
CDB/RePart

Introduction

Multiple-dataset tablespaces are the future of DB2. They offer significant advantages by allowing concurrent and independent application and utility processing of the different datasets. Very large databases, used concurrently by large amounts of users, require the data to be placed in many small “vessels” as opposed to a single, large container.

Multi-dataset objects, however, have a down side, and that is complexity. DB2 supports, at the time of this writing, 254 tablespace partitions, which require in turn the same number of clustering index partitions. Tablespaces of this size may have non-partitioning indexes, which are likely to be in multiple pieces or datasets. Therefore, today’s large tablespace may force you to deal with over 500 objects, which may be split into 1000 datasets or so. The Data Base Administrator in your organization must not only define these massive objects, but also manage them over time, and keep them running at optimal performance (which is why the tablespace was partitioned to start with)

CDB/RePart provides multiple tools to help you manage these objects.

Among other functions, it lets you:

- **Rebalance** partitioned tablespaces so that they provide optimal performance at all times.
- **Convert** your fast-growing tables to partitioned.
- **Repartition** partitioned tablespaces (that is, change the number of partitions).
- **Manage** “periodic data,” that is, business data that is generated, used and migrated at periodic intervals.

Rebalancing Partitions

If your installation handles large volumes of data, then you already know the benefits of a partitioned tablespace. The trouble starts when certain parts of the large tablespace become more active than others - certain partitions may contain so many rows that performance is significantly degraded, or even exceed the maximum partition size limit, causing unexpected problems. Large amounts of data shouldn’t stop your enterprise. (See Figure “*The Partitioning Balancing Problem (And Solution)*”)

CDB/RePart solves the problem of “overstuffed” partitions. It evenly distributes the number of rows by creating a workfile with rows written in clustering sequence. Next, it reads the workfile, defines a new index and new partitions and re-writes both. The result is balanced performance across all sections of your database, and the elimination of the risk of a surprise outage, plus a perfectly reorganized tablespace.

CDB/RePart accomplishes all of this without dropping or recreating the tablespace, even in versions of DB2 that do not support the redefinition of limitkeys.

Converting to Partitioned Tablespaces

You can use CDB/RePart to convert simple and segmented tablespaces to their partitioned equivalents, or just evaluate the benefits of such a change. It analyzes the clustering index, determines its own limitkeys and calculates the number of rows in each partition.

When used for conversion purposes, CDB/RePart will design the new tablespace, unload the data, drop, recreate and reload the tablespace. Before the tablespace is dropped, CDB/RePart will pause to allow you to perform any additional functions that may be required, such as saving authorizations for later re-definition.

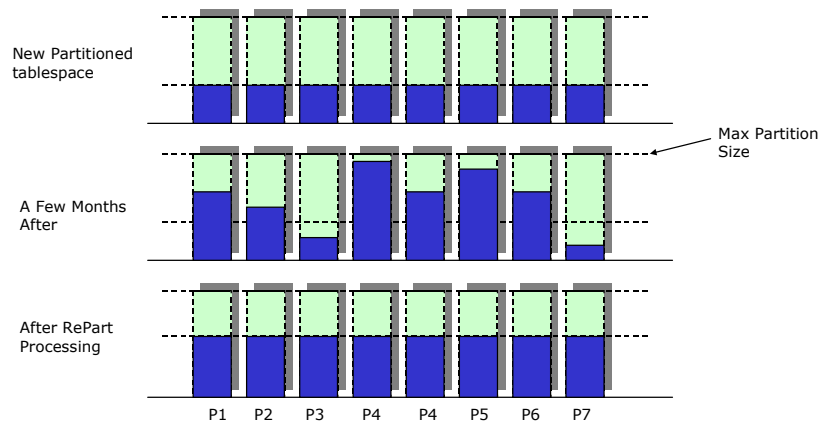
Repartitioning of Tablespaces

You may have partitioned tablespaces that have grown beyond the initial number of partitions. CDB/RePart lets you change the number of partitions of any partitioned tablespace. Like in the conversion case above, it will design the new tablespace, unload the data, drop, recreate and reload the data automatically.

Managing Periodic Data

CDB/RePart manages your periodic data. Periodic data is company data that is generated at some regular interval. The most common example is financial data that is generated monthly, quarterly, or annually. Conceptually, DB2 partitioned tablespaces are ideal for this kind of data. DB2 uses partitioned tablespaces to manage a collection of similar datasets, differentiated only by a range of keys. It seems ideal to add each month to a new partition.

The periodic process is most often a combination of a) delete the oldest period, followed by b) add the data for the newest period. The resultant set of data (e.g., the last 24 months of financial data) is then re-indexed by a set of additional keys to facilitate query access to the data.



The Partition Balancing Problem (and solution)

In this example, the cycle of processing for monthly data would be:

- Unload the oldest month's data to file so that it can be added to a history table,
- Drop the oldest month (partition 1),
- Shift all months down one level,
- Add the newest month's data (partition 24),
- Re-index by customer, account type, or other fields used for reporting and query access.

