

SQLskills Immersion Event IE1: Internals and Performance

Module 8: Index Fragmentation

Paul S. Randal
Paul@SQLskills.com

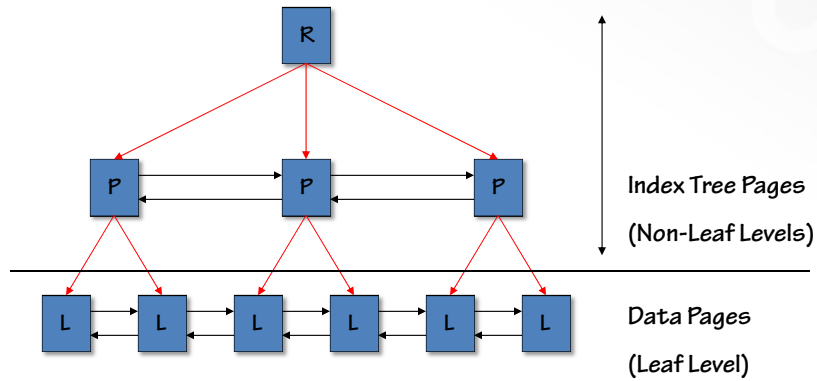


Overview

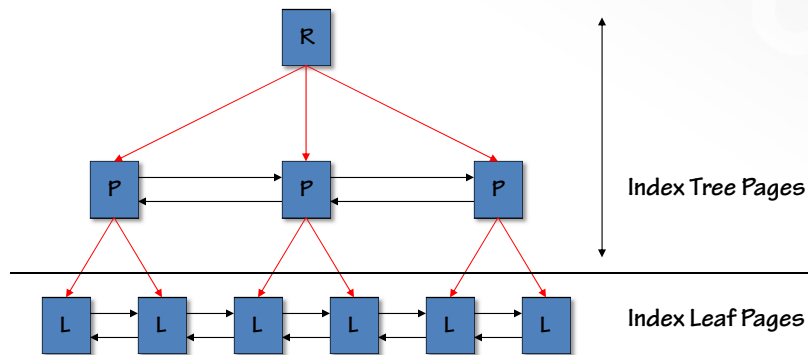
- Data access methods
- What is fragmentation?
- How does fragmentation happen?
- Detecting, mitigating, removing fragmentation
- 2010 PASS presentation on GUIDs
 - **Tales from the Trenches: GUIDs - Use, Abuse and How to Move Forward**
<https://passmedia.sqlpass.org/SummitOnDemand/2010/Top10/AD372S.wmv>



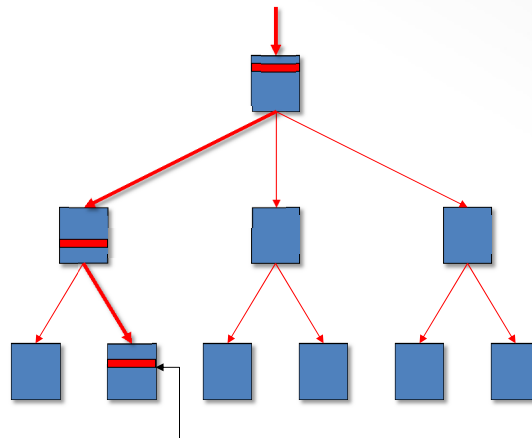
Clustered Index Structure



Nonclustered Index Structure

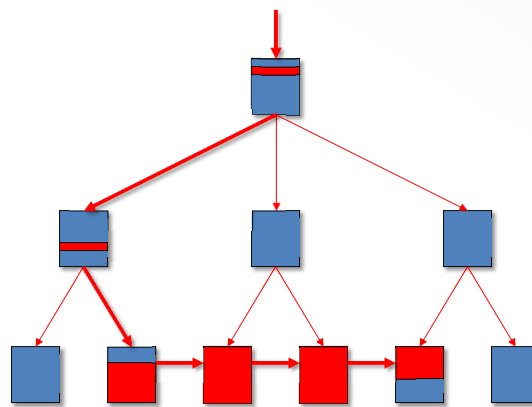


Singleton Lookup (Index Seek/Clustered Index Seek)



