A Self-Tuning Fill Factor Technique for SQL Server – Part 1

By Mike Byrd

ByrdNest Consulting

# Introduction

So what’s all the fuss about Fill Factor? It is a SQL Server parameter I’ve ignored for 20+ years. The main reason was/is I’ve had no idea about what value to use. There is just no documented definitive guidance.

Back in April at SQL Saturday 830 (Colorado Springs) I attended two sessions by Jeff Moden titled “Black Arts” Index Maintenance: How the “Best Practice” Methods are Silently Killing Performance. These were outstanding and are downloadable from the SQLSaturday web site (<https://www.sqlsaturday.com/830/Sessions/Schedule.aspx>). A Reader’s Digest version of his presentations is contrary to the popular concept to reorganize at 10% fragmentation and rebuild indexes at 30%. He showed (in the 2 presentations) that page splits caused tremendous performance degradations and that you should probably rebuild indexes at 1% fragmentation. I don’t want to steal his “thunder” (and it is forthcoming in another series by him).

But on the flight back home I got to thinking about everything he said and how fill factor also “factors” into index performance and maintenance. It got me to thinking about a brute force optimization technique I implemented way back when I worked at the Air Force Rocket Propulsion Laboratory (yes, I was once a rocket scientist and still a geek!). It was a technique that involved tweaking various parameters (plus and minus percentages) and reducing the tweaks to an “optimum” solution. It was not always perfect as in a multi-dimensional scenario there are valleys that this solution could drill down on, but it did offer “near” optimum solutions that let us explore options.

Frankly speaking, what happened is somewhat amazing. Using the “still experimental” methods that I explain in this article on a client’s system, the end result is that the overall database wait times showed improvement (decreases) of about 30%. That’s not a trivial improvement, and I hope this article will serve as an impetus for others to try these and other experiments on their indexes to come up with other improvements they might wish to share with the community at large.

Part 1 of this series will include an Overview and Code Details. Part 2 will cover an analysis of the code and its results over a 3 month period as well as performance details and sections on performance, current limitations, and anticipated future improvements.

# Overview

So what was needed was to collect index parametrics and look for patterns. This involved using the sys.dm\_db\_index\_physical\_stats and sys.dm\_db\_index\_operational\_stats views and capturing these index parametrics before and after each index rebuild. Prior to this task, a modified version of Ola Hallengren’s defrag script only starting rebuilds at 10% and no reorganizes was my choice for index fragmentation. For this project, this script was modified to capture and store in a table the before and after parametrics for each index rebuild (heaps excepted) within a specified database. Initially I started looking at the top 15 average fragmentation indexes greater than 1.0%, but eventually changed it to 1.2% because of logical fragmentation issues. In the meantime, if a fill factor has not established, the previous fill factor (for that index) is decremented and the resulting average fragmentation compared to the previous average fragmentation. (All indexes prior to this task had 100% fill factors.) This process is repeated every 24 hours (SQL Agent job) until the index’s new average fragmentation is greater than the previous one. The previous fill factor is notated and fixed for that specific index. A secondary look at the data over 90 days (since the fill factor was established) inserts an index for a once again review (data collection). While this may temporarily degrade that index’s performance it ensures that a once again “near optimum” fill factor can be established taking into account any new data skew and application utilization of the database.

# Details

This process (index rebuild and fill factor perturbation) was incorporated into a SQL Agent job that runs daily during a period of low server activity. It stores data from the SQL Agent job and inspects that data for possible fill factor adjustments. The following discussion describes the script in Appendix A (and also attached) in detail.



Figure : Line 4 from TSQL code

Line 4 is the specification of the database name. Currently the script is setup for only one database, however, coding within the working tables will allow the script to eventually be expanded to multiple databases within a server. If this line remains uncommented the script will use the current database.

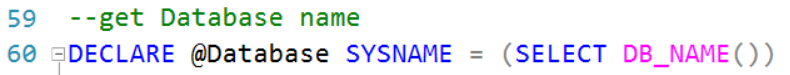


Figure : Lines 59-60

Line 60 gets the name of the current database.

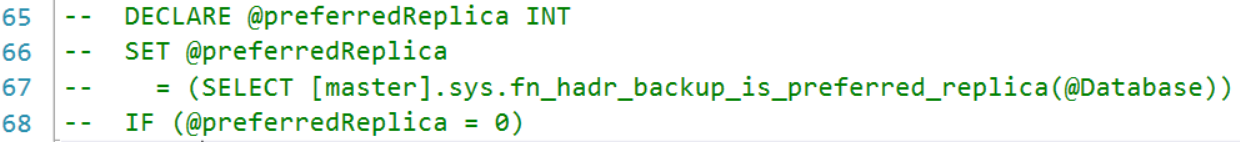


Figure : Lines 65-68

The commented code in lines 65-68 should be uncommented when used in an Always On environment.

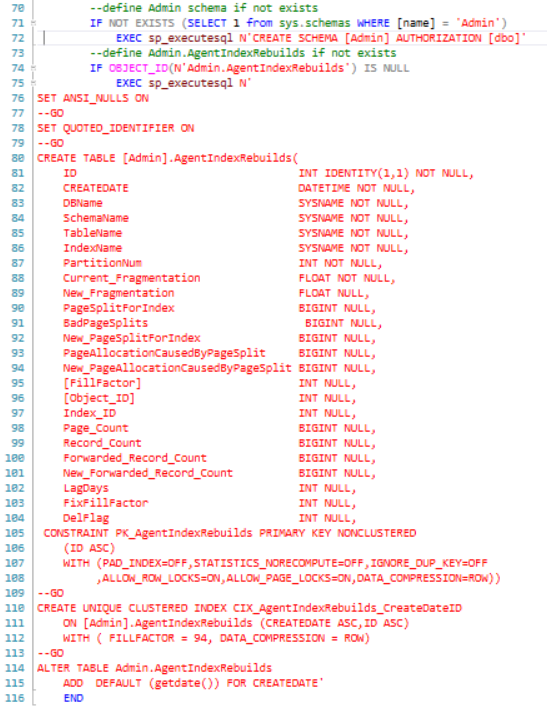


Figure : Lines 70-116

Lines 70-116 define the Admin schema and also the Admin.AgentIndexRebuilds table. This table has evolved during the study to this final version. Not all parameters are used to determine each index’s fill factor, but I thought (and still do) that too much data collection allows sufficient analysis rather than not enough. Note that the database name is retained so that this code can be extended to multiple DBs on the same server. Variables prefaced with “New” are the results after the particular index rebuild has completed (with or without a new fill factor). There will be more discussion on this table later in the Analysis section (Part 2).

