Db2 with BLU Acceleration Best Practices for Storing Data in Column-Organized Tables

Version 1.0

Db2 BLU Engine Storage Development

Chris Drexelius cdrexeli@us.ibm.com
Christina Lee tinalee@us.ibm.com
Ron Liu ronliu@us.ibm.com

Table of Contents

Prerequisite

Some tasks depicted in this paper are detailed in the Best Practices for Compression for column-organized tables paper which is a prerequisite. Ensure you refer to the proper version of that paper based on your Db2 form factor.

Introduction

Extreme compression and advanced query performance are two important characteristics of IBM Db2 with BLU Acceleration that are tightly coupled. In many cases, fast query performance is directly tied to the level of compression for the column-organized tables involved in the queries. For this reason, it is critical to maximize compression by following the Best Practices for Compression paper noted in the Prerequisites section above to significantly improve query performance. In addition, the higher the compression ratio, the lower the storage that will be consumed.

However, compression ratio is not the only factor that determines the amount of storage used. In some cases, it is possible to have a good compression ratio but still consume an excessive amount of storage due to the infrastructure designed to support massive parallelism. To understand how to best reduce storage consumption, this paper first explains basic concepts on how the table data is stored and how to retrieve information regarding storage space. Then it describes some common database operations and their impact to storage. Finally, it provides best practices for users to minimize their table storage consumption.

1. How is Table Data Stored?

Figure 1 – BLU Storage Layout

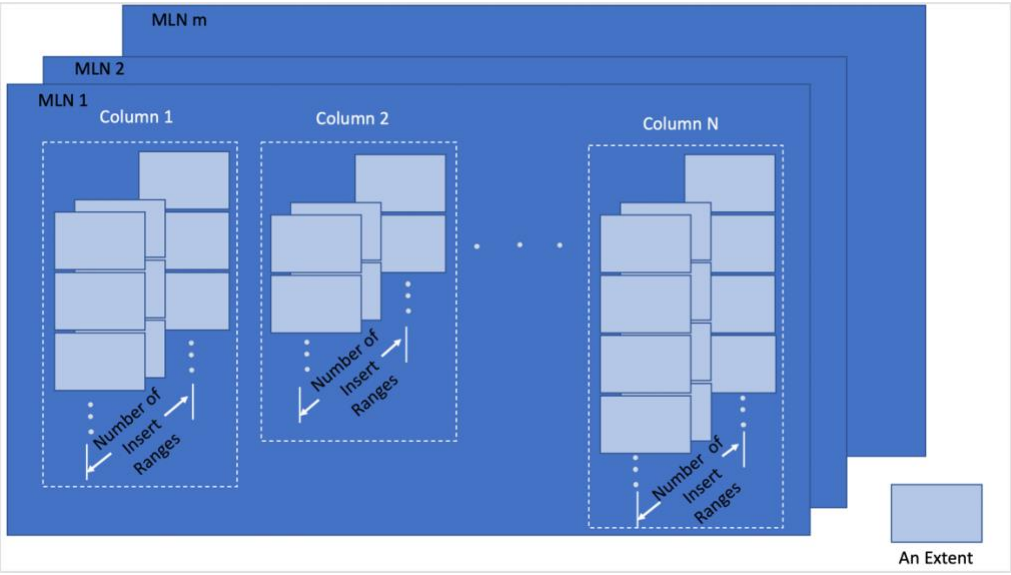


Figure 1 depicts the storage layout of BLU tables. This section discusses the concepts used in BLU storage and how they come together to construct the layout of BLU storage to support massive parallelism. Note that starting from Db2 11.5.6, in IIAS and IBM Data Warehouse on Cloud, an optimization for trickle inserts was enabled where in some cases data pages may contain data from multiple columns. This results in less data pages needed to store the data in some cases (i.e., small tables) compared to previous BLU versions.

Base table: A base table is another name for a user-defined column-organized table.

Synopsis table: A synopsis table is a metadata table for a base table that is typically created when a base table is created. Synopsis tables are real column-organized tables in Db2. Synopsis tables are automatically populated as a base table is populated. Refer to later sections in this document for more info on what metadata is captured in a synopsis table and how it is used.

