

Which Table (Space) Type and Page Size Shall I Choose for DB2 on z/OS?

John Iczkovits
(iczkovit@us.ibm.com)
IBM

August 8, 2012
12068

- Title: Eenie Meenie Miney Mo, With Which Table (Space) Type and Page Size Shall I Choose for DB2 on z/OS?
- Abstract: Choosing the correct DB2 table (Space) type is very confusing. Should I go with a segmented table space, a UTS (if so, a PBG or PBR), HASH, or maybe good old classic partitioned? How about the page size? I now have a choice of 4K, 8K, 16K, or 32K for my table space and/or index. Should I choose a multi or single table option? Should I compress the object? Choosing the data's correct type and size the first time is essential to avoid costly outages. Come learn about best practices in choosing the correct types and settings for your data.

Confusion Corner!

- There are many combinations to choose from when deciding how your data will be housed. For example:
 - What type of Tables and table space?
 - Segmented
 - Classic partitioned
 - UTS PBR (Partitioned By Range)
 - UTS PBG (Partitioned By Growth)
 - UTS HASH
 - XML
 - LOB
 - Etc.
 - What type of index?
 - NPSI
 - DPSI
 - PI
 - Unique
 - Index on expression
 - Etc.
 - How large should the objects be?
 - Should the objects be partitioned?
 - Should the objects be compressed?
 - What should the relationship between table space and index attributes be?
 - How many indexes if any?
 - What page size for the table? How about the indexes?
 - Etc. etc. etc.
 - There is no “one size fits all” answer. Much of what we are going to discuss may help you decide what works best for a specific set of objects, but ultimately you must do one very important thing – TEST, TEST, and TEST some more!



First, what is a space map?

- Space map pages are real pages that exist in both table spaces and indexes
- Space maps can be added as pages are preformatted
 - For DB2 9 and 10, up to 16 cylinders are preformatted at one time
 - HASH is the exception. HASH space is all formatted when created and all of the space map pages created at CREATE time.
- The number of pages a space map covers is outlined in the Diagnosis Guide and Reference
 - When using MEMBER CLUSTER:
 - Classic partitioned - each space map page covers 199 pages
 - UTS – each space map page covers 10 segments
- Space maps do not point to specific rows or RIDs. They contain the status of a page.

Table space definition – TRACKMOD



- Applies to both sequential or random insert
- DB2 keeps track of changed pages in the space map page
- It is used by incremental COPY to efficiently determine which pages to be copied i.e., avoid scanning every page
 - <YES> – DB2 keeps track of updated data pages
 - Is default prior to V10
 - Dirty bit on the space map page is updated when the data page is changed.
 - <NO> – is recommended if do not require incremental COPY
 - DB2 does not keep track of updated pages
 - DB2 does not track changed pages in the space map pages. DB2 uses the LRSN value in each page to determine whether a page has been changed.
 - Less space map page updates which will improve performance
 - Less data sharing overhead
 - Incremental copy can still be used, but can run significantly slower
 - Can be altered via ALTER TABLESPACE DDL
- The default can be controlled for implicitly created table spaces by⁵ zparm (IMPTKMOD) in V10

Complete your sessions evaluation online at SHARE.org/AnaheimEval

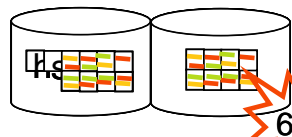

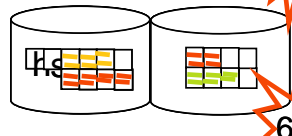

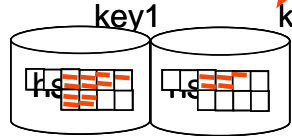



Past Table Spaces Options

- **Past table space options**

- Simple
 - Multi table, interleaved
 - Very simple space map
- **Simple table spaces can not be created (deprecated in DB2 9)**

- Segmented
 - Multi table, no page sharing
 - Better space maps
 - Good with mass deletes and inserts
 - 64GB

| | | | | |
|--------|--------------|--|---|--|
| Simple | Multitable |  | Single deletes | <MC>  |
| Seg | Multitable |  | Mass deletes  | No MC |
| Part. | Single table |  | Single deletes | <MC>  |

*MC = Member Cluster

- Partitioned
 - One table per table space
 - 128TB maximum for 32K objects
 - Does not have the internal space map like that of a segmented table space to allow for mass deletes.

FAQ – What does maximum of 64G mean?



- For simple and segmented objects:
 - Each data set can grow to 2GB
 - When the 2GB maximum is reached, a new data set is automatically created for DB2 managed data sets. User must still use IDCAMS DEFINE when data is user managed.
 - Each table can create up to 32 data sets of data. The LLQ (data set Low Level Qualifier) starts with A001, successive data sets add one to the LLQ value when created (i.e. A001, A002 ... A032).
 - 64GB = 2GB per data set * 32 data sets total
- For partitioned objects:
 - Each data set can grow to 64GB
 - **NOTE!** With DB2 10 APAR PM42175, partitions can grow to 128GB and 256GB with a smaller number of total partitions.
 - Data sets grow based on PRIQTY, SECQTY, DSSIZE, PIECESIZE (some indexes), and sliding secondary.
 - Data sets > 4GB must be SMS managed, with Data Class EF (Extended Format) and EA (Extended Addressability) on
 - *Best practice – EF/EA all DB2 VSAM and sequential data sets with the exception of temporary (&&) data sets*
 - Total size for one table with multiple data sets:
 - 4K page – 16TB
 - 8K page – 32TB
 - 16K page – 64 TB
 - 32K page – 128TB

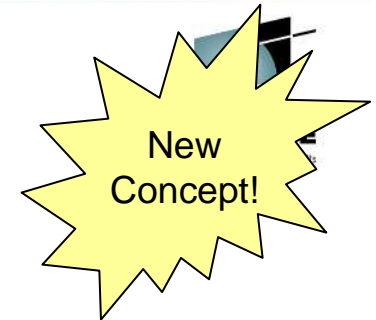
Universal Table Spaces



- **Universal Table Spaces**

- Combination of segmented with partitioning options (hybrid approach)
 - Better space management
 - Support of mass deletes / TRUNCATE
 - One table per table space
- Two options:
- Range-partitioned (PBR)
 - All the features of classic partitioning, but segmented
 - Table controlled partitioning only – no PI (Partitioning Indexes)
 - Using partition column(s)
 - Partitioned and segmented
- Partition-by-growth (PBG)
 - Partitions added as space is needed
 - No partitioning key
 - Partitioned and segmented
 - **New concept!**
- DROP / CREATE to migrate existing page sets. ALTER allowed under some circumstances starting in DB2 V10.