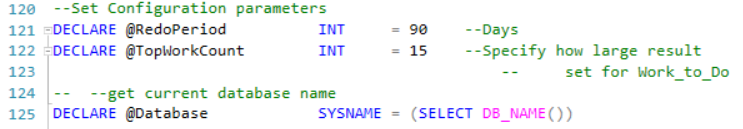


Figure : Lines 120-125

Lines 120-125 are the primary configuration parameters for this script. @RedoPeriod is the number of days to wait until an identified index is re-evaluated (more on this in the code discussion below). @TopWorkCount is a specified number for the TOP command to return the specified number of most fragmented rows. Initially I started with a value of 15, but within 3 weeks most of the time the result set returned less than 15 rows (this was primarily due to the improved fill factor and decreased split pages). @Database is the current database name.

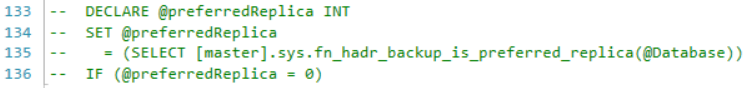


Figure : Lines 133-136

The commented code in lines 133-136 should be uncommented when used in an Always On environment.

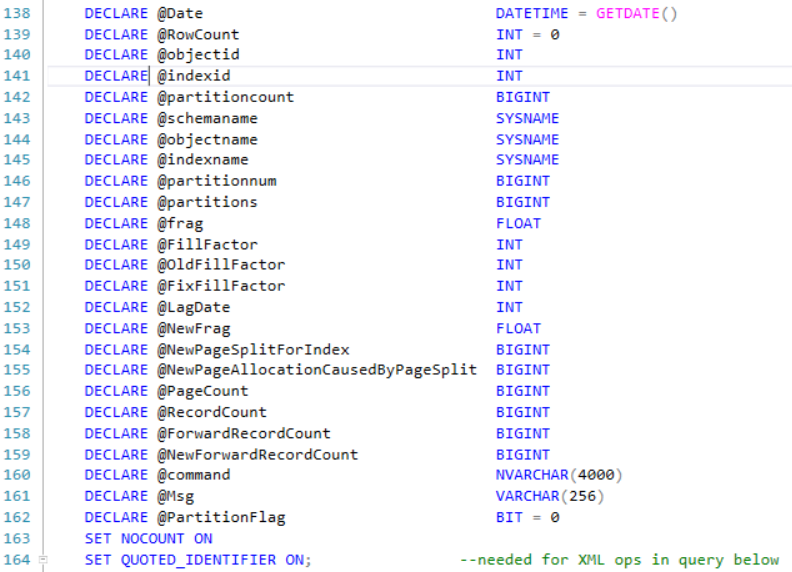


Figure : Lines 138-164

Lines 138-162 are variable declarations used in the subsequent code.



Figure : Lines 171-237

Lines 171-237 is the code used to select the TOP worse fragmented and split page indexes at the current time. The query calls sys.dm\_db\_index\_physical\_stats to get the current fragmentation and sys.dm\_db\_index\_operational\_stats to get page splits as well as sys.indexes and sys.objects to get table and index data. This data is put into a working table (#work\_to\_do) for subsequent utilization. This query is duel-functional; it always does an index rebuild (for the specified index) that does a de facto defrag, but also a fill factor perturbation if not already fixed. If you examine the ORDER BY clause it is also dual-functional – even days the TOP n rows are returned by maximum average fragmentation descending and on odd days the TOP n rows are returned by maximum page-splits per index descending.

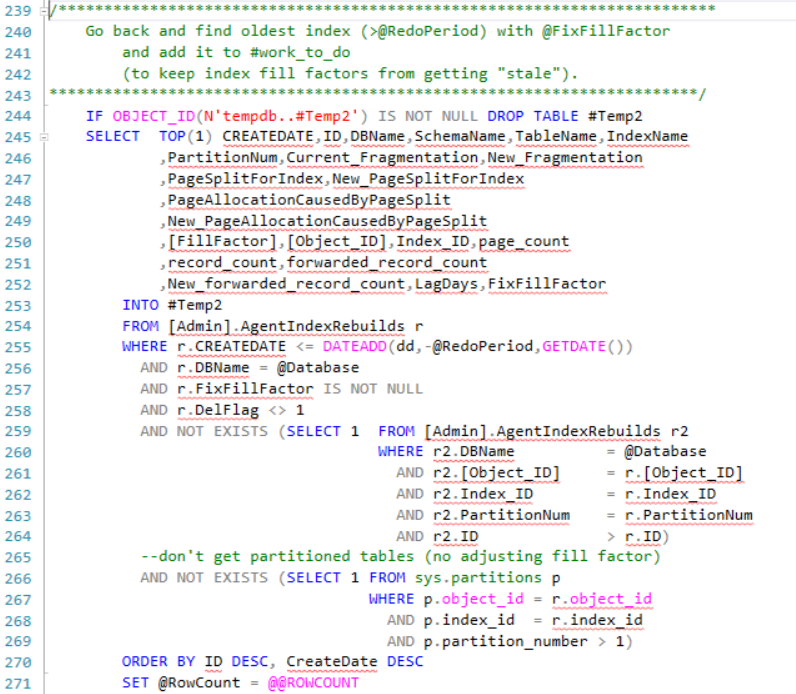


Figure : Lines 220 - 301

Lines 239-271 review the existing data to see if there is a candidate index for re-evaluation of its fill factor (where the fill factor is already set).

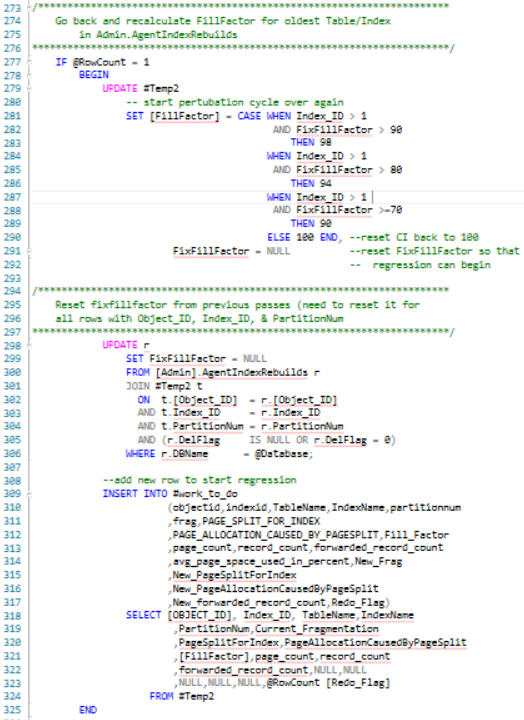


Figure : Lines 273-325

If there is, a row is added to #work\_to\_do for that index and the @Redo\_Flag is set to one. As configured, the @RedoPeriod is 90 days – this looks at any existing index in the Admin.AgentIndexRebuilds table to see if last adjustment > @RedoPeriod .

