

DB2 for z/OS Data and Index Compression

Session # 11958
Wednesday, August 8, 2012: 11:00 AM - 12:15 PM

Willie Favero
Dynamic Warehouse on z/OS Swat Team (DB2 SME)
IBM Silicon Valley Laboratory
713-940-1132
wfavero@attglobal.net



 my TWITTER
#db2zos

Disclaimer



© Copyright IBM Corporation [current year]. All rights reserved.
U.S. Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY. WHILE EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS PRESENTATION, IT IS PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. IN ADDITION, THIS INFORMATION IS BASED ON IBM’S CURRENT PRODUCT PLANS AND STRATEGY, WHICH ARE SUBJECT TO CHANGE BY IBM WITHOUT NOTICE. IBM SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS PRESENTATION OR ANY OTHER DOCUMENTATION. NOTHING CONTAINED IN THIS PRESENTATION IS INTENDED TO, NOR SHALL HAVE THE EFFECT OF, CREATING ANY WARRANTIES OR REPRESENTATIONS FROM IBM (OR ITS SUPPLIERS OR LICENSORS), OR ALTERING THE TERMS AND CONDITIONS OF ANY AGREEMENT OR LICENSE GOVERNING THE USE OF IBM PRODUCTS AND/OR SOFTWARE.

IBM, the IBM logo, ibm.com, System z, Lotus, AIX, AS/400, DATABASE 2, DB2, e-business logo, Enterprise Storage Server, ESCON, FICON, OS/390, OS/400, ES/9000, MVS/ESA, Netfinity, RISC, RISC SYSTEM/6000, iSeries, pSeries, xSeries, SYSTEM/390, IBM, NOTES, WebSphere, z/Architecture, z/OS, zSeries, are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at “Copyright and trademark information” at www.ibm.com/legal/copytrade.shtml

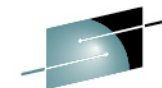
Other company, product, or service names may be trademarks or service marks of others.



Objectives

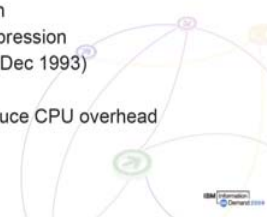
- Describe compression fundamentals
- Explain how DB2 implements data compression
- Describe how a dictionary is created and how data compression uses it
- Describe how DB2 implements index compression
- Determine if using data and/or index compression accomplishes the disk savings you were anticipating.

The Basics



The Setup

- First compression:
 - EDITPROC
 - High CPU overhead
 - HUFFMAN sample (DSN8HUFF in hlq.SDSNSAMP)
- Hardware compression
 - DB2 hardware compression
 - ESA Compression (Dec 1993)
 - DB2 Version 3
 - Hardware helps reduce CPU overhead
- CMPSC instruction

A network diagram with several nodes connected by lines. The nodes are represented by circles with numbers inside. One node is highlighted with a green circle and the number 2. There is also a small IBM logo in the bottom right corner of the diagram area.

The Setup



→ First compression:

– EDITPROC

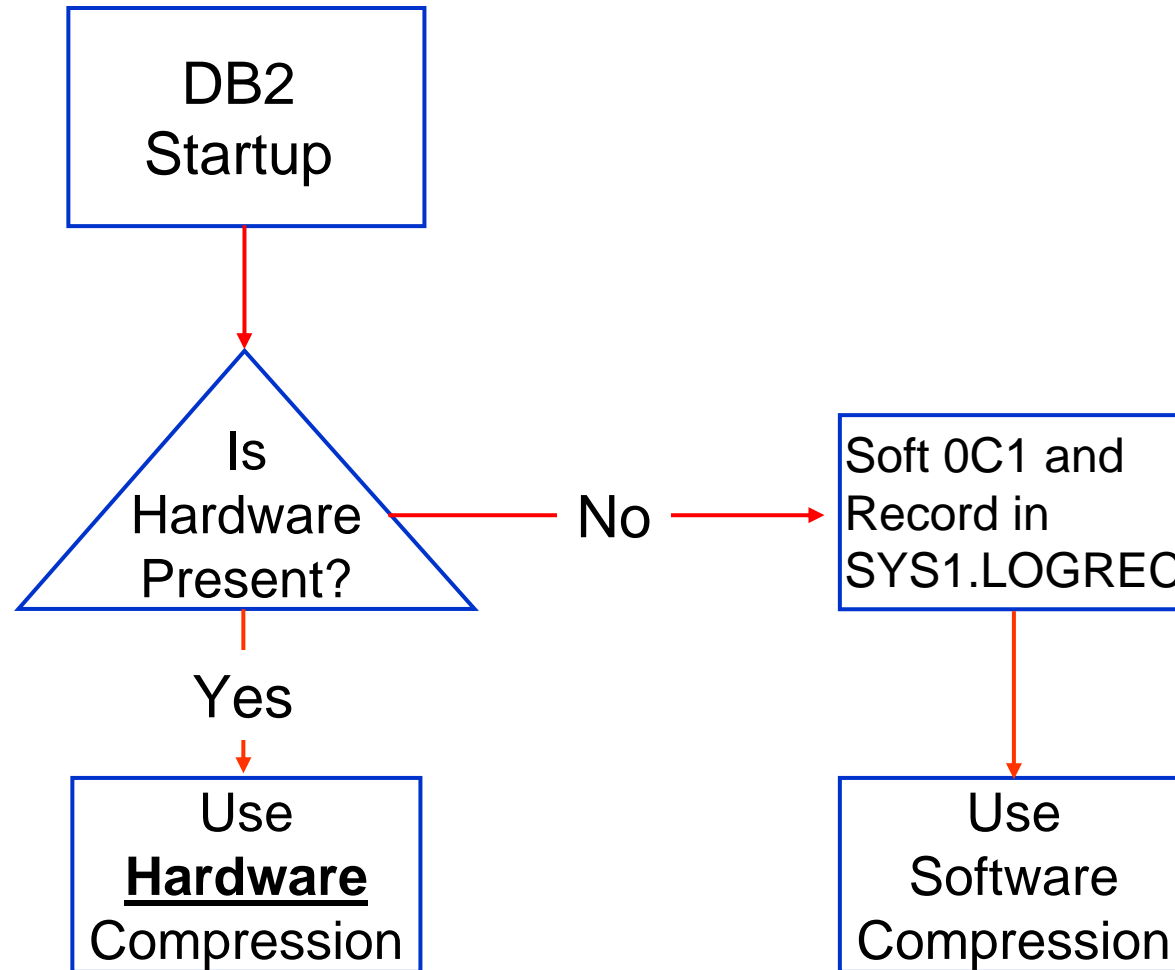
- High CPU overhead
- HUFFMAN sample (DSN8HUFF in hlq.SDSNSAMP)

→ Hardware compression

- DB2 hardware compression
- ESA Compression (Dec 1993)
- DB2 Version 3
- Hardware helps reduce CPU overhead

→ CMPSC instruction

Just Some Interesting History



Compression automatically comes with hardware today.



The Basics

- In 1977 two information theorists, Abraham Lempel and Jacob Ziv, developed lossless data compression techniques
 - LZ77 (LZ1) and LZ78 (LZ2)
 - still very popular and widely used today
 - LZ stands for Lempel-Ziv (some believe it should be Ziv-Lempel)
 - 77 & 78 - years their lossless compression algorithm was developed and improved
- LZ77 is an adaptive dictionary based compression algorithm that works off a window of data using the data just read to compress the next data in the buffer.
- LZ78 variation is based on all of the data available rather than just a limited amount
- LZW (Lempel-Ziv-Welch) variation was created to improve the speed of implementation, not usually considered optimal

The Basics



- "lossless compression" – expanding compressed data gives you the exact same thing you started with
- "lossy compression" loses some information every time you compress it
 - JPG (JPEG) is a form of lossy compression
- Are you familiar with the Lempel-Ziv algorithm?
 - GIF
 - TIFF
 - PDF (Adobe Acrobat)
 - ARC, PKZIP, COMPRESS and COMPACT on the UNIX platform
 - Stuffit for the Mac folks
 - all use the Lempel-Ziv algorithm and some form of LZ compression