Universal Table Spaces – Partitioned By Growth



- **Partition By Growth (PBG)**
 - Single-table table space, where each partition contains a segmented page set (allows segmented to increase from 64GB to 128 TB with 32K pages)
 - Eliminates need to define partitioning key and assign key ranges
 - Partitions are added on demand
 - A new partition is created when a given partition reaches DSSIZE
 - *See the SQL Reference for DSSIZE rules given the page size & number of partitions*
 - Up to MAXPARTITIONS
 - Retains benefits of Utilities and SQL parallelism optimizations (child tasks) for partitioned tables
 - Not used for query parallelism for partition pruning
 - SEGSIZE defaults to 4 in DB2 9, 32 in DB2 10
 - LOCKSIZE defaults to ROW for implicitly created tables, otherwise it defaults to ANY

Universal Table Spaces

- What kind of Table Space will be created? (* optional)

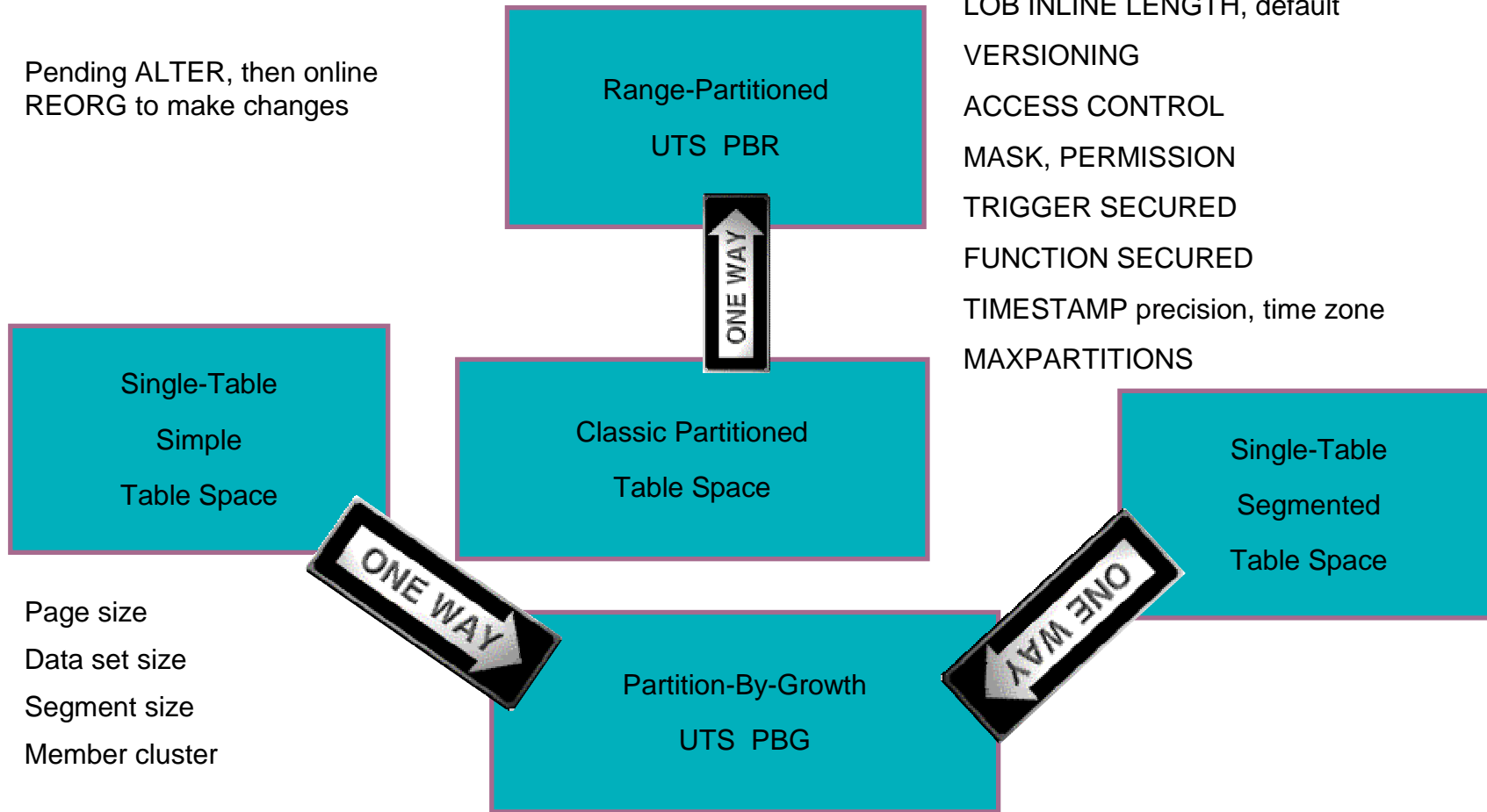
| CREATE TABLESPACE... | SEGSIZE | NUMPARTS | MAXPARTITIONS | Comments |
|------------------------|---|--|---|---|
| Segmented |  * | | | *SEGSIZE is optional. Default for explicitly created TS & implicitly created TS for CM8. SEGSIZE defaults to 4. |
| UTS PBG |  * | Optional to indicate # of initial partitions |  | Default for CM9 and NFM with implicitly created TS. Single table TS. *SEGSIZE will default to 32. |
| UTS PBR |  * |  | | Single table TS *SEGSIZE will default to 32. |
| Classic Partitioned TS |  * |  | | Partitioning TS prior to V9 *SEGSIZE 0 will create classic partitioned and CM8 behavior is same as V8 NFM |



Improved availability in DB2 10 ALTER + REORG



Pending ALTER, then online REORG to make changes



- LOB INLINE LENGTH, default
- VERSIONING
- ACCESS CONTROL
- MASK, PERMISSION
- TRIGGER SECURED
- FUNCTION SECURED
- TIMESTAMP precision, time zone
- MAXPARTITIONS

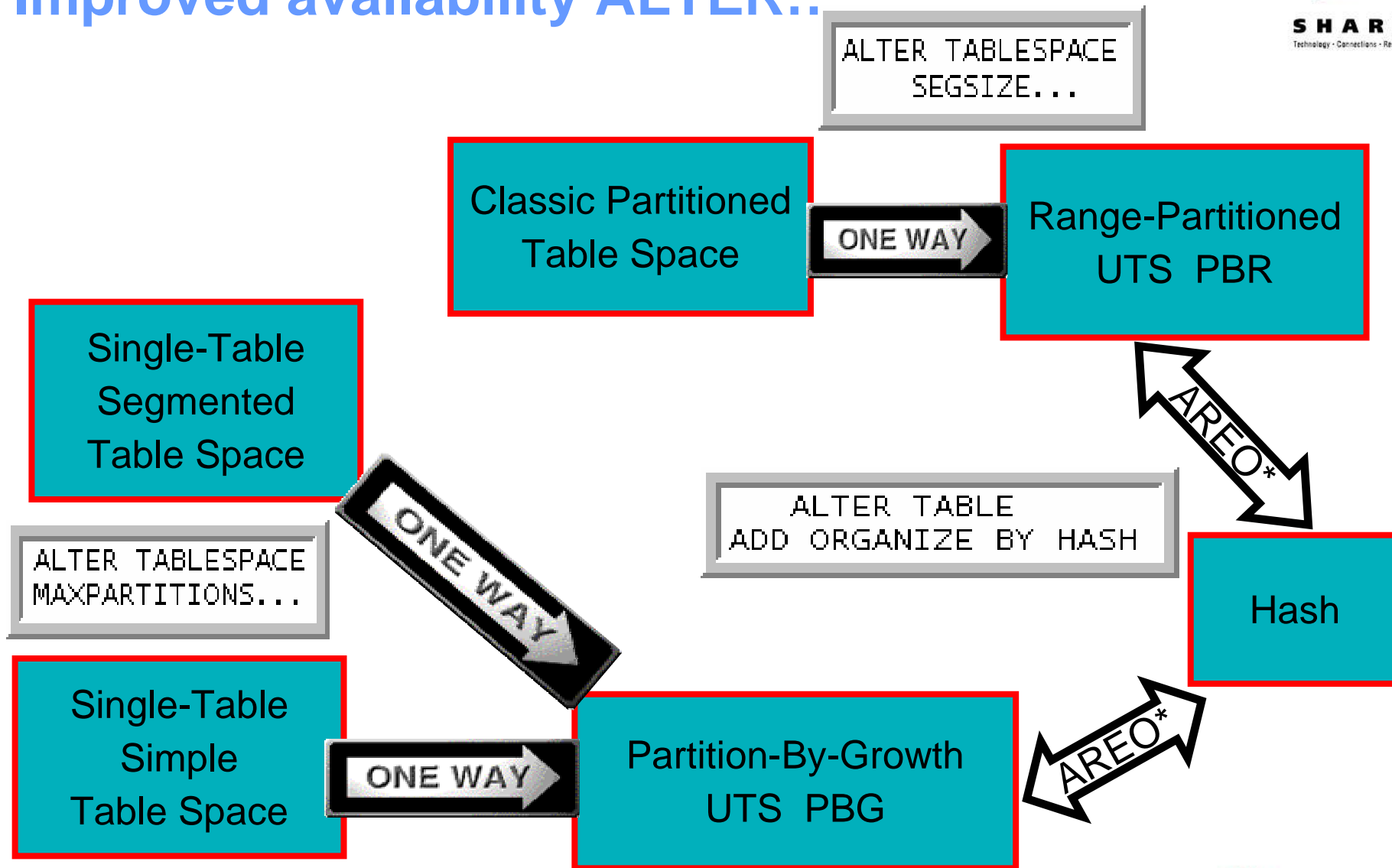
- Page size
- Data set size
- Segment size
- Member cluster

- INDEX page size
- INCLUDE cols

- ADD active log
- MODIFY DDF ALIAS
- BUFFERPOOL PGSTEAL NONE



Improved availability ALTER..