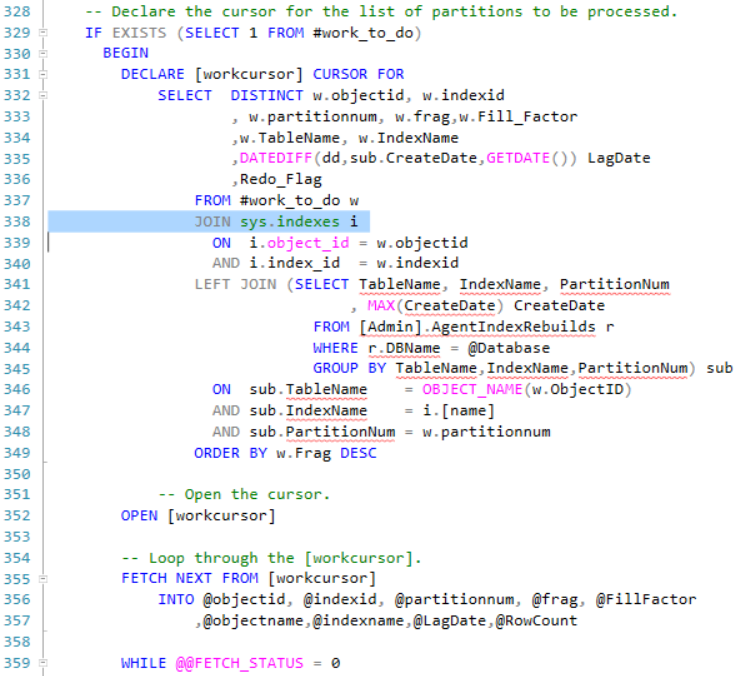


Figure : Lines 328-359

Lines 328-359 define the cursor used to rebuild each table/index in #work\_to\_do.

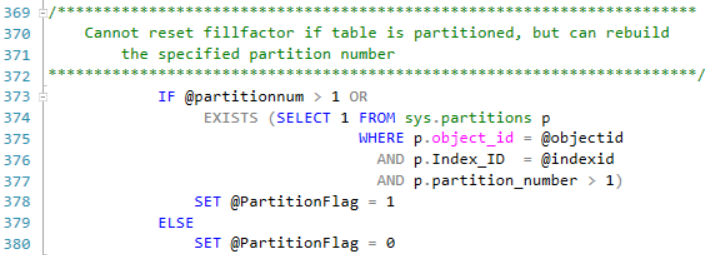


Figure : Lines 369-380

Lines 369-380 set up a flag to specify if an index is partitioned or not. Since you cannot change fill factor for a single partition within a partitioned index, the flag signals the code to only rebuild that particular index partition without a perturbation of the fill factor.

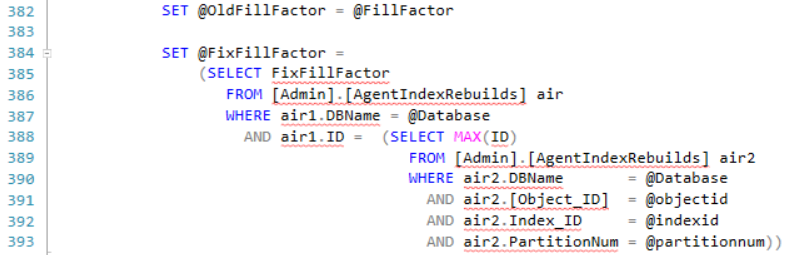


Figure : Lines 382-393

Lines 382-393 test to see if the fill factor is already fixed or not.

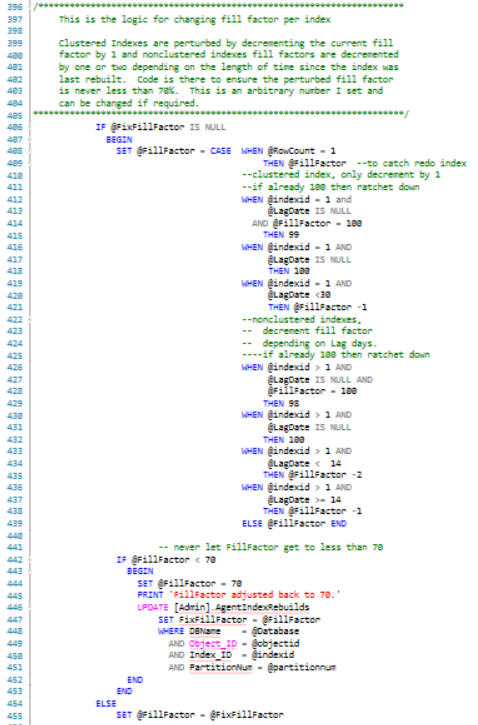


Figure : Lines 396-455

Lines 396-455 contain the logic for perturbing the fill factor. The code is very readable – Clustered Indexes are perturbed by decrementing the current fill factor by 1 and nonclustered indexes fill factors are decremented by one or two depending on the length of time since the index was last rebuilt. Code is there to ensure the perturbed fill factor is never less than 70%. This is an arbitrary number I set and can be changed if required.

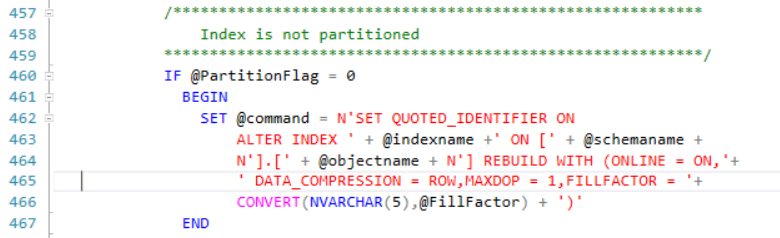


Figure : Lines 457-467

Lines 436-443 setup the dynamic SQL to rebuild the index with the specified fill factor if the table is not partitioned.