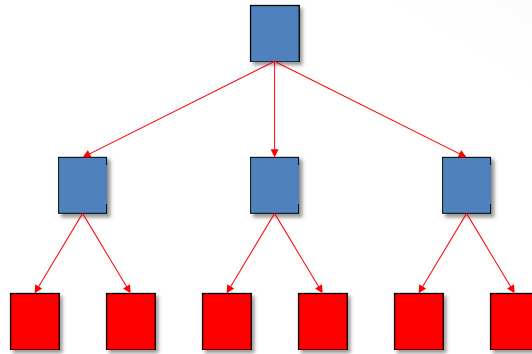
Matching record

Range Scan (Index Scan/Clustered Index Scan)



Matching records (in red)

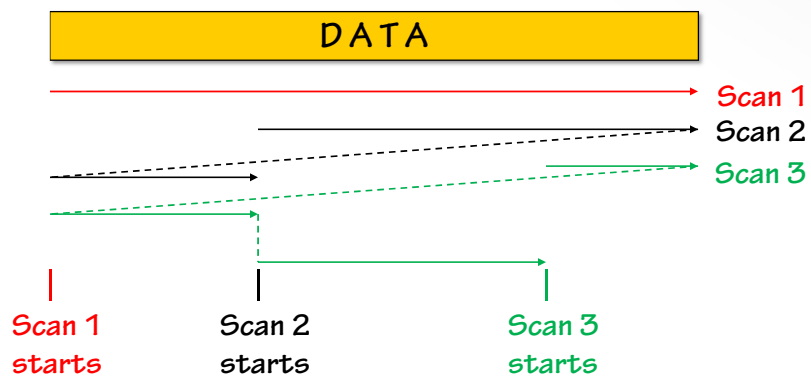
Allocation Order Scan (Table scan or unordered Clustered/Index scan)



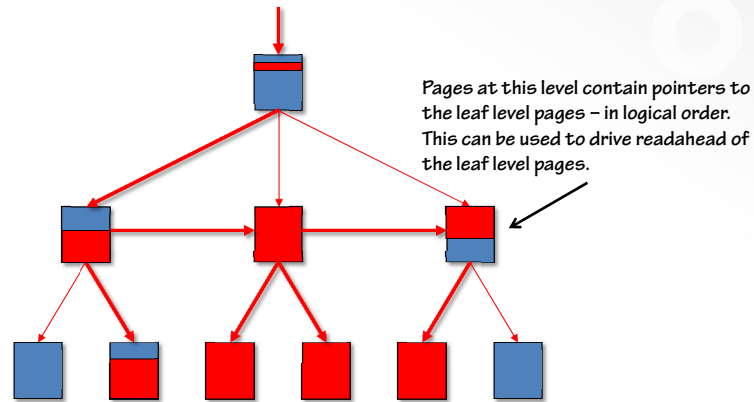
Matching records (in **red**)

Side Note: Merry-Go-Round Scans

- One more reason to not get index order with 'SELECT *'?
- Enterprise only, no control or monitoring possible



Range-Scan: Readahead (1)



Range-Scan: Readahead (2)

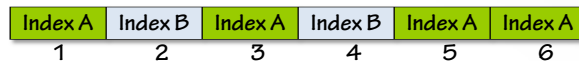
- **Why use readahead?**
 - Keep the CPUs busy, maximize throughput, avoid I/O waits
 - More efficient to issue 1 x 8-page I/O than 8 x 1-page I/Os
- **Feedback mechanism to determine how far ahead of the scan point to read**
- **Driven from parent level during range scans**
 - Parent level pages contain logically-ordered links to the leaf level
- **Issues 1, 8, 32, etc up to 1024 page I/Os**
 - 8, 32+ page I/Os only possible with contiguous pages
 - Better contiguity = bigger I/Os
- **Problem: fragmentation affects range scans**

Logical Fragmentation Defined

- (Sometimes called “external” fragmentation)
- Occurs when the next logical page is not the next physical page
- Prevents optimal readahead
 - Reduces range scan performance
- Does not affect pages that are already in cache
 - Smaller indexes affected less (e.g. 1-5000 pages or less)
- Reported as `avg_fragmentation_in_percent` in the `sys.dm_db_index_physical_stats` DMV in 2005+ for indexes