DSNZPARAM for SEGSIZE Default



- When SEGSIZE is NOT specified
- DB2 10 – The picture changes considerably
 - If ZPARAM DPSEGSZ = 0
 - If MAXPARTITIONS is not specified
 - *If NUMPARTS is not specified*
 - *SEGSIZE 4 for segmented table space*
 - *If NUMPARTS is specified*
 - *Classic partitioned table space*
 - If MAXPARTITIONS is specified
 - *With or without NUMPARTS being specified*
 - *partition-by-growth table space w/ SEGSIZE = 32*
 - If ZPARAM DPSEGSZ > 0 (a greater than zero value)
 - If MAXPARTITIONS is not specified
 - *If NUMPARTS is not specified*
 - *SEGSIZE 4 for segmented table space*
 - *If NUMPARTS is specified*
 - *Partitioned by range-partitioned table space w/ SEGSIZE = DPSEGSZ*
 - If MAXPARTITIONS is specified
 - *With or without NUMPARTS being specified*
 - *partition-by-growth table space w/ SEGSIZE = DPSEGSZ*

DB2 9
Default SEGSIZE 4

Partition-by-growth Table Space



explicit specification

```
CREATE TABLESPACE ...  
MAXPARTITIONS integer
```

implicit specification

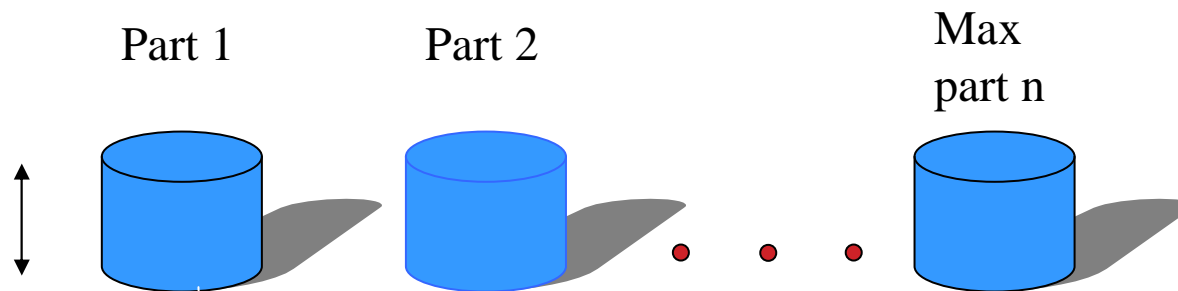
```
CREATE TABLE ...  
PARTITIONED BY SIZE EVERY  
integer G
```

- ✓ Associated SYSTABLESPACES columns
 - MAXPARTITIONS =max number of partitions
 - PARTITIONS =actual number of partitions
 - TYPE =G
- ✓ Only single-table table space
- ✓ Universal table space organization: although the table space is partitioned, the data within each partition is organized according to segmented architecture
- ✓ Incompatible with ROTATE PARTITION

How Partition-By-Growth Works



- ✓ The table space starts with one partition, additional partitions will be added on demand until the maximum partition is reached.



Partitioned Table Space (parts added on demand)

Partition-By-Growth CREATE



- SQL CREATE TABLESPACE statement for PBG

```
CREATE TABLESPACE TS1 IN DB1
  MAXPARTITIONS 55
  SEGSIZE 64
  DSSIZE 2G
  LOCKSIZE ANY;
```

Makes PBG

Partition size

- ✓ A new key word MAXPARTITIONS - specifies the maximum # of partition for a table space.
- ✓ Maxpartitions can be changed by ALTER TABLESPACE
 - Keep in mind that ALTER MAXPARTITIONS may require down time because it needs to physically close the datasets

PBG WARNING!

- Be careful when setting MAXPARTITIONS. Although you can ALTER the value to a higher number, you cannot decrease it to less partitions!
- NOTE – resolved by APAR **PM57001!**
- See Willie Favero's Blog:
- <http://it.toolbox.com/blogs/db2zos/be-careful-how-you-define-your-partitionbygrowth-universal-table-space-50164>

Partition-By-Growth Create



- ✓ SQL CREATE TABLE statement for PBG

```
CREATE TABLE Mytable  
PARTITION BY SIZE EVERY integer G;  
where integer ≤ 64
```

- ✓ Only available when you don't specify a table space name on the CREATE TABLE
- ✓ Table space is implicitly created
- ✓ mG specifies DSSIZE of the table space

More on Implicitly Created PBG

- Implicitly created table space defaults to PBG
- It defaults to row locking for implicitly created tables, otherwise ANY
- The LOCKMAX defaults to SYSTEM
- Default value for MAXPARTITIONS = 256
- Default SEGSIZE = 4 if not specified on DDL

Note

- In DB2 Version 10, the default SEGSIZE value for universal table spaces has changed from 4 to 32
 - New DSNZPARAM – DPSEGSZ (default 32) on DSN6SYSP macro
 - DPSEGSZ affects the SEGSIZE default chosen
 - DPSEGSZ becomes available in DB2 10 new function mode (NFM)

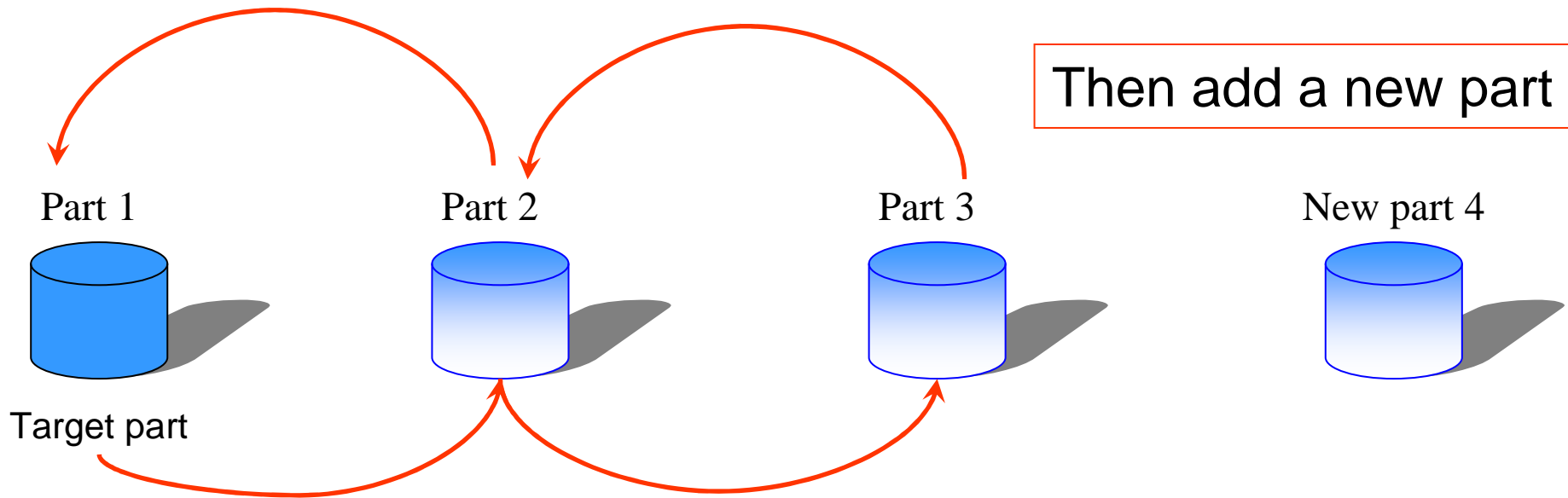
- Default DSSIZE = 4G if not specified on DDL
 - Note: DSSIZE and SEGSIZE require a DROP to change, no ALTER option

