CA Datacom®/DB Data Reorganization

- UNDERSTAND THE CA DATACOM/DB DATA REORGANIZATION PROCESS
- USE DBUTLTY FOR CA DATACOM/DB REORGANIZATION
LEGAL NOTICE

This publication is based on current information and resource allocations as of its date of publication and is subject to change or withdrawal by CA at any time without notice. The information in this publication could include typographical errors or technical inaccuracies. CA may make modifications to any CA product, software program, method or procedure described in this publication at any time without notice.

Any reference in this publication to non-CA products and non-CA websites are provided for convenience only and shall not serve as CA’s endorsement of such products or websites. Your use of such products, websites, and any information regarding such products or any materials provided with such products or at such websites shall be at your own risk.

Notwithstanding anything in this publication to the contrary, this publication shall not (i) constitute product documentation or specifications under any existing or future written license agreement or services agreement relating to any CA software product, or be subject to any warranty set forth in any such written agreement; (ii) serve to affect the rights and/or obligations of CA or its licensees under any existing or future written license agreement or services agreement relating to any CA software product; or (iii) serve to amend any product documentation or specifications for any CA software product. The development, release and timing of any features or functionality described in this publication remain at CA’s sole discretion.

The information in this publication is based upon CA’s experiences with the referenced software products in a variety of development and customer environments. Past performance of the software products in such development and customer environments is not indicative of the future performance of such software products in identical, similar or different environments. CA does not warrant that the software products will operate as specifically set forth in this publication. CA will support only the referenced products in accordance with (i) the documentation and specifications provided with the referenced product, and (ii) CA’s then-current maintenance and support policy for the referenced product.

Certain information in this publication may outline CA’s general product direction. All information in this publication is for your informational purposes only and may not be incorporated into any contract. CA assumes no responsibility for the accuracy or completeness of the information. To the extent permitted by applicable law, CA provides this document “AS IS” without warranty of any kind, including, without limitation, any implied warranties of merchantability, fitness for a particular purpose, or non-infringement. In no event will CA be liable for any loss or damage, direct or indirect, from the use of this document, including, without limitation, lost profits, lost investment, business interruption, goodwill or lost data, even if CA is expressly advised of the possibility of such damages.

COPYRIGHT LICENSE AND NOTICE:

This publication may contain sample application programming code and/or language which illustrate programming techniques on various operating systems. Notwithstanding anything to the contrary contained in this publication, such sample code does not constitute licensed products or software under any CA license or services agreement. You may copy, modify and use this sample code for the purposes of performing the installation methods and routines described in this document. These samples have not been tested. CA does not make, and you may not rely on, any promise, express or implied, of reliability, serviceability or function of the sample code.

Copyright © 2011 CA. All rights reserved. All trademarks, trade names, service marks and logos referenced herein belong to their respective companies. Microsoft product screen shots reprinted with permission from Microsoft Corporation.

TITLE AND PUBLICATION DATE:

CA Datacom®/DB Data Reorganization Green Book Edition 1.0
Publication Date: April 27, 2011
ACKNOWLEDGEMENTS

Principal Authors

Kevin Shuma

The principal author and CA would like to thank the following contributors:

Joseph Lynn, Robert Florian, Vit Gottwald, David Haraga and Ken Eaton.

CA Services

Development

Marketing

Quality Assurance

Support

Engineering Services

Technical Sales

Technical Information
CA TECHNOLOGIES PRODUCT REFERENCES

This document references the following CA Technologies products:

- CA Datacom®/DB
- CA Datacom® Datadictionary™ (Datadictionary)
- CA Datacom® SQL (SQL)

FEEDBACK

Please email us at greenbooks@ca.com to share your feedback on this publication. Please include the title of this publication in the subject of your email response. For technical assistance with a CA Technologies product, please contact CA Support at http://ca.com/support. For assistance with support specific to Japanese operating systems, please contact CA Technologies at http://www.casupport.jp.
Chapter 1: Introduction

