Best Practices to Compress Db2 Column-Organized Tables

Version 1.0

Db2 Engine Storage Development

Ron Liu ronliu@us.ibm.com
Bob Lyle blyle@us.ibm.com
Christina Lee tinalee@us.ibm.com

TABLE OF CONTENTS

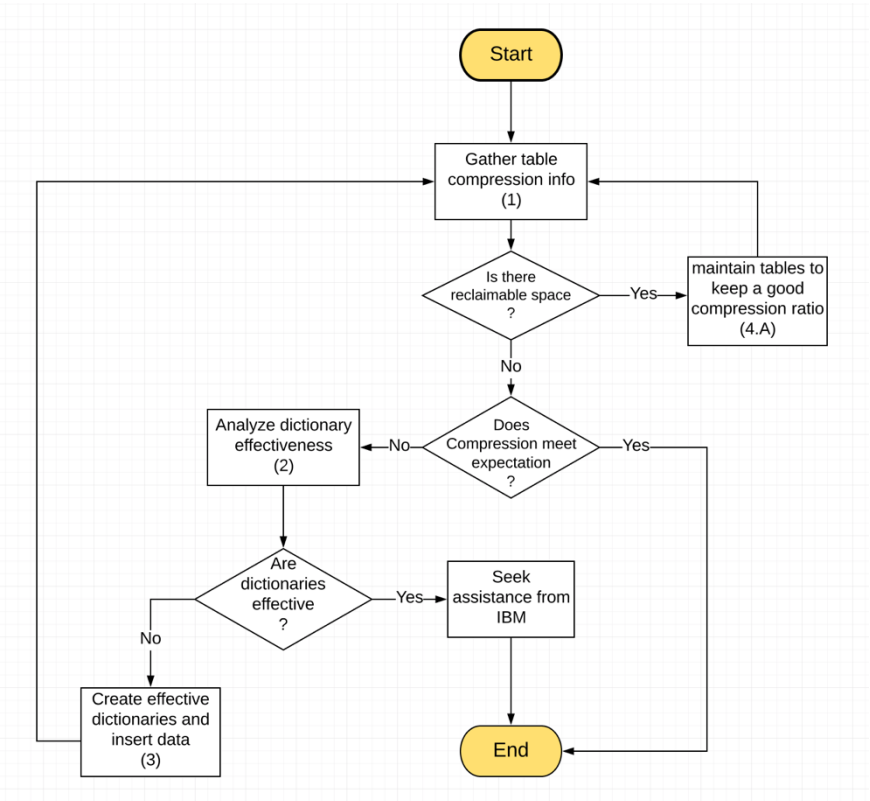
INTRODUCTION

This document describes the recommended practices to achieve the best possible compression for BLU tables. It will start with flow charts that highlight the steps to look at the current compression ratio, determine the effectiveness of the column dictionaries, and build more effective dictionaries, followed by detailed descriptions of those steps. It will end with instructions to best maintain the BLU tables to keep a good compression ratio. Terminology used in BLU compression will be defined in each section.

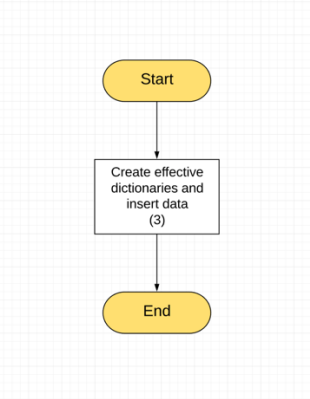
WHAT ARE THE STEPS TO ACHIEVE THE BEST POSSIBLE COMPRESSION?

The following flow charts depict two scenarios, one for existing tables and one for new tables. The numbers within the flow charts correspond to the numbered steps described in the following sections.

A. The following flow chart highlights the steps necessary to evaluate compression on existing tables and how to improve the compression if needed.



B. The following flow chart illustrates the simplified steps when creating and populating new tables. In this scenario, you can proceed directly to step #3.