Note

- DB2 10 has ALTER DSSIZE/SEGSIZE

Partition-By-Growth Space Search

- ✓ No more space in the partition...
 - Search forward to next partition if there is one
 - Search backward to previous partitions



Note: If there is any restricted DBET state of any part during the backward space search, a new part will not be added.

Additional Characteristics of PBG

- PBG is partitioned according to space requirements
 - A partition is allocated when one is needed due to growth
- Each partition has a one-to-one correspondence to a VSAM data sets and **MUST** be DB2-managed
- No partitioning key to bound the data within a table space, so no PI index
- Only non-partitioned indexes can be created
 - No data-partitioned secondary index (DPSI)
- Only single table allowed per table space
 - can not totally replace segmented table space

PBG – Additional Function Limitations

- **No** partition key range can be defined
- **No** ALTER ADD PART
- **No** ALTER ROTATE PART
- **No** ALTER Stogroup
- **No** LOAD PART
- **No** user-directed define partition
 - Required to use UNLOAD/LOAD instead of DSN1COPY for copying data between table space if source table space has more than 1 partition

Note

DB2 10 allows a partition to be added up to the value of MAXPARTITIONS

Practical Applications for PBG

- When no obvious partitioning column exists
- When a table requiring > 64G
 - Lift 64G size limitation of segmented table space
 - Increase overall size of table space on demand
- Space on Demand
- Large table space and manage utilities at a data subset is needed
 - Partition level utility
- There's a need for CLONE table
- HASH table use (added in DB2 10)

Note

- **MYTH:** Inserting the same number of rows into a PBG data set will take more space than a segmented table space.
- **FACT:** Generally the amount of space should be the same
 - For verification, you can compare DB2 Catalog numbers, as well as the HI-U-RBA portion of IDCAMS LISTCAT.

Create Range-partitioned UTS

✓ SQL CREATE statement

```
CREATE TABLESPACE PRB_TS1 IN UTS_DB1  
  NUMPARTS 3  
  SEGSIZE 64  
  LOCKSIZE ANY;
```

Makes it PBR

- ✓ Create a partitioned table space and just add the SEGSIZE clause = Range-partitioned table space
- Range-partitioned table space is now **DEFAULT in DB2 10**

Note

- Classic partitioned table spaces still supported
 - Create classic by specifying **SEGSIZE 0** on CREATE

Create Table in Range-partitioned UTS

```
CREATE TABLE MyTable
  ( C1 CHAR(4),
    C2 VARCHAR(20),
    C3 INTEGER )
PARTITION BY (C1)
  ( PARTITION 1 ENDING AT ('DDDD'),
    PARTITION 2 ENDING AT ('HHHH'),
    PARTITION 3 ENDING AT ('ZZZZ') )
  IN UTS_DB1.PRB_TS1 ;
```

- Must use table-controlled partitioning

PBR – Additional Function Limitations



- No index-controlled (PI – Partitioning Indexes) partitioning definition – only allowed for classic partitioned

Example of invalid way to create partition range:

```
CREATE UNIQUE INDEX TBIX1 ON MyTable
(C1)
  CLUSTER
    (PARTITION 1 ENDING AT ('DDDD'),
     PARTITION 2 ENDING AT ('HHHH'),
     PARTITION 3 ENDING AT ('ZZZZ'))
  BUFFERPOOL BP0
  CLOSE YES;
```

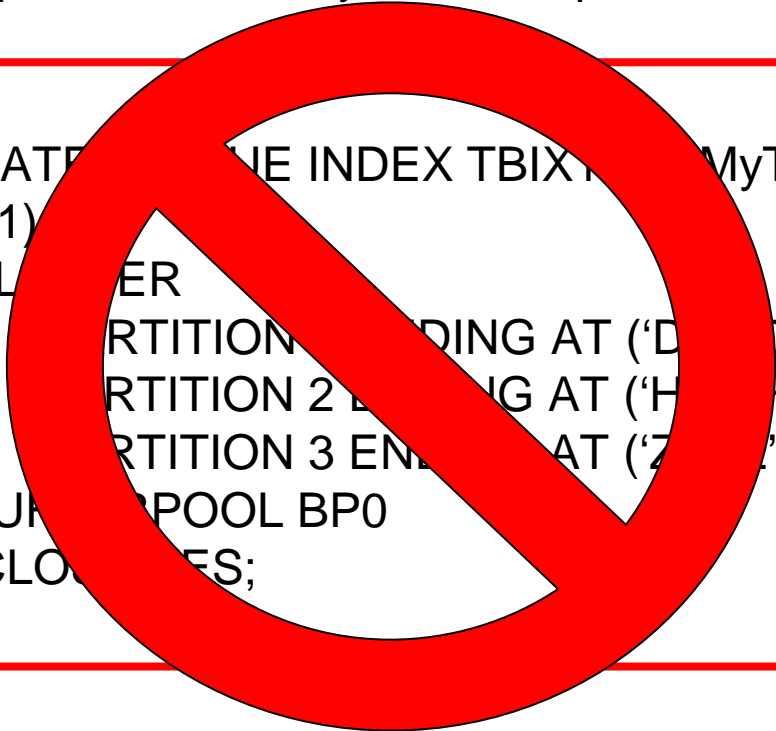
PBR – Additional Function Limitations



- No index-controlled (PI – Partitioning Indexes) partitioning definition – only allowed for classic partitioned

Example of invalid way to create partition range:

```
CREATE PARTITIONED INDEX TBIX ON MyTable  
(C1)  
CLUSTERED  
PARTITION 1 ENDING AT ('D'),  
PARTITION 2 ENDING AT ('H'),  
PARTITION 3 ENDING AT ('Z'))  
BUFFER POOL BP0  
CLOSED;
```



SQLCODE = -662

A PARTITIONED INDEX
CANNOT BE CREATED ON A
NON-PARTITIONED,
PARTITION-BY-GROWTH, OR
RANGE-PARTITIONED
UNIVERSAL TABLE SPACE

Converting index-controlled partitioning to table-controlled partitioning



- DB2 UDB for z/OS Version 8: Everything You Ever Wanted to Know, ... and More SG24-6079
 - 3.6.3 Converting to table-controlled partitioning
- Read other articles on the web for some additional best practices

Range-partitioned Practical Applications

- When a partitioned table space and a partitioning key is required
- When better performance than classic partitioned table space is required – especially for mass deletes
- Parallelism and partition-independence capabilities
- When a CLONE table is required

Note

- HASH table use (added in DB2 10)

