Partitioning in DB2 V8 for z/OS – The Good, The Bad and The Ugly

Steen Rasmussen
Senior consultant @ CA

Geel, March 22, 2007

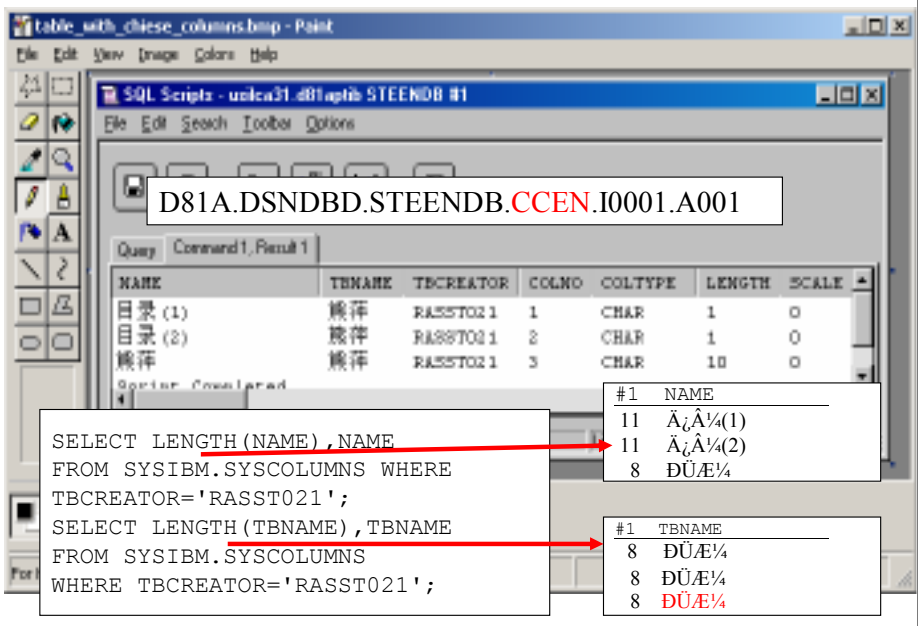
Abstract

- DB2 V8 introduces a completely new way of thinking about partitioning. This is a case study of everything that's worth knowing to administer DB2 V8 table partitioning – covering SQL, Commands and Utilities as well as physical and logical partitions.

Agenda

- The Good, the Bad and the Ugly about partitioning prior to V8 came along
- What is new in V8 partitioning
- The case study – let's look at the real thing
  - DDL, Commands, Utilities
  - Catalog changes for V8 partitioning
- Questions and Answers – but first UNICODE

UNICODE – new challenges



In DB2 V8 the catalog is UNICODE. This opens a completely new world since it is possible to create objects with national characters.

The challenge is, that TSO and ISPF does not support UNICODE so the translation to EBCDIC when using TSO to look into the catalog can make it difficult to recognize the objects names created.

In this example, a Windows application was used to create a table with Chinese characters and Chinese column names.

The select statement illustrates the conversion of UNICODE to EBCDIC in TSO and it also illustrates the 2-Chinese character TBNAME is 8 byte in the catalog even though we only can count 2 characters i.

Also notice how the UNICODE characters are translated into EBCDIC.

DB2 V8 UNICODE

```
SELECT DBNAME,NAME FROM SYSIBM.SYSTABLESPACE
WHERE DBNAME ='PTDB' AND NAME LIKE 'PTG%' ORDER BY NAME;
```

DBNAME	NAME	
PTDB	PTG300UH	DB2 V7
PTDB	PTG500TS	
PTDB	PTG500T2	
DBNAME	**NAME**	
PTDB	PTG300UH	DB2 V8
PTDB	PTG500T2	
PTDB	PTG500TS	

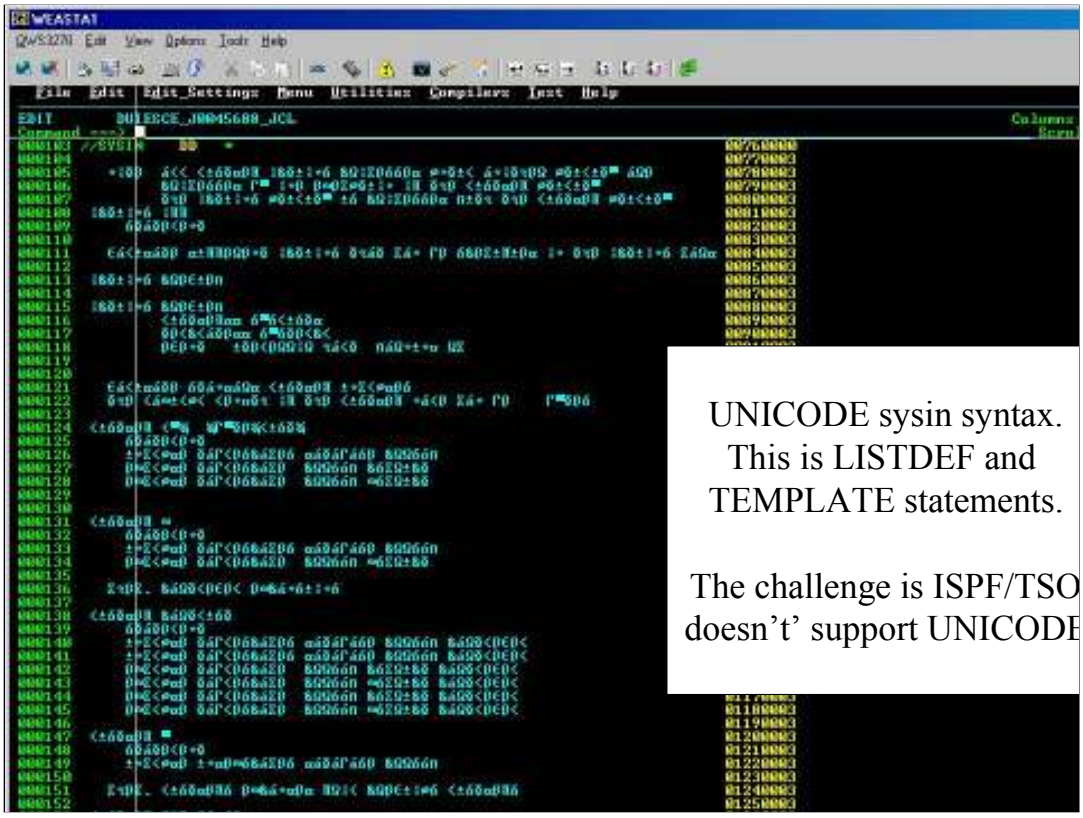
CCSID can be assigned during BIND or by using a special register. Use CAST in SQL statements to get EBCDIC.

5

Another important issue to have in mind once the DB2 catalog is in UNICODE – every feature where DB2 will have to do a sort or comparison – the result MIGHT be different once you are on V8.

The above example illustrates that alphabetic characters are sorted prior to numbers while the opposite is the case in UNICODE.

This will indeed influence ANY application using SQL where the content is a mix of characters, numbers, national characters and special symbols. Just think of BETWEEN, ORDER BY, GROUP BY, <, >, ……



UNICODE sysin syntax. This is LISTDEF and TEMPLATE statements.

The challenge is ISPF/TSO doesn't support UNICODE

The Good, the Bad.....

- Manageability
  - Smaller pieces to administer – storage management
  - Shorter housekeeping outage (copy, reorg, recover,....) when executed on partition level
- Performance
  - More utilities executed in parallel
  - SQL select exploitation of parallelism
  - Batch jobs can execute in parallel
- NPI's
  - Utilities suffer processing huge NPI's
  - Not only the partition touched will be unavailable
    - Partition load will cause outage for all NPI's
    - Partition recover not any better
    - Reorg partition will cause BUILD2 phase – and NPI fragmentation

7

So far most the main reason for partitioning objects has been due to the amount of data. Many smaller VSAM datasets make many daily tasks easier and quicker to perform like imagecopy, reorg etc.

Also – its hard to predict how long a recover or a reorg of a 4000 cylinder tablespace takes, while it's a lot easier to predict if no VSAM dataset is greater than e.g. 100 cylinders. If one utility fails, late in the phase, restarting the utility will not have a major impact if we compare a 100 cylinder object with one of 4000 cylinders.

Having many partitions allows us to execute batch jobs in parallel if a parameter can control which partitions to operate on.

The DB2 Optimizer is very clever at exploiting parallelism when being enabled, and partitioned objects tend to better exploit this feature.

Most utilities can execute in parallel too, allowing the utility to finish faster.

The "problem child" is the NPI's. These indexes are often huge making the BUILD2 phase of a reorg execute for a long time. Recovering or reorganizing a NPI can be a huge challenge due to the size – and as mentioned earlier – if it fails late in the phase, we're having another long outage.

..... and the really Ugly

- Partition Load Replace:

```
LOAD DATA REPLACE
INTO TABLE RASST02.MYTABLE PART 1
```

```
LOAD DATA
INTO TABLE RASST02.MYTABLE PART 1 REPLACE
```

- REPLACE can be specified in two places
  - Syntax accepted by DB2
  - Job finishes very fast
  - TWO completely different results
  - Ooooooops.....