Extent Fragmentation Defined

- Occurs when the extents in an index are not contiguous



- Also affects readahead but not as much
 - When writing the DMV, we decided to remove it to avoid confusion
- Reported as `avg_fragmentation_in_percent` in the DMV in 2005+ for heaps ONLY
- (2000: Extent fragmentation algorithm in DBCC SHOWCONTIG is documented as not working for multiple files)

Page Density Defined

- (Sometimes called “physical” or “internal” fragmentation)
- Page fullness is below the optimal level so lots of wasted space
- Effect is:
 - Increased disk space (more pages required to hold the same number of rows)
 - Increased I/Os to read the same amount of data
 - Greater memory usage if most of the index is memory resident
- Reported as `avg_page_space_used_in_percent` in the DMV

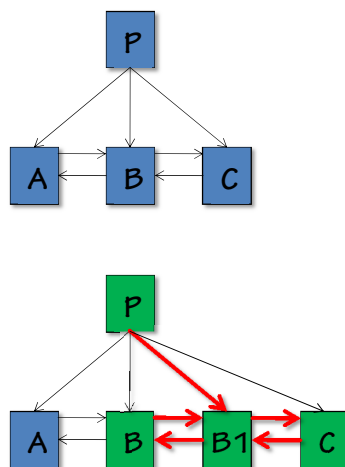
Using -E Startup Option

- <http://support.microsoft.com/kb/329526>
- SQL CAT advises using it, as does Fast Track DW
- During index build/rebuild (and all other operations):
 - 2005: 4 extents allocated before round-robin (256KB)
 - 2008+: 64 extents allocated before round-robin (4MB)
 - NOT single allocations of 4 or 64 extents
- Useful for very large indexes and range scans
- Combine with large RAID stripe size
- Also consider low MAXDOP for maximum contiguity
 - Based on anecdotal evidence only

What is a Page Split?

- **Occurs when a record must be inserted onto a specific page in the index (determined by index key value) and there is not enough space**
 - Could be caused by a new record or an updated record that is now longer than it was before
 - Could also be caused by enabling snapshot isolation, which makes updated records 14-bytes longer
- **The page has to 'split' to make room**
 - Split point is usually as close to 50/50 as possible, but may be skewed if Storage Engine can determine an obvious split point)

Updates During a Page Split



- **Page B splits into B and B1**
- **New page B1 is unlikely to be physically adjacent to B**
- **Pages P, B, C must be updated to change/create page links**
 - Page P might split...
- **All changes are fully logged**

Page Split Mechanism

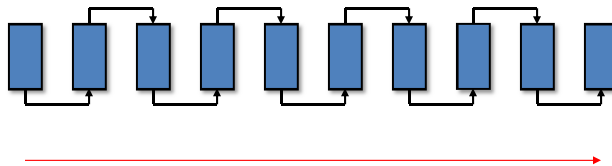
- **A new page is allocated to the index**
 - Usually not physically contiguous to the splitting page so immediately it is logically fragmented
- **All records before the split point stay on the original page**
- **All records after the split point are moved to the new page**
- **New record is written to either page, determined by key value**
- **Both pages (old, and newly allocated) now have low page density**
- **New record must be inserted into index level above the leaf**
 - Could also cause a page split
- **Summary: page splits are very expensive and are the main cause of fragmentation**

Tracking Page Splits

- **There are 'good' and 'nasty' page splits...**
 - 'Good' split is when a page is allocated as part of an append-only insert pattern
 - 'Nasty' split is when a real page split occurs
- **Unfortunately, all documented methods of tracking page splits prior to SQL Server 2012 do not allow differentiation between 'good' and 'nasty' page splits**
 - Perfmon counter
 - Sys.dm_db_index_operational_stats
 - Extended event (possibly with post-processing)
- **Either use log/log backup scanning or 2012 Extended Events**
 - See my blog post at <http://bit.ly/UG1SyG>

Page Splits in Action (1)

Index leaf level of newly built index

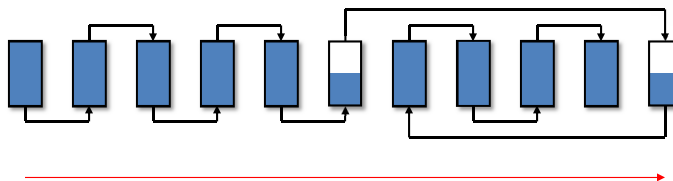


Red arrow is the allocation order

Black arrows are following the logical order

Page Splits in Action (2)