Purpose of this Document........................................................................................................9
Audience ................................................................................................................................9
Requirements .........................................................................................................................9
CA Datacom/DB and Data Row Access through Indexes ......................................................10
CA Datacom/DB and Data Row Storage and Ordering ........................................................12
CA Datacom/DB and Data Row Storage Methods (DSOPs) ................................................13
CA Datacom/DB and the Effects of Data Row Placement ...................................................16
CA Datacom/DB Table Partitioning Considerations ..........................................................17
Modern DASD and the Effects of Data Row Placement ......................................................17
Chapter Summary ..............................................................................................................18

Chapter 2: Determining if a Data Row Reorganization Is Needed

What is Sequential Processing .............................................................................................19
Using the Optional Accounting Facility ..............................................................................22
High Level Accounting Tables (A90 and A91) ..................................................................23
High Level Accounting Tables with Key Information (A92 and A93) .............................25
High Level Access with Key Information for Non-CICS (A94 and A95) .......................26
High Level Access with Key Information for Selected DBIDs (A96 and A97) ..............27
High Level Access with Key Information for Selected Time Periods (A98, A99) .........28
Greater Flexibility with the Accounting Facility .................................................................29
Sample Implementation and Use to Determine Sequential Processing .........................29
Finding Data Tables with Significant Sequential Access .................................................30
Breaking Down the Sequential Access by the Key Name ................................................32
Using the Optional History Database ..............................................................................35
Break Down the Sequential Access Efficiency and Compare it to Overall Access Efficiency 38
Using the ADS Table Information to Answer Questions on Sequential Performance ....40
Comparing the Accounting Facility with the ADS Table of the History Database ..........43
Recognizing the Effect on the Environment .......................................................................44
Recognizing the Effect of Larger Buffer Pools .................................................................45
Considering the Use of MRDF .........................................................................................46
Recognizing the Effect of the Application Access ............................................................47
Chapter Summary ..............................................................................................................48

Chapter 3: New Data Table Native Efficiency Report for DBUTLTY

Native Efficiency Report ......................................................................................................49
Native Efficiency Report and Single-Table Data Areas (Non-Compressed) ....................50
Native Efficiency Report and Single-Table Areas (Compressed) ....................................51
How to Compare Efficiency Ratings if the MRB Changes ...............................................52
Native Efficiency Report and Multi-Table Areas ...............................................................52
Sequential Access and the Data Block ............................................................................52
Sequential Access Efficiency by the Native Sequence Key .............................................53
Buffered Sequential Access Efficiency ...........................................................................54
Sample Native Efficiency Reports ....................................................................................56
Chapter Summary .................................................................................................................. 113

Chapter 7: New Data Area Space Usage Report ................................................................. 115
  Data Area Space Report (DATASP) .................................................................................. 115
  DATASP and Single-Table Data Areas (Non-Compressed) .............................................. 115
  DATASP and Single-Table Areas (Compressed) ............................................................. 116
  Using the New DBUTLTY TYPE=DATASP Utility Function ......................................... 116
  Measuring Data Row Available Space in a Data Area Before and After an OLREORG with DSOP 4/5 .. 117
  Chapter Summary .......................................................................................................... 121

Chapter 8: DBUTLTY OLREORG and the Moved Record Index ......................................... 123
  The Moved Records Index is Built in the IXX Area......................................................... 123
  A Recommendation for All OLREORG Users .................................................................. 124
  Chapter Summary .......................................................................................................... 124

Chapter 9: Utilizing Multiple Data Block Sizes ................................................................. 125
  Two Data Buffer Sizes – Two Data Buffer Pools ............................................................ 125
  Reblocking REO 1022 from 4096 to 8192 .................................................................... 126
  Chapter Summary .......................................................................................................... 129