8

My personal nightmare with partitions is this one.

If a partition needs to be emptied (LOAD REPLACE with a SYSDUMMY DD-card) or a partition needs to be replace with new data – it really matters where the replace keyword is placed.

The first statement will replace the entire tablespace with the data residing in the input dataset – meaning if only ONE partition was intended to be replaced, the entire tablespace will be wiped out.

The second LOAD statement will ONLY replace the partition mentioned after the PART keyword.

What is New in V8 Partitioning

- Part of "Online Schema Evolution"

- **ALTER without having to DROP:**
  - Change/drop the CLUSTERING index
  - Change/drop the PARTITIONING index
- Now 2 different types of partitioning :
  - Index partitioning (been here for years)
  - Table partitioning (the new kid on the block)
  - ADD and ROTATE partitions
  - DPSI (Data Partition Secondary Index)
  - No indexes needed – CLUSTER not needed
  - Switch from index to table partitioning – not visa versa
    - Converts from INDEX partitioned to TABLE partitioned
    - Any ALTER only valid for "new table partitioning"
    - DROP INDEX , ROTATE PARTITION, ADD PART
    - Can not convert back again
    - Might end in REORP if LARGE not used

9

All the new statements implemented in V8 for partitioned objects are part of what is named “Online Schema Evolution”, where attributes can be altered without dropping and re-creating the object.

CREATE TABLESPACE

- Create tablespace for V8 table based partitioning identical to V7

```
CREATE TABLESPACE PART9TS
    IN STEENVOL
    NUMPARTS   9
   (PART 1 USING STOGROUP SYSDEFLT
            PRIQTY 88 ERASE NO
            FREEPAGE 0  PCTFREE 5     ) BUFFERPOOL BP1
            LOCKSIZE ANY  LOCKMAX SYSTEM   CCSID UNICODE  ;
DSNT400I SQLCODE = 000,  SUCCESSFUL EXECUTION
```

- If 4096 partitions – a lot of typing, BUT
  - Only necessary to specify ONE partition
  - PRIQTY and SECQTY picked up from previous part, default OR
  - From new ZPARM MGEXTSZ if enabled
    (QTY columns in SYSTABLEPART and SYSINDEXPART will have -1)

10

The syntax for a partitioned tablespace in DB2 V8 is identical to how the tablespace was created prior to DB2 V8.

Consider 4096 partitions which is the maximum of partitions allowed in DB2 V8, typing all the keywords for all 4096 partitions might take days. Most DB2 users are not aware they only need to specify the attributes for the first partition (for STOGROUP defined objects). Based upon the NUMPARTS attribute, DB2 will allocate the partitions not explicitly specified in the create statement using the default specified in DSNZPARM.

DB2 V8 offers a new feature MGEXTSZ (Smart Managed Extent Size) which basically increases the SECQTY when too many extents are allocated to make sure the object doesn't run into an out-of-space condition too fast. When this parameter is utilized, a value of -1 (minus 1) can be seen in the QTY columns in SYSTABLEPART and SYSINDEXPART.

CREATE TABLE – NEW SYNTAX

LIMITKEY and columns defining partitions specified on **create table**. *(Must be in DB2 V8 NFM mode – otherwise SQL - 4700).*

What is wrong with the sequence of the limitkeys ?

```
CREATE TABLE RASST02.TBPART9
  (COL01     CHAR(10  )     NOT NULL WITH DEFAULT
  ,COL02     DECIMAL(9,2)  NOT NULL WITH DEFAULT
  ,COL03     INTEGER
  ,COL04     DATE
  ,COL05     VARCHAR(120  )   )
  PARTITION BY (COL01 ASC , COL02 DESC)
 ( PART 1 VALUES ( '1000000000' )
  ,PART 2 VALUES ( '5000000000' )
  ,PART 3 VALUES ( 'D999999999' )
  ,PART 4 VALUES ( 'H999999999' )
  ,PART 5 VALUES ( 'R999999999' )
  ,PART 6 VALUES ( 'T999999999' )
  ,PART 7 VALUES ( 'V999999999' )
  ,PART 8 VALUES ( 'X100000000' )
  ,PART 9 VALUES ( 'X200000000' )       )
   IN STEENVOL.PART9TS
   DATA CAPTURE CHANGES CCSID UNICODE ;
```

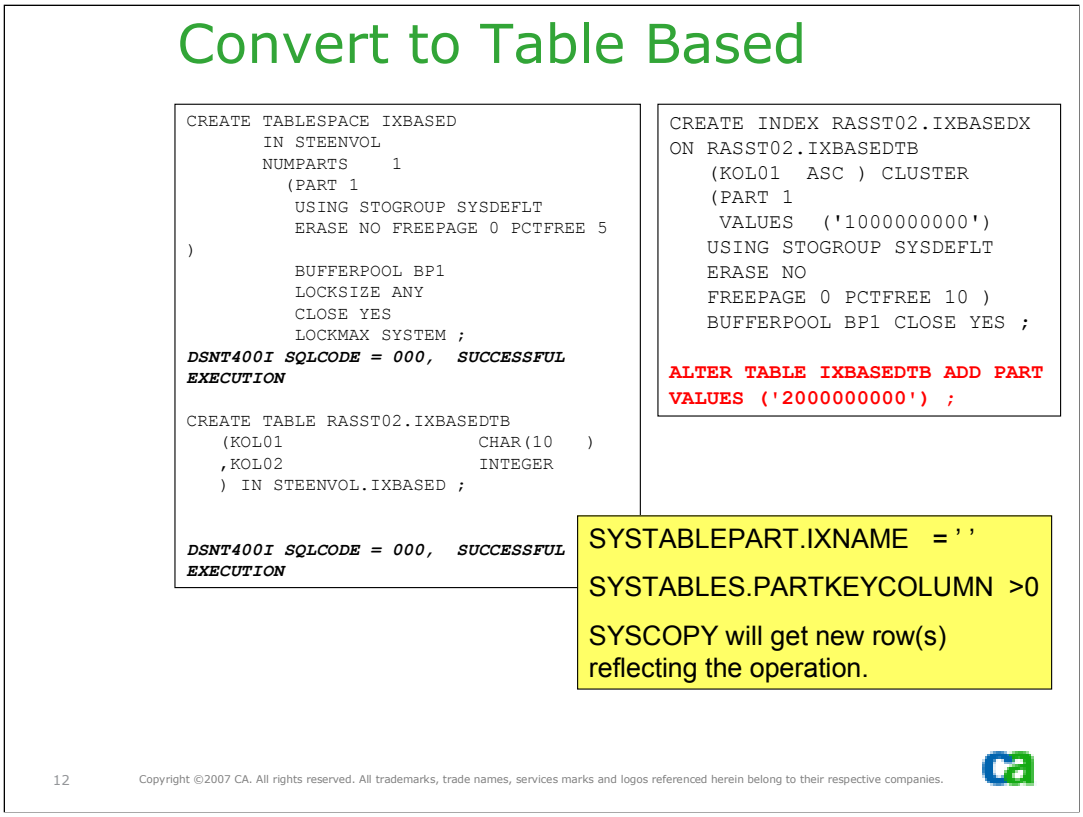
11

The CREATE TABLE statement has the biggest changes when table based partitioning is chosen. The first new parameter is PARTITION BY which describes the columns dictating the partition limits (this used to be the clustering partitioned index columns).

The PART x VALUES (limitkey value) is no different from what used to be specified on the partitioned index.

Notice the sequence of the limitkeys : Since this table is being defined to have UNICODE data, the UNICODE representation of numbers is smaller than the letters in the alphabet, which is the reverse situation compared to EBCDIC.

Also – the new features for table based partitioned objects are only available in DB2 V8 NFM (New Function Mode) – when used in Compatibility mode or enabling new function mode, the DDL statement will fail.



If index based partitioning is used, these objects can be altered to be table based partitioned objects no matter if the objects are created under DB2 V7 or DB2 V8.

The method to convert from index based to table based is very simple. When any of the new DB2 V8 ALTER statements for partitioned objects is executed, DB2 will convert from index based to table based partitioning.

Theses statements are : ADD partition, ROLL partition, DROP CLUSTER index, DROP PARTITIONED index.

One way to see if an object is table based partitioned is to look into the catalog:

1) TABLE based partitioned objects have a blank value in SYSIBM.SYSTABLEPART.IXNAME, while INDEX based partitioned objects have the actual partitioned index name in this column.

2) For TABLE based partitioned objects,column PARTKEYCOLUMN has a value greater than ZERO.

If partitions have been ADD'ed or ROTATE'd, this can be viewed in SYSCOPY, unless the entries have been removed using MODIFY (we will get back to this later).