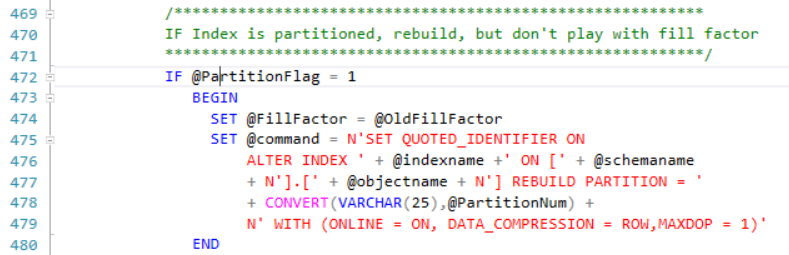


Figure : 469-480

Lines 469-480 set up the dynamic SQL to rebuild the specified index for partitioned indexes. The Fill Factor is not changed for partitioned indexes.



Figure : Lines 482-483

Lines 482-483 executes the generated dynamic SQL.

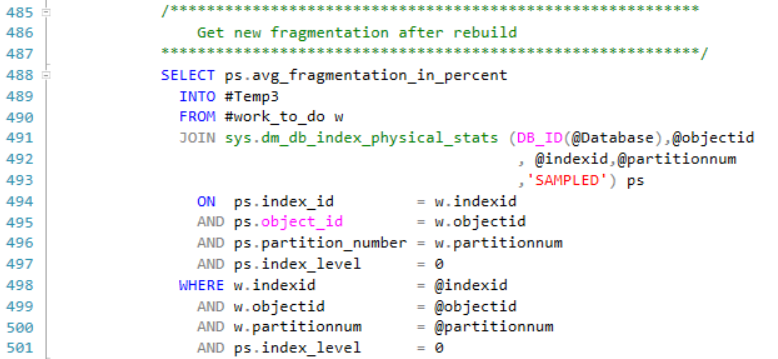


Figure : Lines 485-501

Lines 464-477 gets the new index fragmentation.

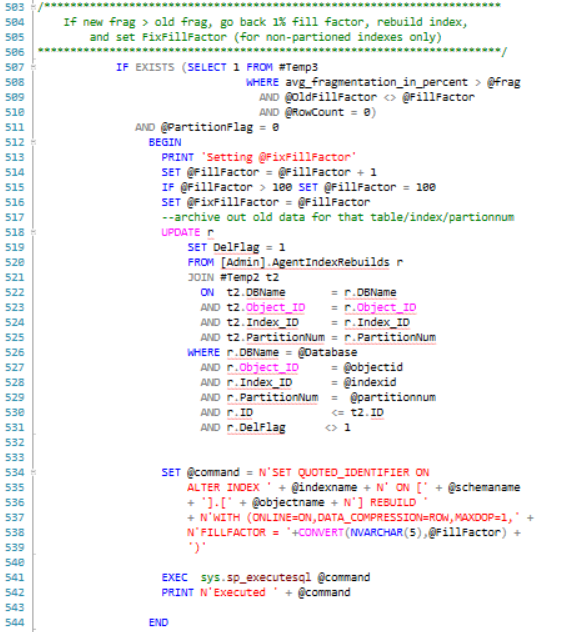


Figure : Lines 503-544

Lines 503-544 contain the logic to determine if the current fill factor causes more fragmentation. If so, the fill factor is backed up (decremented by 1), specified as fixed, and rebuild with the fixed fill factor.

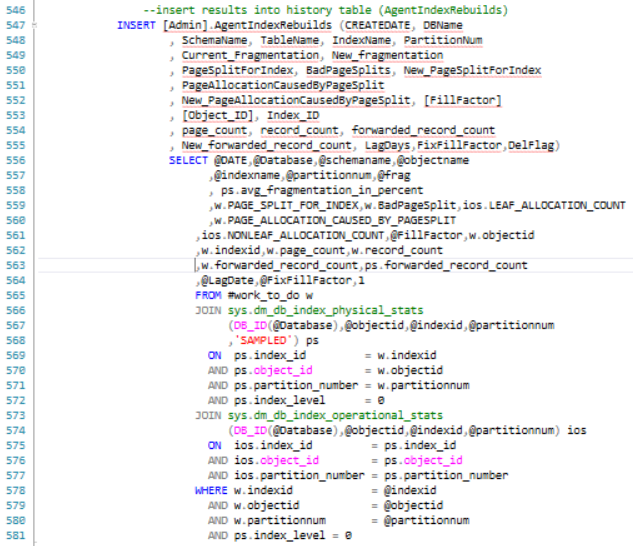


Figure : Lines 546-581

Lines 546-581 inserted the old and new index parametrics into the [Admin].AgentIndexRebuilds table for subsequent review by the next script run.

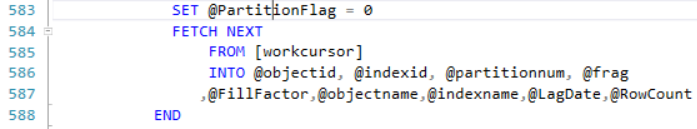


Figure : Lines 583-588

Lines 583-588 take care of some administrative book-keeping and fetch the data for the next cursor loop.

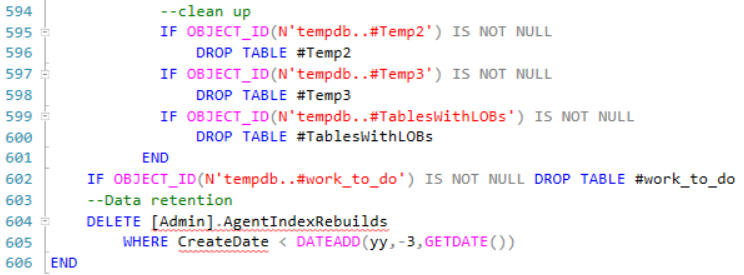


Figure : Lines 594-606

Lines 594-606 are just clean-up code.

# Requirements:

This code was developed and tested with SQL Server 2017 Enterprise Version. However, it is applicable to all versions of SQL Server from 2012 and upward. It can also be used for Standard Editions (same versions) except the WITH Statement in the Index Rebuilds needs to have the “ONLINE = ON, Data\_Compression = ROW,” statement removed (3 occurrences). Data\_Compression = ROW option can (and should be) retained in SS 2016, SP 2 and upward for Standard Edition. In the event you are using Standard Edition consider scheduling this task when there is minimal database activity as without the ONLINE option you will have schema locking during the index rebuilds. Otherwise, there are no other restrictions that I am aware of.