Chapter 10: Table Partitioning and Data Row Reorganizations .......................................... 131
  The Partition Key ............................................................................................................ 131
  Data Area Partition Key Range ...................................................................................... 132
  Offline Data Row Reorganization .................................................................................. 132
  Data Row Placement .................................................................................................... 133
  Sequential Data Row Access for the Parent Table ....................................................... 134
  Online Data Row Reorganization .................................................................................. 135
  Native Efficiency Report (DATANE) ............................................................................... 136
  Chapter Summary .......................................................................................................... 136

Chapter 11: Data Reorganization Green Book Summary ..................................................... 137

Appendix A: Implementing and Using DBUTLTY REPORT=DATANE ............................... 139
  REPORT TYPE=DATANE (Native Sequence Efficiency Report) ..................................... 139
  Restrictions .................................................................................................................... 140
  When to Use .................................................................................................................... 140
  How to Use ..................................................................................................................... 140

Appendix B: Implementing and Using DBUTLTY REPORT=DATASP ............................... 153
  REPORT TYPE=DATASP (Data Area Space Report) ..................................................... 153
  Restrictions .................................................................................................................... 154
  When to Use .................................................................................................................... 155
  How to Use ..................................................................................................................... 155

Glossary .............................................................................................................................. 163

Index .................................................................................................................................. 167
Chapter 1: Introduction

Purpose of this Document

The purpose of this Green Book is to outline the processes associated with the CA Datacom/DB data reorganization. CA Datacom/DB provides multiple ways to reorganize data rows stored in database tables.

While this document is focused on the data reorganization process, several of the techniques and utilities mentioned will be beneficial to users of the CA Datacom/DB environment even if they are not actively doing data reorganizations.

This Green Book has been written for the CA Datacom/DB Version 12.0 (and above) user. Much of the information provided in the book could be useful to users of previous CA Datacom/DB versions. However, the new CA Datacom/DB batch database utility (DBUTLTY) functions are only available at CA Datacom/DB Version 12.0 and above.

Audience

The intended audience is anyone who needs to become familiar with the requirements and use of the CA Datacom/DB utilities and processes used to do data reorganization in the CA Datacom/DB environment. For most sites, this would include the operations staff, database administrators and application programmers who work with the database.

Requirements

This Green Book introduces two new CA Datacom/DB DBUTLTY report functions as well as enhancements to the Data Area Space Management Options (DSOP) and DBUTLTY’s online reorganization (OLREORG) function.

These enhancements have been developed as part of the CA Datacom/DB Version 14.0 product.

For the CA Datacom/DB Version 12.0 user these enhancements are delivered using standard SMP/E maintenance to the CA Datacom/DB Version 12.0 software environment.

A CA Datacom/DB Version 12.0 information solution has been created to document the required maintenance (PTFs) as well as the documentation updates (PDCs). Please see information solution RI29501 for these requirements.
From its earliest beginnings, CA Datacom/DB has supported the use of multiple indexes containing occurrences of the key entity-type to access the data rows stored in its data tables. The key entity-occurrences are created using column values in the data row.

By creating indexes with index values based on row content, data rows can be quickly accessed when the index value is known. For example, if there are 5,000 cattle ranches in the CATTLE_RANCHES table with an index over the column CATTLE_BREED_ID, you could quickly locate the first ranch that raises that breed of cattle, such as Belgian Blue, Hereford, or another breed. Since indexes are always maintained in sequential order, the same index could then be used to sequentially locate the next ranch that raises this breed of cattle, and so on. This type of processing is called skip-sequential access, since a location is skipped to by the first access which locates the first position in the index and the matching data row and then rows are returned sequentially by the same index after that.

In another case, you may want to report all of the breeds of cattle raised and each ranch listed for that breed. In that case, you would start at the beginning of the CATTLE_BREED_ID index in the CATTLE_RANCHES table and read sequentially through the entire index accessing each associated data row and listing each unique breed and the ranches that raise that breed.

