SQLskills Immersion Event

IEPTO1: Performance Tuning and Optimization

Module 1: Database Structures

Paul S. Randal
Paul@SQLskills.com

Why Cover Internals?

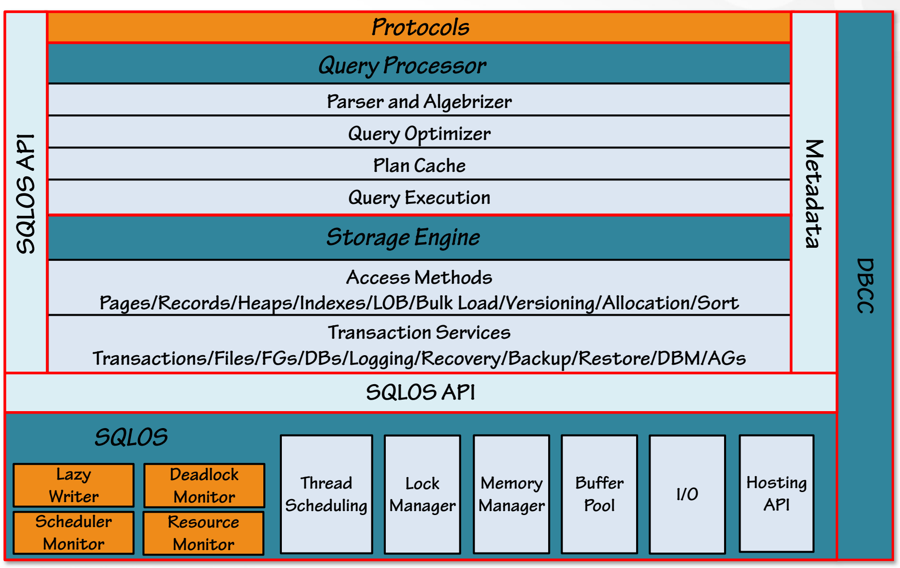
- Internals aren't just to geek-out on (although that's fun to do too! ☺)
- Understanding how data is stored, accessed, and optimized at all levels is key when architecting a system so that it will perform well and be more easily maintained
  - Explains why some decisions are good or bad…
  - Helps to 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

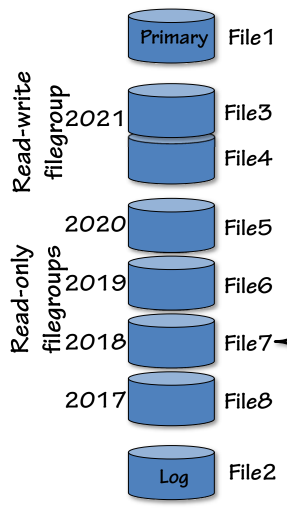
- Records
- Pages
- Extents
- Allocation bitmaps
- IAM chains and allocation units

- Note:
  - In-memory OLTP tables have opaque and entirely different set of structures
    - Good primer at https://sqlskills.com/p/001
  - Columnstore indexes have opaque and entirely different set of structures
    - Good primer at https://sqlskills.com/p/002

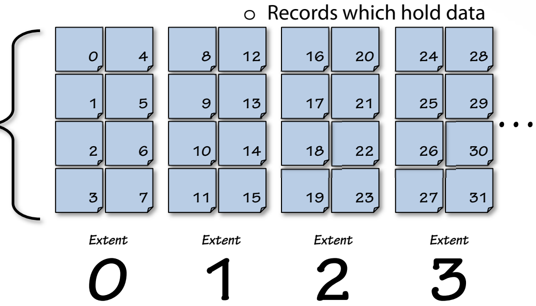
Server Architecture



Database Components

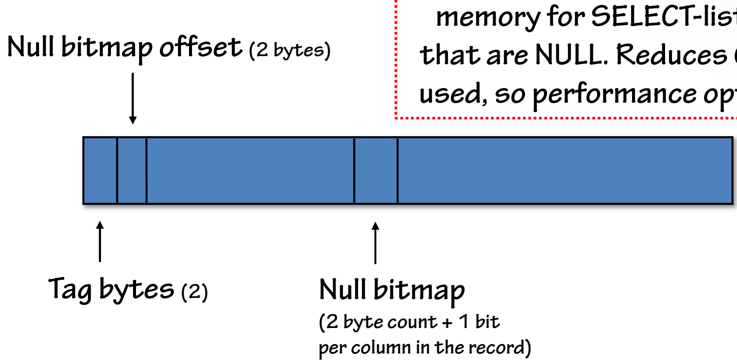


- Databases consist of…
  - Filegroups consist of…
    - Files consist of…
      - Extents consist of…
        - Pages consist of…
          - Records which hold data

