



Keep IT Going
29 oktober 2009
Speulderbos Garderen



ACTIVATE BUSINESS WITH THE POWER OF I.T.™



Be Different, Partition Like Everybody Else

DB2 Session

Agenda – Be different, partition like everybody else

- **DB2 Release history for tablespace storage**
 - Partition / Partitioned/ ..
 - Index Partitioning
- **DB2 9 New Object type UTS**
 - UTS Parttion By Growth
 - UTS Partition By Range
- **Impacts and considerations**

◦Note

the DDL in the examples is pseudo code – no comments on missing commas or unbalanced parenthesis!



DB2 TS Storage Options

- **New tablespace storage mechanisms have been introduced since V1**
- **All datasets are physically stored in VSAM Linear Datasets (LDS) with page size ranging from 4k to 32k.**
- **Tablespace Storage mechanism by DB2 release**
 - Simple (V1.1)
 - Partitioned (V1.1)
 - Segmented (V2.1)
 - LOBs (V6)
 - Table Controlled Partitioned (V8)
 - Also several new Index schemes
 - Universal Tablespace (DB2 9)
 - Partition by Growth, Partition by Range
 - pureXML™ (DB2 9)
- **Why? VLDB and new datatypes**
 - DB2 V4 max TS size – 64GB
 - DB2 9 max TS size – 128 TB (2048 times larger!)
 - LOBs and XML allow storage of media and documents in DB2

PARTITIONED Tablespace

- **Single table in tablespace using multiple datasets**
 - Max 4096 partitions depending on DSSIZE (1G up to 64G) and page size (4k, 8k, 16k, 32k) – up to 128TB
- **Partitioned Index required for V7 and earlier, also had to be the clustering index**
 - CREATE TABLESPACE TS1 ...NUMPARTS (n)
 - (PART 1 ...
 - PART 2 ...
 - CREATE TABLE TRAN IN TS1 (COL1 DATE, ...)
 - CREATE INDEX IX1 ON T1 (COL1 ASC)
 - CLUSTER (PART 1 VALUES (nnn)
 - PART 2 VALUES (yyy)
 - PART 3