Fastest Available in DB2 V9: Index access

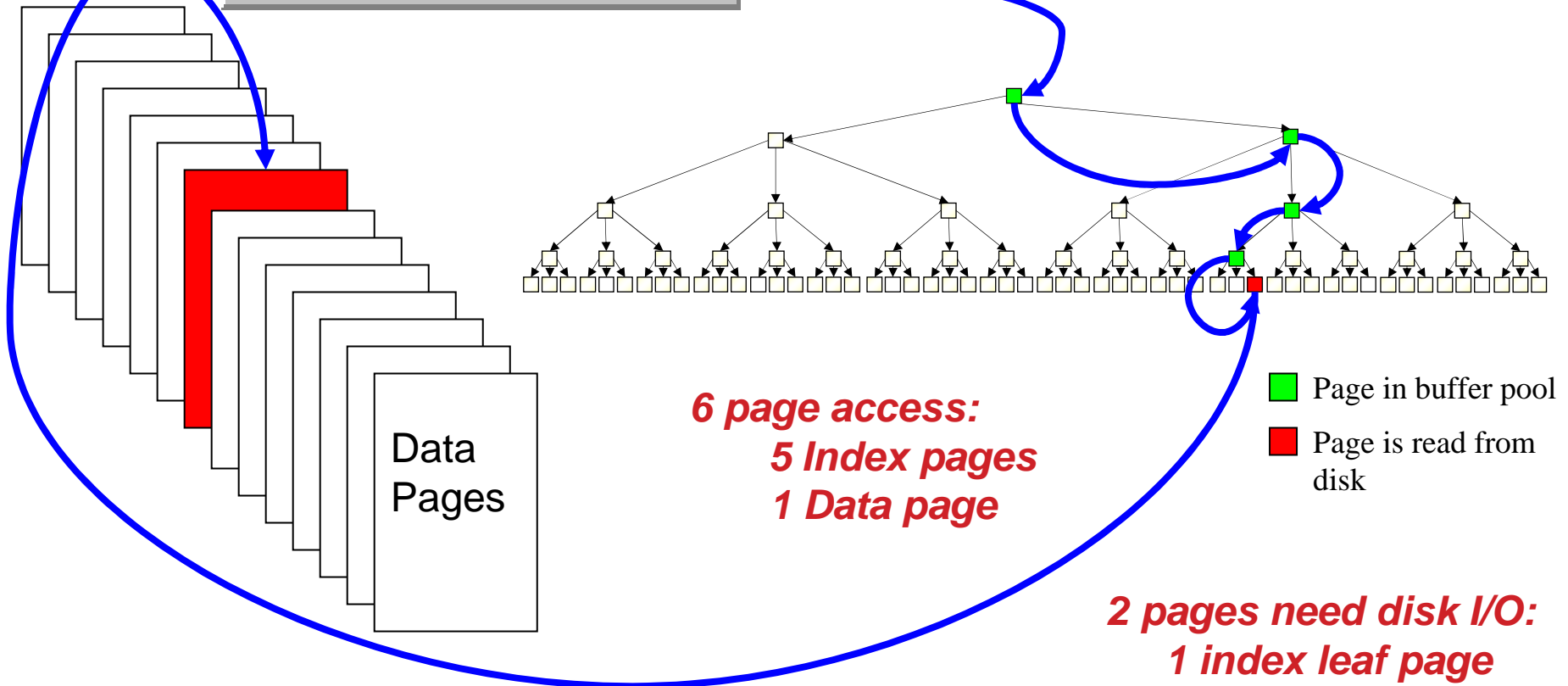
Without HASH access

SHARE
Technology • Connections • Results

Query:

```
SELECT * ...WHERE  
ITEMNO = 'W0133-1662996 '
```

ITEMNO Unique Index
5 levels

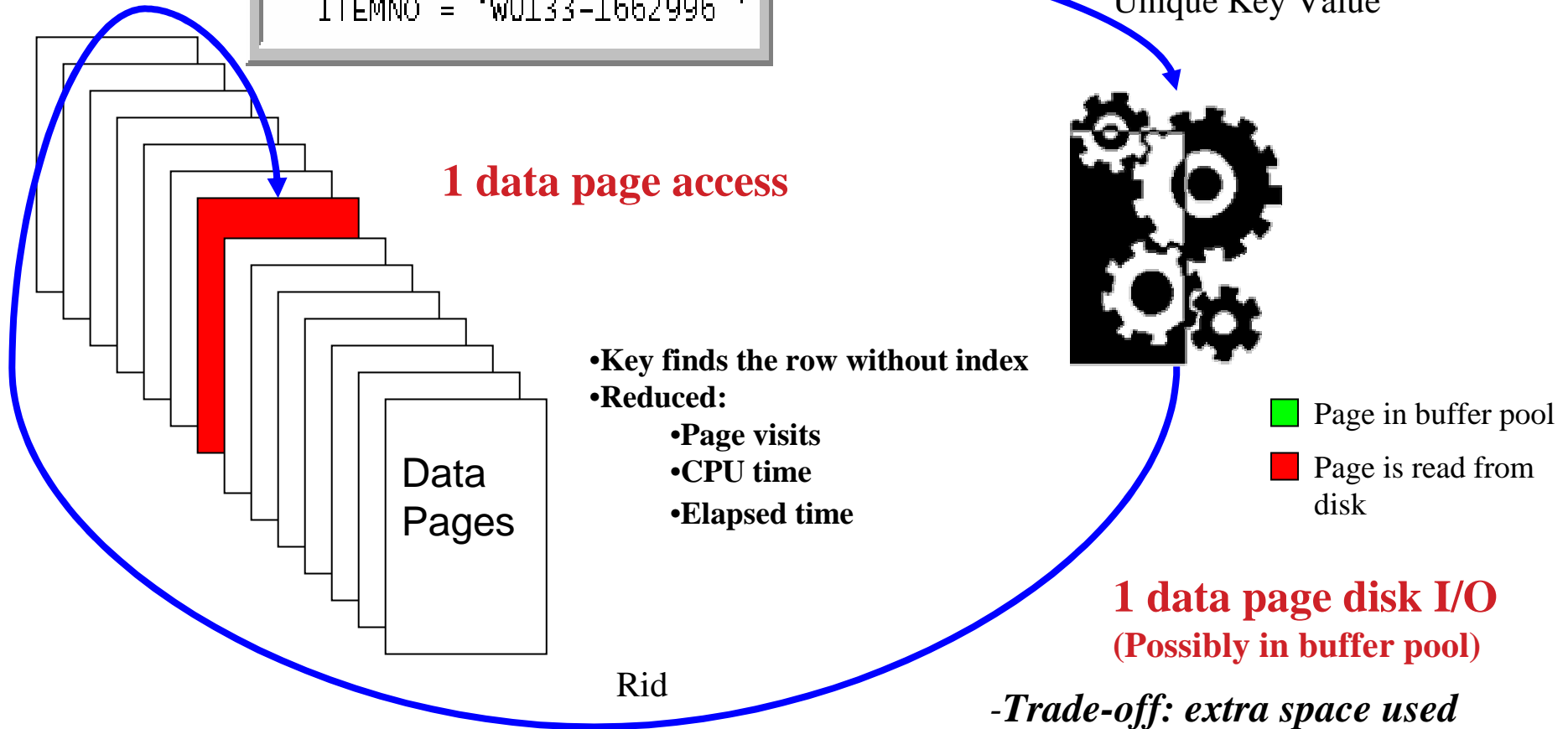


Fastest Available In DB2 V10: Hash Access

Query:

```
SELECT * ..WHERE  
ITEMNO = 'W0133-1662996 '
```

Unique Key Value



Hash Access Candidates

- **Candidate Tables**

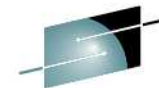
- For queries needing single row access via the unique key
- Queries having equal predicates on keys
- With known and static approximate size of data
- Having large N-level indexes

- **Not for Tables**

- Needing sequential processing
- Frequently updated
- Either using BETWEEN or > and <