CREATE TABLE (cont'd)

- Current status:
  - Tablespace and table created
  - Limitkeys specified on create table
  - No indexes exist (neither cluster nor partition index)
  - Table is fully operational and in a COMPLETE status

```
INSERT INTO RASST02.TBPART9
        (COL01,COL02,COL03,COL04,COL05)
VALUES  ('1234567890',
         12000 , 453 , '2004-12-15' ,
         'ARNOLD FROM CALIFORNIA' ) ;

DSNT400I SQLCODE = 000,SUCCESSFUL EXECUTION
```

  - To verify no index exists – let's have a look at VSAM datasets before digging further into the DDL changes

13

At this point we have created a partitioned tablespace, a table based partitioned table with the limitkeys defined. No indexes exist at this point, which in DB2 V7 would result in the object having an INCOMPLETE status.

When table based partitioned is chosen, the table is ready to have data loaded or inserted – no indexes are necessary.

In order to verify that no indexes actually exist, we will have a look at the VSAM datasets allocated to the database (next page).

VSAM changes in DB2 V8

- Only tablespace VSAM datasets exist

```
DSLIST - Data Sets Matching D81A.DSNDBD.STEENVOL.
Command ===>

Command - Enter "/" to select action     Tracks %U
-------------------------------------------------
D81A.DSNDBD.STEENVOL.PART9TS.I0001.A001       2
D81A.DSNDBD.STEENVOL.PART9TS.I0001.A002       2
D81A.DSNDBD.STEENVOL.PART9TS.I0001.A003       2
D81A.DSNDBD.STEENVOL.PART9TS.I0001.A004       2
D81A.DSNDBD.STEENVOL.PART9TS.I0001.A005       2
D81A.DSNDBD.STEENVOL.PART9TS.I0001.A006       2
D81A.DSNDBD.STEENVOL.PART9TS.I0001.A007       2
D81A.DSNDBD.STEENVOL.PART9TS.I0001.A008       2
D81A.DSNDBD.STEENVOL.PART9TS.I0001.A009       2
************************** End of Data Set list
```

Now up to 4096 partitions - new VSAM dataset naming needed:
==========

vcat.dsndbd.db.space.I0001.**Axxx**

PART 0001 -> 0999	**A001 -> A999**
PART 1000 -> 1999	**B000 -> B999**
PART 2000 -> 2999	**C000 -> C999**
PART 3000 -> 3999	**D000 -> D999**
PART 4000 -> 4096	**E000 -> E096**

- Do NOT forget any internal SMS routines or any other homegrown processes working on VSAM level like DSN1COPY,.....

14

Note that only the tablespace VSAM datasets exist at this time – no indexes and INSERTs are possible.

DB2 V8 offers a maximum of 4096 partitions. Since the low level node only has three numeric digits following the A, a new naming convention is present (please see the frame above).

This might not be a big issue, but consider if any SMS routines need to be corrected or any homegrown routines only operate on Axxx datasets (REXX programs, DSN1COPY, DELETE-DEFINE,……..)

V8 Partitioning

- Not always 4096 partitions
  - Dependent on DSSIZE and LOB's

PAGESIZE DSSIZE	4KB	8KB	16KB	32KB
1GB – 4GB	4096	4096	4096	4096 (4–16TB)
8GB	2048	4096	4096	4096 (32TB)
16GB	1024	2048	4096	4096 (64TB)
32GB	512	1024	2048	4096 (128TB)
64GB	256	512	1024	2048 (128TB)

  - Maximum number of objects in one DB=65535
  - Think about LOB columns
    - Each LOB column : AUXTS + AUXTB + AUXIX
    - If 6 LOB columns ➔ (6 x 4096 x 3 datasets)+4096 base=77770 datasets
    - Maximum of 4 LOB columns in 4096 partitions if **no** indexes

15

As already mentioned the maximum of partitions in DB2 V8 is 4096, but it really depends on the PAGE SIZE and DSSIZE specified for the tablespace.

Another factor which can limit the ability to ADD partitions is if LOB columns are used on the base table. Every LOB column require an auxiliary tablespace, table and index.

DB2 Command output

- A few changes to DB2 Command output
  - 4096 partitions – three bytes not enough anymore
  - New index types and new status (covered later)

```
PTPDBCDD  R11    ---    DB2 Command Processor    ----      2004/12/15   07:52
COMMAND  ==>                                               SCROLL ==> CSR
-------------------------------------------------------  User ID: RASST02
¦ -DIS DB(STEENVOL)                                        ¦
¦                                                          ¦
****************************** TOP OF DATA ********************************
DSNT360I  !D81A ***********************************
DSNT361I  !D81A *  DISPLAY DATABASE SUMMARY GLOBAL
DSNT360I  !D81A ***********************************
DSNT362I  !D81A DATABASE = STEENVOL  STATUS = RW     DBD LENGTH = 16142
DSNT397I  !D81A
NAME     TYPE  PART STATUS            PHYERRLO PHYERRHI CATALOG  PIECE
-------- ---- ----- ------------------ -------- -------- -------- -----
PART9TS  TS    0001 RW
   -THRU       0009
```

  - Adjacent partitions with same status are grouped together – and tied by ”-THRU” keyword

Since the maximum number of partitions has gone up from 254 to 4096, the output from DB2 commands need to reflect this as well.

To make it a lot easier to see the status of partitions, a new keyword is part of the output from DISPLAY. The new word is THRU, which is used to tie all adjacent partitions having the same status. The example above illustrates partition 1 through partition 9 all have a status of RW (Read-Write).

Also note the partition number is 4 byte instead of 3.

Just like its necessary to consider homegrown routines to handle the new VSAM dataset naming convention, it is necessary to consider any applications/programs looking at the output from DB2 Commands, to deal with the 4-byte partition number and the keyword “THRU” (more news to come later in this presentation).

DB2 Command output (cont'd)

- No matter what happens – partitions with identical status are grouped via THRU

```
PTPDBCDD  R11    ---    DB2 Command Processor    ----      2004/12/15   07:52
COMMAND  ==>                                                SCROLL ==> CSR
------------------------------------------------------------  User ID: RASST02
¦ -STA DB(STEENVOL) SPACE(PART9TS) PART(4) ACCESS(RO)
¦ -DIS DB(STEENVOL)
****************************** TOP OF DATA ***********************************
DSNT360I  !D81A **********************************
DSNT361I  !D81A *  DISPLAY DATABASE SUMMARY GLOBAL
DSNT360I  !D81A **********************************
DSNT362I  !D81A DATABASE = STEENVOL  STATUS = RW     DBD LENGTH = 16142
DSNT397I  !D81A
NAME     TYPE  PART STATUS            PHYERRLO PHYERRHI CATALOG  PIECE
-------- ---- ----- ----------------- -------- -------- -------- -----
PART9TS  TS 0001 RW
    -THRU   0003
PART9TS  TS 0004 RO
PART9TS  TS 0005 RW
    -THRU   0009
```


Even though a partition in the middle has a different status, the DB2 Command output will group adjacent parttitions together having the same status.

This example is stopping partition 4 in Read-Only mode, so partition 1 through (THRU) 3 are grouped together and the same is the case for partition 5 through 9.

ROLL PARTITION

- More than one partition can be rolled in one UOW but only one in each statement

```
ALTER TABLE TBPART9 ALTER PART
      ROTATE FIRST TO LAST VALUES ('X300000000') RESET ;
DSNT400I SQLCODE = 000,  SUCCESSFUL EXECUTION
ALTER TABLE TBPART9 ALTER PART
      ROTATE FIRST TO LAST VALUES ('X400000000') RESET ;
DSNT400I SQLCODE = 000,  SUCCESSFUL EXECUTION
```

- The DISPLAY output changes after ROLL - ordered by **LOGICAL** partition number *(limitkey sequence)*

```
NAME     TYPE  PART STATUS
-------- ---- ----- -----------
PART9TS  TS   0003 RW
PART9TS  TS   0004 RO
PART9TS  TS   0005 RW
    -THRU     0009
PART9TS  TS   0001 RW
    -THRU     0002
```

18

In DB2 V8 it is possible to ROLL a partition.

Each ALTER statement can ROLL one partition, but many ALTER statements can exist within one unit-of-work (UOW).

When doing the ROLL (another term is ROTATE), a new limitkey must be specified for the partition which is “moved” from being the lowest partition to become the highest partition. The limitkey must be higher than the current high-limitkey.

Note the DISPLAY output. After two ROLL’s the display output is not in ascending sequence anymore. The display output for a tablespace always displays the partitions in LIMITKEY sequence, so since two partitions are ROLL’ed / ROTATE’d, the “new” lowest partition number is VSAM dataset 3. Let’s dig into the details and what DB2 is really doing during a ROLL / ROTATE.