# Summary:

This whole project started as a proof of concept and as most POCs has made its way into production. While I agree that it’s not complete and isn’t perfect, it did result in a substantial decrease in wait time on a real system. It also demonstrates that effective automatic determination of Fill Factor **IS** possible.

I look forward to constructive critique and suggestions for improvements from the SQL Server Community.

Cheers,

Mike

# Appendix 1: Fill Factor Rebuild script

--© 2019 | ByrdNest Consulting

-- ensure a USE <databasename> statement has been executed first.

--USE <Database>

GO

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Index Rebuild (defrag) script with logic from Jeff Moden and

additional logic to change fillfactor as needed

Designed to work with SS2012 and later, Enterprise Edition and

Developer Edition. If you are using Standard Edition, you

will need to modify the dynamic SQL and remove ONLINE = ON

This script was created to rebuild clustered and non-clustered

indexes with average fragmentation > 1.2%. It picks the top

15 (configurable) worse average fragmented indexes for an index

rebuild and it also varies each index fill factor (not heaps or

partitioned tables) to determine a "near optimum" value for

existing conditions. Once a fill factor value is determined, it

is fixed for each succeeding execution of this script. If the

fill factor value has not changed in last 90 days (configurable),

it is again put in the queue for finding the best fill factor

(rationale for this logic is that data skew and calling patterns

from applications may change over time).

If a table and its indexes are partitioned, this script rebuilds the

appropriate index partition with no adjustment to the fill factor.

This script should be executed from a SQL Agent job that runs daily

-- recommend time when server is least active. It also depends on

a table (created by this script first time run) to store index

parametrics for the fill factor determination.

You may alter this code for your own \*non-commercial\* purposes. You may

republish altered code as long as you include this copyright and give

due credit.

THIS CODE AND INFORMATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF

ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED

TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND/OR FITNESS FOR A

PARTICULAR PURPOSE.

Created By Comments

20190424 Mike Byrd Created

20190513 Mike Byrd Added additional data columns to

AgentIndexRebuilds table

20190604 Mike Byrd Added additional logic for setting FixFillFactor

20190616 Mike Byrd Revised FillFactor logic

20190718 Mike Byrd Added logic to get bad page splits (thanks to Jonathan Kehayias)

https://www.sqlskills.com/blogs/jonathan/tracking-problematic-pages-splits-in-sql-server-2012-extended-events-no-really-this-time/

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

SET QUOTED\_IDENTIFIER OFF

GO

--check to see if Admin schema exists

DECLARE @Database SYSNAME = (SELECT DB\_NAME())

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

code setup for Always On Primary Node; comment out next 4 statements

if not an Always On Node

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

-- DECLARE @preferredReplica INT

-- SET @preferredReplica

-- = (SELECT [master].sys.fn\_hadr\_backup\_is\_preferred\_replica(@Database))

-- IF (@preferredReplica = 0)

BEGIN

--define Admin schema if not exists

IF NOT EXISTS (SELECT 1 from sys.schemas WHERE [name] = 'Admin')

EXEC sp\_executesql N'CREATE SCHEMA [Admin] AUTHORIZATION [dbo]'

--define Admin.AgentIndexRebuilds if not exists

IF OBJECT\_ID(N'Admin.AgentIndexRebuilds') IS NULL

EXEC sp\_executesql N'

SET ANSI\_NULLS ON

--GO

SET QUOTED\_IDENTIFIER ON

--GO

CREATE TABLE [Admin].AgentIndexRebuilds(

ID INT IDENTITY(1,1) NOT NULL,

CREATEDATE DATETIME NOT NULL,

DBName SYSNAME NOT NULL,

SchemaName SYSNAME NOT NULL,

TableName SYSNAME NOT NULL,

IndexName SYSNAME NOT NULL,

PartitionNum INT NOT NULL,

Current\_Fragmentation FLOAT NOT NULL,

New\_Fragmentation FLOAT NULL,

PageSplitForIndex BIGINT NULL,

BadPageSplits BIGINT NULL,

New\_PageSplitForIndex BIGINT NULL,

PageAllocationCausedByPageSplit BIGINT NULL,

New\_PageAllocationCausedByPageSplit BIGINT NULL,

[FillFactor] INT NULL,

[Object\_ID] INT NULL,

Index\_ID INT NULL,

Page\_Count BIGINT NULL,

Record\_Count BIGINT NULL,

Forwarded\_Record\_Count BIGINT NULL,

New\_Forwarded\_Record\_Count BIGINT NULL,

LagDays INT NULL,

FixFillFactor INT NULL,

DelFlag INT NULL,

CONSTRAINT PK\_AgentIndexRebuilds PRIMARY KEY NONCLUSTERED

(ID ASC)

WITH (PAD\_INDEX=OFF,STATISTICS\_NORECOMPUTE=OFF,IGNORE\_DUP\_KEY=OFF

,ALLOW\_ROW\_LOCKS=ON,ALLOW\_PAGE\_LOCKS=ON,DATA\_COMPRESSION=ROW))

--GO

CREATE UNIQUE CLUSTERED INDEX CIX\_AgentIndexRebuilds\_CreateDateID

ON [Admin].AgentIndexRebuilds (CREATEDATE ASC,ID ASC)

WITH ( FILLFACTOR = 94, DATA\_COMPRESSION = ROW)

--GO

ALTER TABLE Admin.AgentIndexRebuilds

ADD DEFAULT (getdate()) FOR CREATEDATE'

END

GO

--Set Configuration parameters

DECLARE @RedoPeriod INT = 90 --Days

DECLARE @TopWorkCount INT = 20 --Specify how large result

-- set for Work\_to\_Do

-- --get current database name

DECLARE @Database SYSNAME = (SELECT DB\_NAME())

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

code setup for Always On Primary Node; comment out next 4 statements

if not an Always On Node

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

-- DECLARE @preferredReplica INT

-- SET @preferredReplica

-- = (SELECT [master].sys.fn\_hadr\_backup\_is\_preferred\_replica(@Database))

-- IF (@preferredReplica = 0)

BEGIN

DECLARE @Date DATETIME = GETDATE()