Page: A page is the smallest unit of I/O that Db2 can handle. Page size is configurable at various points during database, bufferpool, and tablespace creation.

Extent: An extent is a unit of Db2 space allocation. The extent size is the number of pages that are allocated to a database object per allocation. For example, the default extent size for DB2_WORKLOAD=ANALYTICS is 4 (pages). Default extent size is configurable via configuration parameter dft_extent_sz as well as during tablespace creation.

MLN: Multiple Logical Node is a logical data partition in a Db2 MPP (Massive Parallel Processing) environment.

Insert range: An insert range is a logical cross-section of a table that enables multiple parallel threads to each work on their own section independently with reduced resource contention. Within each MLN, a table is logically separated into insert ranges. Insert ranges cover complete rows, so each column for a given row resides within an insert range.

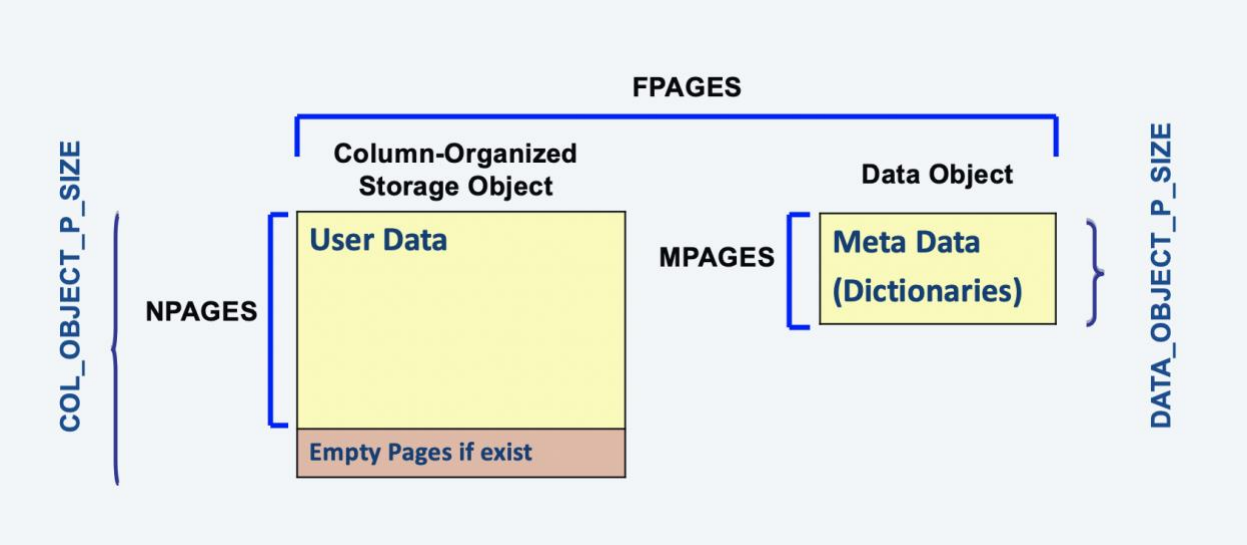
Column group: A column group is a subgroup of columns that Db2 internally separates to optimize performance. Db2 BLU does not support user-defined columns groups like in row-organized tables. From an external perspective, each column group usually contains only a single external column.

Each parallel thread allocates extents as needed for its insert range. To compute the total size of a table, the size of the extents allocated on each MLN for each column group for each insert range must be combined.

2. How is the Storage for a BLU Table Measured?

This section discusses the BLU storage objects and how to measure them.

Figure 2 – BLU Storage Objects Measurement



User data: The actual user data that is stored in a BLU table, i.e., the tuples and rows.

Table Metadata: The metadata that contains the column dictionaries and other table storage information.

COL_OBJECT_P_SIZE: Physical size of column-organized data object containing user data.

DATA_OBJECT_P_SIZE: Physical size of data object containing table metadata.