PARTITIONED TS example

TRAN Table in TS1

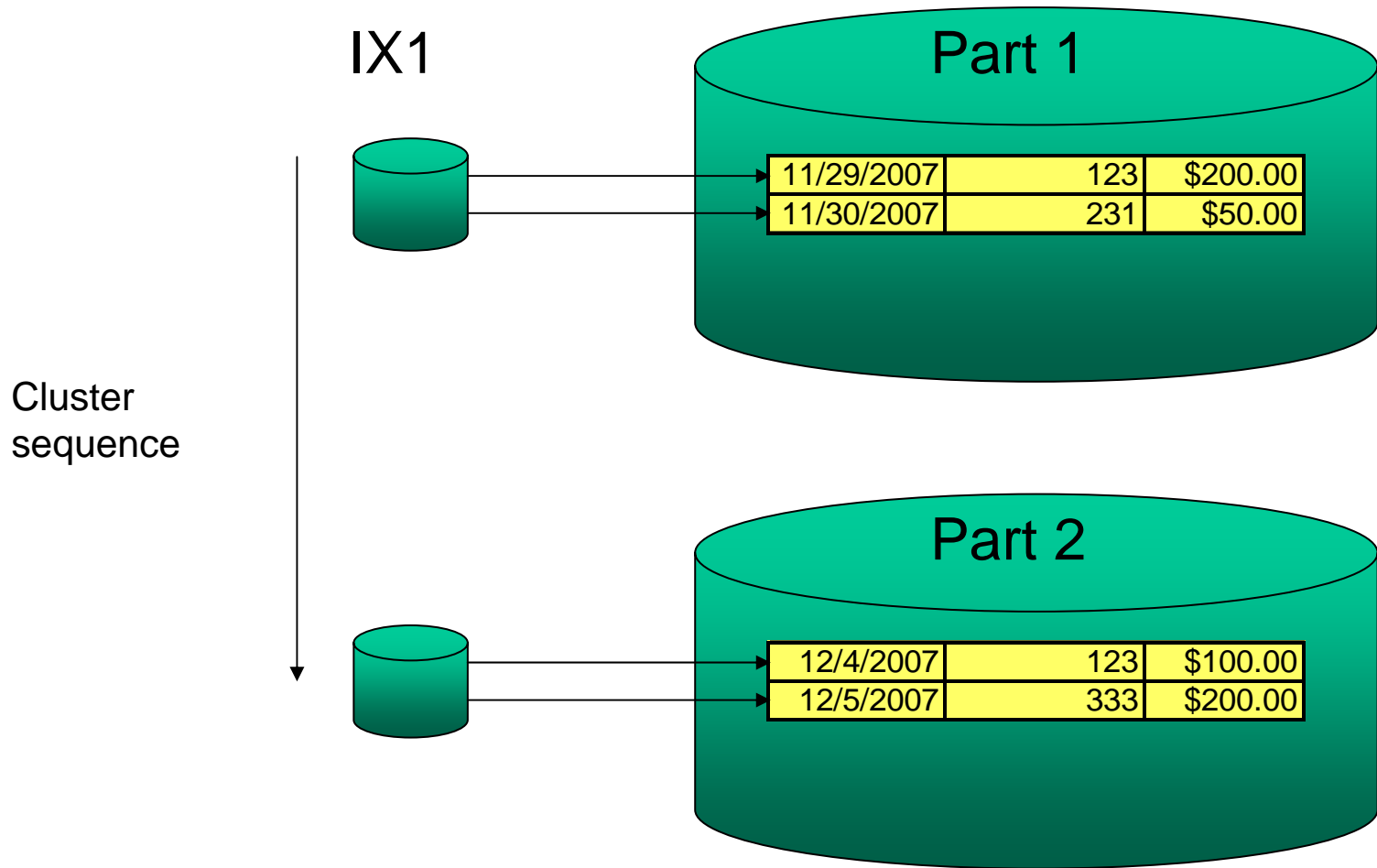


Table Controlled TS Partitioning (new with DB2V8)

- **Limit keys defined by CREATE TABLE, drives TS physical partitioning**
- **No partition index required (saves disk space)**
 - CREATE TABLESPACE TS1 NUMPARTS (n) (PARTITION 1 USING ...
 - PARTITION n USING ...)
 - CREATE TABLE ACCT (COL1 DATE ...)
 - PARTITION BY (COL1 ASC)
 - (PARTITION 1 ENDING AT (nnn),
 - PARTITION n ENDING AT (yyy))

Table Controlled Partitioning - example

ACCT Table in partitioned TS1

Part 1

11/29/2007	123	\$200.00
11/30/2007	231	\$50.00

Part 2

12/4/2007	123	\$100.00
12/5/2007	333	\$200.00

- **DB2 will automatically convert an existing index-controlled partitioned object to table-controlled partitioning if you DROP INDEX or ALTER INDEX NOT CLUSTER the partitioning index**

DB2 IX Storage Options

- **Indexes evolved too**

- Basic Index (Unique, Cluster)
- Partitioning Index (limit keys drive TS parts)
- Non-Partitioning Index (NPI – across all parts)

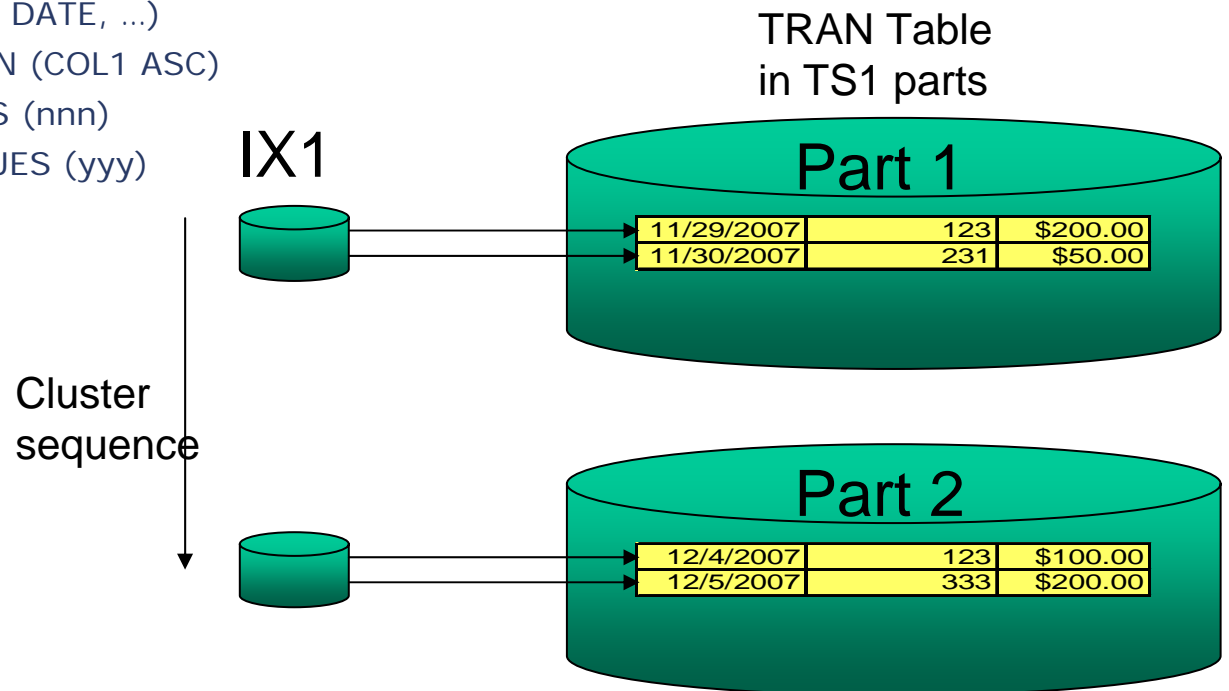
- **Why? Partitioning good for utilities and some batch but ...**