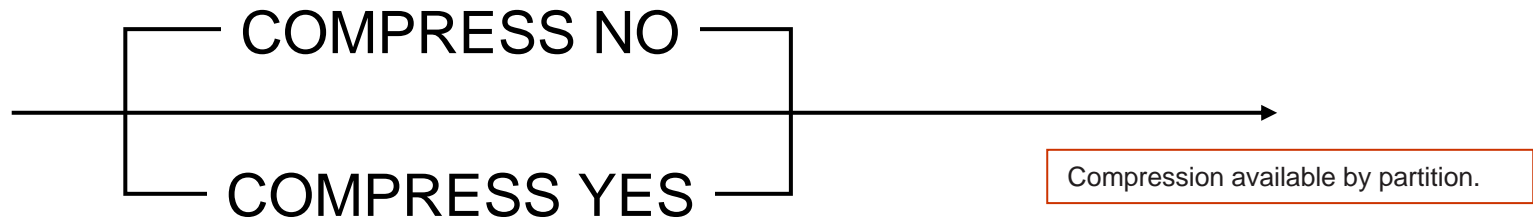
Table Space Compression



Setting Compression On

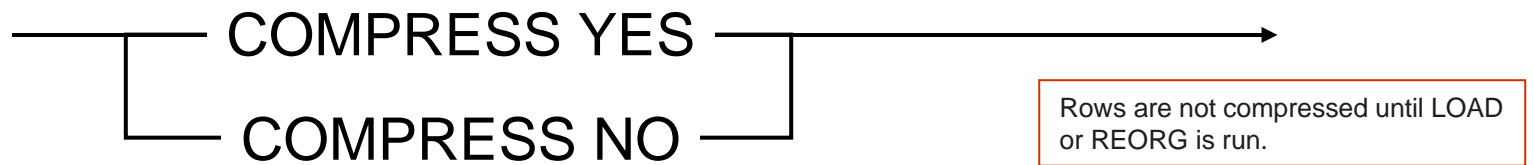
CREATE ... TABLESPACE *tablespace_name*

.....



ALTER ... TABLESPACE *tablespace_name*

.....



The Dictionary



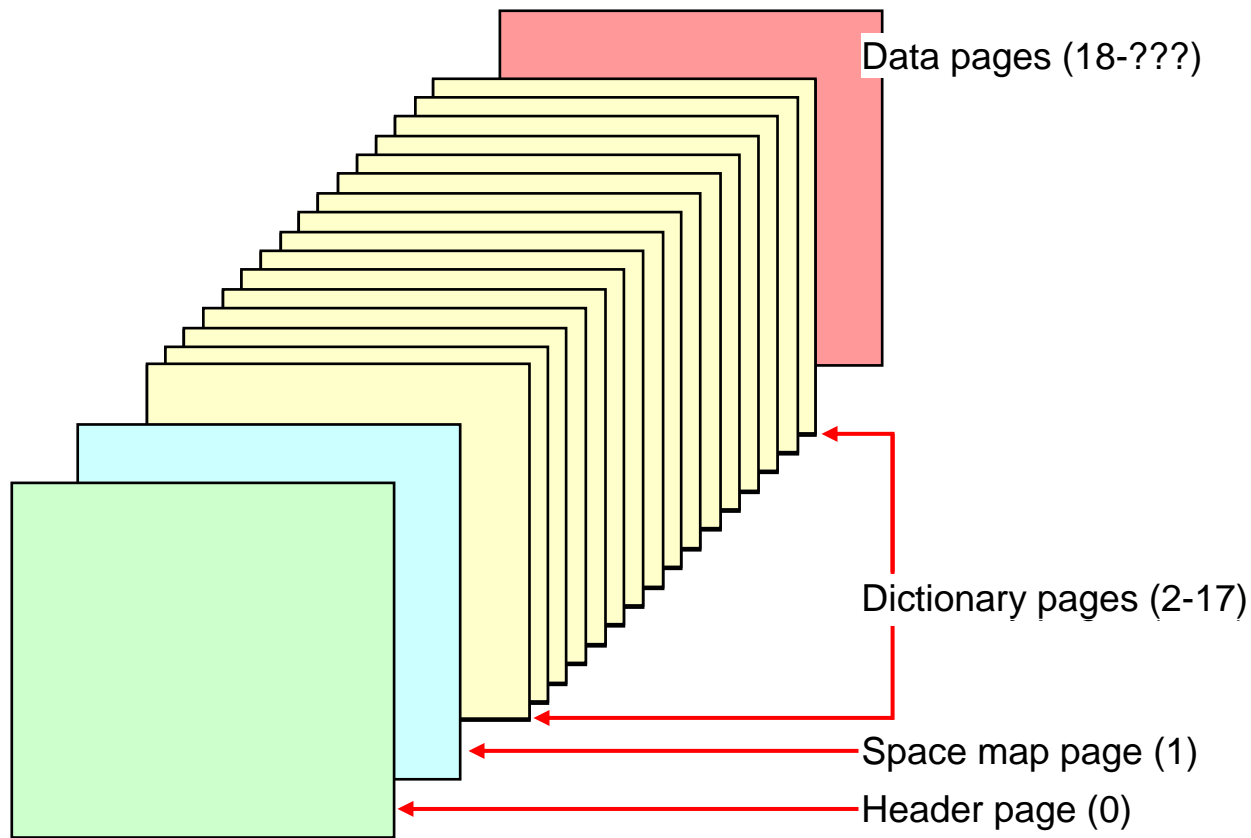
- The dictionary is created by the LOAD and/or REORG utilities only
 - It occupies:
 - 4K - 16 pages
 - 8K - 8 pages
 - 16K - 4 pages
 - 32K - 2 pages
 - The compression dictionary follows the header and first space map pages
 - Dictionaries can be at the partition level (Careful, you could have 4096 partitions meaning 4096 dictionaries)
 - Not all rows in a table spaces can be compressed. If the row after compression is not shorter than the original uncompressed row, the row remains uncompressed.
- Compression dictionary size
- 64K (16 X 4K page) of storage in the DBM1 address space
 - Dictionary goes above the bar in DB2 Version 8 and later releases

