REINDEX PROBLEMS AND SOLUTIONS

From the precon "The Lifecycle Approach: Maintenance Problems & Solutions", by Sean McCown and Jen McCown. Please do not publish this document.

CHANGE THRESHOLDS FOR INDIVIDUAL DB

We have one or two databases that should have different thresholds for reindex and reorganize.

Solution: Your system should provide for thresholds at the database level using **table based parameters** ("uncoding"), instead of hardcoding or soft-coding (using SP parameters in jobs).

Also note:

- The table based parameter system we recommend handles new and "retired" databases seamlessly through the use of default values. Having default values (in MR's case, the "MinionDefault" row) means the system must have rules for settings hierarchy: default values apply for any database that doesn't have DB-specific configurations defined.

ORDER OPERATIONS

We have some VERY IMPORTANT databases (or very important tables) that need to be maintained first!

Why? Well, you know how some shops say, "We have a tight maintenance window! We have a hard stop at 2:00am!" The shop doesn't really have a hard stop at 2am; usually, it's only a set of core tables that need to be done by that time, and the rest of the tables can still be processed because they're not used much and they're much smaller.

Solution: Provide for operation ordering, so that the important tables are maintained *first*. MR doubles down on the concept or ordering by providing a way to define groups of databases, order those operations, and define the ordering within groups, too. (The same thing applies to ordering tables: groups of tables, then ordering within those groups.)

Also note:

- A flexible enough system allows for some really creative problem solving. With MR, for example, you can create an SP to dynamically order the operations in any way you like. The SP can run before each batch of that type, if you put the SP call in the DBPreCode column for that database (in Minion.IndexSettingsDB).

EXCLUSIONS

We have just a couple of databases (or a handful of tables) that should NEVER have index maintenance. Examples:

- Temporary or duplicate databases that are going to be destroyed anyway.
- Archived databases.
- Read only databases.

Solution: Your system should create operation exclusions using **table based parameters** ("uncoding"), instead of hardcoding or soft-coding (using SP parameters in jobs).

Also note:

- Because everything is table based, we can also *dynamically* exclude databases or tables from index maintenance, based on any criteria we like.
- Your system should automatically *not* generate index maintenance operations for read only and archive databases.
- The table based parameter system we recommend handles new and "retired" databases seamlessly through the use of default values. Having default values (in MR's case, the "MinionDefault" row) means the system must have rules for settings hierarchy: default values apply for any database that doesn't have DB-specific configurations defined.

MAXIMIZE YOUR MAINTENANCE WINDOW

Maybe you really do have a tight maintenance window, and you're having trouble getting index maintenance done in time. So, you need to maximize the maintenance window in such a way that the *entire* window is used for the reindexing instead, of using part of it for gathering fragmentation statistics.

Solution: Gather fragmentation statistics during the day, outside of your maintenance window, and save them to a table. Your routine should then use those saved statistics to perform maintenance during the maintenance window. Gathering fragmentation stats can take a very long time, so using "PrepOnly" can help a lot.

RECOVERY MODEL

A fully logged reindex can take a very long time.

Solution: For specific databases, automatically switch the recovery model to bulk-logged before index maintenance, and switch it back to full recovery afterward. The rest of the databases' recovery models won't change.

LOG FRAGMENTATION STATISTICS

We need to see how much each table is fragmenting throughout the day (in fact, we might even want more detailed information on a subset of tables, to see *when* during the day they become most fragmented).

Solution: Gather your fragmentation stats regularly through the day and save those to a table.

Also note:

- This can really help you determine what your fillfactor should be and whether you need to adjust your run frequency.

ALLOW PAGE LOCKS

Your vendor requires you to keep page locks turned off (usually, to reduce blocking and/or deadlocking). However, you can't run index reorganization with page locks off, because they require page-level locking.

Solution: Your system must provide a way to toggle the page locks: on for index operations, then off again.

Also note:

- This should be configurable per database.
- Automate as much as possible; there is *no reason* to manually turn pages locks on and off in a situation like this!

CUSTOM THRESHOLDS

It's not uncommon to need custom thresholds for specific tables.

Solution: Do that. As table based parameters. You do NOT want to manage custom thresholds for dozens of tables, in job form!

CUSTOM SCANMODE

From time to time, a default scan won't detect fragmentation because it doesn't read all levels of the index.

Solution: In these cases, you need to use a detailed scan. For this, MR uses the max fragmentation from all the index levels to determine the fragmentation level to be used.

DYNAMIC OPERATIONS

Flexible and dynamic settings are essential to fine-tune index maintenance operations.

Solution: A table-based system with enough flexibility built in can, for example, provide for:

- Delete on Rowcount
- Dynamic Delete on Table pagecount
- Dynamic Delete on Index pagecount
- Dynamic Order by Index Usage
- Dynamic Delete by Schema
- Dynamic SortInTempDB
- Dynamic Order by Schema
- Dynamic Drop_Existing

REFERENCES

- Precon site: minionware.net/dbmaintprecon/
- Download free backup and maintenance:
    - minionware.net/backup
    - minionware.net/reindex
    - minionware.net/checkdb (soon)
- Sean: www.Twitter.com/KenpoDBA
- Jen: www.Twitter.com/MidnightDBA