- Index controlled partitioning limited online query
- Non-Partitioned Index(es) caused utility headaches (e.g. REORG BUILD2)

Partitioned Index

- For VLDB, partitioning driven by index key ranges.
- Partitioned tablespace definition incomplete until index defined

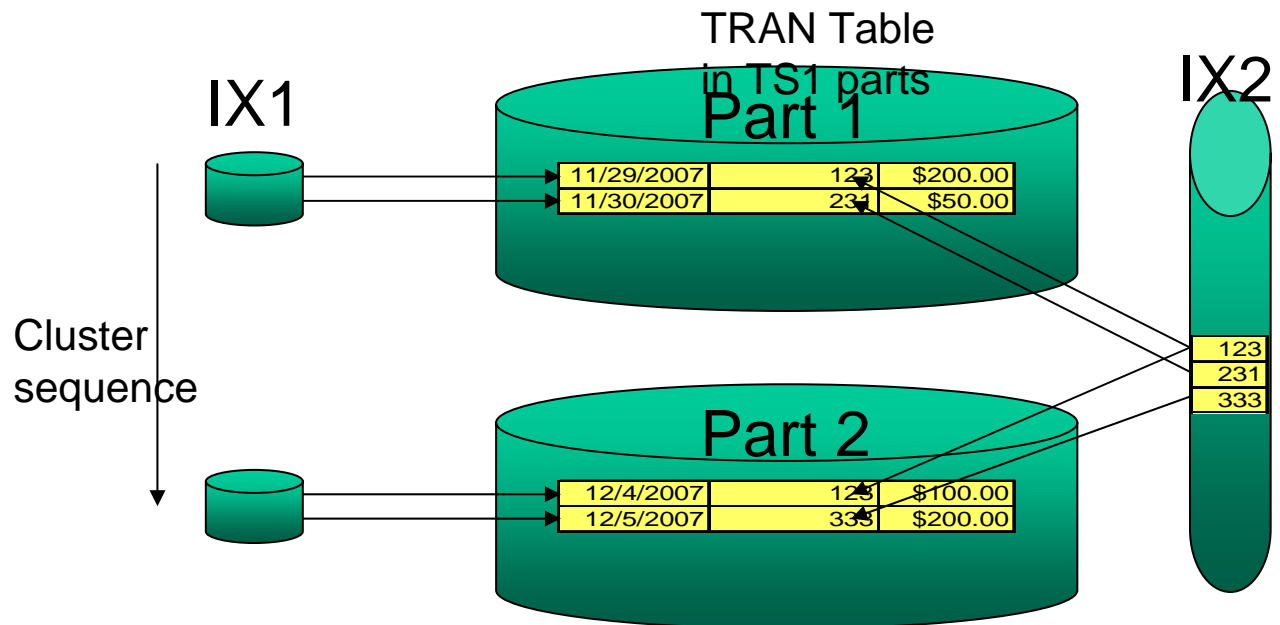
```
CREATE TABLESPACE TS1 ...NUMPARTS (n)
    (PART 1 ...
    PART 2 ...
    PART 3 ...)
CREATE TABLE TRAN (COL1 DATE, ...)
CREATE INDEX IX1 ON TRAN (COL1 ASC)
    CLUSTER (PART 1 VALUES (nnn)
    PART 2 VALUES (yyy)
    PART 3 ...)
```



Non-Partitioned Index (NPI)

- A 'secondary' index used to improve access across all parts

```
CREATE INDEX IX2 ON TRAN (ACCTNO);
```



New nomenclature with V8

- **Partitioned**

- The index is physically stored in multiple datasets

- **Partitioning**

- The index is used to drive the partitioning of the tablespace

- **General naming convention**

- Physical storage (partitioned, non-partitioned), Logical usage (partitioning, secondary)

- **Index evolution continues ...**

- Partitioned Partitioning Index (PPI)
- Non-Partitioned Partitioning Index (NPPI)
- Non-Partitioned Secondary Index (NPSI)
- Data-Partitioned Secondary Index (DPSI)