Dictionary



- Rows are compressed on INSERT
- For an UPDATE
 - Expand row, do update, then re-compress row
 - UPDATE has the potential to be expensive
- Changes (INSERT & UPDATE) are logged in compressed format
- Larger page sizes may result in better compression.
 - Resulting rows after compression are variable length
 - You might be able to fit more rows with less wasted space in a larger page size.
- You cannot turn compression on for the catalog, directory, work files, or LOB table spaces
- Index compression does not use a dictionary

Dictionary – 4K Page Size



Dictionary – Not Always a Good Thing



DB2 V8 and above

64K

Compression
Dictionary

Is problem resolved?
Make sure virtual is
backed by 100% real



No Longer an Issue

storage constrained

Dictionary

16 pages x 4096 bytes/page = 64KB

When to Use Compression

Tables with small rows

255 row/page max

No increase in rows, no savings

4054 bytes/page available

255 15 byte rows = 3825 bytes total

3825 bytes / 80% compression = ????

Doesn't matter, still have 255 row limit

When to Use Compression

Tables with very large rows

255 row/page max

No increase in rows, no savings

4054 bytes/page available

4000 byte row at 45% compression
2200 bytes

2 rows/page: $2200 * 2 = 4400$
4400 bytes > 4054 bytes

No Increase in Rows, No Savings



- If number of rows does not increase, do not use compression
 - For short rows, no help
 - For long rows, consider moving to a larger page size 8K, 16K, or even 32K
 - Use DSN1COMP... coming up in a minute

When to Use Compression



→ Encryption

- Little gain (if any) from compressing encrypted data
 - Usually few repetitive characters

Option:

- DSN1COMP has new option EXTNDICT
 - For encryption tool to support encryption and compression combined
 - Requires PTFs UK41354 - V8 and UK41355 – V9

Logging



- ➔ UPDATES and INSERTs logged in compressed format
 - Possible results:
 - reduced logging
 - reduced log I/O
- ➔ Any active log reduction would carry over to the archive logs
- ➔ However, UPDATES need to expand and compress to complete

Possible Performance Gain



- When compression is on, data pages are brought into buffer pool in compressed state
 - Increasing the number of rows in the same size pool could increase buffer pool hit ratio
 - Increasing hit ratio could reduce I/O necessary to satisfy the same number of getpage requests.
- If compression doubles the number of rows per page
 - When DB2 loads that page in a buffer pool, it will be loading twice as many rows
- Less I/O is always a good thing



Building the Dictionary

LOAD & REORG



- The critical part of data compression is building the dictionary
- The better the dictionary reflects your data, the higher your compression rates are going to be
- There are two choices for building your dictionary
 - LOAD utility
 - REORG utility
- These two utilities are the only mechanism available to create a dictionary

LOAD Utility



- LOAD utility uses the first “x” number of rows
- There are no rows compressed while the LOAD utility is building the dictionary
- Once dictionary is created, the remaining rows being loaded will be considered for compression
- With the dictionary is in place, any rows inserted (SQL INSERT) will be compressed assuming the compressed row is shorter than the original uncompressed row

DB2 9 Enhancement



→ LOAD ... COPYDICTIONARY

- Allow priming of a partition with a compression dictionary
- Can only be used with INTO TABLE PART

→ REPLACE dictionary example:

- LOAD RESUME NO
- COPYDICTIONARY 3
- INTO TABLE PART 4 REPLACE
- INTO TABLE PART 5 REPLACE

→ Requires APARs PK63324 and PK63325

REORG Utility



- REORG utility should be your first choice
 - It builds a better dictionary
- REORG sees all of the rows because the dictionary is built during its UNLOAD phase
- REORG can create a more accurate, and therefore more efficient, dictionary than the LOAD utility
 - The more information used to create the dictionary, the better compression should be
- REORG will compress all of the rows in the table space during the RELOAD phase because the dictionary is now available
- Any row inserted after the dictionary is built will be compressed assuming the compressed row is shorter than the original row



Creating a Dictionary

→ Creating dictionary could potentially be CPU intensive

- If dictionary works for you, reuse it
 - Don't pay the expense to rebuild it

→ **KEEPDICTIONARY**

- REORG and LOAD REPLACE use this utility keyword to suppress building a new dictionary
- Not specifying **KEEPDICTIONARY** could make REORG/LOAD more costly to run and increase elapsed time

DB2 9 Caution



- ➔ REORG or LOAD REPLACE will migrate pre-V9 table spaces to reordered row format
 - Careful if using compression
 - Dictionaries automatically converted
 - Even if KEEPDICTIONARY is specified (APAR PK41154)
 - New DSNZPARM keyword available on DSN6SPRM machine - HONOR_KEEPDICTIONARY



Avoid Converting Existing Compressed Table Spaces

- PK78958 (Hiper – closed March 30, 2009)
- REORG and LOAD REPLACE will not convert an existing compressed table space to RRF when migrating to DB2 9 NFM
- Reasons are...
- Possible work-around (from Willie not IBM)
 - Turn compression OFF for the table space
 - Run REORG (or LOAD REPLACE) against the existing table space migrating it to RRF
 - Turn compression back ON for the table space
 - Rerun REORG to rebuild the dictionary and compress all of the rows in the table space.

PK80375: Compress SPT01



Now you see it,
Now you don't...
And now you see it again

Dictionary On-The-Fly

- ➔ Compression dictionary is built **WHILE** data is being inserted
 - Valid for
 - INSERT SQL statement,
 - MERGE SQL statement,
 - LOAD SHRLEVEL CHANGE utility statement
 - Dictionary is asynchronously built AFTER 1.2 MB of data has been loaded
 - RTS memory statistics used to determine threshold
 - DSNU message issued at completion to console
 - DSNU231I for non-partitioned table space
 - DSNU241I for partitioned table space
 - INSERTed rows NOT compressed until dictionary is built
- ➔ Requires DB2 10 New Function Mode (NFM)



Dictionary On-The-Fly

- REORG and LOAD REPLACE still valid for building dictionary
 - Dictionary pages follow space map
- With Dictionary on-the-fly
 - Dictionary pages randomly placed throughout page set
 - Dictionary pages may not be continuous
- REORG will arrange dictionary pages properly
- For DSN1COMP
 - Keyword REORG if dictionary built by REORG
 - Keyword LOAD (default) for LOAD and COMPRESS ON INSERT

Index Compression

DB2 Index Compression.....



- Index compression is new to DB2 9 for z/OS
- Page level compression
- Unlike data row compression:
 - Buffers contain expanded pages
 - Pages are decompressed when read from disk
 - Prefetch performs the decompression asynchronously
 - A buffer hit does not need to decompress
 - Pages are compressed by the deferred write engine
- Like data row compression:
 - An I/O bound scan will run faster
- DSN1COMP utility can be used to predict space savings

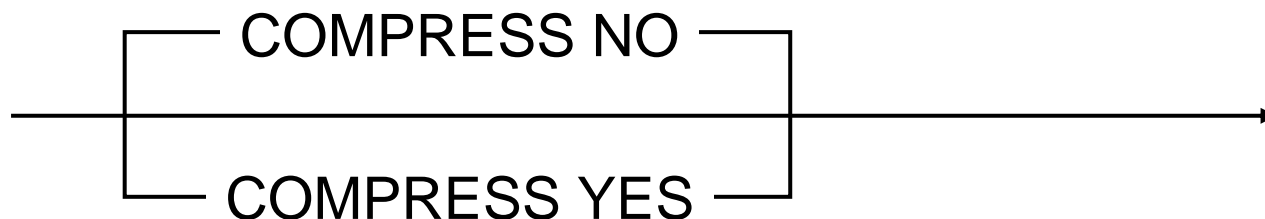
Index compression saves space, it's not for performance

CREATE/ALTER INDEX Compression

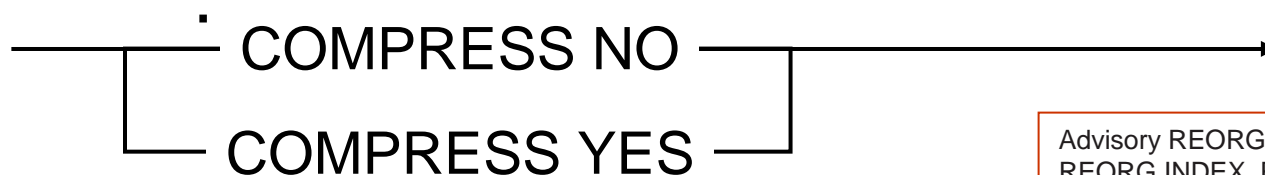


CREATE ... INDEX *index_name*

.....