DECLARE @RowCount INT = 0

DECLARE @objectid INT

DECLARE @indexid INT

DECLARE @partitioncount BIGINT

DECLARE @schemaname SYSNAME

DECLARE @objectname SYSNAME

DECLARE @indexname SYSNAME

DECLARE @partitionnum BIGINT

DECLARE @partitions BIGINT

DECLARE @frag FLOAT

DECLARE @FillFactor INT

DECLARE @OldFillFactor INT

DECLARE @FixFillFactor INT

DECLARE @LagDate INT

DECLARE @NewFrag FLOAT

DECLARE @NewPageSplitForIndex BIGINT

DECLARE @NewPageAllocationCausedByPageSplit BIGINT

DECLARE @PageCount BIGINT

DECLARE @RecordCount BIGINT

DECLARE @ForwardRecordCount BIGINT

DECLARE @NewForwardRecordCount BIGINT

DECLARE @command NVARCHAR(4000)

DECLARE @Msg VARCHAR(256)

DECLARE @PartitionFlag BIT = 0

SET NOCOUNT ON

SET QUOTED\_IDENTIFIER ON; --needed for XML ops in query below

-- ensure the temporary work table does not exist

IF OBJECT\_ID(N'tempdb..#work\_to\_do') IS NOT NULL DROP TABLE #work\_to\_do

-- get worse avg\_fragmentation indexs (TOP @TopWorkCount)

-- conditionally select from the function,

SELECT TOP (@TopWorkCount) \*

INTO #work\_to\_do

FROM ( SELECT

ps.object\_id objectid,

ps.index\_id indexid,

o.[name] TableName,

i.[name] IndexName,

ps.partition\_number partitionnum,

ps.avg\_fragmentation\_in\_percent frag,

ios.LEAF\_ALLOCATION\_COUNT PAGE\_SPLIT\_FOR\_INDEX,

tab.split\_count BadPageSplit,

ios.NONLEAF\_ALLOCATION\_COUNT PAGE\_ALLOCATION\_CAUSED\_BY\_PAGESPLIT,

CASE WHEN i.Fill\_Factor = 0 THEN 100

ELSE i.Fill\_Factor END Fill\_Factor,

ps.page\_count,

ps.record\_count,

ps.forwarded\_record\_count,

ps.avg\_page\_space\_used\_in\_percent,

NULL New\_Frag,

NULL New\_PageSplitForInde,

NULL New\_PageAllocationCausedByPageSplit,

NULL New\_forwarded\_record\_count,

0 [Redo\_Flag],

ROW\_NUMBER() OVER (PARTITION BY ps.object\_id,ps.index\_id,ps.partition\_number,tab.split\_count ORDER BY tab.split\_count DESC) [RowNumber]

-- --get data for all tables/indexes

-- SAMPLED gives same avg fragmentation as DETAILED and is much faster

FROM sys.dm\_db\_index\_physical\_stats (DB\_ID(@Database),NULL,NULL,NULL,'SAMPLED') ps

JOIN sys.dm\_db\_index\_operational\_stats(DB\_ID(@Database),NULL,NULL,NULL) ios

ON ios.index\_id = ps.index\_id

AND ios.[object\_id] = ps.[object\_id]

AND ios.partition\_number = ps.partition\_number

AND ps.index\_level = 0

JOIN sys.indexes i

ON i.index\_id = ps.index\_id

AND i.[object\_id] = ps.[object\_id]

JOIN sys.objects o

ON o.[object\_id] = i.[object\_id]

JOIN sys.partitions p

ON p.[object\_id] = i.[object\_id]

AND p.index\_id = i.index\_id

LEFT JOIN sys.allocation\_units au

ON au.container\_id = p.[partition\_id]

LEFT JOIN (SELECT

n.value('(value)[1]', 'bigint') AS alloc\_unit\_id,

n.value('(@count)[1]', 'bigint') AS split\_count

FROM (SELECT CAST(target\_data as XML) target\_data

FROM sys.dm\_xe\_sessions AS s

JOIN sys.dm\_xe\_session\_targets t

ON s.address = t.event\_session\_address

WHERE s.name = 'SQLskills\_TrackPageSplits'

AND t.target\_name = 'histogram' ) as tab

CROSS APPLY target\_data.nodes('HistogramTarget/Slot') as q(n) ) AS tab

ON tab.alloc\_unit\_id = au.allocation\_unit\_id

WHERE i.index\_id > 0

AND o.[type] = 'U'

AND ps.avg\_fragmentation\_in\_percent > 1.20 --this is rebuild condition

AND ps.index\_level = 0 ) sub

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

The ORDER BY below looks at max avg\_frag and then alternates the

next day with indexes with max page splits.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

WHERE sub.RowNumber = 1

ORDER BY CASE WHEN DAY(getdate()) % 2 = 1

THEN sub.frag

ELSE sub.BadPageSplit END DESC

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Go back and find oldest index (>@RedoPeriod) with @FixFillFactor

and add it to #work\_to\_do

(to keep index fill factors from getting "stale").

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

IF OBJECT\_ID(N'tempdb..#Temp2') IS NOT NULL DROP TABLE #Temp2

SELECT TOP(1) CREATEDATE,ID,DBName,SchemaName,TableName,IndexName

,PartitionNum,Current\_Fragmentation,New\_Fragmentation

,PageSplitForIndex,New\_PageSplitForIndex

,PageAllocationCausedByPageSplit

,New\_PageAllocationCausedByPageSplit

,[FillFactor],[Object\_ID],Index\_ID,page\_count

,record\_count,forwarded\_record\_count

,New\_forwarded\_record\_count,LagDays,FixFillFactor

INTO #Temp2

FROM [Admin].AgentIndexRebuilds r

WHERE r.CREATEDATE <= DATEADD(dd,-@RedoPeriod,GETDATE())

AND r.DBName = @Database

AND r.FixFillFactor IS NOT NULL

AND r.DelFlag <> 1

AND NOT EXISTS (SELECT 1 FROM [Admin].AgentIndexRebuilds r2

WHERE r2.DBName = @Database

AND r2.[Object\_ID] = r.[Object\_ID]

AND r2.Index\_ID = r.Index\_ID

AND r2.PartitionNum = r.PartitionNum

AND r2.ID > r.ID)

--don't get partitioned tables (no adjusting fill factor)

AND NOT EXISTS (SELECT 1 FROM sys.partitions p

WHERE p.object\_id = r.object\_id

AND p.index\_id = r.index\_id

AND p.partition\_number > 1)