ROLL PARTITION

- The partition with the lowest LIMITKEY is rolled to become the partition with the highest LIMITKEY (which has to be specified during ALTER)
- All rows are deleted from the "old" partition – and so will recovery information (SYSCOPY , SYSLGRNX) – *(we will get back to this..)*
- Time for some confusion:
  - DB2 does not rename VSAM datasets
  - Used to have easy match of partition number and corresponding VSAM dataset (.A001 always PART 1)
  - Introducing **LOGICAL partition** in V8
  - After ROLL, VSAM dataset .A001 (part 1) can be a complete different partition (first ROLL for 9-PART tablespace, .A001 will host the new high partition number while .A002 will be host the new low logical partition number).....
- Restrictions / Warnings
  - If table rotated has PK or data capture changes enabled – or delete trigger exist – rows are deleted individually – otherwise ”DELETE/DEFINE” is done
  - DELETE RESTRICT may cause failure

19

Prior to the first roll, VSAM dataset *.A001 holds the lowest limitkey. After the first ROLL, VSAM dataset *.A001 will be the new HIGHEST limitkey, since DB2 does not rename the dataset.

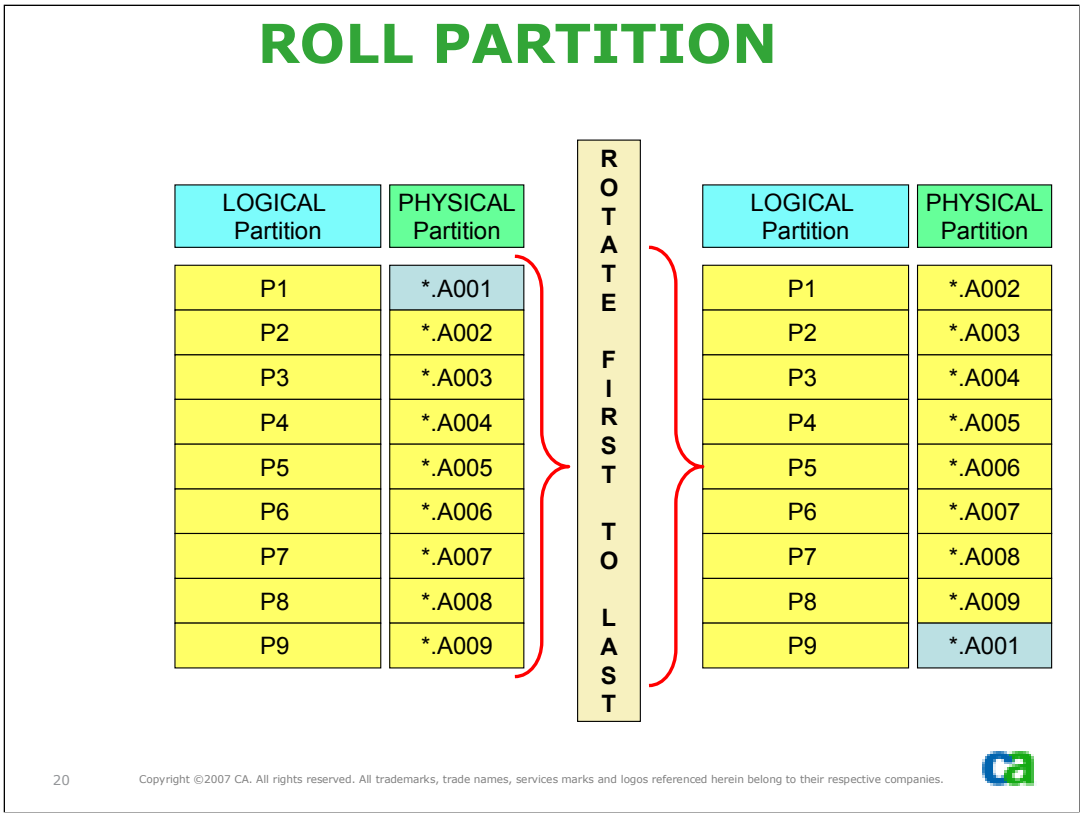
Until now – we have had a match between the PARTITION NUMBER and the VSAM dataset. DB2 V8 introduces a new term : LOGICAL PARTITION.

So – the PHYSICAL partition is the VSAM dataset while the LOGICAL partition refers to the LIMITKEY. Meaning – LOGICAL PARTITION 4 has the limitkey defined for the fourth data partition, and this logical partition 4 could be any PHYSICAL partition (VSAM dataset) depending on how many partitions have been rotated.

DB2 will remove EVERY row from the partition being ROLL’ed, so make sure data is unloaded prior to the ROLL if the data must be accessed at a later point in time.

If the table has DATA CAPTURE CHANGES enabled, every row will be recorded in the LOG – otherwise not. The same goes for tables having a Primary Key defined.

In certain cases, the ROLL might not be possible at all. If a RI-relationship exists with a DELETE RESTRICT rule, the ROLL will fail if dependent child rows exist.



This figure illustrates the first ROLL process:

Logical partition 1 starts up with having physical partition 1 (VSAM dataset) assigned.

The first ROLL will :

1) delete rows from VSAM dataset 1 (physical partition)
2) assign a new HIGHEST limitkey for the rotated partition
3) The LOWEST limitkey after the ROLL is the LOGICAL PARTITION 1.
4) LOGICAL PARTITION 1 will have PHYSICAL PARTITION 2 (VSAM dataset assigned).
5) The highest limitkey after the roll is logical partition 9.
6) Logical partition 9 (highest limitkey) has physical partition 1 (VSAM dataset assigned).

ROLL PARTITION – SYSCOPY changes

- After two ROLL's – imagecopy taken for entire tablespace + per partition.
  - SYSCOPY information ordered by DSNUM and START_RBA

DSNUM	ICTYPE	STYPE	OLDEST_VERSION	LOGICAL_PART
0	F		0	0
1	A	R	0	9
1	F		0	8
2	A	R	0	9
2	F		0	9
3	F		0	1
4	F		0	2
5	F		0	3
6	F		0	4
7	F		0	5
8	F		0	6
9	F		0	7

DSNUM=1 used to be PART 1 has had an alter (ICTYPE=A) via roll (STYPE=R) and was changed to be logical part=9 (highest limitkey).

When the full copy was taken the logical part was 8 – meaning another roll had taken place. This can be confirmed since DSNUM=2 has STYPE=R.

  - ROLL will have X-lock(s) on SYSCOPY
  - The "new" partition is available immediately to have data inserted (with the usual online schema restrictions)

21

SYSCOPY holds a lot of information regarding the sequence of operations.

The output above shows the content of SYSCOPY for the following sequnce of operations:

1) The second row shows: DSNUM=1 (physical partition) with STYPE=R (ROLL is taking place) has logical partition=9 (the highest partition of the table)
2) The third row shows DSNUM=1 with logical partition=8 had an imagecopy taken. Combining the two rows related toDSNUM=1, we can see this partition was rolled due to logical part decreased by 1.
3) DSNUM=2 was rolled too, and had an imagecopy taken. The logical partition is 9 for both the ROLL and imagecopy, and no other rows has logical partition=9 – so the conclusion for this tablespace is, TWO ROLL's were done and then imagecopies were taken on the partition level as well as one imagecopy for the entire tablespace.

4) Bear in mind, a ROLL will take exclusive LOCK(s) on SYSCOPY which might cause contention with other operations.

ROLL PARTITION - RECOVER

- Recover to a RBA prior to a ROLL is NOT possible.

According to the documentation, SYSCOPY info from rolled partitions should be removed…(document change from IBM)
Then RECOVER should not be possible (ALTER found in SYSCOPY...)

```
DSNU050I  DSNUGUTC - RECOVER TABLESPACE ETR.PART5TS DSNUM 1 TORBA X'0026498E1CE5'
DSNU515I  DSNUCBAL - THE IMAGE COPY DATA SET RASST02.PART5TS.P1.D041228.T142052 WI
                     IS PARTICIPATING IN RECOVERY OF TABLESPACE ETR.PART5TS DSNUM
DSNU504I  DSNUCBMD - MERGE STATISTICS FOR TABLESPACE ETR.PART5TS DSNUM 1 -
                             NUMBER OF COPIES=1
                             NUMBER OF PAGES MERGED=3
                             ELAPSED TIME=00:00:00
DSNU1511I !D81A DSNUCALA - FAST LOG APPLY WAS NOT USED FOR RECOVERY
DSNU1510I DSNUCBDR - LOG APPLY PHASE COMPLETE, ELAPSED TIME = 00:00:00
DSNU500I  DSNUCBDR - RECOVERY COMPLETE, ELAPSED TIME=00:00:01
DSNU050I  DSNUGUTC - RECOVER TABLESPACE ETR.PART5TS DSNUM 2 TORBA X'0026498E1CE5'
DSNU556I  !D81A DSNUCASA - RECOVER CANNOT PROCEED FOR TABLESPACE ETR.PART5TS DSNUM2
                  BECAUSE A SYSIBM.SYSCOPY RECORD HAS BEEN ENCOUNTERED WHICH HAS
                  DBNAME=ETR TSNAME=PART5TS DSNUM=2 ICTYPE=A
                  STARTRBA=X'002649998AC5' LOWDSNUM=0 HIGHDSNUM=0
DSNU500I  DSNUCBDR - RECOVERY COMPLETE, ELAPSED TIME=00:00:00
DSNU012I  DSNUGBAC - UTILITY EXECUTION TERMINATED, HIGHEST RETURN CODE=8
```

22

According to the documentation, a ROLL should clean up SYSCOPY information.