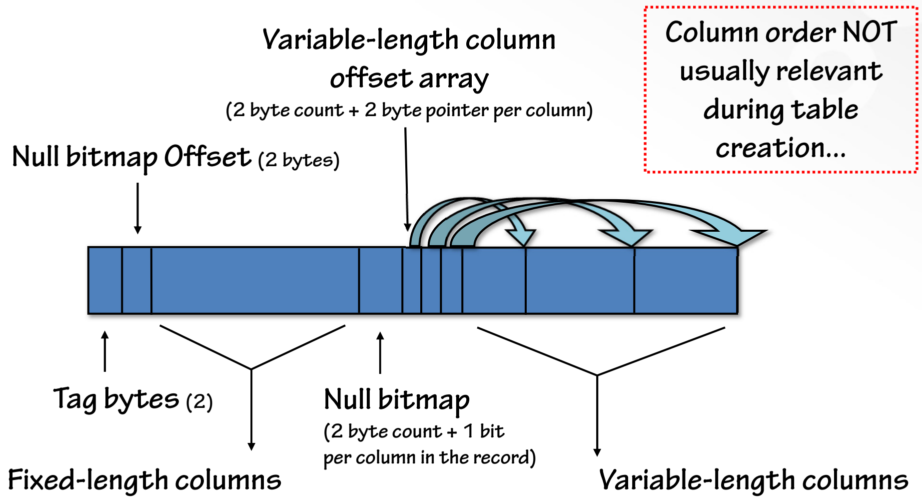


Record Structure (Non-Compressed)

Null bitmap lets SQL Server avoid reading portions of record into CPU memory for SELECT-list columns that are NULL. Reduces CPU cycles used, so performance optimization.



Record Structure (Non-Compressed)



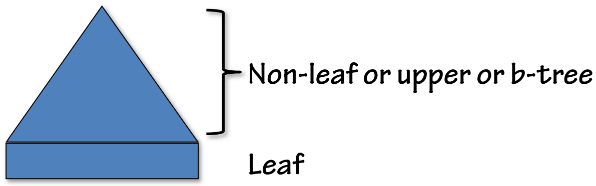
notes for prior slide

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 always exists in nonclustered indexes (SQL 2012+)**
- **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**
  - In a heap, columns are ordered based on column list in CREATE TABLE

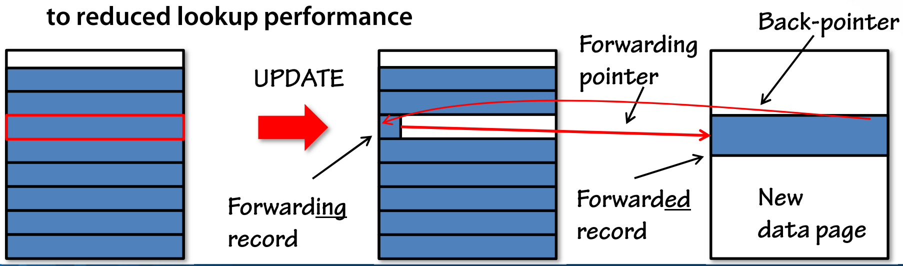
Data Records

- **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'

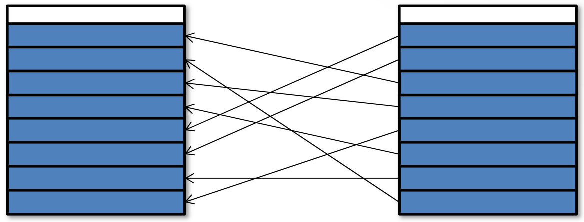


Forwarded/Forwarding Records

- **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**



Back-pointers in Index Records



SELECT columns
WHERE c2 = <val>

Heap (c1, c2, …, c10)

NC index (c2, data record identifier (RID))

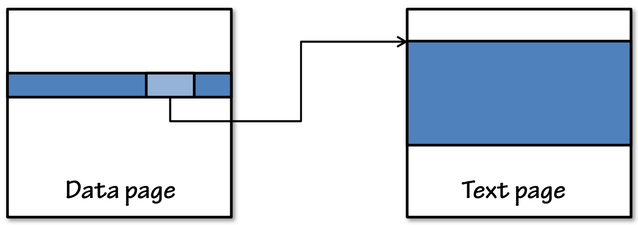
Base table structure	NC index pointer (RID)
Heap	Physical RID (file:page:slot)
Clustered index	Logical RID (cluster key(s))

