



***WDUG : Db2 Virtual Tech Conference
September 2020***

Db2 BLU: Maximizing the Potential of Data



Sanoop Nambiar

IBM Data And AI
Db2 Warehouse Storage Team
Email: snambiar@us.ibm.com

© 2020 IBM Corporation

Agenda

- Columnar Overview
- New Storage Enhancements for 11.5.4
- Compression Best Practices



Please note :

- IBM's statements regarding its plans, directions, and intent are subject to change or withdrawal without notice and at IBM's sole discretion. Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision.
- The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract.
- The development, release, and timing of any future features or functionality described for our products remains at our sole discretion.
- Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon many factors, including considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results similar to those stated here.



Notices and disclaimers

- © 2020 International Business Machines Corporation. No part of this document may be reproduced or transmitted in any form without written permission from IBM.
- U.S. Government Users Restricted Rights — use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM.
- Information in these presentations (including information relating to products that have not yet been announced by IBM) has been reviewed for accuracy as of the date of initial publication and could include unintentional technical or typographical errors. IBM shall have no responsibility to update this information. This document is distributed "as is" without any warranty, either express or implied. In no event, shall IBM be liable for any damage arising from the use of this information, including but not limited to, loss of data, business interruption, loss of profit or loss of opportunity. IBM products and services are warranted per the terms and conditions of the agreements under which they are provided.
- IBM products are manufactured from new parts or new and used parts. In some cases, a product may not be new and may have been previously installed. Regardless, our warranty terms apply.
- Any statements regarding IBM's future direction, intent or product plans are subject to change or withdrawal without notice.
- Performance data contained herein was generally obtained in a controlled, isolated environments. Customer examples are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual performance, cost, savings or other results in other operating environments may vary.
- References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business.
- Workshops, sessions and associated materials may have been prepared by independent session speakers, and do not necessarily reflect the views of IBM. All materials and discussions are provided for informational purposes only, and are neither intended to, nor shall constitute legal or other guidance or advice to any individual participant or their specific situation.
- It is the customer's responsibility to insure its own compliance with legal requirements and to obtain advice of competent legal counsel as to the identification and interpretation of any relevant laws and regulatory requirements that may affect the customer's business and any actions the customer may need to take to comply with such laws. IBM does not provide legal advice or represent or warrant that its services or products will ensure that the customer follows any law.



Columnar Overview

Db2 BLU Core Concepts

• Dynamic In-Memory

In-memory columnar processing with dynamic movement of data from storage data



• Actionable Compression

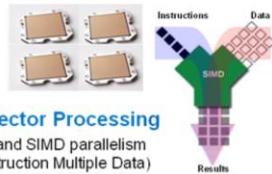
Patented compression technique that preserves order so that the data can be used without decompressing



Encoded

• Parallel Vector Processing

Multi-core and SIMD parallelism (Single Instruction Multiple Data)



• Data Skipping

Skips unnecessary processing of irrelevant data



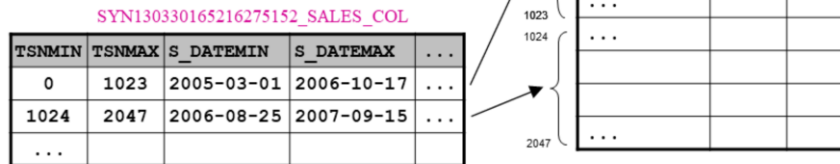
Easy to use – load and go!!

Improvements to all of I/O, RAM, and CPU. Maximizes cache and cacheline efficiency.

Only access required columns since each extent = 1 column.

Synopsis Table

- Meta-data that describes which ranges of values exist in which parts of the user table
- Enables Db2 to skip portions of a table during query processing
- Benefits from data clustering



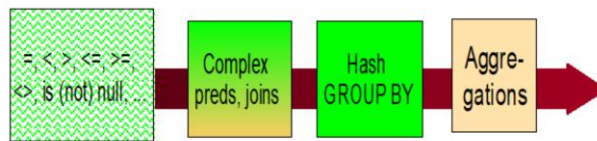
Predicate WHERE S_DATE = 2007-01-01 would skip first range.