ALTER ... INDEX *index_name*



Advisory REORG-pending state until REORG INDEX, REBUILD INDEX, or REORG TABLESPACE is run.

Index Compression: Performance



→ CPU cost is mostly inconsequential. Most of the cost is asynchronous, the exception being a synchronous read. The worst case is an index with a poor buffer hit ratio.

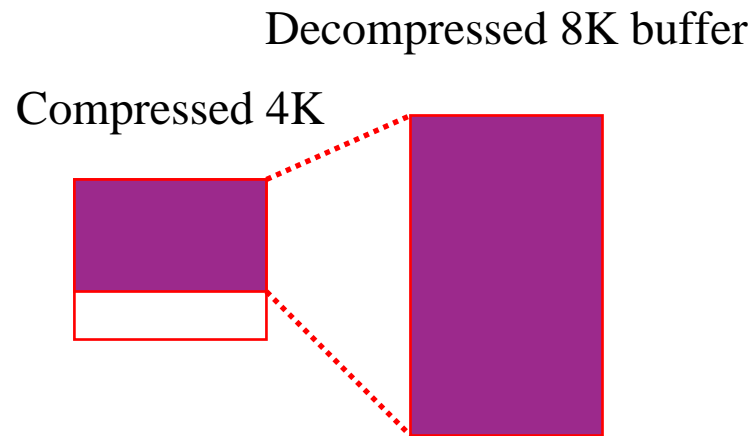
Example: Suppose the index would compress 3-to-1. You have three options.....

1. Use 8K buffer pool. Save 50% of disk. No change in buffer hit ratio or real storage usage.
2. Use 16K buffer pool and increase the buffer pool size by 33%. Save 67% of disk, increase real storage usage by 33%.
3. Use 16K buffer pool, with no change in buffer pool size. Save 67% of disk, no change in real storage used, decrease in buffer hit ratio, with a corresponding increase in synchronous CPU time.

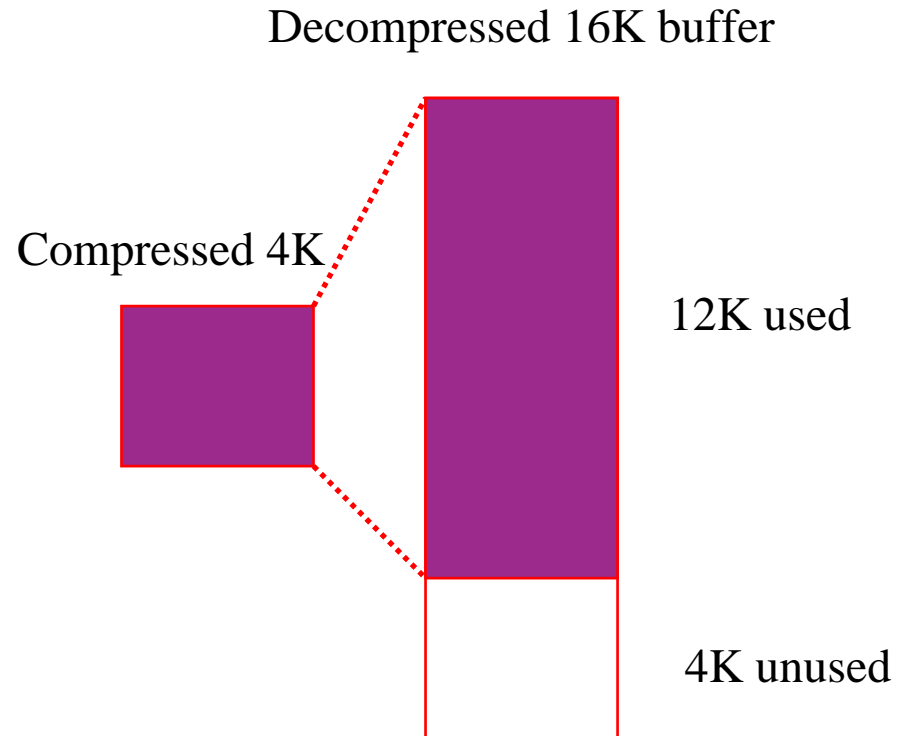
Index Compression Example:



Suppose an index could compress 3-to-1



8K Buffer pool
50% disk space reduction
No increase in virtual storage cost



16K Buffer pool
67% disk space reduction
33% increase in virtual storage cost

.....DB2 Index Compression



- The CI Size of a compressed index on disk is always 4K
- A 4K expands into a 8K or 16K buffer, which is the DBA's choice. This choice determines the maximum compression ratio.
- Compression of key prefix and RID Lists
 - A Rid List describes all of the rows for a particular index key
 - An index with a high level of non-uniqueness, producing long Rid Lists, achieves about 1.4-to-1 compression
 - Compression of unique keys depends on prefix commonality

>4K Page Size for Indexes



- V9 supports 4K, 8K, 16K and 32K page sizes for indexes
- A large page size is very good for reducing the frequency of CI splits, which is costly in data sharing environment.
- The downside: As with large pages for table spaces, the buffer hit ratio could degrade.



Non-padded indexes

- ➔ Non-padded indexes were introduced in DB2 V8
 - Useful when an index contains one or more VARCHAR columns
 - Facilitates index only access
 - Saves DASD space
- ➔ The CPU cost is significant. It grows as a function of the number of VARCHAR columns.
- ➔ Index compression and non-padded indexes are complementary

The Catalog

Catalog Information of Interest



| Catalog table | Column | Description |
|---------------|------------|---|
| SYSINDEXES | COMPRESS | Compression is in use (Y) or not in use (N) |
| SYSTABLEPART | COMPRESS | “Y” compression in use “blank” compression not used Can be at partition level |
| SYSTABLEPART | *PAGESAVE | % pages saved 0 no savings + is an increase Includes overhead bytes |
| SYSTABLEPART | AVGROWLEN | Average row length with or without compression |
| SYSTABLES | PCTROWCOMP | % rows compressed within total rows active |

*There are other columns in history tables

DSN1COMP

Should Compression Be Used?



- DSN1COMP
- Stand-alone utility
- Will estimate disk savings
- Works for table spaces and indexes
- Can be run against:
 - Table space underlying VSAM dataset
 - Index space underlying VSAM dataset
 - Output from a full image copy
 - Output from DSNCOPY
- Cannot be run against:
 - LOB table spaces
 - Catalog (DSNDB06)
 - Directory (DSNDB01)
 - Workfiles (i.e. DSNDB07)
- Using DSN1COMP with image copies and DSN1COPY outputs can make gathering compression information unobtrusive

DSN1COMP Keywords



- Choose the keywords that best resemble your table space
 - PAGESIZE
 - DSSIZE
 - FREEPAGE
 - PCTFREE should be set exactly as the object you are running DSN1COMP against to insure that its estimates are accurate

DSN1COMP Keywords



- Choose the keywords that best resemble your table space
 - If the REORG utility is used to build dictionary, specify REORG at run-time
 - Omitting REORG keyword defaults to LOAD utility
 - DSN1COMP input a full image copy
 - Specify FULLCOPY keyword to obtain the correct results

REORG SHRLEVEL CHANGE Warning



- ➔ When choosing a VSAM LDS to run against, be careful if you are using online REORG (REORG SHRLEVEL CHANGE)
 - Online REORG flips between the I0001 and J0001 for the fifth qualifier of the VSAM data sets. You can query the IPREFIX column in SYSTABLEPART or SYSINDEXPART catalog tables to find out which qualifier is in use.

DSN1COMP with an Index



- LEAFLIM keyword specifies the number of leaf pages that should be scanned
- Omit LEAFLIM and the entire index will be scanned
- Specifying LEAFLIM could limit how long it will take DSN1COMP to complete

DSN1COMP Table Space Output



- Statistics with and without compression, and the percentage you should expect to save in kilobytes
- Number of rows scanned to build the dictionary
- Number of rows processed to deliver the statistics in the report
- The average row length before and after compression
- The size of the dictionary in pages
- The size of the table space in pages
 - Before compression
 - After compression
- Percentage of pages that would have been saved

