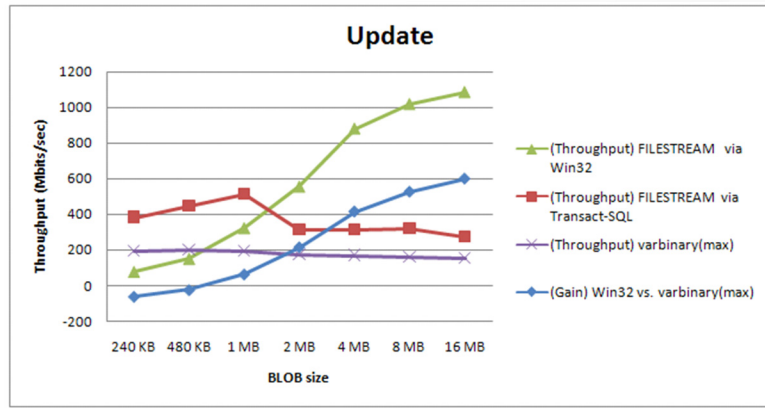
Local Read Performance



Local Insert Performance



Local Update Performance



Resources

- **FILESTREAM**
 - WP: FILESTREAM Storage in SQL Server 2008
 - <http://msdn.microsoft.com/library/hh461480>
 - Blogs:
 - <http://www.sqlskills.com/blogs/paul/category/filestream/>