If clean-up was performed, RECOVER should not be able to tell an ICTYPE=A (alter) was performed.

Anyway – recover to a point-in-time prior to a ROLL cannot be done.

ADD PARTITION

- New partitions can be added and data can be inserted immediately.

```
ALTER TABLE TBPART9 ADD PART VALUES ('X600000000') ;
 DSNT400I SQLCODE = 000,  SUCCESSFUL EXECUTION
ALTER TABLE TBPART9 ADD PARTITION ENDING AT ('X700000000');
 DSNT400I SQLCODE = 000,  SUCCESSFUL EXECUTION
```

- No QTY specified for the new partitions
  - Allocation picked up from previous partition
  - Adding PART 10 will pick up from PART 9 ???
  - …. Or from PART 4 (due to three ROLL's) ???
  - Remember to ALTER after ADD if bulk data expected and the previous logical partition is small
    (SMART managed extent sizes another option [-1 in SECQTY in the catalog])

23

Beside ROLL'ing partitions, it is also possible to ADD new partitions to a partitioned object (can not convert from segmented to partitioned).

Just like the ROLL procedure, adding a new partition a new high limitkey must be specified.

Note the ALTER ADD PARTITION works on the table level, so it is not possible to specify any PRIQTY/SECQTY for the tablespace.

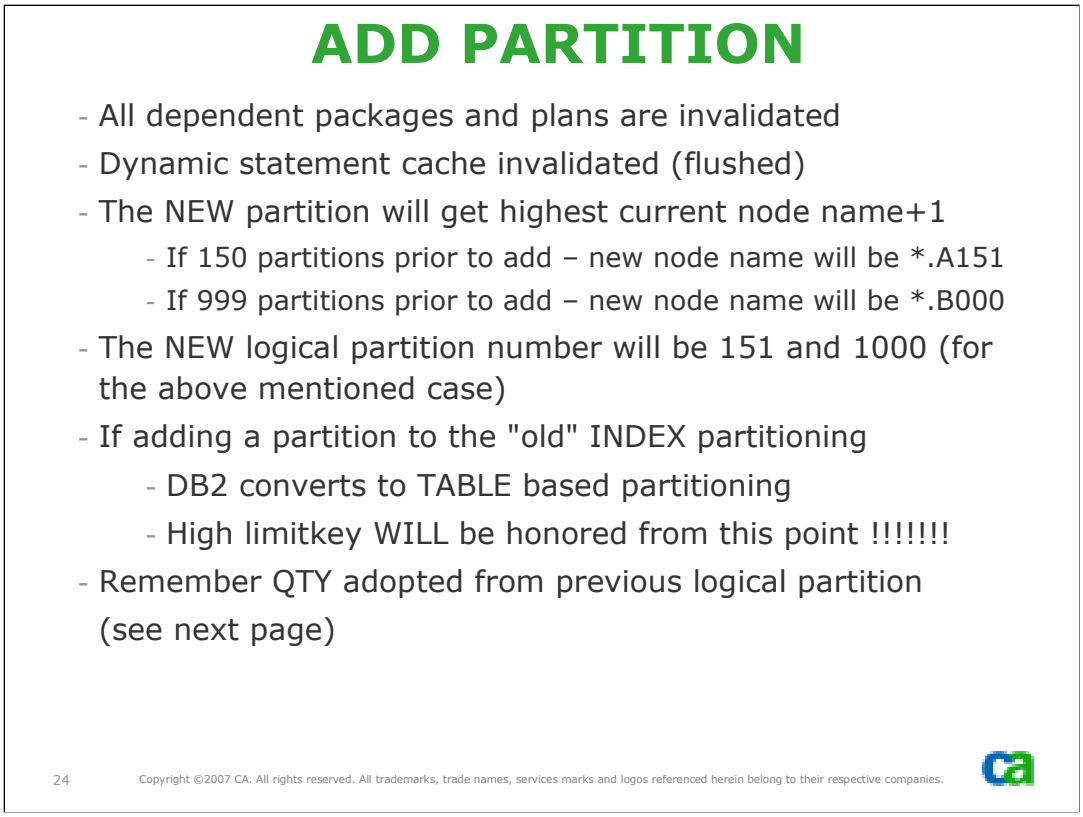
The QTY is picked up from the previous partition – but if a partition has been rolled prior to adding the new partition – is the previous partition the

a) Previous physical partition (VSAM dataset) or..

b) Previous logical partition (the second highest limitkey)

The QTY is picked up from the previous LOGICAL partition.

If bulk data is expected to be loaded into the new partition, it is advisable to alter PRIQTY of the newly added partition. This is not so important if MGEXTSZ in DSNZPARM has been enabled (Smart Managed Extent Size) since DB2 will dynamically increase SECQTY when extents are taken.



Just like all the other features in “Online Schema Evolution”, dependent packages will be invalidated and the statement cache is flushed - when adding a partition.

Adding a new partition will get the next available node name, so if 150 partitions existed prior to the add, the new partition will get *.A151. If the partition being added is partition number 1000, the logical partition 1000 will get VSAM dataset *.B000.

The early documentation describes only one partition can be added in one UOW. Adding two partitions and committing goes well, but if a rollback is issued after adding more than one partition, DB2 will abend (unless the mentioned PTF is applied).

COMMAND output

- Commands work on PHYSICAL partitions but lists in LOGICAL sequence
  (Physical part 6 is logical part 2 in this case)
- Interpreting output from display command might require a little more time in the future.
- Just wait until indexes are involved.....

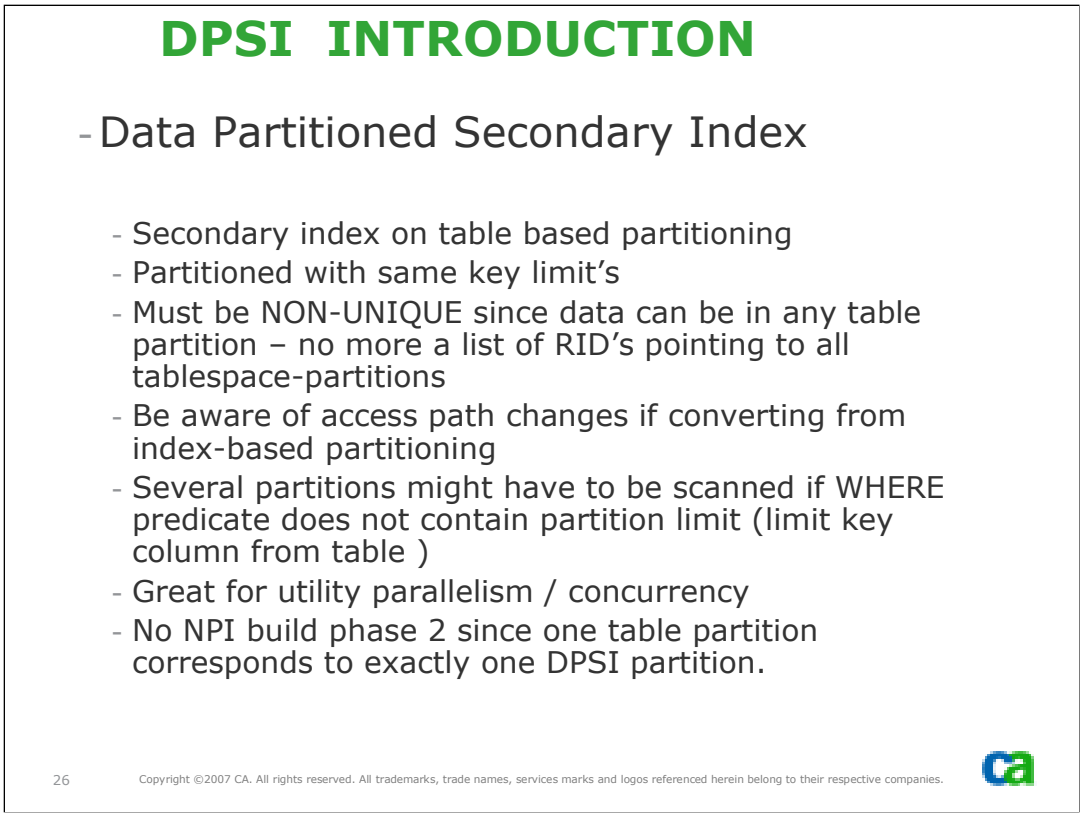
```
-STA DB(STEENVOL)
 SPACE(PART9TS)
 PART(6) ACCESS(RO)

NAME     TYPE PART   STATUS
-------- ---- ----- -------
PART9TS  TS    0005 RW
PART9TS  TS    0006 RO
PART9TS  TS    0007 RW
    -THRU      0009
PART9TS  TS    0001 RW
    -THRU      0003
PART9TS  TS    0010 RW
    -THRU      0011
PART9TS  TS    0004 RW
PART9TS  TS    0012 RW
```

25

Even though all partitions have the same status, once partitions have been rolled and added, the output from a DISPLAY command grows. The reason is the tablespace partitions are listed in LOGICAL (LIMITKEYT) sequence.