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**

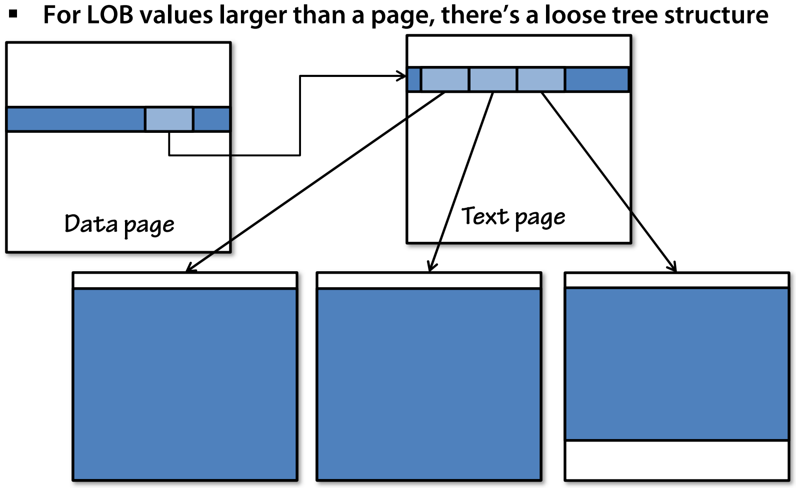
Text Records

- **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



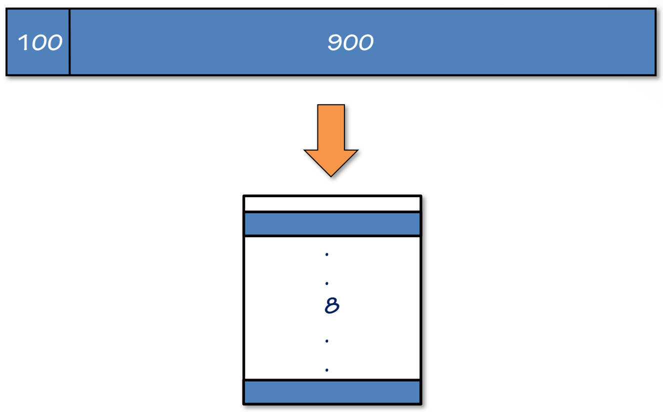
Text Records

- **For LOB values larger than a page, there's a loose tree structure**



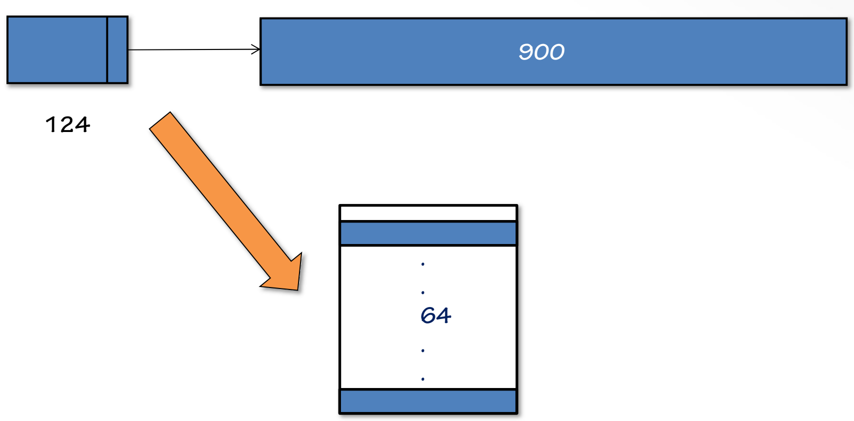
In-row vs. Off-row

- Choice can affect performance through low data-density
- How often is the 900-byte CHAR column used?