INDEX_OBJECT_P_SIZE: Physical size of the indexes. Note that all BLU tables have one internal associated index object that provides a mapping that enables us to locate data in data pages. This value may also include uniqueness or secondary indexes if defined for the table.

LOB_OBJECT_P_SIZE: Physical size of the LOB data.

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 that contain user data. This is computed as the total number of data pages minus any empty pages.

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

FPAGES: Total number of pages in both data and column-organized objects. FPAGES includes empty pages.

For example, the following query retrieves NPAGES, MPAGES, and FPAGES of a table.

```
select NPAGES, MPAGES, FPAGES from SYSCAT.TABLES where tabschema='TPCH' and
tabname='CUSTOMER'

NPAGES               MPAGES               FPAGES
-------------------- -------------------- --------------------
              382930                    3               385350

  1 record(s) selected.
```

Column-Organized Table Total Physical Storage Size = COL_OBJECT_P_SIZE + DATA_OBJECT_P_SIZE + INDEX_OBJECT_P_SIZE + LOB_OBJECT_P_SIZE

Table Function *ADMIN_GET_TAB_INFO* reports physical size of the objects.

For example, the following query retrieves the table size based on the calculation of the object sizes.

```
select TABSCHEMA, TABNAME, sum(PHYSICAL_SIZE_MB) as PHYSICAL_SIZE_MB 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 FROM TABLE (SYSPROC.ADMIN_GET_TAB_INFO('TPCH','CUSTOMER') ) ) group by
tabschema, tabname

TABSCHEMA        TABNAME              PHYSICAL_SIZE_MB
---------------- -------------------- --------------------
TPCH             CUSTOMER                            12053

  1 record(s) selected.
```

Note *ADMIN_GET_TAB_INFO* can run very slowly for very large tables that are being updated frequently. In such an environment, an alternative to get the table size is through Db2 snapshot monitoring.

3. What Additional Storage Overhead Can Come with a BLU Table?

BLU storage is designed to support massive parallelism and superior query performance. While the design favors performance, it uses storage more aggressively in certain circumstances and hence there is potentially some overhead. This section discusses the potential storage overhead, and subsequent sections present how it can be minimized.

Page sparsity overhead: If a data page is not fully filled during insert processing, it is categorized as having page sparsity. Page sparsity can also occur when some rows are deleted. When a row is deleted, its state is updated accordingly in the table's hidden TupleState column that represents the state of that row. However, the storage is not actually released until the extent is reclaimed. Note that an extent can only be reclaimed when all values in the pages contained within the extent have been deleted and are no longer needed.

Extent overhead: Extent overhead occurs when an extent contains pages with tuples, but also contains pages that do not yet contain tuples.

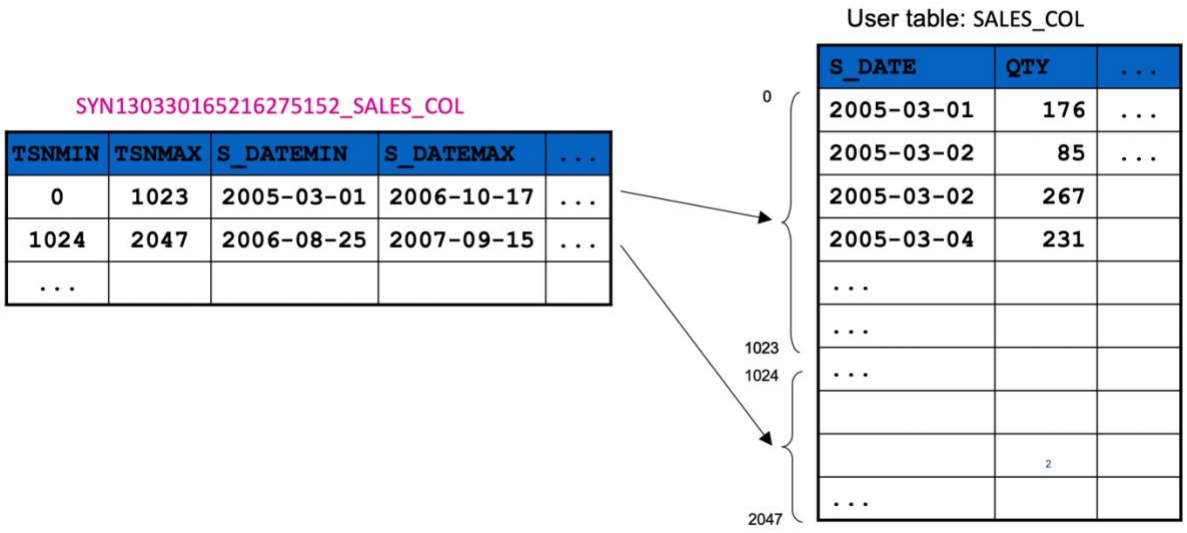
Last extent overhead: After a table is populated and all the rows have been deleted from the table, the last allocated extent in each insert range containing the high water mark can still not be reclaimed and released.