The listed command starts partition 6 in READ-ONLY mode, and by looking at the output, we can conclude the DB2 COMMANDs are working on the PHYSICAL partition level since the partition in RO-mode is the LOGICAL PARTITION 2.

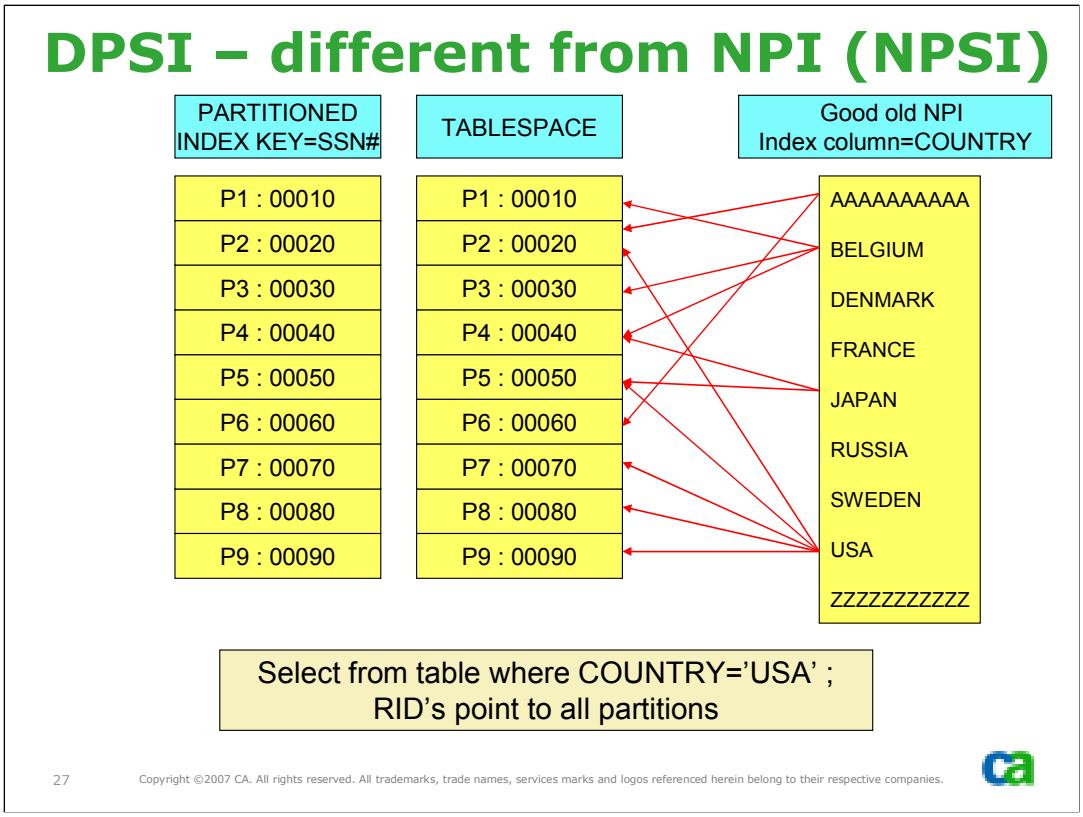


DB2 V8 offers a new index type – Data Partitioned Secondary Index (aka DPSI).

As the name says – it is an index, partitioned the same way the data is. Meaning – each index partition holds data for the corresponding table partition.

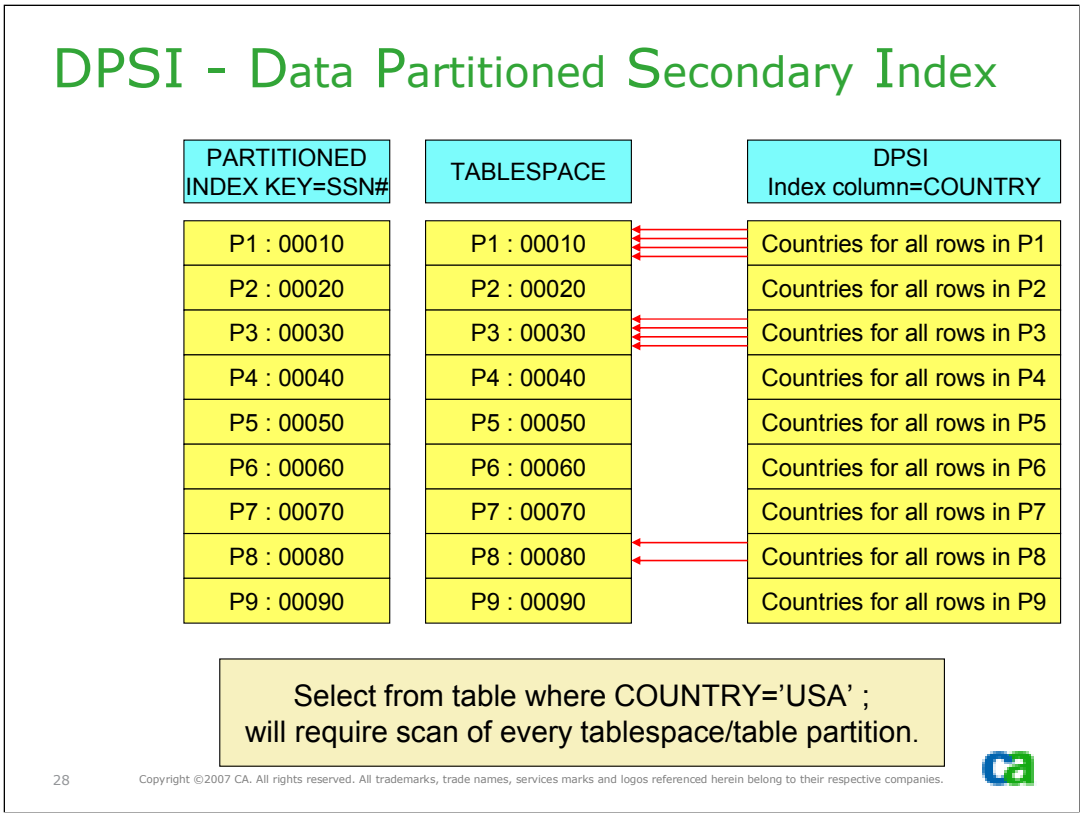
A DPSI can NOT be unique due to it's nature. All index keys holds RID's for exactly one table partition. If a SQL statement does not contain predicates limiting the partition it references, DB2 will scan the entire object.

The MAJOR advantage having a DPSI instead of the old NPI is, the BUILD2 phase of reorg is eliminated. Every utility executing on the partition level will only have to maintain the DPSI partition(s) reflecting the table partitions being operated on.



This picture illustrates how the Non Partitioned Index (NPI) looks like. The NPI is one big index with all the KEYS having a list of RID's pointing to the data in the tablespace.

When the DB2 Optimizer decides to use the NPI for the access path, the KEY(s) are located in the NPI and a list of RID(s) are pointing to the data to be retrieved.



A DPSI is very different in nature compared to a NPI.

Every Index entry only holds RID's for the corresponding data partition.

For performance reasons, queries should be considered to include predicates on both the DPSI column(s) and the partition limit key. In this example:

Select from table where COUNTRY='USA'

and SSN# between '00070' and '00080' ;

Eliminating predicates which reflect the limitkey, will cause DB2 to scan the entire table which might lead to a huge performance degradation compared to using a NPI.

TABLE PARTITIONING - INDEXES

- Create DPSI index:

```
CREATE UNIQUE INDEX RASST02.PART9XD2
       ON RASST02.TBPART9
      (COL05 DESC)
       USING STOGROUP SYSDEFLT
       PRIQTY 48  SECQTY 96  ERASE NO
       BUFFERPOOL BP1 CLOSE YES  PARTITIONED ;
```

UNIQUE will cause SQL-628 – mutually exclusive with PARTITIONED .

- All DPSI partitions created…….

```
D81A.DSNDBD.STEENVOL.PART9XD2.I0001.A001          1    ?   1  3390
D81A.DSNDBD.STEENVOL.PART9XD2.I0001.A002          1    ?   1  3390
D81A.DSNDBD.STEENVOL.PART9XD2.I0001.A003          1    ?   1  3390
D81A.DSNDBD.STEENVOL.PART9XD2.I0001.A004          1    ?   1  3390
D81A.DSNDBD.STEENVOL.PART9XD2.I0001.A005          1    ?   1  3390
D81A.DSNDBD.STEENVOL.PART9XD2.I0001.A006          1    ?   1  3390
D81A.DSNDBD.STEENVOL.PART9XD2.I0001.A007          1    ?   1  3390
D81A.DSNDBD.STEENVOL.PART9XD2.I0001.A008          1    ?   1  3390
D81A.DSNDBD.STEENVOL.PART9XD2.I0001.A009          1    ?   1  3390
D81A.DSNDBD.STEENVOL.PART9XD2.I0001.A010          1    ?   1  3390
D81A.DSNDBD.STEENVOL.PART9XD2.I0001.A011          1    ?   1  3390
D81A.DSNDBD.STEENVOL.PART9XD2.I0001.A012          1    ?   1  3390
```

29

Creating a DPSI is very simple. The only difference from creating a non-partitioned index is the keyword PARTITIONED.