Newly built index leaf after a single page split

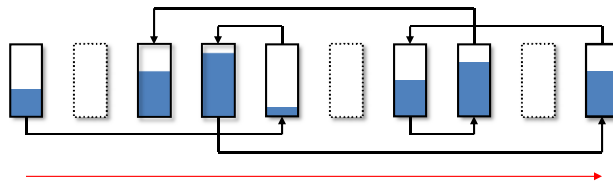


Red arrow is the allocation order

Black arrows are following the logical order

Page Splits in Action (3)

Index leaf level after random inserts/deletes



Red arrow is the allocation order

Black arrows are following the logical order

Demo

Increased logging during page splits

What Causes Fragmentation?

- **Schemas/workloads that cause page splits on full pages**
 - GUID as high-order key (or any other random key)
 - Can even affect nonclustered indexes
 - Updates to variable-length columns
 - Badly configured FILLFACTOR (more in a few slides)
- **Wide schemas that only fit a few records per page**
 - E.g. a fixed-size 5000 byte row = 3000 bytes lost per page!
- **Real-world example:**
 - Social networking site that has a homepage comments table with the member ID as the high-order key
 - Patient check-in company using GUID as clustering key

Can DML Cause Fragmentation?

- **Yes, data modifications can lead to fragmentation**
- **INSERT**
 - YES – if key value is not ever increasing/decreasing (e.g. GUID)
 - NO – if key is ever increasing/decreasing (e.g. INT IDENTITY)
- **UPDATE**
 - YES – if updates make variable-length columns wider on full pages
 - NO – if columns are fixed width or columns have 'place holder' values (i.e. DEFAULT values) to minimize row expansion on update
- **DELETE**
 - YES – if deletes are singleton deletes (Swiss-cheese problem – page density issues)
 - NO – if deletes are range deletes for archival purposes

FILLFACTOR

- Makes the Storage Engine leave space on each leaf-level page to allow row inserts/expansions without causing page splits
- Specified at index creation or rebuild time
 - NOT maintained during regular DML
- Can specify with `sp_configure` for entire instance
 - Not recommended – specify it per index
- Use `PAD_INDEX` to affect the upper levels of the b-tree (uses the `FILLFACTOR` value)
 - Rarely used
- 0 = 100 = default value with special meaning of 'leave no space'
 - Excellent for data warehouse, but not ideal for OLTP
- For OLTP, which value to use?

Configuring FILLFACTOR

- Balancing act between how often page splits occur and how often you can rebuild/defrag the index
- What is going to cause page splits in your schema?
 - UPDATES to variable-width data types?
 - Random INSERTs?
 - The more volatile ⇒ lower FILLFACTOR
- How often can you rebuild/defrag?
 - The more frequent ⇒ higher FILLFACTOR
- Pick a value, try it, monitor fragmentation, tweak it
 - Use DMVs to see how fast the fragmentation increases
 - The faster fragmentation occurs ⇒ lower FILLFACTOR or decreased time between rebuilds/defrag
 - 70% is a common first guess

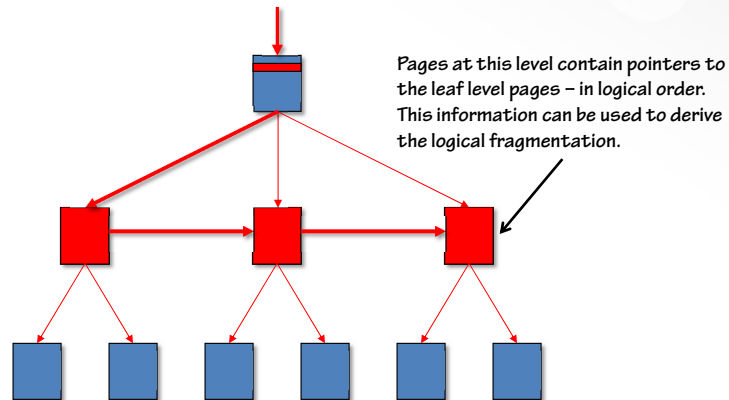
Symptoms of Fragmentation

- **Poor/degrading query performance over time**
 - Longer run-times
 - More disk activity
 - SET STATISTICS IO ON
 - More frequent checkpoints occurring
 - Increased logging (from page split activity)
 - Depending on the average record length and the split point, a page split could log up to 50 times more than a regular insert!
 - Increased buffer pool usage
- **Worsening results from the sys.dm_db_index_physical_stats DMV**
 - Keys to success are knowing which indexes to look at and how to interpret the results

sys.dm_db_index_physical_stats (1)