BLU storage will benefit from minimizing these potential overheads. More details can be found in the following sections.

4. How do Synopsis Tables Impact BLU Table Storage Usage?

In BLU, a synopsis table is created when the base table is created, except for base tables without any eligible synopsis columns (i.e., if all base table columns are variable-length columns with length greater than 1000). It acts like a filter that can narrow down the search range during query processing and hence improve query performance. Synopsis tables are real tables and therefore, have storage costs associated with them as any BLU base table would. This storage overhead ranges from very small (for large base tables) to potentially significant (for small base tables).

Figure 3 – Synopsis Table and Base Table



As shown in Figure 3, the base table is on the right whereas its corresponding synopsis table is on the left. Essentially a synopsis table is a table of metadata that describes minimum and maximum values for chunks of rows in the base table. As noted above, variable-length base table columns with length greater than 1000 do not have corresponding columns in the synopsis table. Each row in the synopsis table describes a range of rows in the base table which usually has a 1K threshold. It contains the MIN and MAX values for each column in that range. When searching for qualifying rows during query processing, if the desired predicate range is not between the MIN and MAX value in the synopsis table, the corresponding range of rows in the base table can

be filtered out. The query only processes rows that are not filtered out which can greatly reduce the I/O cost and improve the query performance. Filtering through synopsis columns benefits from data clustering in the base table.

Given the base table, the name of its synopsis table can be retrieved from the catalog and always starts with “SYN” followed by a series of numbers. Synopsis tables always reside in the SYSIBM schema.

For example, the following query retrieves the name of the synopsis table of TPCH.CUSTOMER.

```
SELECT tabname FROM syscat.tabdep WHERE bschema='TPCH' and bname='CUSTOMER' and DTYPE = '7'

TABNAME
------------------------------------------------------------------------------------------------
----------------------------------
SYN200528175642625463_CUSTOMER

  1 record(s) selected.
```

Once the name of the synopsis table is found, its storage information can be retrieved with the methods discussed in Section 2.

A synopsis table stores the MIN and MAX values of the columns of its base table. In the best-case scenario, the synopsis table is roughly 1/512 of the size of the base table (excluding two additional metadata columns) because the synopsis table stores the MIN and MAX values for every 1024 tuples in the corresponding base table column, and there are 2X columns in the synopsis table (512 = 1024 / 2).

Page sparsity overhead can occur for a synopsis table because of some operations on the base table (e.g., insertion, deletion, update, LOAD utility processing, etc.).

Extent overhead can also happen to a synopsis table if the tuplets for a column group in the synopsis table cannot fill all the pages in an extent. Extreme synopsis table extent overhead is most likely to occur with small base tables. For example, say the extent size is 4 pages and a small base table with 20 columns only contains 10 rows where those 10 tuplets for each column all fit into one data page. With an extent size of 4 pages, 1 page for each column group will contain data, but the 3 remaining pages will be empty and unused. Since each insert range allocates at least one extent for each column group and the insert range concept applies to each MLN in an MPP system, this can result in excessive storage waste and consumption with no query performance benefit. Note that as soon as we write the first tuplet for a column group, we must allocate an entire extent of pages. Aside from the storage considerations, from a query performance perspective it doesn’t make sense to do extra synopsis query filtering for very small base tables.