A DPSI cannot be UNIQUE due to reasons explained earlier.

When the PARTITIONED keyword is used, DB2 will allocate the same number of VSAM datasets / partitions as the corresponding table has partitions.

TABLE PARTITIONING - INDEXES

- Create CLUSTER index :

```
CREATE UNIQUE INDEX RASST02.PART9XC1
       ON RASST02.TBPART9 (COL05 )
       USING STOGROUP SYSDEFLT PRIQTY 48  SECQTY 96  ERASE NO
       FREEPAGE 0  PCTFREE 10  BUFFERPOOL BP1 CLOSE YES CLUSTER ;
DSNT408I SQLCODE=-623,ERROR:A CLUSTERING INDEX ALREADY EXISTS
                          ON TABLE RASST02.TBPART9
```

- CLUSTER can be removed / added :

```
ALTER INDEX RASST02.PART9XD1 NOT CLUSTER ;
 DSNT400I SQLCODE = 000,  SUCCESSFUL EXECUTION
CREATE UNIQUE INDEX RASST02.PART9XC1
       ON RASST02.TBPART9 (COL03 )
       USING STOGROUP SYSDEFLT PRIQTY 48  SECQTY 96  ERASE NO
       FREEPAGE 0  PCTFREE 10  BUFFERPOOL BP1 CLOSE YES CLUSTER ;

 DSNT400I SQLCODE = 000,  SUCCESSFUL EXECUTION
```

- Still possible to create Partitioned Cluster index
  - LIMITKEYS on create index ONLY if NOT specified on the table

30

This example illustrates the CLUSTER indicator for an index can be removed and allocated to another index. It is not necessary to drop the index itself – all that needs to be done is to execute an ALTER statement to remove the CLUSTER indicator and another index can be altered to be the CLUSTER or another index can be created with the CLUSTER keyword.

TABLE PARTITIONING - INDEXES

- Display output grows and grows and ……
- Stop partition 4 of the DPSI

-STO DB(STEENVOL) SPACE (PART9XD2) PART(4)

```
NAME     TYPE PART   STATUS
-------- ---- ------ ------
PART9TS  TS    0005 RW
    -THRU      0009
PART9TS  TS    0001 RW
    -THRU      0003
PART9TS  TS    0010 RW
    -THRU      0011
PART9TS  TS    0004 RW
PART9TS  TS    0012 RW
PART9XC1 IX    L*   RW
PART9XD1 IX    L*   RW
PART9XD2 IX   D0001 RW
    -THRU      0003
PART9XD2 IX   D0004 STOP
PART9XD2 IX   D0005 RW
    -THRU      0012
```

Tablespace partitions listed in LOGICAL partition order

Unique indexes, Non-unique and partitioned indexes identified by L

DPSI (Data partitioned Secundary Indexes) recognized by D

31

When both tablespace partitions and indexes exist, the display output gets even bigger and another new parameter can be viewed on the output.

Tablespace partitions are listed in LOGICAL (increasing limitkey) sequence.

All DPSI partitions are listed in PHYSICAL sequence with a “D” in front of the partition.

All NON-DPSI indexes are listed with a preceeding “L”. If all logical partitions have the same status, an asterisk will illustrate this (no changes here).

TABLE PARTITIONING – NEW COMMAND

```
PTPDBCDD  R11 --- DB2 Command Processor - 2005/01/05
COMMAND  ==>                                  SCROLL ==> CSR
-------------------------------------------------------------
¦ -DIS DB(STEENVOL) SPACE(PART9*) OVERVIEW                    ¦
¦                                                             ¦
************************ TOP OF DATA ***************************
DSNT360I  !D81A **********************************
DSNT361I  !D81A *  DISPLAY DATABASE SUMMARY
                *    OVERVIEW
DSNT360I  !D81A **********************************
DSNT362I  !D81A    DATABASE = STEENVOL  STATUS = RW
                   DBD LENGTH = 20180
DSNT397I  !D81A
NAME     TYPE PARTS
-------- ---- -----
PART9TS  TS    0013
PART9XC1 IX    L*
PART9XD1 IX   L0013
PART9XD2 IX   D0013
******* DISPLAY OF DATABASE S
```

The difference between L* and L0013 :

L* : If ALL partitions of NON-DPSI or NON-TABLESPACE have same status.

Lxxxx : Used for Tablespaces and DPSI's. Used for NON-DPSI and NON-PARTITIONED indexes where all partitions have the same status.

32

A new keyword is available when executing DISPLAY commands.

The REVIEW parameter will display a summary of the number of partitions which is very useful once a table based partitioned object has had partitions rolled and added over time.

REORG and REBUILD INDEX

- Reorg DPSI partition 4 – which is stopped :

```
DSNUGUTC -  REORG INDEX RASST02.PART9XD2 PART 4 SORTDEVT SYSDA
D81A DSNUGDTC - RESOURCE NOT AVAILABLE, REASON=X'00C90081',
                ON STEENVOL.PART9XD2

DSNUGBAC - UTILITY EXECUTION TERMINATED, HIGHEST RETURN CODE=8
```

  - UTILITIES on DPSI's are working on PHYSICAL too (Great news)

- Rebuild index – new syntax possible :

DPSI partitions can be rebuild (less outage compared to NPI's)

```
REBUILD INDEXSPACE (STEENVOL.PART9XD2 PART 4)
REBUILD INDEXSPACE (STEENVOL.PART9XD2 PART 6)
REBUILD INDEXSPACE (STEENVOL.PART9XD2 PART 7)
REBUILD INDEXSPACE (STEENVOL.PART9XD2 PART 12)
```

33

DPSI partitions reference the same physical partitions of the table which is illustrated by stopping DPSI partition 4 and trying to reorg table partition 4, which fails.

The new utility REBUILD INDEXSPACE can operate on individual DPSI partitions.

REORG entire tablespace – no clean up of logical/physical partitions ”out of sequence”

```
For Table   ==> SYSIBM.SYSTABLEPART        Row number==> 1 OF 7
Browse Mode ==> C                           Max Char  ==> 070
SSID: D81A ----------FETCH STATUS: COMPLETE--------------------
  PARTITION TSNAME    LOGICAL_PART BEFORE   LOGICAL_PART AFTER
        1   PART1          3                     3
        2   PART1          4                     4
        3   PART1          7                     7
        4   PART1          1                     1
        5   PART1          2                     2
        6   PART1          5                     5
        7   PART1          6                     6
************************** BOTTOM OF DATA ********************
```

Reorganizing the entire tablespace does NOT change the Logical partitionss to be in the same sequence as the node name (physical partitions).

Once partitions have been rotated and added in between rotated partitions, the logical partitions are completely out of sync with physical partitions.

Unfortunately a reorganization of the entire tablespace does not make the physical partitions match the logical ones.

REORG UTILITY - REBALANCE

- Instead of SQL ALTER LIMITKEY – REBALANCE keyword as part of REORG can be a great solution…….

PARTITION	CARD	LIMITKEY	LOGICAL PART
1	0	'X300000000'	5
2	0	'X400000000'	6
3	4	'X500000000'	7
4	34	'X800000000'	10
5	0	'Y100000000'	12
6	0	'T999999999'	1
7	0	'V999999999'	2
8	0	'X100000000'	3
9	0	'X200000000'	4
10	40	'X600000000'	8
11	1	'X700000000'	9
12	0	'X900000000'	11
13	0	'Y200000000'	13

- Data in partition 4 needs to be spread out between partition 3 thru partition 6

35

Prior to DB2 V8, the easiest method to spread out data from one partition is to execute ALTER statements to change the LIMITKEY and then perform a reorg to redistribute rows across the paRTITIONS CHANGED.

DB2 V8 offers a new parameter to be used with the REORG utility.

REBALANCE will try to spread the rows between the partitions listed in the utility using the partition-range syntax..

REORG UTILITY - REBALANCE

- SYSCOPY DD required for REBALANCE (download latest doc)

```
//SYSREC   DD  DSN=RASST02.SYSREC.NUM01,
//             UNIT=SYSDA,DISP=(,KEEP),SPACE=(CYL,(4,20))
//SYSCOPY  DD  DSN=RASST02.PART9TS.COPY.REO001,
//             UNIT=SYSDA,DISP=(,CATLG),SPACE=(CYL,(4,20))
//SYSUT1   DD  UNIT=SYSDA,DISP=(,PASS),SPACE=(CYL,(4,20))
//SORTOUT  DD  UNIT=SYSDA,DISP=(,PASS),SPACE=(CYL,(4,20))
 REORG TABLESPACE STEENVOL.PART9TS PART (3:6) REBALANCE
       SORTDEVT SYSDA  SORTNUM 03
```

- And then we're ready to reorg…….

```
DSNU1129I !D81A DSNURFIT - PARTITION RANGE NOT CONTIGUOUS -
                           REBALANCE IGNORED
DSNU012I  DSNUGBAC - UTILITY EXECUTION TERMINATED,

                     HIGHEST RETURN CODE=8
```

- Must be contiguous LOGICAL PARTITIONS due to LIMITKEYs being altered

36

It is mandatory to include a DD-CARD SYSCOPY when REBALANCE is to be used for the reorg utility. The REORG syntax does not specify an inline copy to be taken, but DB2 will always create an imagecopy for the range of partitions being reorganized.