The CA Datacom/DB relational DBMS allows the user to have many indexes per table, but it also requires that the user select one specific index as the Master Key. The Master Key is an index where each data row can be uniquely identified by a single index value. One index value equals one data row. CA Datacom/DB does support an extension that allows multiple rows to exist per Master Key index value, but we do not recommend it. In the preceding example, the CATTLE_BREED_ID would occur on many rows and would not be a good choice as the Master Key. Since each ranch listed in the CATTLE_RANCHES table has a unique ranch identification number stored in the RANCH_ID column and each ranch only has one row in the table for each cattle breed it raises, an index built on the RANCH_ID and CATTLE_BREED_ID columns would be more appropriate as the Master Key. The decision on which column is first in the key is dependent on how the data rows are most likely to be accessed. Either order (RANCH_ID, CATTLE_BREED_ID or CATTLE_BREED_ID, RANCH_ID) of the columns in the index would still provide the uniqueness preferred for a Master Key.
CA Datacom/DB also supports the use of secondary indexes on a given table. The physical limitation is 999 defined indexes for all the tables in a CA Datacom/DB database. CA Datacom/DB has a current limit of 5000 databases per CA Datacom/DB environment.

In a CA Datacom/DB database, the defined indexes are stored in the Index Area. The primary Index Area, also called the IXX, is a global storage area that is maintained by the DBMS. Individual indexes are stored and managed by the DBMS in this area. Each of the defined indexes stores the appropriate information in an index entry from the table row’s data columns and the address of the row or rows that the index entry represents.

With CA Datacom/DB Version 12.0, a new enhancement known as the “multi-dataset index” was delivered. This enhancement allows multiple index areas to be defined to hold a database’s indexes. In this case, the DBA defines the IXX area and one or more “Inn” areas, where the nn is a number between 00 and 99. The DBA can select in which index area, either IXX or an Inn, that a specific index and its associated index entries will be stored.

This enhancement was added to give a DBA the ability to assign indexes to a specific data set so that the performance options, such as the Memory Resident Data Facility (MRDF) covering, could be used to improve the performance on certain highly referenced indexes. In most implementations, the multi-dataset will be in use only on databases with large index areas, that is, where many indexes are defined, where separating a certain index to its own data set provides a significant performance benefit.

You define memory resident data areas or index areas to the MRDF with the MUF startup options COVERED and VIRTUAL. For information, see the CA Datacom/DB Database and System Administration Guide.

While the creation of the index group, that is, the IXX plus one or more Inn data sets, changes the physical storage of the database index, it does not change the way the index processing works or how the index participates in the data row access and data row reorganization processes.

For the remainder of this book, we will refer to the index area as the typical single IXX data set implementation.
Like the indexes, the database tables’ data rows are stored in a physical data set known as a data area. There can be multiple tables in a data area. The limit is 240 tables per data area. We recommend that you have one data table per data area for user data tables.

The DBMS manages the placement of the table data rows in the data area according to the method selected by the DBA. We will discuss these various methods later in this document.

Since there is only one copy of the data rows, the data row order can only follow one ordering or sequence. In CA Datacom/DB, we call this sequence the native sequence. To implement the native sequence, the user selects one of the indexes that has been defined, to be the native sequence key, known as the Native Key.

When selecting the index that is to be the Native Key, the user determines what index will be used the most by applications that need to process large amounts of data in sequence. Ordering data rows to match this sequence improves performance for these sequential applications processing by this index.

In some cases, the data rows in a given table are not accessed in large quantities in any given sequence. In those cases, the user typically selects the Master Key to be the Native Key as well.

In CA Datacom/DB, the index entries must remain in sequence to provide reliable and efficient data row access. When new index entries are added, we must insert the entries in the appropriate place. Sometimes for this to occur, index blocks must be split into two blocks to create room for the new entries. This process occurs in the background and, in most cases, does not affect user processing. The index area of a CA Datacom/DB database is constantly changing as index entries are added and deleted.

For more information on designing and using the keys that are used in indexes, see the CA Datacom/DB Database and System Administration Guide.
CA Datacom/DB and Data Row Storage Methods (DSOPs)

Since the majority of CA Datacom/DB processing is index based, there is not a requirement for data rows to remain in any specific sequence. In fact, the amount of overhead required to constantly adjust data row placement to match the Native Key would be significant and in some cases prohibitive.