In Db2 11.5.4, an enhancement called Deferred Synopsis Tuple Creation (enabled by default in IIAS and IBM Data Warehouse on Cloud and may be enabled by registry variable in other form factors) was introduced to help minimize this synopsis table extent overhead. With this feature enabled, for each synopsis insert range the first tuple will only be added when the number of

rows in the corresponding insert range in the base table reaches a higher threshold (10K rows). Note that in some cases the first row for a synopsis table's insert range may still correspond to fewer than 10K rows in the base table if we determine during insert processing that the insert range will include more than 10K rows. The idea is that Db2 defers adding synopsis tuples until it knows that the storage cost of adding the first tuple to the synopsis table is worth it from a space consumption and query filtering perspective. After the first synopsis table row is written for each insert range that covers a larger range of base table rows, subsequent tuples will be written using the regular 1K row threshold.

The following query can be used to retrieve statistics for the rows in a synopsis table.

```
db2 "select TSNMIN, TSNMAX, (TSNMAX - TSNMIN) as Diff from SYSIBM.<syn-table-name> order by
TSNMIN"
```

For a small base table, the query retrieves no rows from its synopsis table because the base table does not have enough rows to reach the threshold for creating a row in the synopsis table.

Figure 4 – No rows in synopsis table for small table

```
TSNMIN               TSNMAX               DIFF
-------------------- -------------------- --------------------

  0 record(s) selected.
```

For a large base table, the query retrieves the statistics of rows in the synopsis table.

Figure 5 – Rows were created in synopsis table for large table

TSNMIN	TSNMAX	DIFF
0	7999	7999
8000	9023	1023
9024	10047	1023
10048	11071	1023
11072	12095	1023
12096	13119	1023
13120	14143	1023
14144	15167	1023

As shown in Figure 5, for large tables, rows were created in the synopsis table. The large synopsis tuple (as shown in row 1) is an indication that the deferred synopsis tuple creation feature is active.

This section provided details for synopsis tables and insert/update processing. Subsequent sections below provide more details on LOAD utility processing for synopsis tables. In general much of what is described here still holds true, but it is more likely that the LOAD utility may cause the synopsis table's rows to correspond to varying numbers of base table rows.

5. What Operations Impact Storage?

This section discusses the impact on BLU storage for some operations.

LOAD utility: The LOAD utility does not append to existing data pages. This may lead to page sparsity in both the base and synopsis tables. The LOAD utility also has not been enhanced to use the page-based string compression enhancements (discussed in the Best Practices for Compression paper) introduced in Db2 11.5.4 so compression may not be optimal for string data types. Also, no matter how many rows are loaded at a time, the LOAD utility must write at least one synopsis tuple. This becomes extremely problematic when the LOAD utility is used to load small amounts of data.

Insert: The insert operation writes new data to a table across all its MLNs. Within each MLN, the insert operation may potentially be parallelized across multiple insert ranges at the discretion of the Db2 optimizer. Within each insert range, we typically write data into extents for each column group as described in previous sections. Of particular note is the Db2 11.5.6 enhancement discussed in section 1 above optimizes trickle inserts and stores data in a different more optimal format for small inserts. When that enhancement is enabled, once we get enough data in the more compact format where pages may contain multiple column groups, we split the data into our traditional column group format where each column group resides in a separate extent.

Delete: When deleting a row in a BLU table, the row is marked deleted. The space will not be reclaimed until all tuples in a column group's extent are deleted. This means that deleted rows will still occupy storage in many cases.

Update: The update operation is decomposed and internally works in 2 steps: first delete then insert. It is therefore subject to the same storage behavior as Delete and Insert.

Truncation: Running the truncate command with the "drop storage immediate" option will release all the pages used by the column objects. There is also a flavor of truncate (using the "without immediate" option) that does not truncate storage until commit processing.

Reorg Table Reclaim Extents: The REORG TABLE RECLAIM EXTENTS command can reclaim full extents of space that have been fully freed up due to deletion or update activity. This can be manually or automatically run. When run on a base table, we reclaim its freed extents. We then mark rows in the corresponding synopsis table as deleted accordingly. REORG TABLE RECLAIM EXTENTS also attempts to reclaim extents for the synopsis table in a similar way.