- **Follow-up**

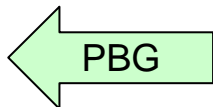
- Run REBIND with EXPLAIN option and query the PLAN_TABLE to check access path
- SYSTABLESPACESTATS.REORGHASHACCESS
 - *Number of times data is read using hash access in the last time interval*
- Check LASTUSED & REORGINDEXACCESS on overflow and other indexes to validate HASH access
- PM25652 adds REORG recommendations to DSNACCOX



```

CREATE TABLESPACE JOHNIHAS
  USING STOGROUP SYSDEFLT
  PRIQTY          720000
  SECQTY          7200
  DSSIZE          1G
  MAXPARTITIONS  3
  LOCKSIZE       ANY
  CLOSE          NO
  BUFFERPOOL     BP0
  FREEPAGE       0
  PCTFREE        0;

```



```

CREATE TABLE JOHNIHASH
(FNAME CHAR(20) NOT NULL,
 LNAME CHAR(20) NOT NULL,
 MNAME CHAR(20) NOT NULL)

```

ORGANIZE BY HASH UNIQUE

(LNAME,FNAME)

HASH SPACE 1G

IN DSNDB04.JOHNIHAS;

HASH must be UTS
(PBG or PBR)

```

DB2A10.DSNDBD.DSNDB04.JOHNIHAS.I0001.A001
DB2A10.DSNDBD.DSNDB04.JOHNIHAS.I0001.A002

DB2A10.DSNDBD.DSNDB04.JOHNIHAS.I0001.A001

ALLOCATION
SPACE-TYPE-----CYLINDER      HI-A-RBA-----1074216960
SPACE-PRI-----1000          HI-U-RBA-----1073479680
SPACE-SEC-----10

DB2A10.DSNDBD.DSNDB04.JOHNIHAS.I0001.A002

ALLOCATION
SPACE-TYPE-----CYLINDER      HI-A-RBA-----737280000
SPACE-PRI-----1000          HI-U-RBA-----11796480
SPACE-SEC-----10

Index created by DB2 for HASH:
DB2A10.DSNDBC.DSNDB04.JOHN1WMZ.I0001.A001
ALLOCATION
SPACE-TYPE-----CYLINDER      HI-A-RBA-----737280
SPACE-PRI-----1              HI-U-RBA-----737280
SPACE-SEC-----1

```

Preformat 1G

Preformat more

Preformatting 1G is just for the data, more Space is required for DB2 overhead.
Partition 3 is not created until required.

UTS – the wave of the future!



- Simple table spaces are deprecated starting in DB2 9
- Segmented table spaces can still be used
- Classic partitioned may eventually be deprecated
- UTS is the wave of the future
 - Most of the DB2 Catalog and Directory in DB2 10 are UTS after ENFM
 - DB2 work files (DSNDB07 or equivalent) are PBG starting in DB2 10
 - Most performance problems and other inhibitors with UTS were fixed in DB2 10
 - PBGs are very easy to create and use. DB2 handles the heavy lifting
 - Certain data sets, such as for CLONE tables and HASH tables require UTS
 - With DB2 10, converting (ALTER) single table simple tables space, segmented, or classic partitioned are all to UTS



FAQ – Should all of my data sets be UTS?

- It depends
- If the data set is small and does not need to be partitioned, segmented is a good way to go
- Locking - No LOCKSIZE TABLE (uses partitioned table space locking scheme)
- There were some CPU concerns in DB2 9 for UTS objects that were resolved in DB2 10. Test and verify results.
- There is a performance issue in DB2 9 and 10 for large sequential insert throughput using row level locking with PBGs. PBRs do not run into this problem. Even a lower level of throughput, PBG still outperformed segmented.

FAQ: How many tables should I have in my table space



- UTS and classic partitioned only allow one table per table space
- Segmented tables allow for multiple tables per table space
 - Generally create only one table per table space
 - For small tables that are not frequently updated, multiple tables per table space can still be an option
 - Large tables or ones with more frequent updates should only have one table per table space
 - **NOTE!** When more than one table exists in a table space, with DB2 10 ALTER cannot convert the table space to PBG. Only single table table spaces can be converted to PBG.

Index sizes



- Common Problem: 99% of the customers that **I** talk to do not use indexes > 4K
- Starting with DB2 9, indexes can now be 4K, 8K, 16K, or 32K.
- Only indexes > 4K can be compressed
- Dictionaryless, software managed index compression at the page level.
- Indexes are compressed at write time, decompressed at read time. They are uncompressed in the buffer pools.
- Compression of indexes for BI workloads
 - Indexes are often larger than tables in BI
- Solution provides page-level compression
 - Data is compressed to 4 KB pages on disk
 - 4K indexes cannot be compressed
 - 32/16/8 KB pages results in 8x/4x/2x disk savings
 - No compression dictionaries – compression on the fly
- DSN1COMP can be used for indexes as well starting with DB2 9
- Index compression is strictly for disk cost savings, not performance

Index Compression Versus Data Compression

- There are differences between index and data compression

| | Data | Index |
|----------------------|---------------|-------------------------------|
| Level | Row | Page (1) |
| Comp in DASD | Yes | Yes |
| Comp in Buffer Pool | Yes | No |
| Comp in Log | Yes | No |
| Comp Dictionary | Yes | No (2) |
| 'Typical' Comp Ratio | 10 to 90% | 25 to 75% (3) |
| CPU Overhead (4) | In Accounting | In Accounting and/or DBM1 SRB |

1. No compression or decompression at each insert or fetch; instead, it is done at I/O time
2. LOAD or REORG not required for compression; compression on the fly
3. Based on very limited survey and usage
4. CPU time impact under study – sensitive to index BP Hit Ratio, larger index BP recommended, higher impact on relatively unique indexes with long keys

Table space compression



- Data compression, unlike index compression can result in performance improvements.
- When do you compress a table space?
 - Compression requires CPU cycles. Is your LPAR or CEC running at 100% busy? Can it afford extra cycles for compression?
 - Compression information resides above the 2GB bar. Make sure there is enough real memory to house the information.
 - Do not compress small table spaces. It is not worth the tradeoff of CPU.
 - DSN1COMP on the object or image copy of the object provides potential savings. What is your magic number for compression saving? Mine is generally a liberal 40%. For some people the magic number is 50%, 60%, etc.
 - Starting in DB2 10, you can compress table space on the fly without requiring REORG. Do you know if your data set will compress well without knowing exactly what your data will look like?

INDEX FAQ



- When do I use indexes >4K?
 - When indexes have high number of split pages
 - When less index levels and more RIDs per page are important
- How do I tell from the RTS (Real Time Statistics) if page splits are a problem?
 - $((\text{REORGLAFFAR} * 100) / \text{NACTIVE}) > 10$
 - Calculates the ratio of index page splits in which the higher part of the split page was far from the location of the original page to the number of active pages. REORG when > 10%.
 - *RRILeafLimit* when running RTS Stored Procedures DSNACCOR or DSNACCOX
 - For potentially more stale statistics, you can also use the DB2 Catalog sysibm.sysindexpart column FAROFFPOSF
- Before using an index > 4K for page splits, test using variations on FREEPAGE and PCTFREE which at times can greatly reduce the number of page splits.
- There can be dramatic performance degradation when choosing the wrong page size. TEST, TEST, and TEST again! Test with different variations in page size.

Large Index Page Size

- Available in V9 NFM
- Potential to reduce the number of index leaf page splits, which are painful especially for GBP-dependent index (data sharing)
 - Reduce index tree latch contention
 - Reduce index tree p-lock contention
- Potential to reduce the number of index levels
 - Reduce the number of getpages for index traversal
 - Reduce CPU resource consumption
- Possibility that large index page size may aggravate index bufferpool hit ratio for random access

FAQ – I know what my table space size is, is my index the same size?