Typically, database update processing does not affect a data row’s placement within the data area. Delete processing removes the data row and its index entries, but does not affect the remaining data rows’ placements. Therefore, the majority of the data row placement decisions are based on where new rows should be added.

To provide flexibility to the user, CA Datacom/DB provides the user with various choices on how new data row storage will be handled. We call these management choices Data Space Options, or DSOPs.

For each data area, the user selects one of the following DSOP options:

No Reuse (DSOP 0)

This option indicates that there is no attempt to reuse space created in data blocks by deleted rows. All new rows are added to the last active data block. When the last active data block is full, the next unused block is allocated and rows inserted there until it is full, and so on. Data row order for these added rows is based on the order of processing and does not reflect any particular index sequence.

DSOP 0 always inserts new rows to the back of the data set, there is no overhead spent in looking at the existing blocks to see if space is available for a new row.

Since DSOP 0 does not reuse deleted space within the data block, the user can easily determine the fullness of a data area by comparing the total blocks in the data area to the number of data blocks in use. The CXX report provides this information. You can also retrieve the information from the Dynamic System Tables.

DSOP 0 is usually best used for tables where the data rows are not deleted. For example, a personnel master table that must keep all personnel records on file, or a transaction file where rows are collected all day and extracted from the table at night, and the table is marked empty using the DBUTLTY LOAD FORMAT=NONE function to start the next day.
Random Reuse (DSOP 1)

This option indicates that there will be active attempts by the DBMS to reuse deleted row space. When data rows are added they can be randomly inserted into any data block that has space or to an empty block as needed. Once a row is added to an empty block, the block is marked as in use.

DSOP 1 is the space management default option and provides the best way to reuse data row space in the exiting in-use blocks with minimal amount of overhead. With DSOP 1, most in-use data blocks will remain full, as long as there are as many adds as deletes.

Since DSOP 1 does its very best to add new rows in the deleted space in the in-use blocks before using space in empty blocks, the blocks in-use percent in the CXX report provides a relatively accurate measurement for used space in the data area. Once DSOP 1 selects a block (partially full or empty) to use to receive new rows, it continues to use that block for add processing until the next add will not fit in that data block. At this point, the DBMS selects a block from the “in use, but not full” set of blocks for add processing. If there are no “in use, but not full” data blocks, the processing selects the first available empty block.

DSOP 1 is usually best used for tables where data rows are deleted and added frequently and without any specific regard to the Native Key sequence. For example, an activity table that has one row for each person that is online in an application. As people enter the application, rows are added, as people leave, rows are deleted.

Sequential Reuse (DSOP 2)

This option indicates that the DBMS tracks the deleted row space. However, when data rows are added, they are added to the last active data block, similar to the way DSOP 0 operates. In DSOP 2, when the data blocks at the back of the data area are full, the process goes to the first block of the data area. If there is available space from deletes or other processes, new data rows are placed in that block until it is full. The process moves forward through each data block filling available space. The process is continuously wrapping around through the data area to store new data rows.

DSOP 2 is usually best used for tables where data rows are always added with a next higher Native Key value and rows are typically deleted from the lowest Native Key values. For example, a Purchase Order table where PO# is the Native Key. All PO#s are always ascending. When goods are received and processed the PO#s for those goods are deleted. Another example would be a sales history table where rows are always added with the sales date as the Native Key. Each month the user deletes all sales history rows that are over 6 months old.

Since DSOP 2 places new rows at the end of the data area and does not attempt to reuse deleted space until processing has wrapped back to the first block of the data area, the blocks-in-use percent in the CXX report may not provide an accurate measurement for available space in the data area.
Clustered Reuse (DSOP 3)

In this option, the user selects a high-order portion of the Native Key as the cluster key. The DBMS assigns a data block for each unique cluster key value when the first row is added with that value. All additional rows with the same cluster key value are added to the data block. When the assigned data block becomes full, an additional data block is assigned. This process continues until all data blocks are assigned to a cluster key value. To avoid false area full conditions once all data blocks have been assigned, DSOP 3 will allow new rows to be added to data blocks not assigned to this cluster key value.