Predicate WHERE S_DATE = 2006-09-12 would scan both ranges.

Each base table column has a synopsis min/max column.

BLU Query Processing Order

- Synopsis scan to skip tuples
- Predicates on compressed data
- Join and group-by
- Apply after decoding
 - Complex expressions, arithmetic, aggregations



Late materialization means we decode data only for operations that can not be executed on encoded values after other processing complete.

Dictionary-Based Compression

- Frequency compression: Most common values use fewest bits
- Multiple compression techniques:
 - Approximate Huffman-encoding
 - Prefix compression
 - Offset compression

0 = California 1 = New York	} 2 High Frequency States (1 bit covers 2 entries)
000 = Arizona 001 = Colorado 010 = Illinois 011 = Kentucky ... 111 = Washington	
000000 = Alabama 000001 = Alaska ...	} 8 Medium Frequency States (3 bits cover 8 entries)
	} 40 Low Frequency States (6 bits cover 64 entries)

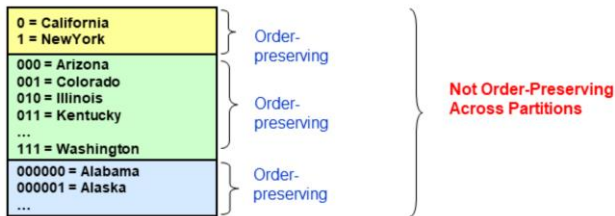
Exploiting skew in data distribution improves compression ratio
Very effective since all values in a column have the same data type
Maps entire values to compressed codes

Dictionary-Based Compression

- Prefix compression
 - Similar to approximate Huffman-encoding for common prefixes
 - Prefix bits for encoded prefixes concatenated with uncompressed suffix bits
 - Example values: MAR01, MAR02, JUN10, JUN15, etc.
 - 4 prefix bits, 16 suffix bits
- Offset compression
 - Dictionary includes base value and number of bits that define range of coverage
 - Example: Base value 0 and 10 offset bits (0-1023)
 - May include extended partition to provide some prediction for future values



Actionable Compression

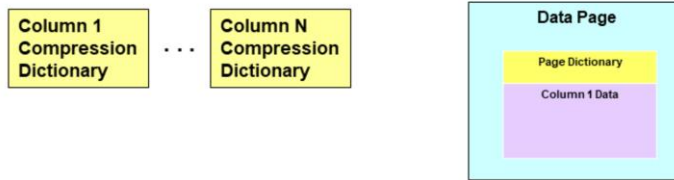


- **Actionable compression allows the following actions to be performed on compressed data**
 - Predicate evaluation (=, <, >, >=, <=, Between, LIKE)
 - Group by and Join processing on encoded data use global and On-The-Fly (OTF) encoding
- **Order-preserving encoding allows range predicates to be evaluated on compressed data**
 - Order is preserved within a dictionary partition but not across partitions
- **Avoiding decompression during predicate evaluation provides significant query performance gains**

Dictionary Encoding Support by Data Type

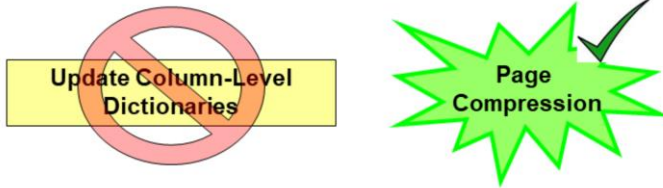
Column Data Type	Pure Dictionary Coding	Minus Coding	Prefix Coding
Integer (2, 4, 8 bytes)	Yes	Yes	Yes
Decimal	Yes	Yes	Yes
Date/Time, Timestamp (precision <=6)	Yes	Yes	Yes
Double, Real	Yes	No	Yes
CHAR, VARCHAR, GRAPHIC, VARGRAPHIC	Yes	No	Yes
BINARY, VARBINARY	No	No	No
LOB and all variants (including inlined LOBs)	No	No	No