In-row vs. Off-row

- Splitting out the uncommonly-used data to off-row means higher data density for the commonly-used data so better performance

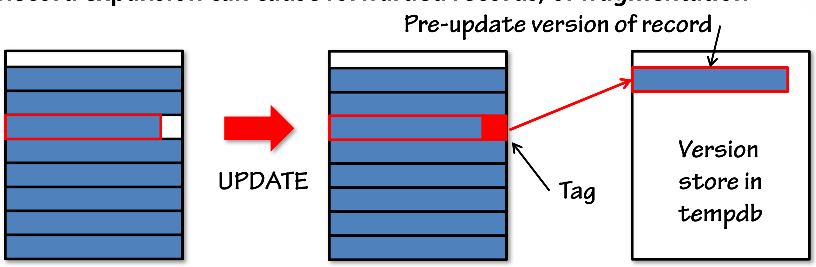


LOB Data Storage Settings

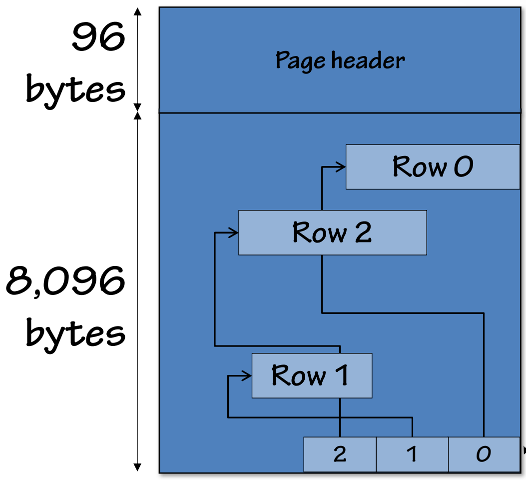
- **Regular and legacy types differ for default on/off-row storage**
  - Legacy types (n/text, image) off-row by default
  - Regular types (n/varchar(max), varbinary(max), XML) on-row by default as long as there is space, and up to 8,000 bytes only
- **For legacy LOB data types:**
  - Use the 'text in row' table option (defaults to OFF)
  - 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
- **Should LOB data be stored in-row or off-row? It depends!**

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 AG readable secondaries – see https://sqlskills.com/p/003
- **Latest version of record on a page has 14-byte tag on the end**
  - Tag contains 'timestamp' and pointer into version store
  - Can be a chain of previous versions
- **Record expansion can cause forwarded records, or fragmentation**



Page Structure

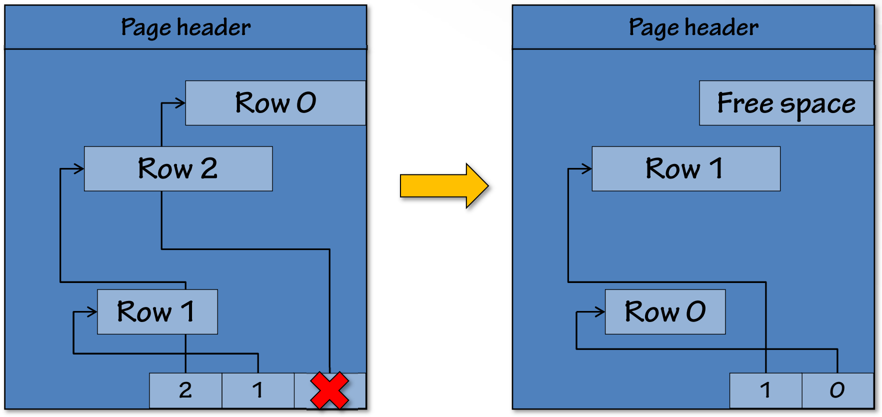


- 8,096 bytes available for records and slot array
- Single record size limit is 8,060 for in-row portion (+version tag +forwarded record back-pointer +slot array entry +future use)
- Slot array stores offsets to rows
  - 2 bytes per row, stored sorted in the order defined by the index keys. No special order for a heap.
- NOTE: rows do not have to be stored on the page in sorted order, only the offsets

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
- Ghost record removal occurs after commit by ghost cleanup 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…
  - Could be blocked by long-running query on AG secondary
  - Ghost cleanup takes page locks, can cause blocking (2012+ is aggressive)
  - Ghost cleanup can be disabled using TF 661, watched using TF 662
    - DBCC TRACEON (662 or 661, -1) so background task picks up the trace flag
  - Ghost cleanup can be forced using:
    - Force an index scan, index rebuild/reorganize, DBCC FORCEGHOSTCLEANUP
    - sp_clean_db_file_free_space and sp_clean_db_free_space

Ghost Record Removal



- Slot entry of deleted record removed and old record is now free space
- Log record: LOP_EXPUNGE_GHOST

Common Page Types

- **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)
- **Allocation bitmaps**
  - PFS, GAM, SGAM, IAM, DIFF_MAP, ML_MAP
  - More on these later

Boot Page

- **Most important page in the database**
- **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 cut-and-paste from older restored copy of the same database**
- **Dump using DBCC PAGE or DBCC DBINFO**
  - Must also enable trace flag 3604 to get output