- **Replacement for DBCC SHOWCONTIG**
 - select * from sys.dm_db_index_physical_stats (dbid, objectid, indexid, partitionid, samplemode)
- **No longer need insert/exec to analyze/process DBCC SHOWCONTIG results**
 - DMVs are programmatically “composable”
 - However, this is a DMF, not a true DMV so must do work for results
- **Ability to control how much data is read using sample mode (LIMITED, SAMPLED, DETAILED)**
 - LIMITED (default) does not read the leaf level so is fastest mode
 - SAMPLED reads 1% of the leaf-level pages if the index/partition has more than 10000 pages
 - DETAILED reads everything and is the slowest mode
 - Beware of running against an entire database, could ‘flush’ the buffer pool

sys.dm_db_index_physical_stats (2) LIMITED Scanning Mode



sys.dm_db_index_physical_stats (3) Output and How to Interpret it

- **Logical fragmentation**
 - avg_fragmentation_in_percent (should be low)
- **Page density**
 - avg_page_space_used_in_percent
 - Should be high for data warehouse
 - Should have some free space for OLTP
- **Other counters exist (e.g. fragments, avg. fragment size) but these were only invented to be more accessible to users – somewhat unsuccessfully**

Demo

Detecting fragmentation using `sys.dm_db_index_physical_stats`

How to Correct Fragmentation?

- **2 realistic choices**
 - Rebuild the index: `ALTER INDEX ... REBUILD`
 - Defrag the index: `ALTER INDEX ... REORGANIZE`
- **Also**
 - `CREATE INDEX ... WITH (DROP_EXISTING = ON)`
 - Commonly used to move or (re)partition an index
- **Can also choose not to remove fragmentation**
 - If the index isn't used for range scans, and page density isn't an issue, why spend the resources?
- **Don't necessarily just rebuild everything every day**
 - Although for an involuntary DBA with enough resources, this is can be better than nothing

When To Rebuild vs. Defrag

- Much debate on this, basically it depends!
- I had to come up with numbers for Books Online so I chose:
 - < 10% do nothing
 - 10% <> 30% defrag/reorganize
 - 30%+ rebuild
 - And don't do anything if the index has < 1-5000 pages
- Your mileage may (and will) vary

Paul's Method...

- Create a table with one row per index you want to work on
 - I call it the 'driver table'
- Call the DMV for the indexes listed in the driver table
- Use per-index fragmentation thresholds to determine whether to rebuild, reorganize, or do nothing
- Log what you decide to do for future reference
- Optional: keep a counter of how many times in succession an index is rebuilt and programmatically reduce fillfactor
- Much easier: use code someone's already written...

ALTER INDEX ... REBUILD

- Replaces DBCC DBREINDEX in SQL Server 2005 onwards
- **Pros**
 - Atomic operation
 - No knowledge of index schema or constraints required
 - Can use multiple CPUs, and control MAXDOP
 - Rebuilds index statistics (with equivalent of full scan)
 - Can rebuild a single partition or all partitions
 - Can be done online (except for single partition and indexes with LOB columns)
 - Rebuilding using the same sort order does not require a sort
 - Can be minimally-logged (but log backup will be the same size)
- **Cons**
 - Atomic operation – potentially long rollback on interrupt, all or nothing semantics
 - Requires creating complete new index before dropping old one
 - When offline – S table lock for nonclustered index rebuild, X table lock for clustered index rebuild