Compression Dictionaries for Column-Organized Tables



- Column-level dictionaries: **Always one per column**
 - ▶ Dictionary created during load replace, load insert, SQL insert/update
- Page-level dictionaries: **May also be created during load or insert**
 - ▶ Used if space savings outweighs cost of storing page-level dictionaries
 - ▶ Exploit local data clustering at page level

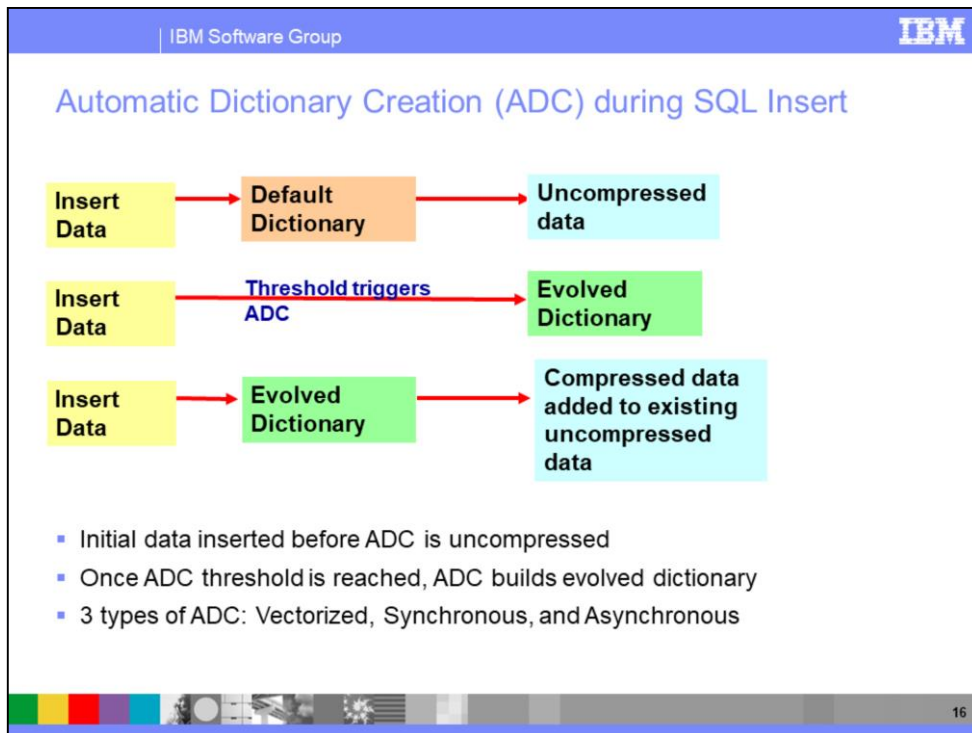
Column-Level Dictionaries are Static



- Once created, evolved column-level dictionaries are static
- Compression ratio may deteriorate if newly-inserted values are not covered by the column-level dictionary
 - Page compression can reduce need to rebuild column dictionaries
 - New offset encodable values not covered by column-level dictionaries can still be compressed by page-level dictionaries

Offset encodable = numeric datatypes such as integer, date, etc.

NOTE while compression ratio is important, column-level dictionary coverage is critical to maximize query performance.



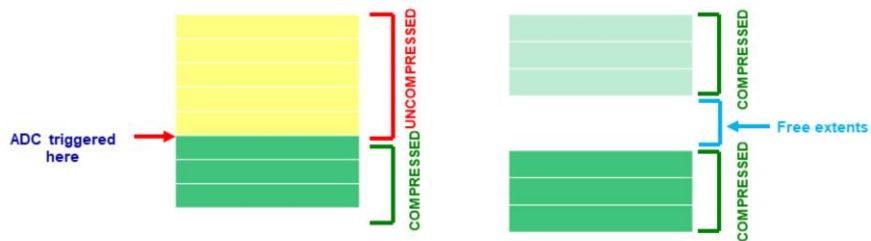
Per-MLN ADC threshold = 500K rows for one MLN, 1 million rows/# MLNs for multiple MLNs

Vectorized ADC method designed for bulk insert.

Synchronous ADC is used for non-bulk insert cases where we have a Z-lock on the table or DGTs. Examples may include CTAS, NLI tables, and inserting into an uncommitted table.