Alter Tablespace Reduce Max: The ALTER TABLESPACE REDUCE MAX command can be used to remove freed extents from the tablespace to reduce the physical size of the tablespace.

As discussed in this section, some operations on the tables will have implications on BLU storage. Understanding database applications and their operations against the database will shed some light on the status quo of the storage usage. Once the workload is understood, the best

practices (Section 6) can be followed to adjust the workload to minimize overhead and improve storage utilization.

6. What are the Best Practices for BLU Storage?

While harnessing the massive parallelism capability and superior performance of BLU, some steps can be taken to minimize storage overhead. This section will discuss some best practices that can help with improving BLU storage usage.

a. Enable all optional BLU storage enhancements for your release if possible

As noted in various sections above, we have introduced various BLU storage enhancements over time, particularly within the Db2 11.5 mod packs. These enhancements provide critical performance and storage features for customers, but are not enabled by default especially for on-prem installations since they may change what is stored on disk, which removes fallback support. If such fallback between mod packs is critical for you, please do not enable these enhancements by default. However, if such fallback is not required, please do enable all of these enhancements so they can be fully leveraged as soon as they are available. Refer to the Db2 Documentation for more details on such enhancements.

b. When upgrading to a new mod pack or release, consider the BLU storage features it supports

Many customers may upgrade infrequently to a new release. Please do consider what BLU storage features are supported in your target release and compare those available in subsequent releases. Many storage concerns have been addressed over time, so it is critical to make use of the available features as much as possible. As noted above, please consult the Db2 Documentation for more details on new features in each mod pack/release.

c. Minimize extent overhead (through example)

Characteristics of a potential issue: Customers have small base tables with many columns. The compression of the values is good, but the physical storage size is still large. The storage is dominated by extent overhead for each column and multiplied by the number of insert ranges multiplied by the number of MLNs. Although not limited to SAP customers, SAP workloads typically contain databases with many small tables (can be greater than 10K) that are often each smaller than 10 GB in size. They also each have many columns – 300 to 400. Synopsis tables can also consume an excessive amount of storage due to unused pages in extents.

Solutions in recommended order:

Option 1: If you are using IIAS or a Db2 Warehouse on Cloud form factor, move up to Db2 11.5.6 or a later release and rebuild any problematic tables as noted in the Compression Best Practices paper. This will fully utilize the enhancements described earlier in this paper where we are able to store multiple column groups in a single extent, thus minimizing the total number of extents and the extent overhead. This should handle all extent overhead for both the synopsis and

base table. In addition, the Deferred Synopsis Tuple Creation enhancement may further help to minimize extent overhead. If this option is selected, options 2 and 3 below should not be necessary.

Option 2: Move up to Db2 11.5.4 or a later release and enable the Deferred Synopsis Tuple Creation enhancement described above. Note that it is enabled by default for IIAS and Db2 Warehouse on Cloud form factors. Rebuild any problematic tables as noted in the Compression Best Practices paper. This will specifically address extent overhead issues only for synopsis tables. If there are extent overhead issues for base tables, option 3 below is also recommended.

Option 3: Create a separate tablespace with extent size of 2 instead of using the default size of 4 and place the small base tables in that tablespace. The performance impact of placing the small base tables in the separate tablespace is insignificant, but requires customers to manually change their database design.

d. Reclaim Empty Extents

The REORG TABLE RECLAIM EXTENTS command can be used to reclaim extents that are no longer used (either empty or all data has been deleted) and return them back to the tablespace.