ALTER INDEX ... REORGANIZE

- Replaces DBCC INDEXDEFRAG in SQL Server 2005 onwards
- **Pros**
 - ALWAYS online – only requires table IX lock
 - Interruptible with no loss of work – stops instantly
 - Has progress reporting
 - 2000 – output every 30 seconds to the processing connection
 - 2005 – sys.dm_exec_requests (percent_complete)
 - Optionally compacts LOB storage
 - Usually faster for a lightly fragmented index
 - Can reorganize one or all partitions
 - Does not require any extra disk space
- **Cons**
 - Usually slower for a heavily fragmented index
 - Always fully-logged, single CPU only, does not update statistics
 - Does not do as good a job as removing fragmentation when there is plenty of contiguous free space

CREATE ... WITH DROP_EXISTING

- **Pros**

- ❑ Same as ALTER INDEX ... REBUILD
- ❑ Can move the index to a new location
- ❑ Can rebuild the index with a new partitioning scheme
- ❑ Can change the index schema (keys, sort order, etc)
- ❑ Can do all of this online (with same limitations as regular index rebuild)

- **Cons**

- ❑ Same as ALTER INDEX ... REBUILD
- ❑ Need to know the index schema

Comparison Points

- Space required
- Log generated
- Algorithm speed on amount of fragmentation
- Locks required (i.e. online or not)
- Interruptible or not
- Progress reporting or not

Setting FILLFACTOR

- **Can be set when creating or rebuilding an index**
 - Stores the fillfactor in the index metadata
- **Cannot be set directly with REORGANIZE**
- **REBUILD and REORGANIZE use the metadata-stored fillfactor, if there is one, otherwise they will use the instance-wide fillfactor**
 - Unless a fillfactor is specified on the REBUILD
 - I.e. REBUILD specified fillfactor overrides metadata-stored fillfactor, which overrides instance-wide fillfactor
- **Fillfactor is only set in index metadata by a CREATE or REBUILD**

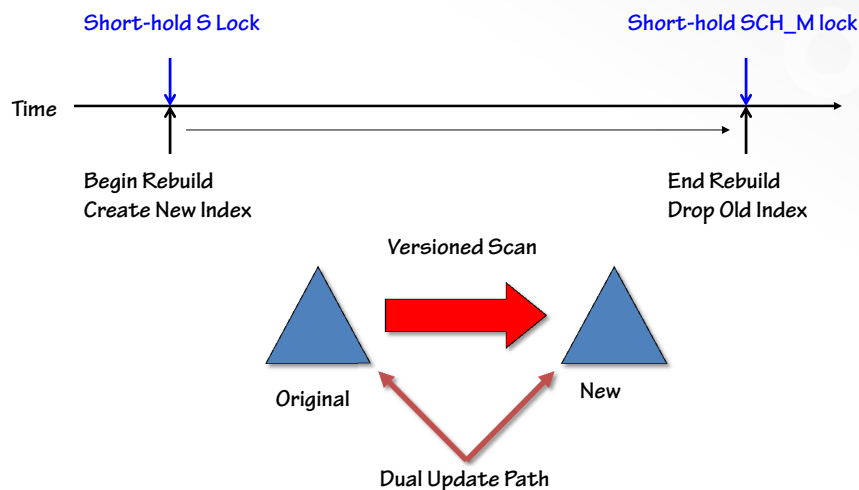
Index Rebuild Misconceptions

- **Rebuilding a clustered index rebuilds all nonclustered indexes**
 - No, only in 2000 and before for non-unique clustered indexes
 - There was a bug in 2000 SP1 that *did* rebuild always
- **Index rebuild pre-allocates the necessary space**
 - No, it allocates space as it goes so may run out
- **Rebuilding in BULK_LOGGED reduces the size of the log backups**
 - No, only the size of the transaction log. Remember how log backups work after a minimally-logged operation?
- **Index rebuilds in a multi-file filegroup only use a single file**
 - No, the allocations will follow regular round-robin proportional fill
- **Online index build doesn't take any locks at all**
 - No, as we'll see in a couple of slides

