SQLskills Immersion Event

IEPTO2: Performance Tuning and Optimization

Module 9: Index Analysis

Kimberly L. Tripp

Kimberly@SQLskills.com

Overview

- **Nonclustered indexes: key to better performance**
- **Indexing for performance:**
  - During design
  - During QA
  - In production
- **Key production strategy – an ordered approach:**
  1. Index cleanup
  2. Index health
  3. Missing indexes

Nonclustered Indexes: Key to Better Performance

- **In a row-based indexing strategy performance hinges on your choice of nonclustered indexes:**
  - Indexing strategies are extremely challenging
    - Users lie ☺
    - Workload specific
      - Data modifications are impacted by indexes (indexes add overhead to INSERTs/UPDATEs/DELETEs)
      - The type and frequency of the queries needs to be considered
        - This can change over time
        - This can change over the course of the business cycle
    - Need to have an understanding of how SQL Server works in order to create the "RIGHT" indexes, you CANNOT just guess!
  - **To do a good job at tuning you must:**
    - Know your data
    - Know your workload
    - Know how SQL Server works!

Indexing for Performance: At Design

- First and foremost: choose a GOOD clustering key
- Create your primary keys and unique keys
- Create your foreign keys
  - Manually index your foreign keys with nonclustered indexes
- Create any nonclustered indexes needed to help with highly selective queries (lookups are OK for highly selective queries)
- STOP: this is your "design" base
- Add indexes slowly and iteratively over time while learning and understanding your workload as well as query priorities and always re-evaluate if/when things change!

Indexing for Performance: At QA

- **While testing primary workload characteristics monitor query performance:**
  - By duration and IO (at a minimum)
  - Review the cumulative costs of frequently executed queries
    - sys.dm_exec_query_stats (also review query_hash)
    - sys.dm_exec_procedure_stats (this is cumulative)
- **Identify key performance problems**
- **Consider wider indexes through testing / analysis**
  - Be sure to evaluate the impact to OLTP as wider indexes are added
  - Be sure to evaluate the disk / memory costs for wider indexes

Indexing for Performance: In Production

1. **Make sure you don't have any "dead weight"**
   - Remove duplicate indexes
   - Consider the removal or consolidation of redundant indexes
2. **Make sure you have a good maintenance strategy**
   - How are you analyzing for fragmentation?
     - Use a limited scan (no need for "sampled" or "detailed")
   - Are you dealing with statistics appropriately?
     - Rebuilding indexes updates the statistics of the rebuilt index with the equivalent of a full scan but does not update other statistics
     - Reorganzing indexes does NOT update statistics at all
     - Do you have any other statistics (column-level statistics?)
3. **Only after steps 1 and 2 are done can you add indexes slowly and iteratively over time while learning and understanding your ACTUAL workload as well as query priorities. And, always re-evaluate if / when things change!**

(1) Remove Unused Indexes

DMVs Don't Tell You Everything…

- **Removing redundant/duplicate indexes: *SQL Server allows you to create as many useless indexes as you like…***
- **Duplicate indexes can still show as used**
- ***MUST* review your indexes manually**
  - Don't forget INCLUDEd columns (in 2005) or filtered indexes (in 2008)
  - sp_helpindex doesn't show these columns
  - Use my updated version of sp_helpindex (blog category: sp_helpindex rewrites) to get better information and determine if one index really is redundant/duplicate
  - Kimberly's blog post: Removing duplicate indexes (http://bit.ly/rusI9U)
- **Don't rely on sys.dm_db_index_usage_stats alone**

(1) Remove Unused Indexes

DMVs Don't Tell You Everything…

- **In addition to completely duplicate indexes, must prune out the redundant indexes…**
- **Pruning existing indexes is not quite as simple as removing all left-based subsets:**
  - It's true that a query using an index on LastName alone COULD use an index on LastName, Firstname OR an index on LastName, Firstname, MiddleInitial but, always be careful of how much larger the indexes are and the type of queries using them
  - Should you drop indexes that are left-based subsets of others?
    - Typically yes BUT, consider the width of the additional columns
    - If the additional column(s) are relatively wide and only needed for a couple of queries whereas the narrower version is used by a lot of queries, you might want both!