- For NPSI, user can specify the index PIECESIZE. Review the SQL manual, PIECESIZE in DB2 10 can reach the equivalent of 256GB.
- For partitioned index (DPSI), DB2 uses a formula to calculate the index piece size based on the table spaces DSSIZE. PIECESIZE cannot be specified for DPSI.
- For indexes for segmented table spaces, same as the allocation for the table space – maximum of 2GB per data set * 32 data sets = 64GB total.
- PIECESIZE used to be used commonly to reduce the size of indexes and therefore create more index data sets to avoid disk contention. PIECESIZE now allows for very large data sets. Does it make sense to worry about more index data sets in today's virtualized disk world?

Table and Index page sizes

- MYTH: Index page sizes must always match table page sizes
- FACT: Index page sizes are independent of table page sizes
 - Larger index page sizes provide benefit for large number of page splits and reducing leaf pages
 - Tables can be at one page size and indexes at different pages sizes
 - Compressing the table does not require that the index be compressed as well

CREATE TABLESPACE JOHNIPBG USING STOGROUP

SYSDEFLT

PRIQTY 72000
SECQTY 7200 64G
DSSIZE 64G
MAXPARTITIONS 3 PBG
LOCKSIZE ANY
COMPRESS YES compress
CLOSE NO
BUFFERPOOL BP0 4K
FREEPAGE 0
PCTFREE 0
SEGSIZE 32;

CREATE TABLE JOHNITB
(FNAME CHAR(20) NOT NULL,
LNAME CHAR(20) NOT NULL,
MNAME CHAR(20) NOT NULL)
IN DSNDB04.JOHNIPBG;

CREATE INDEX JOHNIX1
ON JOHNICZ.JOHNITB
(FNAME)
USING STOGROUP SYSDEFLT

PRIQTY 72000
SECQTY 7200
PIECESIZE 16G 16G
CLOSE NO
COMPRESS NO No compression
BUFFERPOOL BP8K0 8K
FREEPAGE 0
PCTFREE 0;

CREATE INDEX JOHNIX2
ON JOHNICZ.JOHNITB
(FNAME)
USING STOGROUP SYSDEFLT

PRIQTY 72000 4G - default
SECQTY 7200
CLOSE NO
COMPRESS NO No compression
BUFFERPOOL BP32K 32K
FREEPAGE 0
PCTFREE 0;

CREATE INDEX JOHNIX3

ON JOHNICZ.JOHNITB
(LNAME)
USING STOGROUP SYSDEFLT

PRIQTY 72000 4G - default
SECQTY 7200
CLOSE NO
COMPRESS NO No compression
BUFFERPOOL BP8K0 8K
FREEPAGE 0
PCTFREE 0;

CREATE INDEX JOHNIX4

ON JOHNICZ.JOHNITB
(LNAME)
USING STOGROUP SYSDEFLT

PRIQTY 72000 4G - default
SECQTY 7200
CLOSE NO
COMPRESS NO No compression
BUFFERPOOL BP32K 32K
FREEPAGE 0
PCTFREE 0;

CREATE INDEX JOHNIX5

ON JOHNICZ.JOHNITB
(LNAME)
USING STOGROUP SYSDEFLT

PRIQTY 72000 4G - default
SECQTY 7200
CLOSE NO
COMPRESS YES compress
BUFFERPOOL BP32K 32K
FREEPAGE 0
PCTFREE 0;

Objects before REORG



| Object | BP | COMPRESS | CISIZE | Cylinders allocated | HI-A-RBA | Cylinders used | HI-U-RBA |
|-----------------|--------------|------------|-------------|---------------------|----------|----------------|----------|
| JOHNIPBG | BP0 | YES | 4096 | 100 | 73728000 | 80 | 58982400 |
| JOHNIIX1 | BP8K0 | NO | 8192 | 100 | 73728000 | 32 | 23592960 |
| JOHNIIX2 | BP32K | NO | 32768 | 103 | 74252288 | 32 | 23068672 |
| JOHNIIX3 | BP8K0 | NO | 8192 | 100 | 73728000 | 32 | 23592960 |
| JOHNIIX4 | BP32K | NO | 32768 | 103 | 74252288 | 32 | 23068672 |
| JOHNIIX5 | BP32K | YES | 4096 | 100 | 73728000 | 16 | 11796480 |

Objects after REORG

| Object | BP | COMPRESS | CISIZE | Cylinders allocated | HI-A-RBA | Cylinders used | HI-U-RBA |
|-----------------|--------------|------------|-------------|---------------------|----------|----------------|----------|
| JOHNIPBG | BP0 | YES | 4096 | 100 | 73728000 | 12 | 8372224 |
| JOHNIIX1 | BP8K0 | NO | 8192 | 100 | 73728000 | 5 | 3162112 |
| JOHNIIX2 | BP32K | NO | 32768 | 103 | 74252288 | 5 | 3244032 |
| JOHNIIX3 | BP8K0 | NO | 8192 | 100 | 73728000 | 5 | 3162112 |
| JOHNIIX4 | BP32K | NO | 32768 | 103 | 74252288 | 5 | 3244032 |
| JOHNIIX5 | BP32K | YES | 4096 | 100 | 73728000 | 0.75 | 552960 |

NOTE – 32K objects have space + 2.2%. This is not true for compressed 32K Index objects as they are allocated as 4K objects

| TSNAME | CARD | COMPRESS | PAGESAVE |
|----------|--------|----------|----------|
| JOHNIPBG | 512512 | Y | 76 |

systablespace

| NAME | BPOOL | PGSIZE | DSSIZE |
|----------|-------|--------|----------|
| JOHNIPBG | BP0 | 4 | 67108864 |

sysindexes

| NAME | NLEAF | NLEVELS | BPOOL | PGSIZE | PIECESIZE | SPACEF | COMPRESS |
|----------|-------|---------|-------|--------|-----------|---------------------------------|----------|
| JOHNIIX1 | 380 | 3 | BP8K0 | 8 | 16777216 | +0.72000 00000000 000E+05 | N |
| JOHNIIX2 | 95 | 2 | BP32K | 32 | 4194304 | +0.72512 00000000 000E+05 | N |
| JOHNIIX3 | 380 | 3 | BP8K0 | 8 | 4194304 | +0.72000 00000000 000E+05 | N |
| JOHNIIX4 | 95 | 2 | BP32K | 32 | 4194304 | +0.72512 00000000 000E+05 | N |
| JOHNIIX5 | 129 | 3 | BP32K | 32 | 4194304 | +0.72000 00000000 000E+05 | Y |

Test was done with inserts of the same row over and over, then why a large space savings after REORG?



- DB2 10 NFM
- Table space was insert on the fly, last index was compressed
- The larger amount of space before REORG was due to preformat – up to 16 cylinders for DB2 9 and 10
 - Utilities use format write and only need to preformat one page, therefore all of the space savings

Table Space Sizes



- MYTH: Increasing my table and index page sizes will always help in regards to space and performance
- FACT: Table sizes can be 4K, 8K, 16K, or 32K
 - The maximum size for simple and segmented table spaces is 64G
 - The maximum size for UTS, partitions, XML, or LOB depends on the table space size and number of partitions (see next page)
 - Row length determines how much data is stored on each page
 - Table space pages can only hold 255 rows maximum. Indexes do not have this restriction.
 - When locking on the space map becomes an issue, fewer rows per page can be the result and some rows can be out of clustering order.
 - When locking becomes an issue, fewer rows per page can be specified in the DDL
 - Increasing the page size does not necessarily provide more space because of the 255 row per page limitation. Much of the data set can have wasted space as the table size increases.
 - Changing from a 4K to a 32K partitioned table space will increase the total size for a table from 16TB to 128TB, but may not add the capability to add many more rows because of the 255 row inhibitor.
 - There can be dramatic performance degradation when choosing the wrong page size. TEST, TEST, and TEST again! Test with different variations in page size.
 - Data set waste is also a problem with regards to buffer pools
 - Generally disk throughput increases as page size increases. 16K and 32K are treated the same as they have the same physical block size (16K).