Asynchronous ADC is used in all cases where other ADC types are not available and builds column-level dictionaries in a background transaction.

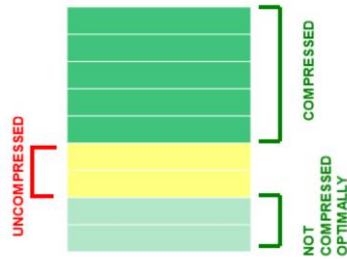
Automatic REORG Recompress



- ADC threshold is set higher to build a better dictionary
- Large number of values at the front of the table left uncompressed
- REORG Recompress automatically uses evolved dictionary to recompress committed data previously encoded using default dictionary
- Frees full extents, but does not deallocate them

Freed extents may be reused during future inserts to the same table.

Future: REORG Recompress Enhancement



- If existing feature or compression schemes are not effective, then column might have mix of well compressed, not optimally compressed as well as uncompressed data.
- In future, if we find that a new feature is beneficial to column in anyway then rewrite part or all of column data using the new feature.
- Also this feature will be fully automatic.

Freed extents may be reused during future inserts to the same table.

New Storage Enhancements for 11.5.4



High Cardinality String Data Types Not Dictionary Encoded

States VARCHAR

001 = Colorado
001 = Colorado
010 = Kentucky
001 = Colorado
011 = Illinois

States have high frequency so
are encoded with dictionary

Product Description VARCHAR

Blue dress with unicorns girls size 6X
Red dress with hearts girls size 6X
Red dress with bears girls size 6X
Blue uniform shirt boys size 5
Red uniform shirt boys size 5

Free flowing text stored unencoded

- Frequency-based compression not effective for high cardinality string datasets so percentage of values encoded < 10%
- String data dominates storage cost

Page-Based String Compression Type 1

- New page-based compression algorithm instead of dictionary-based compression algorithm
- Looks for repeating patterns within string data being inserted
- Significantly improves compression of string data types, especially for columns with high cardinality that have many unique values
- Handles long common prefixes well

Frequency-based compression (even with our existing prefix compression) unable to provide coverage for some datasets.

- Geospatial data
- URLs
- Comments
- Etc.

String datatypes = CHAR, VARCHAR, GRAPHIC, VARGRAPHIC, BINARY, and VARBINARY

Enormous space savings possible for many such datasets!

Page-Based String Compression Type 1 with Industry Benchmark TPC-H - Test Description

- **System**

- Full rack IIAS

- **Workload and Data**

- Industry Benchmark: TPC-H
- Data:
 - Volume: 1 TB dataset.
 - Text Percentage: 55.8% (based on table definition).
- Workload: run in sequential mode and concurrent mode.
- Pre-workload Executions:
 - Pre-build dictionaries per best practices. Load(from ET) -> Runstats.

- **Execution**

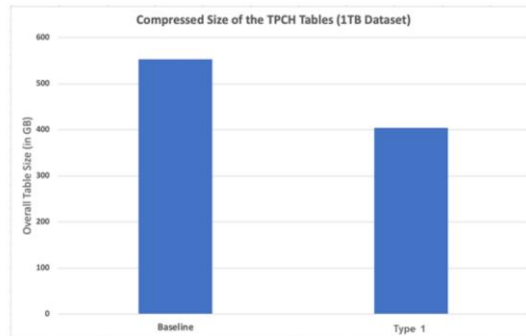
- Sequential mode: 1 client. 1 query at a time.
- Concurrent mode: 10 clients. each runs one query at a time.
- In both execution modes, we intend to isolate the page-based compression type 1 impact in runtime performance and focus on comparing the performance with the same query plans.

Concurrent Mode:

10 clients running concurrently executing the queries in sequential order. To avoid plan reuse different predicate values are used for the queries in each client.

The order of the queries were pre-generated (randomly) and stay unchanged for all iterations.

Compression Results with TPCB



- Overall compressed size of TPCB tables was reduced by 27% with Page based string compression Type 1.