(1) Remove Unused Indexes
What Do the DMVs Tell You?

- **Verify index usage with sys.dm_db_index_usage_stats**
- **Tracks the following and the date/time of their last occurrence:**
  - Seeks (a singleton lookup or range scan)
  - Scans (something like a 'select *')
  - Lookups (a bookmark lookup)
  - Updates (an insert, update, or delete)
- **The cache is flushed at shutdown as well as when**
  - Entries for all indexes in a database are removed when the database is closed (via AUTOCLOSE (if enabled)), taken offline, or detached
  - There's no way to manually flush the cache (but offline/online works…)
  - An object's index usage stats are cleared when the object is rebuilt UNTIL:
    - SQL Server 2012 SP3 + CU3 / SQL Server 2014 SP2 / SQL Server 2016
  - Persist this data and analyze over business cycle

(1) Dropping an Index
What Could Go Wrong?

- **Queries that use index hints will ERROR if the index no longer exists**
  - This is the reason for why SQL Server allows duplicate indexes to be created in general…
- **Plans guide might no longer work**
  - They're just invalidated, a new plan will be used
- **Plans could change**
  - This could be good… or bad?!

(2) Verify Health of Existing (and Useful) Indexes…

- **Fragmentation can mean a lot of things (completely overloaded term) and not an entirely simple thing to address…why?**
- **Yes, solid state can remove the excess cost of random I/O but fragmentation isn't JUST about I/O**
  - Table size and usage patterns
    - The costs of splits on logging, cache, and performance!
  - Impact to availability – can you use an online operation?
    - ALTER INDEX…REBUILD…WITH (ONLINE = ON)
      - Not if the index has a new LOB column in it (fixed in 2012)
        - NOT if the table still has a LEGACY LOB type (text / ntext / image) online operations are not allowed
      - Not if you try to rebuild only a single partition (fixed in 2014)
    - ALTER INDEX…REORGANIZE (always online)
  - Reorganizing always uses 'full' logging but the amount of log information generated will depend on how much fragmentation exists
  - There are trade-offs between rebuilding and reorganizing in terms of log space, disk space, run-time, impact to tempdb and even benefit…
  - Check out Paul's Pluralsight course: SQL Server: Index Fragmentation Internals, Analysis, and Solutions

(2) Verify Health of Existing (and Useful) Indexes…

- **Verify structural details (levels/fragments/density) from sys.dm_db_index_physical_stats**
  - Can run in one of three modes:
    - LIMITED (default): the logical (left to right) order of the pages (as defined by the index keys) is not the same as the physical order in which the pages are allocated
    - SAMPLED: run in detailed mode for tables < 10000 pages but for tables with >= 10000 pages, "samples" every 100th page to get a picture of the index fragmentation (faster but less accurate for some stats)
    - DETAILED: slowest, but shows ALL structural and fragmentation details
  - Tracks the following:
    - avg_fragmentation_in_percent: overall health of the index…this number should be very low and it should not change rapidly
    - avg_page_space_used_in_percent: overall health of the pages…this should be relatively high but might have some freespace (fillfactor) that's specifically been added in an attempt to reduce overall fragmentation
- **Check out our blog post categories on Indexes:**
  Kimberly: https://www.sqlskills.com/blogs/Kimberly/category/Indexes.aspx
  Paul: https://www.sqlskills.com/blogs/paul/category/Indexes-From-Every-Angle.aspx

hidden slide
w/extra details

(2) Verify Health of Existing (and Useful) Indexes…

- **Verify operational details (stats / waits / overflow) from sys.dm_db_index_operational_stats**
  - Tracks the following:
    - Leaf_X_count (essentially rows inserted, deleted or updated – in the leaf level of the index)
    - nonleaf_X_count (this indicates that pages were added/removed in the b-tree…this might indicate heavy fragmentation but you MUST review the index physical stats to be sure)
    - Significant overflow activity could indicate poor IO patterns
    - row_lock_wait_in_ms and page_lock_wait_in_ms show the TOTAL amount of blocking on these structures – this could indicate poor index choices and/or a need to change reader/writer isolation options
    - page_io_latch_wait_count and page_io_latch_wait_in_ms show the total physical I/Os that were necessary to access the data
- **The cache is flushed when the object falls out of metadata cache… this cannot be predicted (per se) but "active" objects will be available in this DMV when queried**
- **Consider using a custom data collection set in Data Collection to persist snapshots of this (and possibly other) DMVs for better trend analysis**