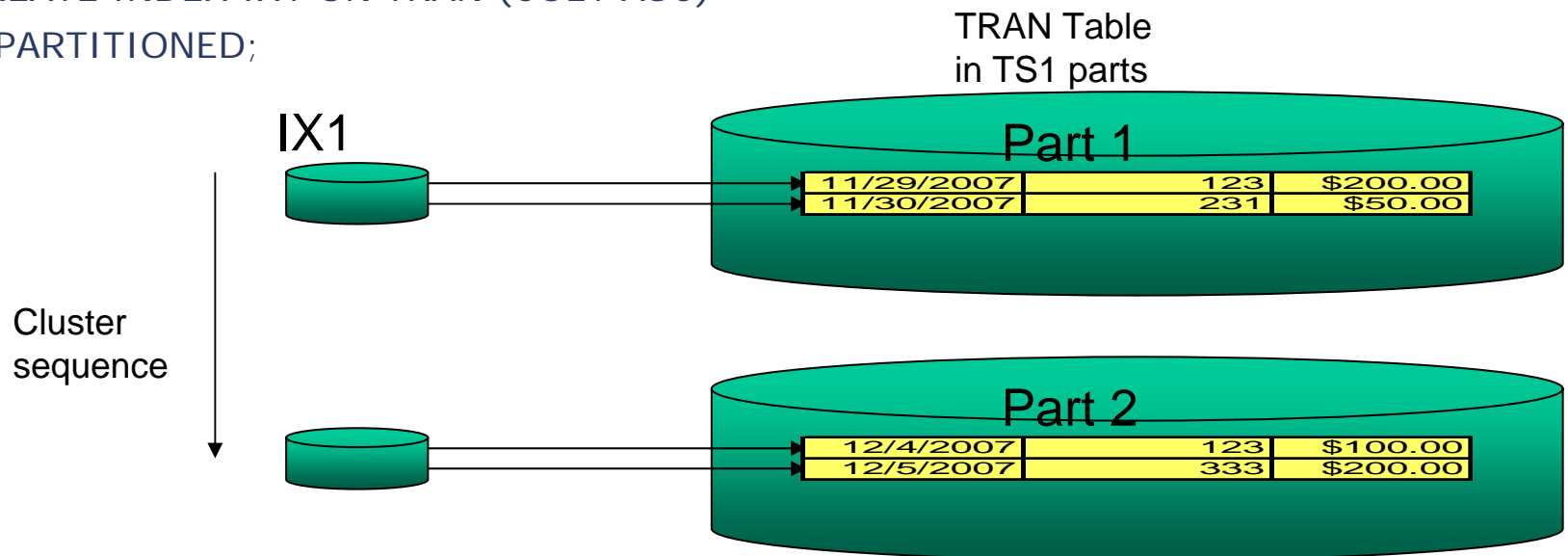
Partitioned Partitioning Index (PPI)

- **Physically partitioned, logically partitioning**

```
CREATE INDEX IX1 ON TRAN (COL1 ASC)
(PART 1 VALUES (nnn)
PART 2 VALUES (yyy)
PART 3 ....)
PARTITIONED;
```

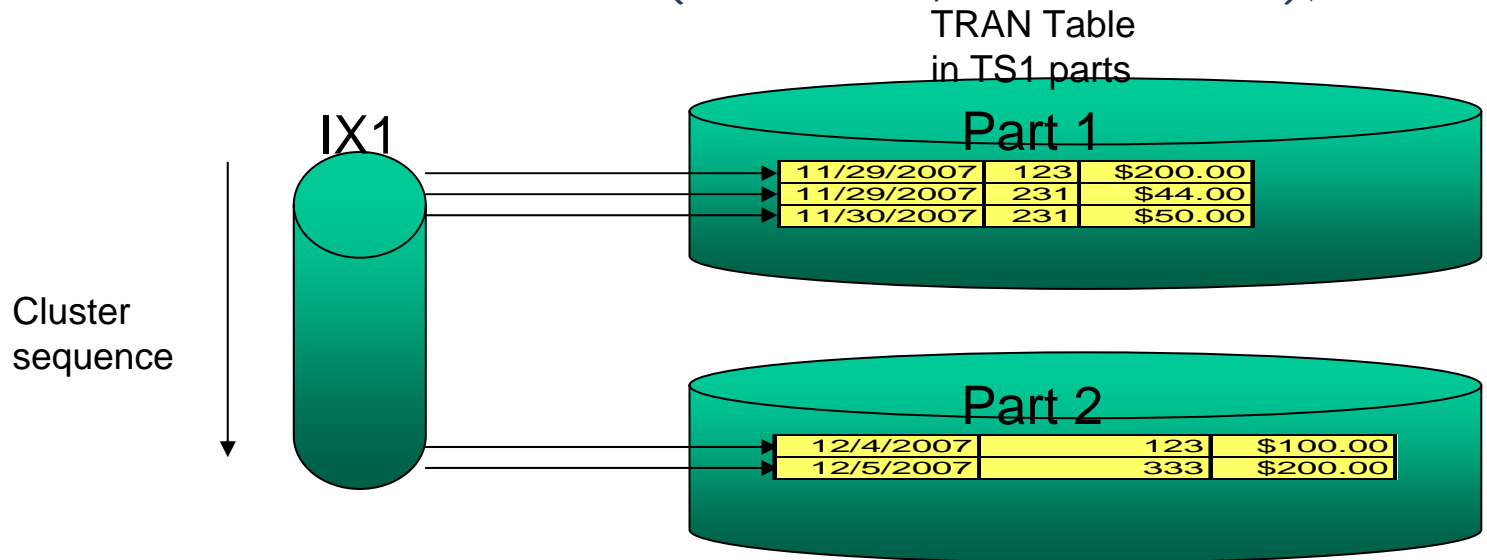
- **Or, if table-based partitioning used ...**

```
CREATE INDEX IX1 ON TRAN (COL1 ASC)
PARTITIONED;
```