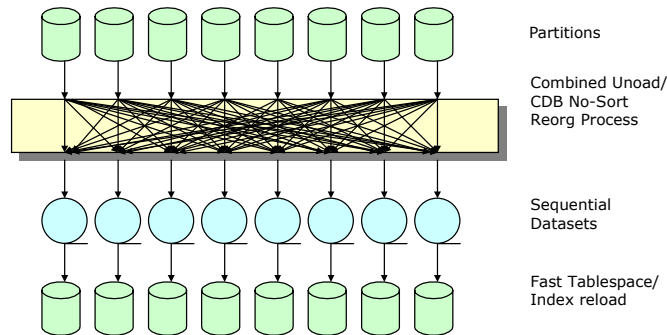
Unfortunately, DB2's own limitations make this process long and error-prone. Many installations have skirted this issue with, in some cases, rather ingenious procedures. Most of these concocted solutions have expensive consequences, even when they work. Some are done with rather extensive reprogramming of applications.

CDB/RePart is the ideal solution. It lets you work with partitioned tablespaces exactly as you would think you should be able to.

CDB/RePart Features

Limitkey Analysis

CDB/RePart will automatically analyze and propose new limitkeys for your tablespace. It does this by scanning the tablespace and evenly assigning rows to each partition. This even distribution is the default, but you can apply your



CDB/Re-Part Process

own criteria to override this recommendation. For example, you may decide that recent data is most likely to be accessed, while aged data is rarely used. Then you may determine that aged data should be stored in a single, large partition, while the rest of the partitions are reserved for active data. The resulting smaller partitions will sustain larger transaction rates with better response times.

Reserved Partitions

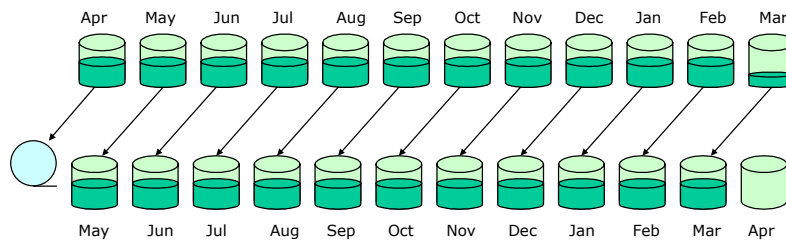
A common technique is to define a tablespace with a maximum number of partitions even though not all of them will be needed initially. The USE parameter allows you to specify how many partitions will be filled by CDB/RePart. The partitions specified in USE will be populated evenly, and the rest will be left empty. If at a later time you decide that additional partitions are required you can simply change the USE parameter.

Performance

CDB/RePart is fast. Although it is generally somewhat slower than CDB/Auto-Reorg, it is faster than any third party reorg in the market. This makes it feasible to use CDB/RePart in lieu of Reorg as part of the normal production schedule. Typically, users of both CDB/Auto-Reorg and CDB/RePart may run one RePart every five to ten reorg runs.

CDB/RePart achieves its high performance through high parallelism and proprietary techniques.

- In the unload phase, all partitions are processed in parallel.
- Unloaded data is reorganized using CDB's no-sort techniques and placed in separate sequential datasets that are ready for the load phase.



Rolling Partitions

- The load phase is extremely fast because all data is already preprocessed and sequenced.
- Indexes and partitions are written in parallel.

These phases are shown schematically in the figure entitled “*CDB/RePart Process*”.

Subset Repartitioning

You may repartition only a subset of partitions. This is useful when, due to high activity, only a sector of your tablespace has become unbalanced, while the rest is in good shape, or when you prefer to split the work required to repartition into multiple days.

Support for Automation

CDB/RePart supports the CDB/Automation scheme. Through Automation, you may automate tasks such as dataset allocation, or the execution of tasks such as rebinds after the repartitioning is complete. In addition, with Automation, you have complete control over the size of the resulting partitions.

Features of Rolling Partitions

CDB/RePart performs the following steps automatically:

- Stops the tablespace.
- Optionally unloads the data from partition 1 to a sequential file in a format suitable for LOAD to another similar table (e.g., a history table).
- Reorganizes all data from Partition 2 to Partition 1 (tablespace and partition index).
- Reorganizes all data from Partition 3 to Partition 2 (tablespace and partition index).

- Reorganizes all data from Partition 4 to Partition 3 (tablespace and partition index).
- Reorganizes partition n to partition n-1.
- Resets (reformats) an empty Partition n (last partition).
- Reorganizes all non-partitioning indexespace, deleting all references to entries that used to be in Partition 1, and adjusting the partition bits for all other entries.
- Updates the Limitkey values in the DB2 Catalog and Directory.
- Does a DB2 MODIFY RECOVERY DELETE AGE(*). Since the definition of the tablespace limitkeys has changed, the tablespace is not recoverable from any prior Image Copy.
- Posts SYSIBM.SYSCOPY for the tablespace with a REORG LOG(NO).
- Sets COPY PENDING for the tablespace.
- Optionally, creates a DB2-compatible Image Copy of the new tablespace, including all new partitions 1 to n-1. Note that this is done concurrently with the above steps, not afterwards. The copy can be to tape or DASD. Up to eight copies can be made at the same time.
- If this copy is done, SYSIBM.SYSCOPY is posted after the REORG with a FULL COPY, and COPY PENDING is turned off.

This process is shown in the figure entitled “*Rolling Partitions*”.