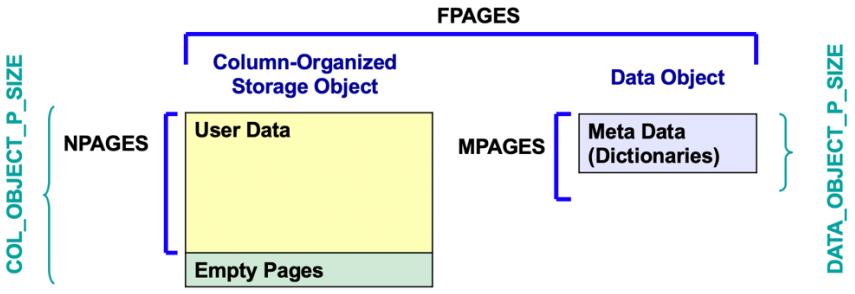


1. GATHER INFORMATION TO DETERMINE HOW WELL TABLES ARE COMPRESSED

This section will begin with some terminology used in BLU compression, then describe the queries and tools that can be used to retrieve information on features such as storage space, table/database sizes, and compression ratio. In addition, this section will illustrate how to determine if compression has become an issue requiring investigation.

Compression ratio: Value used to measure effectiveness of compression. It is calculated by dividing the original data size by the compressed data size. For example, if the original data size is 10GB and the compressed size is 2GB, the compression ratio is 5.

Catalog Statistics for Measuring Number of Pages (SYSCAT.TABLES): After *runstats* on the base table is performed, the catalog is populated with the following statistics which can be used to measure the size of the table.



NPAGES: Number of pages in Column-Organized Object (contains user data) minus any empty pages.

MPAGES: Number of Meta Data pages in Data Object (includes dictionaries).

FPAGES: Total number of pages in both objects.

Table Function ADMIN_GET_TAB_INFO reports physical size of objects

- **COL_OBJECT_P_SIZE:** Physical size of column-organized data object containing user data.
- **DATA_OBJECT_P_SIZE:** Physical size of data object containing meta data.
- **INDEX_OBJECT_P_SIZE:** Physical size of the indexes.
- **LOB_OBJECT_P_SIZE:** Physical size of the LOB data.

Column-Organized Table Total Physical Storage Size = COL_OBJECT_P_SIZE + DATA_OBJECT_P_SIZE + INDEX_OBJECT_P_SIZE + LOB_OBJECT_P_SIZE. This is the most accurate measurement of the storage consumption.

The following are the commands, queries, and tools that are used to retrieve information regarding table size and compression, where BLUDB is the sample database, TPCH is the sample schema, and ORDERS is the sample table.

A. Perform runstats on the table to get the most current statistics. If statistics profile is available, use RUNSTATS USE PROFILE. Note the INDEXES ALL option is required even for columnar tables.

```
RUNSTATS ON TABLE TPCH.ORDERS WITH DISTRIBUTION AND SAMPLED DETAILED INDEXES ALL
DB20000I  The RUNSTATS command completed successfully.
```

B. Use the following query to get the physical size of the data used by the column-organized table. Omitting the table name and just leaving quotes without a space will retrieve the information for all tables under the schema. This query also indicates the amount of space that can be reclaimed via REORG TABLE RECLAIM EXTENTS.

```
select TABSCHEMA, TABNAME, sum(PHYSICAL_SIZE_MB) as PHYSICAL_SIZE, sum(RECLAIMABLE_SPACE) as
RECLAIMABLE_SPACE from (SELECT CAST(TABSCHEMA as char(16)) TABSCHEMA, CAST(TABNAME as char(20))
TABNAME,  (DATA_OBJECT_P_SIZE+COL_OBJECT_P_SIZE+INDEX_OBJECT_P_SIZE+LOB_OBJECT_P_SIZE)/1024  AS
PHYSICAL_SIZE_MB,       RECLAIMABLE_SPACE       AS       RECLAIMABLE_SPACE       FROM       TABLE
(SYSPROC.ADMIN_GET_TAB_INFO('TPCH','ORDERS') ) ) group by tabschema, tabname

TABSCHEMA        TABNAME              PHYSICAL_SIZE        RECLAIMABLE_SPACE
---------------- -------------------- -------------------- --------------------
TPCH             ORDERS                              62468               635008

  1 record(s) selected.
```

If RECLAIMABLE_SPACE is larger than zero, the space can be reclaimed. More details can be found in section 4.