(3) Are You Missing Any Indexes?

- **Have you tried other options?**
- **Ask DTA what it thinks?**
- **Ask SQL Server what it thinks?**
  - SQL Server 2005
    - DMVStats
    - Performance Dashboard Reports (SP2)
    - RML Utilities from PSS
  - Both SQL Server 2005+
    - DMV queries
    - Other resources/blogs/sites…
  - SQL Server 2008+
    - [Performance] Data Collector
  - THIRD-PARTY TOOLS!

(3) Are You Missing Any Indexes?
Ask SQL Server What it Thinks…

- **sys.dm_db_missing_index_group_stats (probably the most detailed):**
  - user_seeks, user_scans
  - last_user_seek and last_user_scan are both datetime
  - avg_total_user_cost: higher costs give relative numbers to determine which are more "costly" to the system
  - avg_user_impact: the improvement (in terms of percent that the user should see from the index addition)
- **sys.dm_db_missing_index_groups**
  - Many to many relationship table to tie together the index details and the usage stats (index group stats is effectively the index usage stats for these needed indexes)
- **sys.dm_db_missing_index_details**
  - Details the table, key columns and included columns that you should consider for these indexes…
- ***NOTE: The important part isn't the query, it's what you do with the results (evaluation, testing, consolidation)***

(3) Asking the Tools

- **USE the tools!**
  - STATISTICS IO
  - Showplan/missing index DMVs
  - Database [Engine] Tuning Advisor
- **BEWARE of the limitations of the tools!**
  - Missing index DMVs (and therefore showplan) only tune the plan that was executed; they do not "hypothesize" about alternatives (like DTA does)
  - All of the index recommendation from tools tend to go for "the best" choice rather than good enough choices
  - NONE of the tools do index consolidation…
- **Resources:**
  - Search "Bart Duncan Missing"
  - Glenn's DMV toolkit
  - A bit of searching as lots of good stuff out there

Summary: Indexing for Performance

- **Extremely challenging**
  - Users lie ☺
  - Workload specific
    - Data modifications are impacted by indexes (indexes add overhead to INSERTs/UPDATEs/DELETEs)
    - The type and frequency of the queries needs to be considered
      - This can change over time
      - This can change over the course of the business cycle
  - Need to have an understanding of how SQL Server works in order to create the "RIGHT" indexes, you CANNOT just guess!
- **To do a good job at tuning you must:**
  - Know your data
  - Know your workload
  - Know how SQL Server works!

Key Takeaways

- **Actually go back and DO this… so many people who have attended IEPTO1 actually LOVE this module as a reminder to do all of the things that they haven't had time to do from IEPTO1**
- **At a minimum**
  - Check for duplicates – this is easy to do and will help you save time in data modifications and maintenance, space in the database as well as in backups
  - Have a good maintenance plan in place
- **Big step 1**
  - Find your top 10 tables (in size) for your top 10 databases
    - Consolidate indexes and reduce [disable, then later drop] some of "similar" indexes
    - Analyze query and procedure performance – find those that have the highest cumulative cost (looking for query classes [query_hash] and procedure "total time / total CPU")
  - As time allows, repeat this for the next set of large tables and possibly even smaller ones (just to clean things up)

Review

- **Nonclustered indexes: key to better performance**
- **Indexing for performance:**
  - During design
  - During QA
  - In production
- **Key production strategy – an ordered approach:**
  1. Index cleanup
  2. Index health
  3. Missing indexes