SQLskills immerse yourself in sql server

23

File Header Pages

- **One per data and log file, always first page (i.e. page 0 in every file)**
- **Log file header page partially mirrors the boot page**
  - This 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 cut-and-paste 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**
  - Must also enable trace flag 3604 to get output

SQLskills immerse yourself in sql server

24

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 undocumented DMV from SQL Server 2012+**
  - sys.dm_db_database_page_allocations (equivalent of DBCC IND)
- **And new documented DMV from SQL Server 2019+**
  - sys.dm_db_page_info (equivalent of page header from DBCC PAGE)

Extents

- **Extents exist to make the allocation system more efficient**
- **Extent is group of 8 contiguous pages, starting at page 0 in data file**
  - Tracked in allocation bitmaps (IAM, GAM, SGAM pages)
- **Mixed extents vs. dedicated extents**
  - Mixed: pages are shared with up to 8 objects/indexes
  - Dedicated: pages are reserved for exclusive use of 1 object/index
- **Default behavior before 2016 (unless disabled with TF 1118)**
  - First 8 pages allocated to a table/index are one-page-at-a-time from anywhere in the filegroup (i.e. mixed extents)
  - Once 8 pages have been allocated, then switch to dedicated extents
    - When dedicated extent is allocate, only first page is actually allocated and used
- **Mixed extents off by default in SQL Server 2016+**
  - ALTER DATABASE … SET MIXED_PAGE_ALLOCATION {ON | OFF}

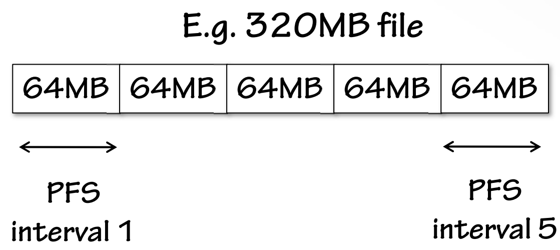
27

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 only
    - No point for indexes, as insertion point is dictated by index key
- **PFS page tracks 64MB of a data file (called a 'PFS interval')**
  - One byte in the PFS page per data file page, in the first extent
  - 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**

28

PFS Intervals



PFS Bits

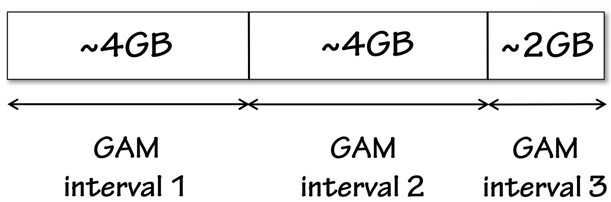
- **Each byte contains the following info:**
  - bits 0-2: how much free space is on the page
    - 0x00: empty
    - 0x01: 1 to 50% full
    - 0x02: 51 to 80% full
    - 0x03: 81 to 95% full
    - 0x04: 96 to 100% full
  - bit 3 (0x08): is there one or more ghost records on the page?
  - bit 4 (0x10): is the page an IAM page?
  - bit 5 (0x20): is the page a mixed-page?
  - bit 6 (0x40): is the page allocated?
  - Bit 7 (0x80): does the page have a row from an aborted transaction (2019+)
- **For example, an allocation IAM page will have a PFS value of 0x70 (IAM + mixed + allocated)**
  - Even on 2016+, where mixed extents are off by default – still used for IAMs

Allocation Bitmaps

- **All other allocation bitmaps have 1 bit per extent over 4GB interval**
  - Called a GAM interval, easier just to think of it as a 4GB interval
  - Equivalent to 511,232 pages in a data file; 63,904 extents; ~3.9GB
- **GAM – Global Allocation Map**
  - Page 2, then every 511,232 pages
- **SGAM – Shared Global Allocation Map**
  - Page 3, then every 511,232 pages
- **DIFF Map – Differential Bitmap**
  - Page 5, then every 511,232 pages
- **ML Map – Minimally Logged Bitmap**
  - Page 6, then every 511,232 pages
- **IAM page – Index Allocation Map**
  - Allocated as needed

GAM Intervals

E.g. 10GB file



GAM Pages

- 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)
  - If the bit is one, it's available for allocation (i.e. it is currently unused)
- GAM page searches are only done when allocations have reached the end of the file and there is free space
  - Before that, the next extent to allocate is found from a pointer in the FCB (File Control Block) instead of searching through GAM pages
  - I.e., what's the current highest-allocated extent in the file?

