[**A Couple of Data Compression Tricks in SQL Server 2008**](https://sqlserverperformance.wordpress.com/2011/08/23/a-couple-of-data-compression-tricks-in-sql-server-2008/)

Posted on [August 23, 2011](https://sqlserverperformance.wordpress.com/2011/08/23/a-couple-of-data-compression-tricks-in-sql-server-2008/) by [Glenn Berry](https://sqlserverperformance.wordpress.com/author/guderian1961/)

FROM: <https://sqlserverperformance.wordpress.com/2011/08/23/a-couple-of-data-compression-tricks-in-sql-server-2008/>

As I have said several times before, I am a **big fan** of data compression in SQL Server 2008 and above. Data compression is an Enterprise Edition only feature that was added to SQL Server 2008, and enhanced with Unicode compression in SQL Server 2008 R2.

It can be very effective in allowing you to trade some CPU utilization for a large reduction in I/O pressure and in memory pressure. Particularly if you have indexes that are relatively static, with data that compresses well, on large tables, with a weak or overstressed storage subsystem. Powerful CPUs are usually a lot less expensive than adding extra I/O capacity.

If you have large existing indexes that are good candidates for compression, you need to consider how to get those indexes compressed with the least performance impact on your system. Compressing an index means that it is rebuilt. The entire index must be read off of the storage subsystem, compressed, and then written back out to the storage subsystem. This can cause I/O pressure, memory pressure, and CPU pressure on your system.

If you use the Data Compression Wizard that is built into the SSMS GUI, the T-SQL script that it will generate can cause big concurrency issues for an active database that must be available 24 x 7 x 365.  This is because it will rebuild the index in OFFLINE mode, and it will not restrict the number of CPU cores that are used for the index rebuild (by using the MAXDOP = xx option). This will rebuild and compress the index as fast as your I/O subsystem and processors will allow, but it will end up locking your table and very likely pushing your CPU utilization to 100% during the operation.

A safer, but more time consuming alternative is to add the ONLINE = ON and MAXDOP = xx (where xx is 25-50% of your logical CPU cores) to the command, like you see in Figure 1 below.

-- Page compress clustered index on BigAddress table

ALTER INDEX [PK\_BigAddress]

ON [dbo].[BigAddress] REBUILD PARTITION = ALL

WITH ( PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, ALLOW\_ROW\_LOCKS = ON,

ALLOW\_PAGE\_LOCKS = ON, ONLINE = ON, MAXDOP = 2, SORT\_IN\_TEMPDB = ON, DATA\_COMPRESSION = PAGE);

**Figure 1: Example of modified ALTER INDEX command**

Adding ONLINE = ON will add about 50% more time to the ALTER INDEX operation, but will keep the table available during the command. Adding the MAXDOP option will restrict the number of logical CPU cores used for the operation (if you have the instance level MAXDOP set to zero, which is the default), thereby putting a ceiling on CPU utilization. This will also slow down the operation. You might be thinking “Why would I want to slow down the index rebuild?”, but believe me, running it ONLINE, with CPU usage throttled is much safer for concurrency purposes!

Once you have the command running in ONLINE mode, you might be curious about how long it will take to complete. Well, I just happen to have a query that will help you estimate that, shown in Figure 2.

-- Get Table names, row counts, and compression status for clustered index or heap

SELECT OBJECT\_NAME(object\_id) AS [ObjectName],

SUM(Rows) AS [RowCount], data\_compression\_desc AS [CompressionType]

FROM sys.partitions

WHERE index\_id < 2 --ignore the partitions from the non-clustered index if any

AND OBJECT\_NAME(object\_id) NOT LIKE 'sys%'

AND OBJECT\_NAME(object\_id) NOT LIKE 'queue\_%'

AND OBJECT\_NAME(object\_id) NOT LIKE 'filestream\_tombstone%'

AND OBJECT\_NAME(object\_id) NOT LIKE 'fulltext%'

AND OBJECT\_NAME(object\_id) NOT LIKE 'ifts\_comp\_fragment%'

GROUP BY object\_id, data\_compression\_desc

ORDER BY SUM(Rows) DESC OPTION (RECOMPILE);

**Figure 2: Getting row counts and compression status for clustered indexes**

It turns out that if you use the ONLINE = ON flag, after SQL Server starts writing out the PAGE compressed version of the clustered index, it will show up when you query sys.partitions, like you see in Figure 3, where you can see that it was about 90% done when I ran this query.

**ObjectName**               **RowCount**         **CompressionType**BigAddress                    330391354          NONE  
BigAddress                    306682536          PAGE

**Figure 3: Duplicate entries in sys.partitions while index is being compressed**

If the ALTER INDEX command has been running for 60 minutes, and you see that it is 90% done writing out the new index, that means that it will probably finish in about 3-4 minutes, since it took some time to read the index in off of the disk subsystem