# DB2 Compression: z/OS versus LUW

**By Craig S. Mullins**

Data compression for non-mainframe DB2 is quite a bit different than it is for DB2 for z/OS. In mainframe DB2, specifying COMPRESS YES on the CREATE TABLESPACE statement will cause DB2 to implement **Ziv-Lempel compression** (http://www.data-compression.com/lempelziv.html) for the table space in question. Data is compressed upon entry to the database and decompressed when it is read.

For DB2 on Linux, Unix, and Windows (LUW), when creating a table, you can use the optional VALUE COMPRESSION clause to specify that the table is using the space saving row format at the table level and possibly at the column level. There are two ways in which tables can occupy less space when stored on disk:

- If the column value is NULL, do not set aside the defined, fixed amount of space.
- If the column value can be easily known or determined (like default values) and if the value is available to the database manager during record formatting and column extraction.

When VALUE COMPRESSION is used, NULLs and zero-length data that has been assigned to defined variable-length data types (VARCHAR, VARGRAPHICS, LONG VARCHAR, LONG VARGRAPHIC, BLOB, CLOB, and DBCLOB) will not be stored on disk. Only overhead values associated with these data types will take up disk space.

If VALUE COMPRESSION is used then the optional COMPRESS SYSTEM DEFAULT parameter can also be specified to further reduce disk space usage. Minimal disk space is used if the inserted or updated value is equal to the system default value for the data type of the column. The default value will not be stored on disk. Data types that support COMPRESS SYSTEM DEFAULT include all numerical type columns, fixed-length character, and fixed-length graphic string data types. This means that zeros and blanks can be compressed.

At this point it would seem that the two platforms vary in how they approach "compression." The mainframe actually applies an algorithm to the data to compress it into another format. Every row that is inserted must first be compressed before storing it; every row that is read must be decompressed. On LUW platforms, DB2 compression is simply a way of avoiding the storage of certain types of data that either can be determined easily, or need not be stored.

But if you are up-to-date with DB2 for LUW, mainframe-like compression becomes a viable option. Row compression, using a compression dictionary as in DB2 for z/OS, was added to DB2 for LUW in Version 9. So for DB2 on LUW you need to know which version of DB2 you are using as well as what type of compression. Actually, once you get to DB2 10 for LUW there is even more to know.

## Adaptive Compression in DB2 10 for LUW

DB2 compression has been improved as of DB2 10 for LUW. IBM calls the new compression in DB2 10, **adaptive compression** (http://www-01.ibm.com/support/knowledgecenter/SSEPGG_10.1.0/com.ibm.db2.luw.admin.dbobj.doc/doc/c0059054.html). With adaptive compression you can expect improved compression rates.

Adaptive compression combines table level compression with an additional page level compression. The compression at the page level can change over time based on your actual data – hence, “adaptive” compression. Significant improvement in compression rates can be achieved with adaptive compression.

## When Should You Compress?

So, when should you consider using compression? In general, it is wise to use DB2 for z/OS compression for larger table spaces where the disk savings can be significant. But it can also make sense for medium sized table spaces (especially when data is accessed sequentially) because compressing the data can enable more rows to be stored on each page. Compressing very small tables usually does not make sense because the amount of space required to store the compression dictionary may exceed the space saved by compressing the data.

What is the compression dictionary? Well, as I mentioned earlier, DB2 for z/OS compression is enabled by specifying COMPRESS YES for the table space in your DDL. When compression is specified, DB2 builds a static dictionary to control compression. This will cause from 2 to 17 dictionary pages to be stored in the table space. These pages are stored after the header and first space map page.

For partitioned table spaces, DB2 will create a separate compression dictionary for each table space partition. Multiple dictionaries tend to cause better overall compression ratios. In addition, it is more likely that the partition-level compression dictionaries can be rebuilt more frequently than non-partitioned dictionaries. Frequent rebuilding of the compression dictionary can lead to a better overall compression ratio.

It is also reasonable to avoid compressing table spaces with multiple tables in them because the compression ratio can be impacted by the different types of data in the multiple tables, and DB2 can only have one compression dictionary per table space.

**But why compress data at all?** (http://www.lostsaloon.com/technology/what-is-data-compression-and-why-use-it/) Consider an uncompressed table with a large row size, say 800 bytes. Therefore, five of this table's rows fit on a 4K page. If the compression routine achieves 30 percent compression, on average, the 800-byte row uses only 560 bytes, because (800-(800*0.3))=560. Now, on average, seven rows fit on a 4K page. Because I/O occurs at the page level, the cost of I/O is reduced because fewer pages must be read for table space scans, and the data is more likely to be in the buffer pool because more rows fit on a physical page. This can be a significant I/O improvement. Consider the following scenarios. A 10,000-row table with 800-byte rows requires 2,000 pages. Using a compression routine as outlined previously, the table would require only 1,429 pages. Another table also with 800-byte rows but now having 1 million rows would require 200,000 pages without a compression routine. Using the compression routine, you would reduce the pages to 142,858 - a reduction of more than 50,000 pages.

Another question I am commonly asked is about overhead. Yes, there is going to be some overhead involved if you turn on compression... CPU is required to apply the Ziv-Lempel algorithm to compress upon insertion - and to de-compress upon access. Of course, this does NOT mean that overall performance will suffer if you turn on compression. Remember the trade-off: additional CPU in exchange for possibly improved I/O efficiency. You see, when more compressed rows fit onto a single page fewer I/O operations may be needed to satisfy your query processing needs. If you are performing a lot of sequential access (as opposed to random access) you can get improved performance because fewer I/O operations are required to access the same number of rows.

Of course. there is always the other trade-off to consider: disk storage savings in exchange for CPU cost of compressing and decompressing data. Keep in mind, too though, that DB2 for z/OS uses hardware-assisted compression. This means that the compression algorithm is in hardware microcode, thereby diminishing the amount of overhead and speeding up the process. So, the overall cost of compression *may* be minimal… Indeed, due to I/O issues, overall elapsed time for certain I/O heavy processes can decrease when data is compressed.

**How Much Compression?**

For DB2 for z/OS, you can use the DSN1COMP utility to estimate how much disk space will be saved by compressing a table space before deciding whether to turn compression on or not. This utility can be run on full image copy data sets, VSAM data sets that contain DB2 table spaces, or sequential data sets that contain DB2 table spaces (such as DSN1COPY output). DSN1COMP does not estimate savings for data sets that contain LOB table spaces or index spaces. Refer to the IBM Utility Guide and Reference for more information on DSN1COMP.

For DB2 LUW, you use Data Studio to estimate how much storage you could save via adaptive compression.  You will use the Data Source Explorer to find Schemas, find SYSPROC and then run the ADMIN_GET_TAB_COMPRESS_INFO user-defined function. Simply provide the name of the table to estimate and the UDF does the rest.

**In Conclusion**

Of course, before you consider compression be sure to examine all of its details -- and be sure to understand all of the nuances of your particular data and applications. I mean, we didn't even touch upon **index compression** (http://it.toolbox.com/blogs/db2zos/index-compression-and-ibm-redpapers-18160) in this article…

But don't be afraid of investigating data compression in DB2... it can be a very handy tool in the DBA's arsenal!