DSOP 3 is usually best used for tables where the user can reasonable estimate how many data rows will be added for each cluster key value. DSOP 3 will handle the occasion where a certain cluster key value has an abnormally high number of rows associated with it. For example, an insurance history table has 6 to 8 rows in the table for each of its customers. A row is added when the customer first opens a contract with the company. Then, over the next few days, another 6 to 7 rows are added for the customer. By having the assigned cluster key, the rows for the customer are stored together in the data block. If an additional row is added and it does not fit in the assigned data block, a new data block is assigned from the empty block list.

To maximize the space in each data block of the cluster, CA Datacom/DB maintains a list of free space in each block of the cluster. When a new row is added to the cluster, it is placed in the data block with the least amount of free space that still allows the new row to fit. This extra check takes additional overhead, but maximizes the ability to cluster rows when the rows are from multiple tables or the rows are in compressed form.

Since DSOP 3 allocates data blocks at the first add of a row with a specific cluster key value, the blocks-in-use percent in the CXX report will not provide an accurate measurement for available space in the data area.

For most implementations, the planning and data row knowledge to successfully implement DSOP 3 is significant. In addition, the allocation of data blocks across cluster key values and the extra overhead involved in the process (CPU and I/O) make clustering a seldom selected option. Successful implementations would be as previously described where the data has a relatively well defined set of rows that can be planned and associated with a cluster key.

Clustered Reuse without a cluster key (DSOP 3)

This is a special implementation of the DSOP 3 space management. In this option, the user does not specify any key value for the cluster key.

Without a cluster key, the data area clusters data rows by the table’s 3-character CA Datacom/DB short name for the TABLE occurrence. This means that no attempt at sequencing of the data will be done. Instead, CA Datacom/DB maps the free space available in each data block assigned to the table’s DATACOM name. When a new row is added, it is placed in the data block that has the smallest amount of free space that will still accommodate the row.
This varies from DSOP 1 and DSOP 2 in that each added row is placed in a block according to its 3-character short name and its size. Data row placement is based purely on data row size. For data areas that have widely varying row sizes, there will be significant randomness to the data blocks selection which could increase the I/O and CPU overhead for storing a data row.

Successful implementation for this special DSOP 3 option would be based on the need to store a set of rows with widely ranging row sizes as efficiently as possible on the DASD with the understanding that data row placement would not be based on any key value or processing order.

For more information on the data area space management options (DSOP), see the CA Datacom/DB Database and System Administration Guide.

Having an understanding of how the data row storage works in the various DSOP settings will assist you in determining when and what type of reorganization should be used for a given data area.

CA Datacom/DB and the Effects of Data Row Placement

From the first database implementation in CA Datacom/DB, there has been a requirement to monitor the data row placement, and the sequence, within the data area and to determine when that data row placement was having a negative effect on the applications that were sequentially accessing the data rows.

Once it was determined that the data row placement was having a negative effect and that re-ordering the rows into the Native Key order would improve that performance, the user was faced with a challenge.

How do I re-order my data rows with the least impact on my applications that are accessing that data?

In past years, this was accomplished by scheduling a set of batch database utility (DBUTLTY) functions that would run during the “dark window” when all databases were down and no applications were running against it.
In today’s world, these available dark windows are becoming smaller and smaller, and the user must look for ways to achieve the data reorganization within these smaller windows or, in some cases, reorganizing the data rows when no window exists at all.

For many sites today, the data access interruption caused by an offline reorganization process can no longer be tolerated. These sites require user data tables to be available 24x7, or as close to 24x7 as possible.

The next set of chapters covers a suggested process for meeting these reorganization needs while supporting the continuous access requirements of today.