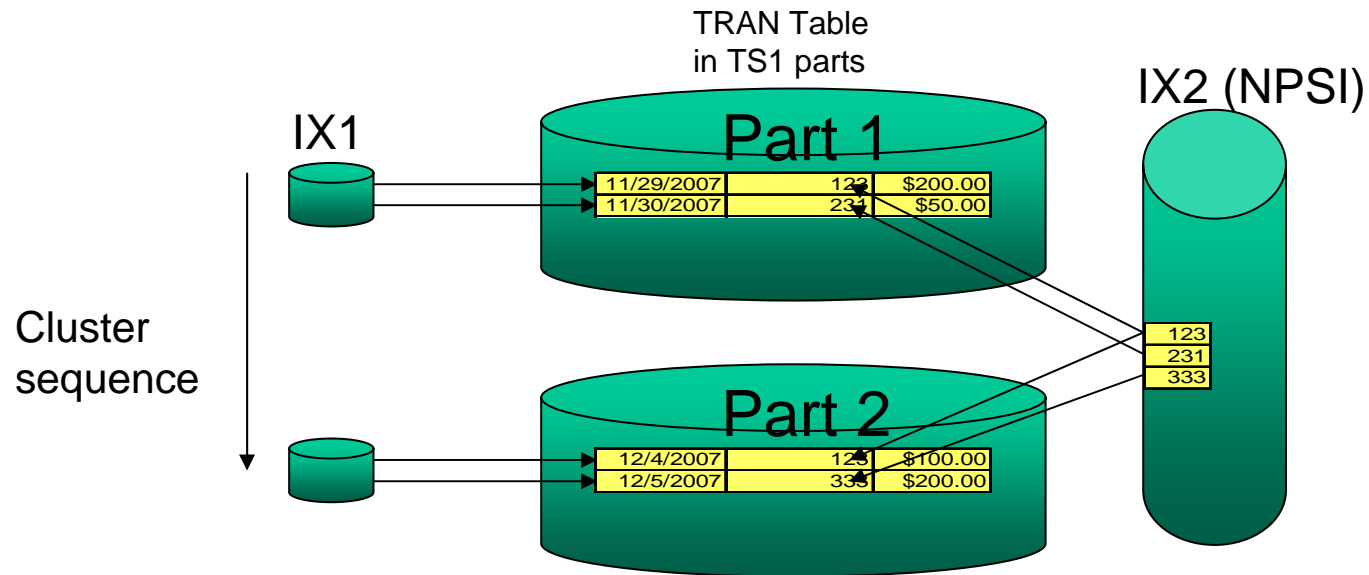
Non-Partitioned Partitioning Index (NPPI)

- **Single physical dataset, logical partitioning key**
 - CREATE INDEX IX1 ON TRAN (COL1 ASC, ACCTNO ASC)
 - (PART 1 VALUES (nnn))
 - PART 2 VALUES (yyy)
 - PART 3);
- **Or, if table-based partitioning used ...**
 - CREATE INDEX IX1 ON TRAN (COL1 ASC, ACCTNO ASC);



Non-Partitioned Secondary Index

- Single physical dataset, logical alternate access
- CREATE INDEX IX2 ON TRAN (ACCTNO);