ORDER BY ID DESC, CreateDate DESC

SET @RowCount = @@ROWCOUNT

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Go back and recalculate FillFactor for oldest Table/Index

in Admin.AgentIndexRebuilds

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

IF @RowCount = 1

BEGIN

UPDATE #Temp2

-- start pertubation cycle over again

SET [FillFactor] = CASE WHEN Index\_ID > 1

AND FixFillFactor > 90

THEN 98

WHEN Index\_ID > 1

AND FixFillFactor > 80

THEN 94

WHEN Index\_ID > 1

AND FixFillFactor >=70

THEN 90

ELSE 100 END, --reset CI back to 100

FixFillFactor = NULL --reset FixFillFactor so that

-- regression can begin

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Reset fixfillfactor from previous passes (need to reset it for

all rows with Object\_ID, Index\_ID, & PartitionNum

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

UPDATE r

SET FixFillFactor = NULL

FROM [Admin].AgentIndexRebuilds r

JOIN #Temp2 t

ON t.[Object\_ID] = r.[Object\_ID]

AND t.Index\_ID = r.Index\_ID

AND t.PartitionNum = r.PartitionNum

AND (r.DelFlag IS NULL OR r.DelFlag = 0)

WHERE r.DBName = @Database;

--add new row to start regression

INSERT INTO #work\_to\_do

(objectid,indexid,TableName,IndexName,partitionnum

,frag,PAGE\_SPLIT\_FOR\_INDEX

,PAGE\_ALLOCATION\_CAUSED\_BY\_PAGESPLIT,Fill\_Factor

,page\_count,record\_count,forwarded\_record\_count

,avg\_page\_space\_used\_in\_percent,New\_Frag

,New\_PageSplitForIndex

,New\_PageAllocationCausedByPageSplit

,New\_forwarded\_record\_count,Redo\_Flag)

SELECT [OBJECT\_ID], Index\_ID, TableName,IndexName

,PartitionNum,Current\_Fragmentation

,PageSplitForIndex,PageAllocationCausedByPageSplit

,[FillFactor],page\_count,record\_count

,forwarded\_record\_count,NULL,NULL

,NULL,NULL,NULL,@RowCount [Redo\_Flag]

FROM #Temp2

END

-- Declare the cursor for the list of partitions to be processed.

IF EXISTS (SELECT 1 FROM #work\_to\_do)

BEGIN

DECLARE [workcursor] CURSOR FOR

SELECT DISTINCT w.objectid, w.indexid

, w.partitionnum, w.frag,w.Fill\_Factor

,w.TableName, w.IndexName

,DATEDIFF(dd,sub.CreateDate,GETDATE()) LagDate

,Redo\_Flag

FROM #work\_to\_do w

JOIN sys.indexes i

ON i.object\_id = w.objectid

AND i.index\_id = w.indexid

LEFT JOIN (SELECT TableName, IndexName, PartitionNum

, MAX(CreateDate) CreateDate

FROM [Admin].AgentIndexRebuilds r

WHERE r.DBName = @Database

GROUP BY TableName,IndexName,PartitionNum) sub

ON sub.TableName = OBJECT\_NAME(w.ObjectID)

AND sub.IndexName = i.[name]

AND sub.PartitionNum = w.partitionnum

ORDER BY w.Frag DESC

-- Open the cursor.

OPEN [workcursor]

-- Loop through the [workcursor].

FETCH NEXT FROM [workcursor]

INTO @objectid, @indexid, @partitionnum, @frag, @FillFactor

,@objectname,@indexname,@LagDate,@RowCount

WHILE @@FETCH\_STATUS = 0

BEGIN

IF OBJECT\_ID(N'tempdb..#Temp3') IS NOT NULL DROP TABLE #Temp3

SELECT @schemaname = s.[name]

FROM sys.objects AS o

JOIN sys.schemas as s

ON s.schema\_id = o.schema\_id

WHERE o.object\_id = @objectid

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Cannot reset fillfactor if table is partitioned, but can rebuild

the specified partition number

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

IF @partitionnum > 1 OR

EXISTS (SELECT 1 FROM sys.partitions p

WHERE p.object\_id = @objectid

AND p.Index\_ID = @indexid

AND p.partition\_number > 1)

SET @PartitionFlag = 1

ELSE

SET @PartitionFlag = 0

SET @OldFillFactor = @FillFactor

SET @FixFillFactor =

(SELECT FixFillFactor

FROM [Admin].[AgentIndexRebuilds] air1

WHERE air1.DBName = @Database

AND air1.ID = (SELECT MAX(ID)

FROM [Admin].[AgentIndexRebuilds] air2

WHERE air2.DBName = @Database

AND air2.[Object\_ID] = @objectid

AND air2.Index\_ID = @indexid

AND air2.PartitionNum = @partitionnum))

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

This is the logic for changing fill factor per index

Clustered Indexes are perturbed by decrementing the current fill

factor by 1 and nonclustered indexes fill factors are decremented

by one or two depending on the length of time since the index was

last rebuilt. Code is there to ensure the perturbed fill factor

is never less than 70%. This is an arbitrary number I set and

can be changed if required.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

IF @FixFillFactor IS NULL

BEGIN

SET @FillFactor = CASE WHEN @RowCount = 1

THEN @FillFactor --to catch redo index

--clustered index, only decrement by 1

--if already 100 then ratchet down

WHEN @indexid = 1 and

@LagDate IS NULL

AND @FillFactor = 100

THEN 99

WHEN @indexid = 1 AND

@LagDate IS NULL

THEN 100

WHEN @indexid = 1 AND

@LagDate <30

THEN @FillFactor -1

--nonclustered indexes,

-- decrement fill factor

-- depending on Lag days.

----if already 100 then ratchet down

WHEN @indexid > 1 AND

@LagDate IS NULL AND

@FillFactor = 100

THEN 98

WHEN @indexid > 1 AND

@LagDate IS NULL

THEN 100

WHEN @indexid > 1 AND

@LagDate < 14

THEN @FillFactor -2

WHEN @indexid > 1 AND

@LagDate >= 14

THEN @FillFactor -1

ELSE @FillFactor END

-- never let FillFactor get to less than 70

IF @FillFactor < 70

BEGIN

SET @FillFactor = 70

PRINT 'FillFactor adjusted back to 70.'