DSN1COMP Report



DSN1944I DSN1COMP INPUT PARAMETERS

512 DICTIONARY SIZE USED

30 FREEPAGE VALUE USED

45 PCTFREE VALUE USED

NO ROWLIMIT WAS REQUESTED

ESTIMATE BASED ON DB2 LOAD METHOD

DSN1COMP Report



DSN1940I DSN1COMP COMPRESSION REPORT

301 KB WITHOUT COMPRESSION

224 KB WITH COMPRESSION

25 PERCENT OF THE BYTES WOULD BE SAVED

1,975 ROWS SCANNED TO BUILD DICTIONARY

4,665 ROWS SCANNED TO PROVIDE COMPRESSION ESTIMATE

4,096 DICTIONARY ENTRIES

81 BYTES FOR AVERAGE UNCOMPRESSED ROW LENGTH

52 BYTES FOR AVERAGE COMPRESSED ROW LENGTH

16 DICTIONARY PAGES REQUIRED

110 PAGES REQUIRED WITHOUT COMPRESSION

99 PAGES REQUIRED WITH COMPRESSION

10 PERCENT OF THE DB2 DATA PAGES WOULD BE SAVED

DSN1COMP Index Output



- Reports on the number of leaf pages scanned
- Number of keys and rids processed
- How many kilobytes of key data was processed
- Number of kilobytes of compressed keys produced
- Reports broken down for possible percent reduction and buffer pool space usage for both 8K and 16K index leaf page sizes
- Considerable help when determining correct leaf page size

DSN1COMP Report



```
DSN1944I DSN1COMP INPUT PARAMETERS  
PROCESSING PARMS FOR INDEX DATASET:  
NO LEAFLIM WAS REQUESTED
```

DSN1COMP Report



DSN1940I DSN1COMP COMPRESSION REPORT

38 Index Leaf Pages Processed
3,000 Keys Processed
3,000 Rids Processed
401 KB of Key Data Processed
106 KB of Compressed Keys Produced

Cont'd on next page....

DSN1COMP Report



Cont'd from previous page...

EVALUATION OF COMPRESSION WITH DIFFERENT INDEX PAGE SIZES:

8 K Page Buffer Size yields a

→ 51 % Reduction in Index Leaf Page Space

The Resulting Index would have approximately

49 % of the original index's Leaf Page Space

→ **No Bufferpool Space would be unused**

16 K Page Buffer Size yields a

→ 74 % Reduction in Index Leaf Page Space

The Resulting Index would have approximately

26 % of the original index's Leaf Page Space

→ **3 % of Bufferpool Space would be unused to
ensure keys fit into compressed buffers**

SMF Compression



SMF Compression

- DB2's SMF records can now be compressed
 - SMF 100 (stats), 101 (accounting), 102 (performance)
 - all data after SMF header
 - Have seen 60-80% saving when used
 - Less than 1% CPU overhead if used
- How to activate
 - Installation panel
 - DSNTIPN
 - COMPRESS SMF RECS field
 - DSNZPARM
 - DSN6SYSP
 - SMFCOMP
 - Values – ON, OFF Default = OFF

SMF Compression

→ DSNTSMFD

- APAR PM27872
 - Closed February 4, 2011
- Accepts all SMF records
- Decompresses all DB2 SMF records
- Output SMF data set
- Sample job DSNTTEJDS provides basic JCL framework

→ DB2 10 and later only

References



- My Blog: “Getting the Most out of DB2 for z/OS and System z”
 - <http://blogs.ittoolbox.com/database/db2zos>

- IBM Redbook, “DB2 for OS/390 and Data Compression” ([SG24-5261](#))
 - although a bit old (circa Nov 1998), it should answer most of your remaining data compression questions

- “z/Architecture Principles of Operation” ([SA22-7832](#))
 - for a complete description of the CMPSC instruction

- “Enterprise Systems Architecture/390 Data Compression” ([SA22-7208](#))

- RedPaper “Index Compression with DB2 9 for z/OS” ([REDP-4345](#))

- “The Big Deal About Making Things Smaller: DB2 Compression”
 - [z/Journal Magazine, February/March 2008 issue](#)

- IBM Journal of Research and Development
 - Volume 46, Numbers 4/5, 2002 - IBM eServer z900
 - [“The microarchitecture of the IBM eServer z900 processor”](#)



Questions

Willie Favero

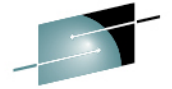
DB2 SME
Data Warehousing for System z Swat Team
IBM Silicon Valley Laboratory

My DB2 Blog
www.it.toolbox.com/blogs/db2zos/

<http://www.WillieFavero.com>



SHARE supplied QR code for this session



SHARE
Technology • Connections • Results