C. Estimate the compression ratio of the table.

```
SELECT CAST(TABSCHEMA as char(16)) TABSCHEMA, CAST(TABNAME as char(20)) TABNAME, PCTPAGESSAVED,
DEC(1.0/(1.0-(PCTPAGESSAVED*1.0)/100.0),31,2) AS compression_ratio FROM SYSCAT.TABLES tab WHERE
TABSCHEMA = 'TPCH' AND TABNAME = 'ORDERS'
```

```
TABSCHEMA        TABNAME              PCTPAGESSAVED COMPRESSION_RATIO
---------------- -------------------- ------------- --------------------------------
TPCH             ORDERS                         64                             2.77

  1 record(s) selected.
```

This section described the queries that can be used to retrieve the information regarding compression and storage space. If the compression ratio of the tables degrades over time or more storage space is used than expected, it can be an indicator that compression might have become an issue requiring investigation.

2. HOW EFFECTIVE ARE THE TABLE-LEVEL COLUMN DICTIONARIES?

A. Get information of how well each column is 'covered' by its column dictionary.

```
SELECT CAST(COLNAME AS CHAR(20)) COLNAME, CAST(TYPENAME AS CHAR(16)) TYPENAME, PCTENCODED,
COLCARD FROM SYSCAT.COLUMNS WHERE TABSCHEMA = 'TPCH' AND TABNAME = 'ORDERS'

COLNAME              TYPENAME         PCTENCODED COLCARD
-------------------- ---------------- ---------- --------------------
O_ORDERKEY           BIGINT                  100            150007839
O_CUSTKEY            INTEGER                 100             77594624
O_ORDERSTATUS        CHARACTER               100                    3
O_TOTALPRICE         DECIMAL                 100             36438016
O_ORDERDATE          DATE                    100                 2400
O_ORDERPRIORITY      CHARACTER               100                    5
O_CLERK              CHARACTER               100              1003520
O_SHIPPRIORITY       INTEGER                 100                    1
O_COMMENT            VARCHAR                   1             58982400

  9 record(s) selected.
```

100% encoding means that the dictionary for that column is able to encode all of the data values for that column. The encoding rate is not an indication of compression per se, but it is typically highly correlated with the compression ratios achieved, provided a good quality dictionary is in place. This is true for columns of data types other than string data types (CHAR, VARCHAR, GRAPHIC, VARGRAPHIC, BINARY, VARBINARY, CHAR FOR BIT DATA, VARCHAR FOR BIT DATA). PCTENCODED includes only the compression done using table-level dictionaries. It does not include page-level compression and string compression.

B. If PCTENCODED for the non-string columns is not high (i.e., < 90%), it indicates the table-level dictionaries are not effective. For string columns, page-based compression might be in effect. Page-based compression does not use column dictionaries and so the effectiveness is not reflected by PCTENCODED. So evaluating the effectiveness of the column dictionaries using PCTENCODED should skip the string columns. The instructions in the following sections can be used to build effective dictionaries.

3. HOW TO REBUILD TABLE, CREATE EFFECTIVE TABLE-LEVEL DICTIONARIES, AND INSERT DATA

A recommended approach that usually results in the best compression is to pre-build the dictionary using LOAD with the RESETDICTIONARYONLY option to build the dictionary from a random sample of the rows in the source table. A random sample of 5 million rows is typically sufficient to build an efficient dictionary. Fewer rows may be used in many cases (depending on the types and frequencies of values in the source table), but at 5 million, typically even some of the more complicated data patterns are handled. For comparison, the Automatic Dictionary Creation (ADC) process used by insertion looks at between 500K (for a single node system) and 1 million rows (multi-node systems). The concern with the ADC approach is that when there is clustering present in the incoming insertion stream due to ordering of the data, the ADC process may only see a small subset of the values, and thus the compression dictionaries may not be truly representative of the underlying data. Pre-building the dictionary with a true random sample can avoid this problem.

The following steps are used to create effective table-level dictionaries and populate the table. For container users who have just upgraded to container 1.0.19.5 or later from an earlier container, these steps will also compress the string data through the page-based compression mechanism when applicable. Note for temporal tables, the same steps need to be completed for both the temporal and history tables. More details can be found in subsection 4d.

A. Create identical table.

```
create table T1 like exttable organize by column;
```

B. Create a sampling cursor from the source table.

Db2 provides a TABLESAMPLE clause to allow sampling the data in the table. If the number of rows in a table is known, a sample rate should be specified that results in about 5 million rows in the sample. For example, if the table had 500 million rows, a sample rate of 1% should be specified to get approximately 5 million rows:

```
declare cursor1 CURSOR FOR select * from exttable tablesample bernoulli(1);  -- <== 1% sample
rate to get 5M of 500M rows
```

Note that a higher sampling rate does not always produce better compression If the result set of the above query is too large. It will not be fully used to build the dictionaries because there is a limit on how large a dictionary can be.