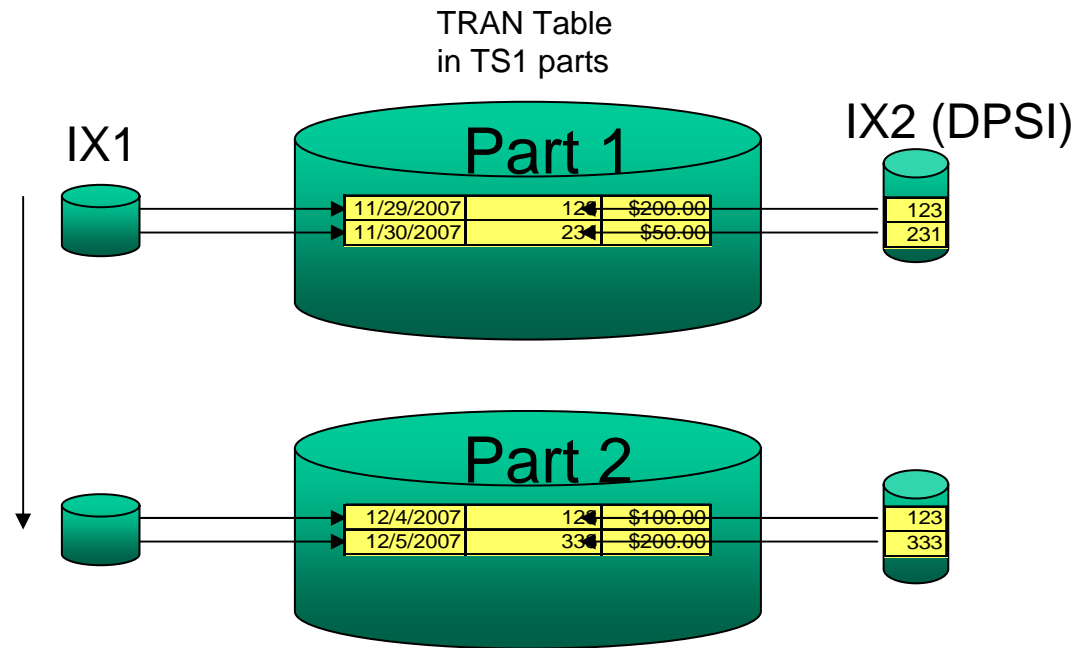


Data-Partitioned Secondary Index (DPSI)

- Physically partitioned, logical alternate index
- Cannot be defined as **UNIQUE**

```
CREATE INDEX IX2 ON TRAN (ACCTNO) PARTITIONED;
```

Cluster
Sequence
(note –
any index
can be the
Clustering
Index on a
Table-Controlled
Partitioned Tablespace



V7 Partitioning - Impacts

- **V7 Partitioned Tablespace**

- Partitioning Index good for utilities and batch – but maybe not so good for query
- NPI maybe help query, but painful for utilities and SQL IUD operations

V8 Partitioning - Impacts

- **Table-controlled partitioning eliminates need for partitioning index**
 - but can still define Partitioned or Non-Partitioned Index if partitioning key helps data access performance
 - Any Index can be the Clustering Index
- **DPSI and NPSI**
 - Flexibility for dataset allocation for Secondary IX
- **Query versus Utility performance**
 - Partitioned Indexes (PPI, DPSI) favors partition level query and utilities
 - Non-Partitioned (NPPI, NPSI) reduce dataset level Index processing
 - Secondary Indexes (DPSI, NPSI) favor broad Table Query processing but can cause overhead for SQL IUD processes

Partitioned TS - Concern

- **What if you don't have a handy Partitioning key?**
 - Many applications code 'made up' keys
 - Adds to application complexity
 - Stores data with no business value
 - Reduces flexibility to change application
 - Index Overhead for driving partitioning can be high
 - And Index may never be used for Query support
 - What if you want the space management of Segmented, but have too much data?
 - Segmented allows for up to 64GB

DB2 9 - New Object types

- **DB2 9 introduces Universal Tablespace**
 - Partition by Range (PBR) – what we are used to, with SEGMENTED benefit
 - Partition by Growth (PBG) – combination of Segmented and Partitioned with no Partitioning Key required
- **Type of TS dictated by combination of CREATE TS Syntax**
 - $SEGSIZE + MAXPARTITIONS = PBG$
 - $SEGSIZE + NUMPARTS = PBR$
 - Note – with DB2 9 TS can be implicitly created via CREATE TABLE syntax
- **DB2 9 Tablespace maximum size – 128 TB**
 - Determined by combination of DSSIZE, PAGESIZE, SEGSIZE and MAXPARTITIONS/NUMPARTS (impact COMPRESSION Y/N)