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. For MPP environment, use db2_all to perform RUNSTATS on all nodes for the RECLAIMABLE_SPACE on all nodes to be reflected in catalog (e.g., `db2_all "db2 connect to bludb; db2 runstats on table TPCH.CUSTOMER"`).

```
select TABSCHEMA, TABNAME, sum(PHYSICAL_SIZE_MB) as PHYSICAL_SIZE_MB,
sum(RECLAIMABLE_SPACE_MB) as RECLAIMABLE_SPACE_MB 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/1024) AS RECLAIMABLE_SPACE_MB FROM TABLE
(SYSPROC.ADMIN_GET_TAB_INFO('TPCH','CUSTOMER') ) ) group by tabschema, tabname

TABSCHEMA        TABNAME              PHYSICAL_SIZE_MB     RECLAIMABLE_SPACE_MB
---------------- -------------------- -------------------- --------------------
TPCH.            CUSTOMER                            12054                 5218

  1 record(s) selected.


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

select TABSCHEMA, TABNAME, sum(PHYSICAL_SIZE_MB) as PHYSICAL_SIZE_MB,
sum(RECLAIMABLE_SPACE_MB) as RECLAIMABLE_SPACE_MB 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/1024) AS RECLAIMABLE_SPACE_MB FROM TABLE
(SYSPROC.ADMIN_GET_TAB_INFO('TPCH','CUSTOMER') ) ) group by tabschema, tabname

TABSCHEMA        TABNAME              PHYSICAL_SIZE_MB     RECLAIMABLE_SPACE_MB
---------------- -------------------- -------------------- --------------------
TPCH.            CUSTOMER                             6833                    0

  1 record(s) selected.
```

The following command is used to reorg a table to return freed extents to the table space. Note if automatic table maintenance is enabled, this will be done periodically and automatically behind the scenes, so no user intervention is required.

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

The following command is used to remove freed extents from the tablespace to reduce the physical size of the tablespace. Note, this command should only be used if there are no concurrent INSERTs running.

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

The ALTER TABLESPACE REDUCE MAX command is run asynchronously in the background once the command is manually issued. Its status can be monitored with the following query. While running, the tablespace state will be MOVE_IN_PROGRESS. When the asynchronous processing completes, the tablespace state is NORMAL.

```
select member, varchar(tbsp_name, 30) as tbsp_name, varchar(tbsp_state, 30) as tbsp_state
from table (mon_get_tablespace('USERSPACE1',-2)) as t order by member

MEMBER TBSP_NAME                       TBSP_STATE
------ ------------------------------- -------------------------------
     1 USERSPACE1                      NORMAL
     2 USERSPACE1                      NORMAL
     3 USERSPACE1                      NORMAL
     4 USERSPACE1                      NORMAL
     5 USERSPACE1                      NORMAL
     6 USERSPACE1                      NORMAL
     7 USERSPACE1                      NORMAL
     8 USERSPACE1                      NORMAL
     9 USERSPACE1                      NORMAL
    10 USERSPACE1                      NORMAL

  10 record(s) selected.
```

e. Avoid Using Db2 LOAD Utility to Load Data into a Table

Using the Db2 LOAD Utility to load data (especially small amounts of data) into a BLU table can result in excessive page sparsity in the synopsis table. Therefore, usage of the LOAD utility should be minimized or converted to using bulk inserts through external tables. Bulk inserts through external tables are also strategic since they often perform better than the LOAD utility, especially when parallelized across multiple streams.

This subsection uses CDC as an example to show how to avoid page sparsity caused by loading of small chunks of data.

CDC Initial Load: Based on the discussion of the LOAD operation impact on BLU storage in the previous section, reducing the frequency of invocations of load operations can reduce the storage usage. The parameter that controls the batch size on CDC initial load is fastload_refresh_commit_after_max_operations. Refer to the IBM Documentation for details:

https://www.ibm.com/docs/en/idr/11.4.0?topic=mtsp-fastload-refresh-commit-after-max-operations

CDC Incremental Update: Incremental updates consist of delete and insert operations. As discussed in the previous section, delete operations can cause sparsity in the data pages. To minimize the sparsity caused by incremental updates, a technique called External table mirror bulk apply can be used.
More information can be found in the IBM Documentation:
https://www.ibm.com/docs/en/idr/11.4.0?topic=modes-external-table-mirror-bulk-apply

Reference

BLU Compression Best Practices paper

IIAS: https://ibm.biz/BdqP2U
Other form factors: https://ibm.biz/BdqPGw