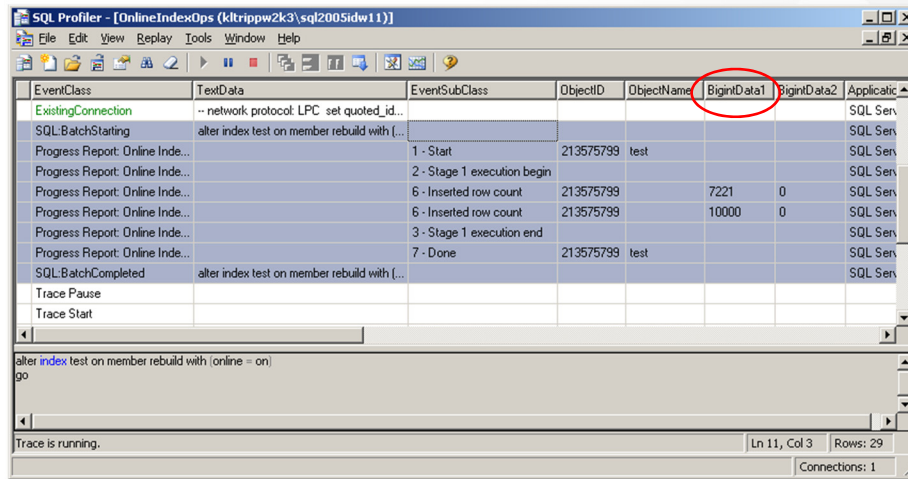
Online Index Operations

- **BEWARE: now a fully logged operation in 2008 onwards!!**
 - See <http://support.microsoft.com/kb/2407439>
- **'Online' is a bit of a misnomer**
 - Concurrent activity is impacted for (hopefully) very brief period at start and completion of rebuild
- **Performance may be impacted during operation as DML is directed to two indexes and versioning is used**
- **Indexes with LOB columns cannot have online operations until SQL Server 2012**
 - E.g. a clustered index on a table with a LOB column
- **Only one online index operation per table at a time**
 - Can online create, drop, rebuild, (re)partition an index
- **Fragmentation might increase!**
 - CSS blog post: <http://bit.ly/POCiPT>

Inside Online Index Operations



Online Progress Report



EventClass	TextData	EventSubClass	ObjectID	ObjectName	BigintData1	BigintData2	Applicatio
ExistingConnection	-- network protocol: LPC set quoted_id...						SQL Ser
SQL:BatchStarting	alter index test on member rebuild with [...]						SQL Ser
Progress Report: Online Inde...		1 - Start	213575799	test			SQL Ser
Progress Report: Online Inde...		2 - Stage 1 execution begin					SQL Ser
Progress Report: Online Inde...		6 - Inserted row count	213575799		7221	0	SQL Ser
Progress Report: Online Inde...		6 - Inserted row count	213575799		10000	0	SQL Ser
Progress Report: Online Inde...		3 - Stage 1 execution end					SQL Ser
Progress Report: Online Inde...		7 - Done	213575799	test			SQL Ser
SQL:BatchCompleted	alter index test on member rebuild with [...]						SQL Ser
Trace Pause							
Trace Start							

alter index test on member rebuild with (online = on)
go

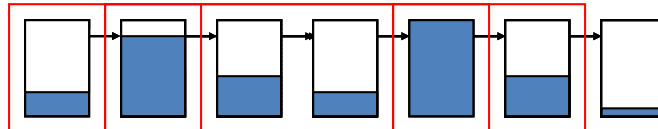
Trace is running. Ln 11, Col 3 Rows: 29 Connections: 1

Inside REORGANIZE (1)

- **Stage 1: leaf-level compaction**
 - Make leaf-level pages have free space near to original FILLFACTOR if they have too much
 - Compacts pages by shuffling rows towards right-to-left
 - Safeguard to prevent cascading row move
 - Deletes ghost rows and deallocates empty pages
- **Stage 2: leaf-level defragment**
 - Makes logical order of leaf-level same as physical order
 - Uses logical order scan + allocation order scan in lockstep
 - Drops and reestablishes scan position after every page so totally online apart from page X locks
 - Reorders pages currently allocated to the index

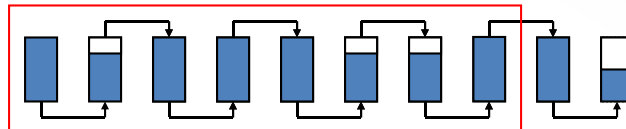
Inside REORGANIZE (2)

- Page compaction example (2000)



Inside REORGANIZE (3)