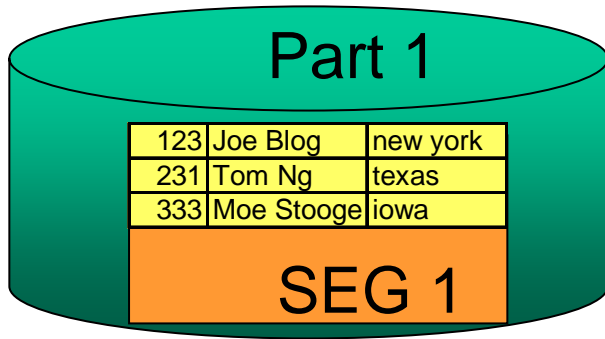
UTS – Partition By Growth

- **Good for large objects with no suitable partitioning key**
- **It is a single table segmented tablespace on steroids.**
- **You can use any SEGSIZE you want**
- **You decide the maximum number of partitions it can grow into.**
- **New partitions are only allocated as you use them.**
 - DB2 manages space, allocates new parts as needed up to MAXPARTITIONS
 - ALTER MAXPARTITION to increase, but cannot decrease without DROP
 - Can be created explicitly ...
 - CREATE TABLESPACE TS1 SEGSIZE (n4) MAXPARTITIONS (n)
 - Or implicitly ...
 - CREATE TABLE ACCT PARTITION BY SIZE EVERY nG

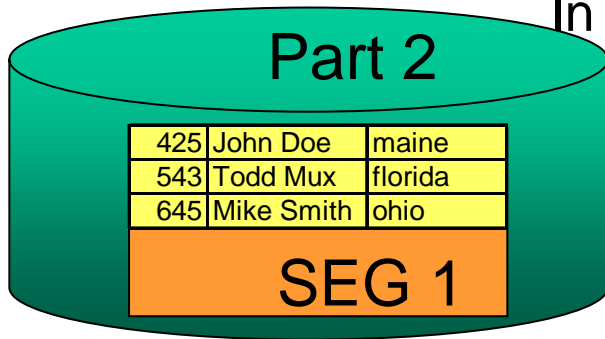
UTS – Partition By Growth

- **DB2 manages space, allocates new parts as needed up to MAXPARTITIONS**
 - ALTER MAXPARTITION to increase, but cannot decrease without DROP
- **Can be created explicitly ...**
 - CREATE TABLESPACE TS1 SEGSIZE (n4) MAXPARTITIONS (n)
- **Or implicitly ...**
 - CREATE TABLE ACCT PARTITION BY SIZE EVERY nG
- **Implicitly created TS defaults to PBG with defaults of:**
 - MAXPARTITIONS=256
 - SEGSIZE=4
 - DSSIZE=4G
 - LOCKMAX – System Default

UTS – Partition By Growth - example



ACCT table
In TS1 parts



- One table per PBG TS
- It is NOT partitioned in the sense of having a partitioning key.
- There is no partitioning index, nor are there any partitioned indexes,
- All indexes are NPSI (nee' NPI)
- One index can be declared as CLUSTERING

UTS – Partition By Range

- **Use if large object with good partitioning key**

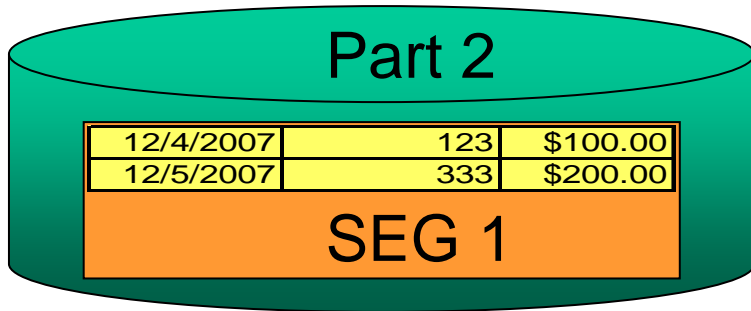
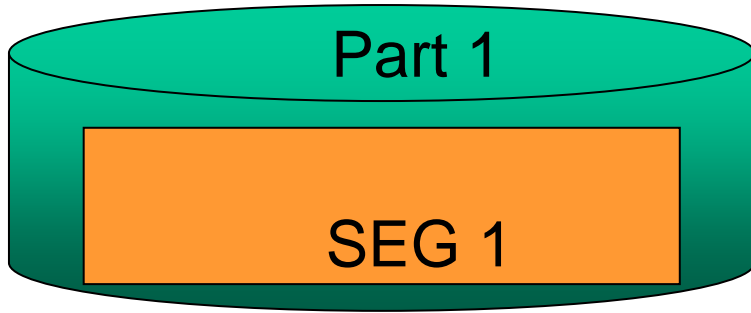
- perfect mix of SEGMENTED + PARTITIONED
 - Better space management and mass delete
 - One table per tablespace
 - Very fast partition rotation

```
CREATE TABLESPACE TS1 ... SEGSIZE (n4),  
NUMPARTS (n)
```

```
(PART 1 ...  
PART 2 ...)
```

- Or implicitly (with APAR 51572) ...
- CREATE TABLE ACCT PARTITION BY RANGE PARTITION n
ENDING AT constant/MAXVALUE/MINVALUE

UTS – Partition By Range - example



- Partition limit ranges can be defined in CREATE TABLE or CREATE INDEX statements
- Activities that are already allowed on partitioned or segmented Tablespaces are allowed on a UTS PBR Tablespace (e.g. utilities and batch jobs can operate at the partition level, mass deletes and drop tables operate at the segment level).