C. Use the sampling cursor with LOAD to pre-build the compression dictionary.

Load has a RESETDICTIONARYONLY option that will only build the compression dictionary. When this option is specified, no rows are actually inserted into the table. Note that internally, LOAD itself will sample incoming data by default. Because the sample has already been generated on the source data, the sampling that is typically done in load needs to be overridden

to avoid 'double sampling'. This is done with another clause (MODIFIED BY CDEANALYZEFREQUENCY=100) that says all rows should be used to build the compression dictionary:

```
load from cursor1 OF CURSOR  MODIFIED BY CDEANALYZEFREQUENCY=100 replace resetDictionaryOnly
into T1;  -- <== Load has its own internal sampling, turn it off to avoid double sampling....
```

Once the load is done, the table T1 will be empty (no truncate table needed), but the compression dictionary will be present.

D. Populate the new table with insert.

```
insert into T1 select * from exttable;
```

If the new table T1 has the same rows as the old table (i.e., verified with the EXCEPT ALL query on the old table), the old table can be dropped, and the new table can be renamed to the old table name.

E. To migrate or REORG temporal tables, the corresponding history table should also be updated. This subsection describes the steps for moving data over to a new temporal table with optimal dictionary. Db2 BLU supports application-period temporal table.

a. Create identical tables for both the temporal and history tables.

```
create table new_T as T;
create table new_T_hist like  T_hist;
```

b. Pre-build the dictionary with data sampling and load the data for both new_T and new_T_hist following the steps described in 4.B and 4.C.

c. Convert new_T into a temporal table.

```
alter table new_T add period SYSTEM_TIME(sys_start, sys_end) maintained by user;
```

d. Link the history table.

```
alter table new_T add user versioning use history table new_T_hist;
```

e. Test the new temporal table with some temporal queries.

f. If the test returns correct results as expected, drop the old tables and rename the new tables as desired.

4. HOW TO BEST MAINTAIN THE COLUMN-ORGANIZED TABLES TO KEEP A GOOD COMPRESSION RATIO

A. Reclaim freed extents.

Currently, there is no full REORG operation for column-organized tables. There is a REORG TABLE ... RECLAIM EXTENTS command that can reclaim full extents of space that have been freed up due to deletion or update activity. The query to retrieve the reclaimable space can be found in Section 1.B.

The following command is used to reorg table to reclaim the freed extents for the tables.

```
reorg table TPCH.ORDERS reclaim extents
DB20000I  The REORG command completed successfully.
```

For all tables, repeat the steps to retrieve the information of the physical storage space and reclaim the freed extents for those with RECLAIMABLE_SPACE. The following command can be used to remove freed extents from the tablespace to reduce the physical size of the tablespace. Note, the max should be only used if no concurrent INSERTs are running.

```
alter tablespace userspace1 reduce max
DB20000I  The SQL command completed successfully.
```

Besides the above manual steps to reclaim extents, the Db2 health monitor is able to detect tables with large amounts of reclaimable space (as indicated in 1.B) and automatically launch a reclaim extents command on tables where appropriate (based on a threshold of reclaimable space).

After reclaiming the extents and removing the freed extents from the tablespace, RUNSTATS on the table to update the statistics. Since the REDUCE MAX command is asynchronous, users need to wait until it completes to do a RUNSTATS. The MON_GET_EXTENT_MOVEMENT_STATUS table function can be used to return the status of the extent movement operation.

B. Rebuild tables with degrading dictionaries.

Once a table-level dictionary is built for a column-organized table, that compression dictionary will continue to be used for all subsequent insert operations. If the new data is similar to the data that was used to build the table level dictionary, the compression dictionary will typically continue to be effective.

However, if the nature of the data changes significantly, it's possible that the compression ratio of the new data will decline. A good indication of compression ratio degradation is that PCTENCODED (reference 2.A) of the columns keep going down. In this case, the best practice is to unload and reload the data back into the table, or alternately, reload the data into the new table. Then, the user must delete the old table and rename the new table (Section 3).