CA Datacom/DB Table Partitioning Considerations

CA Datacom/DB supports a feature known as table partitioning. Table partitioning distributes the data rows of the partitioned table into multiple data areas. The data row placement criteria using a partitioning key can have significant affect on data row sequential access and data row reorganization.

If your site has partitioned tables, see chapter Table Partitioning and Data Row Reorganization (see page 131).

Modern DASD and the Effects of Data Row Placement

When mainframe computing began, DASD storage access and performance was a constant concern when managing database systems. The main reason for this concern was that the majority of the work done by the DBMS revolved around the retrieval and storage of data on a DASD device. These operations triggered Input-Output (I/O) operations which were costly in CPU and response time.

Even with significant DBMS data buffering capabilities and today’s extremely fast DASD, DBMS systems are still I/O bound and sensitive to DASD device performance.

However, there has been a significant change in the last 20 years in the DASD architecture of how the actual physical DASD stores the data blocks. This has changed the way we think about sequential I/O.
Old mainframe DASD devices like the 3375, 3380 and the 3390 (non-emulated) stored the data
blocks on DASD tracks. DASD tracks were stored on DASD cylinders and the DASD cylinders were
part of a DASD volume. In these devices, the physical location of the data blocks was very
specific. The first set of data blocks were on the 1st track of the 1st cylinder, following sets of
blocks were in the next track, and so forth.

A given data table’s access performance was affected the by the physical location of its data
blocks and the nearness of those blocks to other blocks from the same table. This was
particularly evident when doing sequential processing.

Modern DASD devices, such as the IBM Enterprise Storage Server (ESS or Shark) device no longer
have the same physical characteristics of the old 33xx devices. Instead, these devices house
hundreds or thousands of smaller storage units that can be logically configured to emulate
multiple 3380 or 3390 volumes.

These logically configured devices still support the logical image of the
block/track/cylinder/volume architecture found on the 3380 and 3390 devices. The user still sees
his data set as a set of consecutive data blocks that reside on a set of consecutive tracks or
cylinders on a set of DASD volumes. However, the actual physical storage of the data blocks in
the data set is randomly spread, or balanced, across the entire DASD device as a series of small
storage areas, also called storage pages. This is done while still providing high-performance, data
redundancy, and a wealth of other advantages. These advantages in storage management have
reduced the negative effects that were previously associated with secondary extents and
multiple volume data sets.

For the DBMS software, this implementation reduces the need to concentrate on keeping data
blocks in perfect sequential order according to the logical image block/track/cylinder number
and to move the focus to keeping rows in sequential order within the data blocks.

This chapter has explained the basics of data row storage and index access to the data rows in a
CA Datacom database. This type of indexed processing to access data rows is common among
most mainframe database and file management systems.

A significant point to remember is that with modern data storage devices, the physical layout of
data storage pages housing a set of tracks and cylinders will be balanced across the entire
storage device and will not be assigned in any sequential or consecutive placement. This is a
major point of consideration and key to the various online reorganization strategies discussed in
the following chapters of this Green Book.
Chapter 2: Determining if a Data Row Reorganization Is Needed

Data row reorganization is the process where data rows are re-ordered into a selected sequence. In CA Datacom, we refer to this sequence as the Native Key sequence. Data row reorganizations are done to improve performance of application processes that sequentially process the data rows using the Native Key. For most sites, these applications are typically batch programs that process large segments of the data table.

Prior to CA Datacom/DB r11, the determination of which tables needed to be reorganized was based on subjective observations such as:

- The batch sequential programs seem to be taking longer to complete.
- The data area statistics show that there are a lot of data row overflows.
- You have a standard procedure to reorganize this database weekly.

While each of these indications could mean that a data row organization may be needed, they are not consistently accurate. In many cases, users may be doing data reorganization where there is minimal benefit to their applications.

The goal of this chapter is to document some best practices for making the determination of whether data row reorganization is needed.