PBG – How big is big?

- **Up to 128TB**

- 1 TB = 1024 GB
- 1 TB = (10242 MB)
- 1 TB = 1,048,576 MB
- Combination of DSSIZE, PGSIZE, and MAXPARTITIONS (up to 4096) determine max size.
- Table at right helps determine optimum SEGSIZE

DSSIZE	PGSIZE	SEGSIZE with COMPRESS NO	SEGSIZE with COMPRESS YES
1G	4	20	20
1G	8	20	12
1G	16	20	4
1G	32	20	4
2G	4	44	24
2G	8	36	12
2G	16	24	4
2G	32	12	4
4G	4	64	28
4G	8	36	20
4G	16	24	8
4G	32	28	4
8G	4	64	64
8G	8	52	28
8G	16	44	12
8G	32	28	4
16G	4	64	64
16G	8	64	52
16G	16	28	24
16G	32	28	12
32G	4	64	64
32G	8	48	48
32G	16	40	28
32G	32	28	12
64G	4	64	60
64G	8	64	64
64G	16	64	32
64G	32	64	12

What about Online Schema Evolution/DDOD

- **Online Schema Evolution in DB2 v8**
 - Change Data Type of Column
 - Alter Identity Columns
 - Add Column to Index
 - Change Clustering Index for Table
 - Changes to Partitioned and Partitioning TS and IX
- **DB2 9 introduces Database Definition on Demand**
 - Rename columns and Indexes
 - Fast Replacement of one table with another (Clone)
 - Tablespace that can add partitions for growth automatically
 - Improved Ability to rebuild Index Online
 - And more – But not for UTS
- **You can not ADD or ROTATE a PBG**
 - There is no key range associated with a PBG partition, so no reason for ADD or ROTATE
- **You can use ALTER to increase MAXPARTITIONS**
- **You can NEVER decrease MAXPARTITIONS**
- **Once a partition is materialized by DB2 it never goes away**
- **None ALTER-able attributes**
 - SEGSIZE, PIECESIZE, DSSIZE

UTS – Partition By Growth

- **Maintaining a PBG object**

- You can LOAD RESUME and LOAD REPLACE the table as a whole.
 - You can not use the LOAD INTO PART parameter with a PBG object.
- You can REORG the whole table or a set of partitions.
 - If you are REORGing to add free space, then you must include the last partition, either explicitly or by implication.
 - If you do a REORG and the part will not fit back into the part it came from and the last current partition is not part of the set of partitions of the part in question, then the REORG will fail.

UTS – Partition By Growth

- **Compressed Object Considerations**

- LOAD REPLACE will build a dictionary that is repeated in each partition.
 - LOAD REPLACE KEEPDICTIONARY will keep the first partition's dictionary and propagate that dictionary.
- LOAD RESUME will propagate the last existing partition's dictionary to any new partitions.
- A tablespace REORG will build a new dictionary and propagate it to all partitions.
- A partition REORG will build a new dictionary for that partition unless KEEPDICTIONARY is specified.

UTS – Partition By Growth

- **Don't just set MAXPARTITIONS to the largest possible value.**
 - It wastes DBD space
 - Allocation can take a while
 - Slight utility overhead
 - Objects that should be looked at might be ignored

DB2 9 Universal Tablespace impacts

- **UTS has benefits of segmented and partitioning**
 - Each partition has own Header and Space Map page
 - If compressed, each partition has it's own set of dictionary pages
 - Better space management and mass deletes
 - PBG allows for large objects when no good Partitioning range key
- **Cannot ALTER a non-UTS tablespace to convert to UTS**
 - Must UNLOAD/DROP/CREATE/LOAD
 - Impacts associated/dependent objects – TB, VW, IX,
 - But might be able to with DB2 'X'
- **DB2 pushing more and more storage allocation process to DB2-managed storage**
 - User defined VCATs becoming thing of the past

What's it mean to me?

- **If I were defining a new application I would lean towards..**
 - For large tables
 - Table-Controlled Partitioning in a PBR UTS if I had a good range key, regular table in a PBG UTS if not
 - Secondary Indexes as needed for Query
 - Cluster index on most likely sequence column(s)
 - Partitioned (DPSI) or Non-Partitioned (NPSI) depending on how I wanted batch/utilities and part-level query to work
 - For small tables
 - Single table per PBG UTS tablespace MAXPARTITION > 1.
 - DB2 only allocates the first partition at CREATE, so it would look like a single part SEGMENTED TS.
 - Secondary Indexes as needed for Query

Be Different, Partition Just Like Everybody Else



Thank You

Paul Oostvogels
BMC Software
Paul_Oostvogels@bmc.com