Data Page Size

- Assuming insert only workload, use large data page size for sequential inserts to
 - Reduce # Getpages
 - Reduce # Lock Requests
 - Reduce # CF requests
 - Get better space use

Maximum size of a partition table

4K



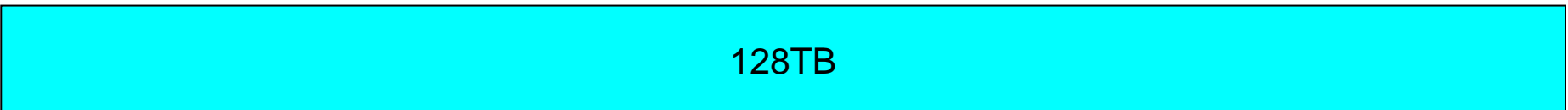
8K



16K



32K



DDL for illustration of 4K, 8K, 16K, 32K and row sizes of 60 bytes vs. 600 bytes



```
CREATE TABLESPACE JOHNI###  
  USING STOGROUP SYSDEFLT  
  PRIQTY      72000  
  SECQTY      7200  
  LOCKSIZE    ANY  
  CLOSE       NO  
  FREEPAGE    0  
  PCTFREE     0  
  
--  BUFFERPOOL BP0  
--  BUFFERPOOL BP8K0  
--  BUFFERPOOL BP16K0  
  BUFFERPOOL BP32K0;
```

```
CREATE TABLE JOHNI###  
  (FNAME      CHAR(20) NOT NULL,  
   LNAME      CHAR(20) NOT NULL,  
   MNAME      CHAR(20) NOT NULL)  
  IN DSNDB04.JOHNI###;
```

```
CREATE TABLE JOHNI###  
  (FNAME      CHAR(200) NOT NULL,  
   LNAME      CHAR(200) NOT NULL,  
   MNAME      CHAR(200) NOT NULL)  
  IN DSNDB04.JOHNI###;
```

systablepart



| TSNAME | PQTY | SQTY | CARD | PERCACTIVE | SPACE | AVGROWLEN |
|----------|-------|------|--------|------------|--------|-----------|
| JOHNITS1 | 18000 | 1800 | 512512 | 57 | 72000 | 66 |
| JOHNI18K | 18000 | 1800 | 512512 | 71 | 72000 | 66 |
| JOHNI116 | 18000 | 1800 | 512512 | 71 | 72000 | 66 |
| JOHNI132 | 18000 | 1800 | 512512 | 45 | 72512 | 66 |
| JOHNI600 | 18000 | 1800 | 512512 | 86 | 349920 | 606 |
| JOHNI28K | 18000 | 1800 | 512512 | 91 | 331200 | 606 |
| JOHNI216 | 18000 | 1800 | 512512 | 91 | 331200 | 606 |
| JOHNI232 | 18000 | 1800 | 512512 | 94 | 319616 | 606 |

- **PERCACTIVE:** Percentage of space occupied by active rows, containing actual data from active tables. This value is influenced by the PCTFREE and the FREEPAGE parameters on the CREATE TABLESPACE statement and by unused segments of segmented table spaces.

| NAME | BPOOL | PGSIZE | NACTIVE | AVGROWLEN |
|----------|--------|--------|---------|-----------|
| JOHNITS1 | BP0 | 4 | 14400 | 66 |
| JOHNI18K | BP8K0 | 8 | 5760 | 66 |
| JOHNI116 | BP16K0 | 16 | 2880 | 66 |
| JOHNI132 | BP32K | 32 | 2266 | 66 |
| JOHNI600 | BP0 | 4 | 87480 | 606 |
| JOHNI28K | BP8K0 | 8 | 41400 | 606 |
| JOHNI216 | BP16K0 | 16 | 20700 | 606 |
| JOHNI232 | BP32K | 32 | 9988 | 606 |

systables

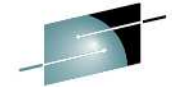
| TSNAME | CARD | NPAGES | PCTPAGES | RECLENGTH | AVGROWLEN |
|----------|--------|--------|----------|-----------|-----------|
| JOHNITS1 | 512512 | 8687 | 99 | 68 | 68 |
| JOHNI18K | 512512 | 4271 | 99 | 68 | 68 |
| JOHNI116 | 512512 | 2136 | 100 | 68 | 68 |
| JOHNI132 | 512512 | 2010 | 99 | 68 | 68 |
| JOHNI600 | 512512 | 85419 | 99 | 608 | 608 |
| JOHNI28K | 512512 | 39424 | 100 | 608 | 608 |
| JOHNI216 | 512512 | 19712 | 100 | 608 | 608 |
| JOHNI232 | 512512 | 9671 | 99 | 608 | 608 |

Difference between NACTIVE in SYSTABLESPACE on previous page and NPAGES in SYSTABLES on this page:

NACTIVE - Number of active pages in the table space. A page is termed active if it is formatted for rows, even if it currently contains none.

NPAGES - Total number of pages on which rows of the table appear.

PCTPAGES - Percentage of active table space pages that contain rows of the table. A page is termed active if it is formatted for rows, regardless of whether it contains any. If the table space is segmented, the percentage is based on the number of active pages in the set of segments assigned to the table. This example was of a segmented table space.



SHARE

| Object | BP | AVGRECLEN | CISIZE | Cylinders Allocated | HI-A-RBA | Cylinders Used | HI-U-RBA |
|----------|--------|-----------|--------|------------------------|-----------|-------------------|-----------|
| JOHNITS1 | BP0 | 68 | 4096 | 100 | 73728000 | 80 | 58982400 |
| JOHNI18K | BP8K0 | 68 | 8192 | 100 | 73728000 | 64 | 47185920 |
| JOHNI116 | BP16K0 | 68 | 16384 | 100 | 73728000 | 64 | 47185920 |
| JOHNI132 | BP32K | 68 | 32768 | 103 | 74252288 | 103 | 74252288 |
| JOHNI600 | BP0 | 608 | 4096 | 486 | 358318080 | 486 | 358318080 |
| JOHNI28K | BP8K0 | 608 | 8192 | 460 | 339148800 | 460 | 339148800 |
| JOHNI216 | BP16K0 | 608 | 16384 | 460 | 339148800 | 460 | 339148800 |
| JOHNI232 | BP32K | 608 | 32768 | 454 | 327286784 | 454 | 327286784 |

Final Thoughts

- Table spaces and indexes are not created equal. What works for one object may not work for another.
- There are many combination of options available
- Table spaces and indexes do not need to look alike, they do not necessarily require the same attributes.
- Choose the right type of table space and index up front
- Larger pages sizes do not necessarily provide much more space for your data set. Test performance implications of 4K vs. 8K vs. 16K vs. 32K. At times 16K might be the right choice, at times, 32K.
- Compare space savings vs. performance
- Compress your data sets for the right reasons
- **Most important of all – TEST, TEST, and verify your TESTs**

Critical UTS related APARS!

- PM51093
- PM45458
- PM55070
- PM56535
- PM58114

Before making you final choice, read on the web:



- DB2 for z/OS Optimizing Insert Performance – Parts 1 & 2 by John Campbell & Frances Villafuerte
- DB2 for z/OS Universal Table Spaces: The Whole Story by Willie Favero
- Realizing the Full Performance Potential of High Performance FICON with DB2 for z/OS by Jeff Berger
- Blogs:
- Robert Catterall's blog: <http://robertsdb2blog.blogspot.com/2010/12/reorg-and-db2-for-zos-partition-by.html>
- Willie Favero's blog: <http://it.toolbox.com/blogs/db2zos/be-careful-how-you-define-your-partitionbygrowth-universal-table-space-50164>