More compression Improvements for High Cardinality String Data Types

Product Description VARCHAR

Blue dress with unicorns girls size 6X
Red dress with hearts girls size 6X
Red dress with bears girls size 6X
Blue uniform shirt boys size 5
Red uniform shirt boys size 5

Page-based string compression
Type 1 compresses well for long
repeating data patterns

Postal Codes VARCHAR

95119
95120
95236
95237
95238

Compression can be improved
even more for short repeating
data patterns

- Compression can have additional improvements for numeric data stored in string data types

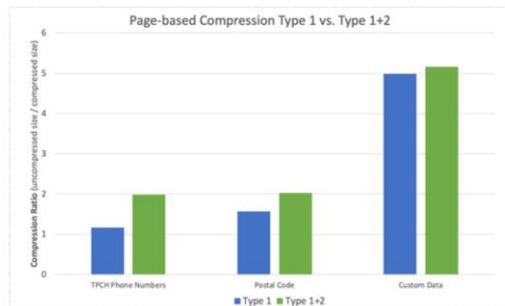
Page-Based String Compression Type 1 requires 4+ byte patterns to compress.

Page-Based String Compression Type 2

- Page-Based String Compression Type 2 very effective for compressing high cardinality hex, date, timestamp, numeric data stored in string data types
- Must have ≤ 16 distinct characters per compressed data page
- Sample data
 - HEX: 59721B038CB9AC5A5DD09055529A64CA4D000000
 - Postal-codes: 95133-7670
 - Telephone Numbers: 1-408-775-6978
 - Date:12/21/2003 (or other date formats)
 - Time:11:32:02 (or other time formats)
 - CDR (Call Detail Record for telecommunications):00650068D34B41799911903603
- Db2 automatically determines which compression scheme to use



Compression Ratio with Page-based String Compression (Type 1 and 2)



- With Type 2, compression ratio for the phone numbers (stored as strings) of the TPCH benchmark was improved from 1.17 (Type 1) to 1.98.
- A custom workload also shows compression ratio for postal codes improves significantly with Type 2.
- Another custom workload shows compression ratio went from 4.99 (Type 1) to 5.17 (Type 1+2). It indicates that Type 2 can potentially increase the overall compression ratio for data that has been compressed considerably well by Type 1 already.

Page-Based String Compression Type 1 and 2 Use Cases

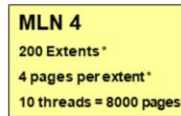
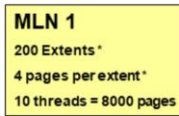
- During bulk inserts and bulk updates, compression can be greatly improved for the following data types:
 - ▶ CHAR/VARCHAR
 - ▶ GRAPHIC/VARGRAPHIC
 - ▶ CHAR FOR BIT DATA/VARCHAR FOR BIT DATA
 - ▶ BINARY/VARBINARY



Space Usage Overhead for Synopsis Tables on Small Tables



Default tablespace extent size
4
with only 1 page out of 4 used

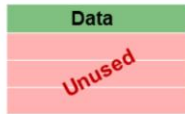


Synopsis Table with 200 columns on Base Table with 100 columns using default of 4 pages per extent and Parallel Degree 10 on MPP system with 4 MLNs = 32000 allocated pages with only 8000 used pages

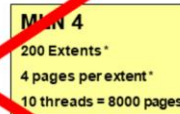
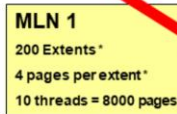
- Synopsis tables can consume significantly more storage than small base tables due to unused pages in extents
- Default of 4 pages per extent * number of parallel insert threads * # of MLNs
- For each column in base table, synopsis table generates 2 columns
- Using synopsis table adds unnecessary performance overhead
- Small tables can be scanned quickly without synopsis table filtering

Space usage overhead may quickly become significant depending on number of concurrent streams used to populate a table.