The REORG REBALANCE will only execute if the partition range listed are contiguous LOGICAL partitions – meaning they will have to be in limitkey sequence so DB2 can change the limitkey definitions without having to move data to other partitions than the ones listed.

REORG UTILITY - REBALANCE

- Partition (8:9) are contiguous PHYSICAL and contiguous LOGICAL (3:4) - but …….

```
REORG TABLESPACE STEENVOL.PART9TS PART(8:9) REBALANCE
SORTDEVT SYSDA SORTNUM 3
UNLOAD PHASE STATISTICS - NUMBER OF RECORDS UNLOADED=0
FOR TABLESPACE STEENVOL.PART9TS
UNLOAD PHASE COMPLETE, ELAPSED TIME=00:00:00
- FEWER PAGES THAN PARTS - REBALANCE IGNORED
INDEXES WILL BE BUILT IN PARALLEL, NUMBER OF TASKS = 6
COPY PROCESSED FOR TABLESPACE STEENVOL.PART9TS PART 8:9
     NUMBER OF PAGES=4 AVERAGE PERCENT FREE SPACE PER PAGE = 0.00
```

- Physical partition (10:11) , logical (8:9) have enough data, but …

```
ERROR FROM SORT COMPONENT RC=16, UTILITY STOPPED
UTILITY BATCH MEMORY EXECUTION ABENDED, REASON=X'00E40005
```

- If indexes exist, reorg abends – wrong keylength parsed to sort

Apply PTF=UQ95755 , APAR=PQ96253 (solved one year ago)

37

Once contiguous partitions have been located, the reorg will not execute if the number of pages in the partition range have fewer pages than partitions.

If indexes exist, make sure PTF=UQ95755 is applied to avoid a SORT abend.

REORG UTILITY - REBALANCE

- Rebalance might not be possible when:
  - A partition ends up with ZERO rows (could be due to very NON-UNIQUE data)
  - If the PHYSICAL and LOGICAL partition does NOT match

```
REORG TABLESPACE STEENVOL.PART9TS PART(1:3) REBALANCE UNLOAD PHASE
STATISTICS - NUMBER OF RECORDS UNLOADED=667
NOT ALL PARTITIONS POPULATED BY REBALANCE - PROCESSING TERMINATES
      COPY PROCESSED FOR TABLESPACE STEENVOL.PART9TS PART 1:3
      NUMBER OF PAGES=15

(RE)LOAD PHASE STATISTICS - NUMBER OF INPUT RECORDS PROCESSED=667
TABLESPACE STEENVOL.PART9TS PARTITION 1 IS IN RECOVER PENDING STATE
TABLESPACE STEENVOL.PART9TS PARTITION 2 IS IN RECOVER PENDING STATE
TABLESPACE STEENVOL.PART9TS PARTITION 3 IS IN RECOVER PENDING STATE
      UTILITY EXECUTION TERMINATED, HIGHEST RETURN CODE=8
```

- CASE : Partition (1:3) has data (94 , 560 and 12 rows)

38

Another issue which might cause the REBALANCE to be impossible is, if the data in the columns described by the limitkeys is highly non-unique. In these cases, it might not be possible for DB2 to find enough distinct values to use for the “new” limitkeys, and DB2 will not leave any partitions participating empty.

This case can be critical depending on what type of reorg is being executed. A SHRLEVEL NONE (which is the default) will cause the partitions to be placed in RECOVER PENDING.

In order not to end up with an outage, SHRLEVEL REFERENCE is recommended.

REORG UTILITY – REBALANCE

PART	CARD-B4	CARD-AFTER	LIMITKEY B4	LIMITKEY AFTER
1	94	322	X300000000	X304000000,411.00
2	560	322	X400000000	X360000000,9918.70
3	12	32	X500000000	X500000000
4	34	34	X800000000	X800000000
5	0	0	Y100000000	Y100000000
6	0	0	T999999999	T999999999
7	0	0	V999999999	V999999999
8	0	0	X100000000	X100000000
9	1	1	X200000000	X200000000
10	40	40	X600000000	X600000000
11	1	1	X700000000	X700000000
12	0	0	X900000000	X900000000
13	0	0	Y200000000	Y200000000

- Once all the "IF's" are gone – reorg rebalance does a great job (in some / most) cases
- Note the changes to limitkey updates – even second column of partitioning key has been adjusted

39

Once all the "IF's" are gone, REBALANCE does a really good job.

This figure illustrates how partition 1:3 have been rebalanced.

Notice the difference how the LIMITKEY's have been updated during the process. DB2 even updates the LIMITKEY columns not populated when the table was created in order to spread data evenly.

Oddities - Rebalance

- REORG REBALANCE (1:13) = all partitions

```
REORG TABLESPACE STEENVOL.PART9TS PART(1:13) REBALANCE
UNLOAD PHASE STATISTICS - NUMBER OF RECORDS UNLOADED=752
UNLOAD PHASE COMPLETE, ELAPSED TIME=00:00:00
KEYWORD 'SORTKEYS' SPECIFIED BUT NO INDEX OR FOREIGN K
      FEWER PAGES THAN PARTS - REBALANCE IGNORED
COPY PROCESSED FOR TABLESPACE STEENVOL.PART9TS
      NUMBER OF PAGES=46
      AVERAGE PERCENT FREE SPACE PER PAGE = 13.60
      PERCENT OF CHANGED PAGES =100.00
      ELAPSED TIME=00:00:18
(RE)LOAD PHASE STATISTICS - NUMBER OF RECORDS=752 FOR
DB2 IMAGE COPY SUCCESSFUL FOR TABLESPACE STEENVOL.PART9TS
(RE)LOAD PHASE STATISTICS - NUMBER OF INPUT RECORDS PROCE
(RE)LOAD PHASE COMPLETE, ELAPSED TIME=00:00:18
```

- The syntax where all partitions specified takes care of the physical/logical challenges

When the logical and physical partitions are completely out of sync, the only possible way to rebalance data might be to REBALANCE all the partitions in one go. Of course it's always possible to DROP and create the objects with the new limitkeys.

REORG REBALANCE SUMMARY

- IBM documentation clearly states
  - Not REORG shrlevel CHANGE
  - No LOB columns in base object
  - Must be contiguous logical
  - No mix of compression / no compression
  - Might not work for very non-unique data
  - Might not work when physical partitions don't match logical
  - Plans, packages are invalidated – statement cache flushed
- My findings (and this is changing)
  - Number of pages must be greater than partitions
  - All partitions must have data – otherwise "touched" partitions end in RECP and indexes in RBDP (solved in current GA version)
  - Wrong SORT parameters parsed if indexes and REORG will abend (solved by PTF)
  - Make sure you have fallback imagecopies
  - Seems to work great for "good old" Index Based partitioned objects

RECOVER SCENARIO

- Sequence of operations:
  1. IMAGECOPY (partition level)
  2. QUIESCE
  3. REORG REBALANCE part(1:3)
  4. RECOVER PARTITION 1,2 TORBA (quiesce RBA done in step 2).

C:\Documents and Settings\rasst02.TANT-A01\Desktop\recovrebal.xls

Catalog Changes for V8 Partitioning

- SYSTABLEPART
  - LIMITKEY changed to VARCHAR(765)
  - LOGICAL_PART SMALLINT
  - LIMITKEY_INTERNAL VARCHAR(512)
    (blank for index partitioned object)
- SYSTABLES
  - PARTKEYCOLNUM SMALLINT
- SYSINDEXES
  - INDEXTYPE 'P' , 'D' , '2' *(partitioning, DPSI, others)*
  - LIMITKEY only for index based partitioning
- SYSCOLUMNS
  - PARTKEY_COLSEQ SMALLINT
  - PARTKEY_ORDERING C(1)

New and changed columns in the catalog concerning table based partitioning.

When to exploit V8 partitioning

- Always use partitioning – perhaps
  - When growth is expected define ONE partition the good old way
  - Can convert to table partitioning any time
  - Free to add or rotate partitions
  - Rebalance even on index based partitioned objects
  - IBM recommends to convert to table based
    - Least impact method
      - ALTER INDEX NON CLUSTER
      - ALTER INDEX CLUSTER
    - Remember invalidation of packages, dynamic cache

V8 Table Partitioning - Conclusion

- Remember ”Hill Street Blues” ?
- When they left the morning briefing.....

Let’s be careful
out there .....

Summary

- New and changed DDL statements
- Adding Partitions
- Rotating Partitions
- Commands
- Utility Changes

  - Hope you got some useful information which will give you a quick start – and avoid the many hours I spent trying to figure out WHAT happens when
  - My intention was NOT to scare you

Questions and Answers