UPDATE [Admin].AgentIndexRebuilds

SET FixFillFactor = @FillFactor

WHERE DBName = @Database

AND Object\_ID = @objectid

AND Index\_ID = @indexid

AND PartitionNum = @partitionnum

END

END

ELSE

SET @FillFactor = @FixFillFactor

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Index is not partitioned

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

IF @PartitionFlag = 0

BEGIN

SET @command = N'SET QUOTED\_IDENTIFIER ON

ALTER INDEX ' + @indexname +' ON [' + @schemaname +

N'].[' + @objectname + N'] REBUILD WITH (ONLINE = ON,'+

' DATA\_COMPRESSION = ROW,MAXDOP = 1,FILLFACTOR = '+

CONVERT(NVARCHAR(5),@FillFactor) + ')'

END

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

IF Index is partitioned, rebuild, but don't play with fill factor

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

IF @PartitionFlag = 1

BEGIN

SET @FillFactor = @OldFillFactor

SET @command = N'SET QUOTED\_IDENTIFIER ON

ALTER INDEX ' + @indexname +' ON [' + @schemaname

+ N'].[' + @objectname + N'] REBUILD PARTITION = '

+ CONVERT(VARCHAR(25),@PartitionNum) +

N' WITH (ONLINE = ON, DATA\_COMPRESSION = ROW,MAXDOP = 1)'

END

EXEC sys.sp\_executesql @command

PRINT N'Executed ' + @command

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Get new fragmentation after rebuild

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

SELECT ps.avg\_fragmentation\_in\_percent

INTO #Temp3

FROM #work\_to\_do w

JOIN sys.dm\_db\_index\_physical\_stats (DB\_ID(@Database),@objectid

, @indexid,@partitionnum

,'SAMPLED') ps

ON ps.index\_id = w.indexid

AND ps.object\_id = w.objectid

AND ps.partition\_number = w.partitionnum

AND ps.index\_level = 0

WHERE w.indexid = @indexid

AND w.objectid = @objectid

AND w.partitionnum = @partitionnum

AND ps.index\_level = 0

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

If new frag > old frag, go back 1% fill factor, rebuild index,

and set FixFillFactor (for non-partioned indexes only)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

IF EXISTS (SELECT 1 FROM #Temp3

WHERE avg\_fragmentation\_in\_percent > @frag

AND @OldFillFactor <> @FillFactor

AND @RowCount = 0)

AND @PartitionFlag = 0

BEGIN

PRINT 'Setting @FixFillFactor'

SET @FillFactor = @FillFactor + 1

IF @FillFactor > 100 SET @FillFactor = 100

SET @FixFillFactor = @FillFactor

--archive out old data for that table/index/partionnum

UPDATE r

SET DelFlag = 1

FROM [Admin].AgentIndexRebuilds r

JOIN #Temp2 t2

ON t2.DBName = r.DBName

AND t2.Object\_ID = r.Object\_ID

AND t2.Index\_ID = r.Index\_ID

AND t2.PartitionNum = r.PartitionNum

WHERE r.DBName = @Database

AND r.Object\_ID = @objectid

AND r.Index\_ID = @indexid

AND r.PartitionNum = @partitionnum

AND r.ID <= t2.ID

AND r.DelFlag <> 1

SET @command = N'SET QUOTED\_IDENTIFIER ON

ALTER INDEX ' + @indexname + N' ON [' + @schemaname

+ '].[' + @objectname + N'] REBUILD '

+ N'WITH (ONLINE=ON,DATA\_COMPRESSION=ROW,MAXDOP=1,' +

N'FILLFACTOR = '+CONVERT(NVARCHAR(5),@FillFactor) +

')'

EXEC sys.sp\_executesql @command

PRINT N'Executed ' + @command

END

--insert results into history table (AgentIndexRebuilds)

INSERT [Admin].AgentIndexRebuilds (CREATEDATE, DBName

, SchemaName, TableName, IndexName, PartitionNum

, Current\_Fragmentation, New\_fragmentation

, PageSplitForIndex, BadPageSplits, New\_PageSplitForIndex

, PageAllocationCausedByPageSplit

, New\_PageAllocationCausedByPageSplit, [FillFactor]

, [Object\_ID], Index\_ID

, page\_count, record\_count, forwarded\_record\_count

, New\_forwarded\_record\_count, LagDays,FixFillFactor,DelFlag)

SELECT @DATE,@Database,@schemaname,@objectname

,@indexname,@partitionnum,@frag

, ps.avg\_fragmentation\_in\_percent

,w.PAGE\_SPLIT\_FOR\_INDEX,w.BadPageSplit,ios.LEAF\_ALLOCATION\_COUNT

,w.PAGE\_ALLOCATION\_CAUSED\_BY\_PAGESPLIT

,ios.NONLEAF\_ALLOCATION\_COUNT,@FillFactor,w.objectid

,w.indexid,w.page\_count,w.record\_count

,w.forwarded\_record\_count,ps.forwarded\_record\_count

,@LagDate,@FixFillFactor,1

FROM #work\_to\_do w

JOIN sys.dm\_db\_index\_physical\_stats

(DB\_ID(@Database),@objectid,@indexid,@partitionnum

,'SAMPLED') ps

ON ps.index\_id = w.indexid

AND ps.object\_id = w.objectid

AND ps.partition\_number = w.partitionnum

AND ps.index\_level = 0

JOIN sys.dm\_db\_index\_operational\_stats

(DB\_ID(@Database),@objectid,@indexid,@partitionnum) ios

ON ios.index\_id = ps.index\_id

AND ios.object\_id = ps.object\_id

AND ios.partition\_number = ps.partition\_number

WHERE w.indexid = @indexid

AND w.objectid = @objectid

AND w.partitionnum = @partitionnum

AND ps.index\_level = 0

SET @PartitionFlag = 0

FETCH NEXT

FROM [workcursor]

INTO @objectid, @indexid, @partitionnum, @frag

,@FillFactor,@objectname,@indexname,@LagDate,@RowCount

END

-- Close and deallocate the cursor.

CLOSE [workcursor]

DEALLOCATE [workcursor]

--clean up

IF OBJECT\_ID(N'tempdb..#Temp2') IS NOT NULL

DROP TABLE #Temp2

IF OBJECT\_ID(N'tempdb..#Temp3') IS NOT NULL

DROP TABLE #Temp3

IF OBJECT\_ID(N'tempdb..#TablesWithLOBs') IS NOT NULL

DROP TABLE #TablesWithLOBs

END

IF OBJECT\_ID(N'tempdb..#work\_to\_do') IS NOT NULL DROP TABLE #work\_to\_do

--Data retention

DELETE [Admin].AgentIndexRebuilds

WHERE CreateDate < DATEADD(yy,-3,GETDATE())

END

GO