SGAM pages

- SGAM = Shared GAM
  - "Shared" is what Books Online uses – pronounce it as "es-gam"
- Used to help 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**

First Extent in a Data File

- **Page 0 = file header page**
- **Page 1 = first PFS page**
  - Repeats as page 0 of extent every 1,011 extents
- **Page 2 = first GAM page**
- **Page 3 = first SGAM page**
- **Page 4 = UNUSED (used to be first fixed page of sysobjects)**
- **Page 5 = UNUSED (used to be first fixed page of sysindexes)**
- **Page 6 = first DIFF map page**
- **Page 7 = first ML map page**

- **Reserved extent every 63,904 extents that have the four map pages as pages 2, 3, 6, 7 of that extent, with pages 0, 1, 4, 5 unused**

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?
    - Because 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
    - Unless mixed extents disabled, 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 (and CHECKDB) validates these relationships

SQLskills immerse yourself in sql server

39

Allocating First Page in a Table

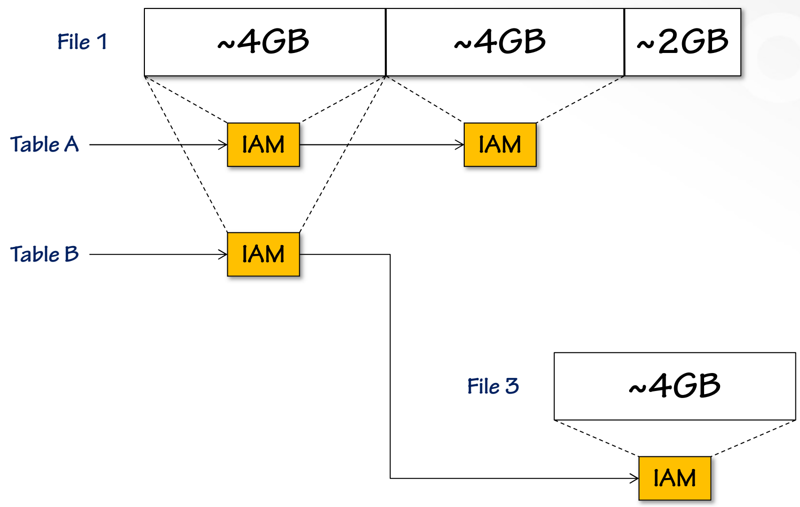
- Find an extent to allocate from
  - Allocate new extent (or from mixed extent if mixed page)
- Allocate the first data page
  - If mixed extents available, find a page from one, otherwise allocate an extent and allocate first page from it
  - Mark it allocated in the PFS (+ mixed if mixed extent)
  - (If mixed, mark the extent as a mixed extent in the SGAM)
- Allocate the IAM page
  - Mark it allocated + mixed + IAM in the PFS
  - Mark the extent as a mixed extent in the SGAM
- In the IAM page, if data page is mixed, enter page ID in the single page slot array, otherwise set the extent's bit in the bitmap
- Enter the IAM page ID in the table's allocation metadata
- Enter the data page ID in the table's allocation metadata

SQLskills immerse yourself in sql server

40

© SQLskills, All rights reserved.
https://www.SQLskills.com

IAM Chains



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

Table

Index 1

Index 2

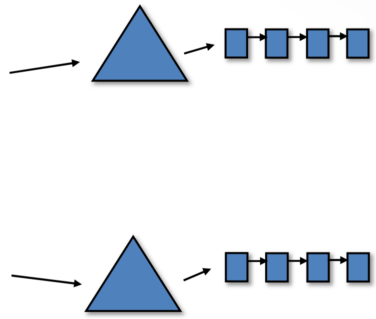
.

.

.

Index 250

Index 255



Total possible IAM chains = 251

Allocation Changes in SQL 2005+

- **Allocation metadata rewritten for SQL Server 2005**
  - No further changes since then
- **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 became 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

SQL 2005

Object

Index 1

Index 2

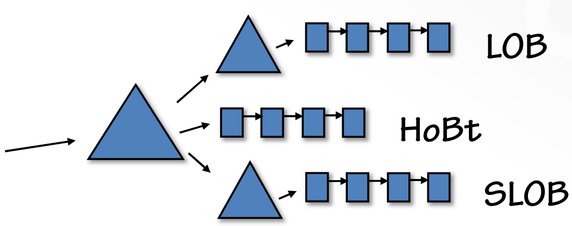
.