Deferred Synopsis Tuple Creation for Small Base Tables



Default tablespace extent size
4
with only 1 page out of 4 used



Synopsis Table with 200 columns using default of 4
pages per extent and Parallel Degree 10 on MPP system
with 4 MLNs = 32000 allocated pages with only 8000 used
pages

- Defer creation of synopsis tuples until certain table size exceeded
- Avoids creating unnecessary synopsis tuples for small base tables
- Small tables don't need synopsis table for query performance

Enablement of New Columnar (BLU) Storage/Compression Features

- These new features have either direct or indirect on-disk impact.
- Enabled by default for Db2 Warehouse and IBM Integrated Analytics Appliance (ie. all "Warehouse" offerings)
- Disabled by default for traditional Db2 install
 - ▶ If enabled, fallback to earlier levels such as 11.5 GA is blocked
 - ▶ New features along with registry variables (and usage) for each one are documented in Knowledge Center

The following registry variable settings are used on-prem to enable the new 11.5.4 features discussed here. NOTE that Db2 must be restarted after these registry variables are set for them to take effect.

To enable Page-Based String Compression Types 1 and 2:

```
db2set DB2_COL_STRING_COMPRESSION="UNENCODED_STRING:ON"
```

To enable Deferred Synopsis Tuple Creation:

```
DB2_COL_SYNOPSIS_SETTINGS="DEFER_FIRST_SYNOPSIS_TUPLE:ON"
```

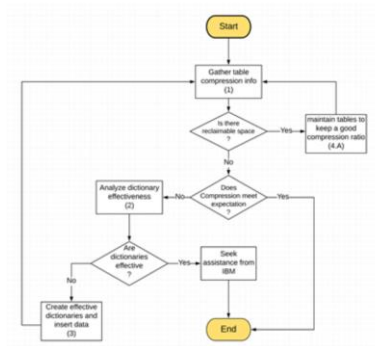
Compression Best Practices



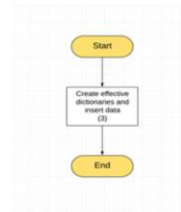
Best Practices Paper

- The paper can be downloaded from:
 - ▶ IIAS: <https://ibm.biz/BdqP2U>
 - ▶ Other platforms: <https://ibm.biz/BdqPGw>

A. For existing tables compression. Numbers corresponding to section numbers in the paper



B. For database migrations from Netezza to IIAS.



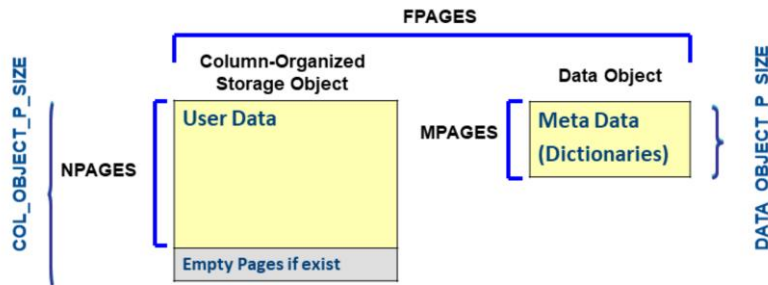
Section 1 - Gather information to determine how well tables are compressed (Measuring Compression 1)

- Statistics for Measuring Number of Pages in SYSCAT.TABLES
 - › **NPAGES:** Number of pages in Column-Organized Object minus any empty pages
 - › **MPAGES:** (M for meta data) Number of pages in Data Object
 - › **FPAGES:** Total number of pages in both objects
- ADMIN_GET_TAB_INFO table function reports
 - › **COL_OBJECT_P_SIZE:** Physical size (KB) of column data object containing **user data**
 - › **DATA_OBJECT_P_SIZE:** Physical size (KB) of data object containing **meta data**

COL_OBJECT_P_SIZE : The size returned takes into account full extents allocated for the table and includes the EMP extents for objects created in DMS table spaces. This size represents the physical size of the base columnar data only. Space consumed by LOB data, Long Data, Indexes and XML objects are reported by other columns.

DATA_OBJECT_P_SIZE : For column-organized tables, the data object physical size is the amount of disk space that is physically allocated for the table metadata only, which is relatively small.

Section 1 - Gather information to determine how well tables are compressed (Measuring Compression 2)



COL_OBJECT_P_SIZE : The size returned takes into account full extents allocated for the table and includes the EMP extents for objects created in DMS table spaces. This size represents the physical size of the base columnar data only. Space consumed by LOB data, Long Data, Indexes and XML objects are reported by other columns.