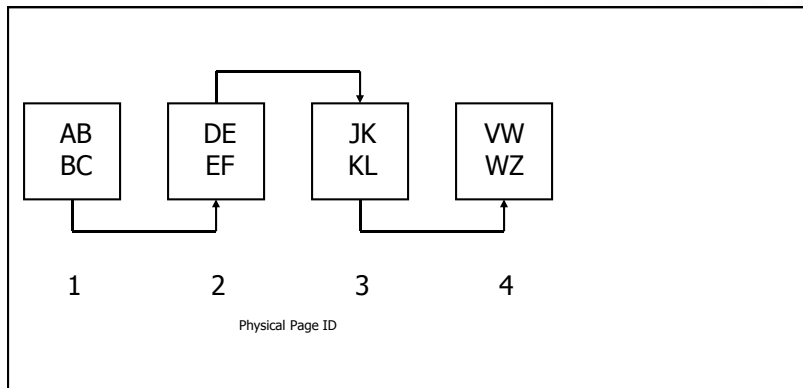
- New 'sliding window' compaction algorithm in 2005
- Found certain applications created pathological cases more frequently than expected (e.g. SAP)



- This new algorithm will only perform compaction if there is enough space in an 8-page window to remove one page

Inside REORGANIZE (4)

- Page reordering example

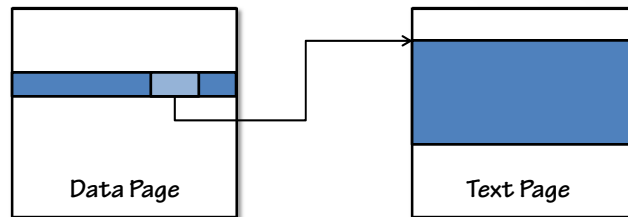


Demo

Removing fragmentation

LOB/Row-Overflow Data (revisited) (E.g. varchar (max), text, XML)

- Data/index record stores a pointer to the actual LOB value on a different page



- You remove fragmentation but scan performance is still poor...
- Accessing the LOB data or row-overflow data requires (potentially) another I/O – usually random!
 - Can be a cause of poor performance even if the index is free of fragmentation

Are Your Indexes Being Used?

- There are lots of bad practices around index strategy, including creating extra indexes
 - E.g. an index for each column in the table
- Extra, unused indexes waste resources as they must be maintained by DML operations
- Use the sys.dm_db_index_usage_stats DMV to tell if an index is being used at all during the business cycle
 - Beware of indexes not being used but enforcing unique constraints

Index Maintenance Survey

What regular index maintenance do you do?

Rebuild all indexes (using DBCC DBREINDEX or ALTER INDEX REBUILD)		29%	25
Defrag all indexes (using DBCC INDEXDEFRAG or ALTER INDEX REORGANIZE)		3%	3
Rebuild based on fragmentation		15%	13
Defrag based on fragmentation		2%	2
Mixture of rebuild and defrag, based on fragmentation		30%	26
Complex maintenance based on a list of indexes, fragmentation, or other factors		9%	8
Any answer above, followed by a database shrink		3%	3
Nothing		8%	7
Other? Enter here...		0%	0
Total: 87 responses			

Whitepapers on This and More...

- **Microsoft SQL Server 2000 Index Defragmentation Best Practices**
 - <http://www.microsoft.com/technet/prodtechnol/sql/2000/maintain/ss2kidb.p.msp>
 - Based on SQL Server 2000, so discusses DBREINDEX vs. INDEXDEFRAG
 - Concepts translate to 2005+, and will be rewritten/updated this year
- **Online Indexing Operations in SQL Server 2005**
 - <http://www.microsoft.com/technet/prodtechnol/sql/2005/onlineindex.msp>
- **Sample scripts to automate index maintenance**
 - <http://ola.hallengren.com/>
 - <http://weblogs.sqlteam.com/tarad/archive/2009/11/03/DefragmentingRebuilding-Indexes-in-SQL-Server-2005-and-2008Again.aspx>
(<http://bit.ly/72ATFB>)
 - <http://sqlfool.com/2010/04/index-defrag-script-v4-0/>

Review

- Data access methods
- What is fragmentation?
- How does fragmentation happen?
- Detecting, mitigating, removing fragmentation

Questions!