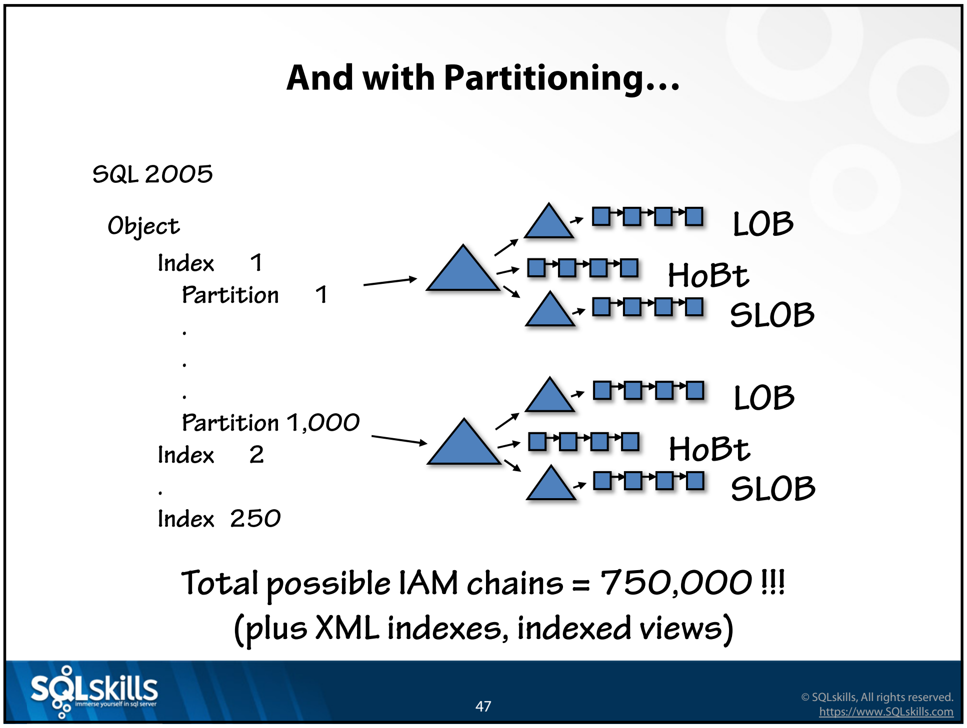
.

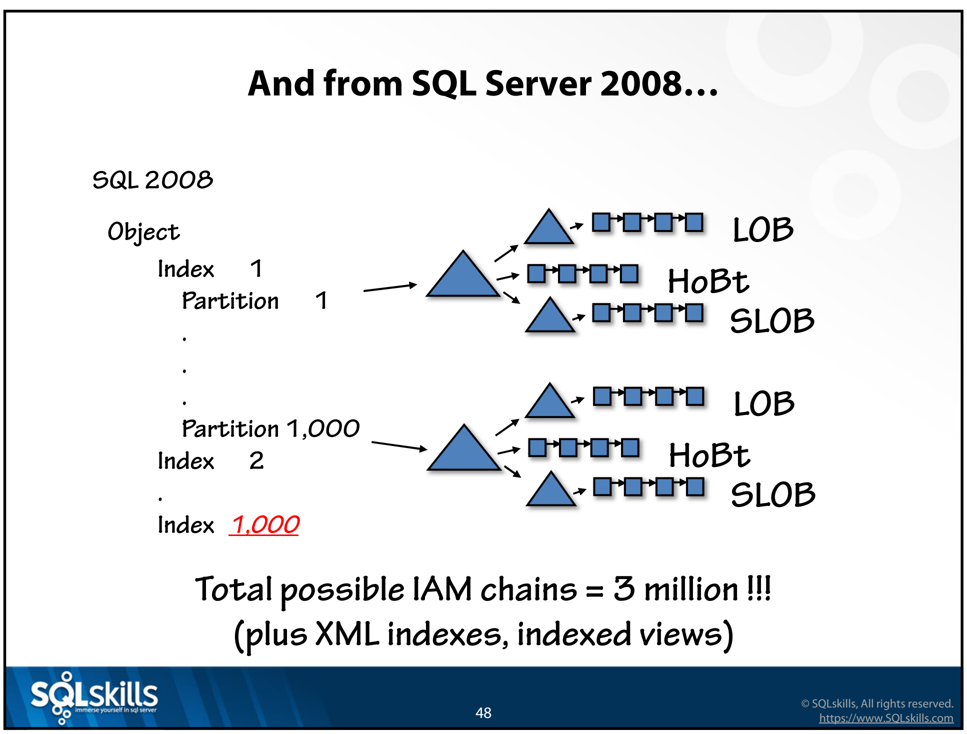
.