DATA_OBJECT_P_SIZE : For column-organized tables, the data object physical size is the amount of disk space that is physically allocated for the table metadata only, which is relatively small.

Section 1 - Gather information to determine how well tables are compressed (Calculating Column-Organized Storage Sizes)

User Data	COL_OBJECT_P_SIZE
User Data + Meta Data + Page Map/Unique Indexes	COL_OBJECT_P_SIZE + DATA_OBJECT_P_SIZE + INDEX_OBJECT_P_SIZE + LOB_OBJECT_P_SIZE

- Be careful using NPAGES to determine table size
 - May underestimate actual space usage especially for small tables
 - Doesn't take meta data or empty pages into account
- Use the table function ADMIN_GET_TAB_INFO or admin view ADMINTABINFO to retrieve
 - COL_OBJECT_P_SIZE + DATA_OBJECT_P_SIZE + INDEX_OBJECT_P_SIZE + LOB_OBJECT_P_SIZE

You may want to run RUNSTATS first to ensure statistics are up to date.

Here is a sample query to compute storage sizes for all tables within a given tablespace (substitute the appropriate tablespace name for BLYLE):

```
SELECT CAST(TABSCHEMA as char(16)) TABSCHEMA,
       CAST(TABNAME as char(20)) TABNAME,
       DBPARTITIONNUM,
       (DATA_OBJECT_P_SIZE+COL_OBJECT_P_SIZE+INDEX_OBJECT_P_SIZE)/1024 AS PHYSICAL_SIZE_MB,
       RECLAIMABLE_SPACE AS RECLAIMABLE_SPACE
FROM TABLE (SYSPROC.ADMIN_GET_TAB_INFO('BLYLE','T1')) ;
```

Section 1 - Gather information to determine how well tables are compressed (Table Compression Statistics in SYSCAT.TABLES)

Row-Organized Table Statistics	Column-Organized Table Statistics
PCTPAGESSAVED	PCTPAGESSAVED
AVGCOMPRESSEDROWSIZE	
AVGROWCOMPRESSIONRATIO	
AVGROWSIZE	
PCTROWCOMPRESSED	

- Only PCTPAGESSAVED applies to column-organized tables too
 - Approximate percentage of pages saved in the table
 - runstats collects PCTPAGESSAVED by estimating the number of data pages needed to store table in uncompressed row orientation
- ADMIN_GET_COMPRESS_INFO not supported yet for column-organized tables and will return zero rows

For column-organized tables, the following compression statistics will all have the value of -1 for Not Applicable:

- AVGCOMPRESSEDROWSIZE
- AVGROWCOMPRESSIONRATIO
- AVGROWSIZE
- PCTROWCOMPRESSED

Section 1 - Gather information to determine how well tables are compressed (Estimating Compression Ratios)

- PCTPAGESSAVED can be converted to a compression ratio
 - ▶ See Notes for sample query

$$\begin{aligned}\text{Compression Ratio} &= \text{Uncompressed Size} / \text{Compressed Size} \\ &= 1 / (1 - \text{PCTPAGESSAVED} / 100)\end{aligned}$$


Here is a sample query to compute the compression ratio for a specific 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 = 'BLYLE' AND TABNAME = 'T1' with UR;
```


IBM Software Group

IBM

Section 2. How effective are the table-level column dictionaries (PCTENCODED Statistic in SYSCAT.COLUMNS)





C1	PCTENCODED = 90
C2	PCTENCODED = 75
C3	PCTENCODED = 100



C1	PCTENCODED = 0
C2	PCTENCODED = 10
C3	PCTENCODED = 0

- Monitor this statistic to determine how many values were left uncompressed in specific columns
- Percentage of values encoded (compressed) by column-level dictionary
- It measures number of values compressed NOT compression ratio

38

Each column could have a different encoded percentage.

WARNING: PCTENCODED does not include page compression since it is used in heuristic decisions for performing join or group by on either encoded or unencoded data. PCTENCODED = 0 could actually have many values compressed at page level.

Would like to have better statistics in future such as PCTCOMPRESSED per column.

If you suspect a problem with the compression ratio after the initial load, check PCTENCODED to make sure the columns were compressed. If you see many columns where PCTENCODED = 0, then something could have gone wrong with the load such as the util_heap_sz value set too small and the AUTOMATIC option wasn't specified. PCTENCODED = 0 means that the values were not encoded using the table-level dictionary but they still could have been compressed using the page-level dictionaries for minus coded values.

Here is a sample query to find the PCTENCODED values for the columns of a specific table:

```
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.