.

Index 250




And with Partitioning…
SQL 2005
Object
Index 1
Partition 1
.
.
.
Partition 1,000
Index 2
.
Index 250
LOB
HoBt
SLOB
LOB
HoBt
SLOB
Total possible IAM chains = 750,000 !!!
(plus XML indexes, indexed views)
SQLskills
47
© SQLskills, All rights reserved.
https://www.SQLskills.com


And from SQL Server 2008…
SQL 2008
Object
Index 1
Partition 1
.
.
.
Partition 1,000
Index 2
.
Index 1,000
LOB
HoBt
SLOB
LOB
HoBt
SLOB
Total possible IAM chains = 3 million !!!
(plus XML indexes, indexed views)
SQLskills
48
© SQLskills, All rights reserved.
https://www.SQLskills.com

And from SQL Server 2008 SP2…

SQL 2008 SP2+

Object

Index 1

Partition 1

.

.

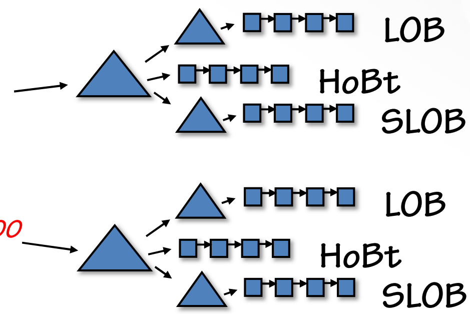
.

Partition *15,000*

Index 2

.

Index *1,000*



Total possible IAM chains = *45 million !!!*

(plus XML indexes, indexed views)

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
  - And others…
- **Hidden unless you connect using the Dedicated Admin Connection**

Demo

Examining IAM chains and table metadata

Database Physical Version Number

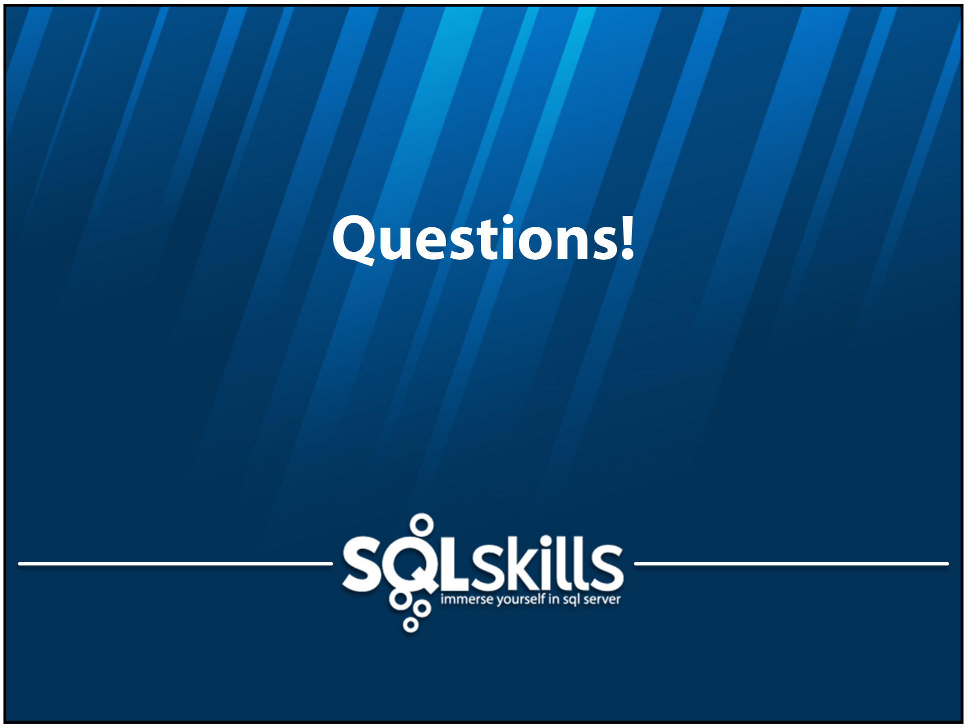
- **All databases have a physical version number**
- **Physical version number is increased during upgrade**
  - And sometimes by SP features…
  - E.g. 2005 = 611/612, 2014 = 782, 2017 = 869, 2019 = 904
- **All SQL Server instances have a maximum physical version number they can understand**
  - Newer 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**
  - https://sqlskills.com/p/004
    - 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!
SQLskills
immerse yourself in sql server