Section 2. How effective are the table-level column dictionaries (PCTENCODED for string data types)

- PCTENCODED does not reflect page level encoding or correlate with page level compression.
- Low PCTENCODED does not necessarily mean poor compression for columns of string data types (CHAR, VARCHAR, GRAPHIC, VARGRAPHIC, BINARY, VARBINARY, CHAR FOR BIT DATA, VARCHAR FOR BIT DATA)



Section 3. How to rebuild table, create effective table-level dictionaries, and insert data (main steps)

- Create an identical table
- Create a cursor from the source table using Bernoulli sampling
 - Sampling rate should be specified to get approximately 5 million rows
 - Note the higher the better
- Use the sampling cursor with LOAD to pre-build the compression dictionary
- Insert the data to the target table with selecting from the source table
- If everything looks good, drop the source table and rename the target table to source.

40

A. Create identical table.

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

B. Create a sampling cursor from the source table.

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

C. Use the sampling cursor with LOAD to pre-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....
```

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

Section 4. How to best maintain the column-organized tables to keep a good compression ratio (Space Reclamation)

- When automatic REORG Recompress frees extents, REORG TABLE...RECLAIM EXTENTS may return pages in freed extents to tablespace storage
- These reclaimed pages may be reused by any tables in same tablespace
- auto_reorg database configuration parameter controls if auto reclamation takes place
 - ▶ Set to ON if DB2_WORKLOAD=ANALYTICS

To manually check for reclaimable space (update schema value as appropriate):

```
SELECT substr(tabname,1,10) as tabname , substr(tabschema,1,10) as  
tabschema, reclaimable_space as reclaimable_space_KB FROM  
SYSIBMADM.ADMINTABINFO WHERE tabschema = 'CDREXELI' order by  
tabname WITH UR;
```

If you locate tables with high amounts of reclaimable space, you may manually run REORG RECLAIM EXTENTS manually. Here is a sample command:

```
REORG TABLE <tabname> RECLAIM EXTENTS;
```

Once this is complete, you may rerun RUNSTATS to verify that statistics are up to date.

Section 4. How to best maintain the column-organized tables to keep a good compression ratio (Reducing Unused Space in a Tablespace)

- Once extents are reclaimed, they are available for reuse within the same tablespace
- However, this unused space may also be released for other consumers
 - A sample query to detect unused space is provided in the Notes
 - To release all unused space and lower the high water mark:
ALTER TABLESPACE <TBSPACE NAME> REDUCE MAX;

Here is a sample query to examine your tablespaces for unused space:

```
select substr(TBSP_NAME,1,14) as TS_NAME , TBSP_TOTAL_PAGES as
Total_pages, TBSP_USED_PAGES as Used_pages , TBSP_PAGE_TOP as
High_water_MARK , reclaimable_space_enabled from table (
MON_GET_TABLESPACE ( NULL , -1 ) ) as ts where TBSP_NAME NOT LIKE
('SYS%') and TBSP_TYPE = 'DMS' ;
```

When considering reducing tablespace usage:

- Look for tablespaces with total pages and/or HWM that is much higher than the number of used pages
- Look for tablespaces with reclaimable space attribute of '1'

REDUCE MAX command is asynchronous, user needs to wait it completes to do a RUNSTATS

The max should be only used if no concurrent INSERTs are running

The MON_GET_EXTENT_MOVEMENT_STATUS table function can be used to return the status of the extent movement operation



WDUG : Db2 Virtual Tech Conference
September 2020

Thank You!!

Speaker: Sanoop Nambiar
Company: IBM
Email Address: snambia@us.ibm.com



© 2020 IBM Corporation