What is Sequential Processing

When a user begins at a place in the data table using the index and selects that row then asks the database to provide the next row in sequence, we call this sequential processing. Sequential processing is based on a selected index order.

Two of the three data retrieval APIs use database indexes to retrieve data rows sequentially.

**RAAT (Record-At-A-Time)**

The RAAT API contains a specific set of next commands (REDNX, RDUNX, GETIT, and so on) that retrieve data rows in sequential order based on the index selected in the request. These commands must be preceded by a command that establishes a position in the index.
What is Sequential Processing

**SAAT (Set-At-A-Time)**

The SAAT API user does not specify a particular index to use to retrieve data rows. Instead, the user provides a set of selection and ordering criteria based on the columns in the data table. The internal SAAT parser, known as the Compound Boolean Selection facility (CBS), analyzes the column selection and ordering criteria and chooses the best index path to select and order the data. In some cases, the data rows can be best retrieved by doing specific keyed retrievals. In other cases, it may be best to retrieve the data rows by doing a sequential process based on the selected index.

**SQL (Structured Query Language)**

Datacom also supports ANSI SQL as an API. However, all SQL requests that need to retrieve data rows do so by issuing one or more SAAT requests. This API conversion is all done under the covers by the SQL and SAAT parsers.

**Sequential Processing by the Native Key**

When the selected index is the Native Key, it is called sequential processing by the Native Key. Since data row reorganizations re-order the data rows by the Native Key, typically it is only sequential processing by the Native Key that is going to be improved by data row reorganizations.

The following table shows two programs that are reading all the rows in a selected table using the sequential commands GSETL and GETIT. The first pair of entries is for a program accessing using the Native Key and the second pair of entries is for a program using a different (non-native sequence) key.

The first entry of each pair shows the resource consumption, that is the physical I/O, to process the table when the data rows are unorganized, that is, not in Native Key sequence. The second entry of each pair shows the processing after the data rows have been reorganized into Native Key sequence.
For this example, we have set the dedicated sequential buffer pool to four buffers and have not placed any data in the Memory Resident Data Facility (MRDF).

<table>
<thead>
<tr>
<th>GSETL Read Program</th>
<th>Buffers</th>
<th>MRDF</th>
<th>Data Logical</th>
<th>Data Physical I/O</th>
<th>Logical Change</th>
<th>Percent Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unorganized Read by native</td>
<td>4</td>
<td>No</td>
<td>31,953</td>
<td>3,509</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reorganized Read by native</td>
<td>4</td>
<td>No</td>
<td>31,953</td>
<td>448</td>
<td>0%</td>
<td>87%</td>
</tr>
<tr>
<td>Unorganized Read by non-native</td>
<td>4</td>
<td>No</td>
<td>31,953</td>
<td>25,373</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reorganized Read by non-native</td>
<td>4</td>
<td>No</td>
<td>31,953</td>
<td>30,783</td>
<td>0%</td>
<td>-21%</td>
</tr>
</tbody>
</table>

The key point to note is that the logical requests (Data Logical) done by each program are the same. The program that is reading the data rows in native sequence shows a significant reduction in physical I/O (Data Physical I/O) after the data row reorganization.

Even with this relatively small table, the resource consumption savings by the reorganization was significant.

We must also note that for the sequential access by the non-native sequence, the resource consumption was negatively affected by the reorganization. This is something that cannot be controlled, since the data can only be stored in one order or sequence. If the batch program that is not using the Native Key is being seriously affected by the reorganization and is critical to the business, the user may need to review whether this program should be changed to use the Native Key.

For CA Datacom/DB tables, typical random access by a key is not directly affected by the data row placement. In most cases, the same is true of sequential processing by a key that is not the Native Key.

For most sites, data row placement only affects processing where the applications are doing heavy sequential access using the Native Key.

**Determining the Negative Effect**

To determine which tables’ access is negatively affected by data rows not in native sequence, you must determine which data tables have a significant amount of sequential processing by the Native Key.
Using the Optional Accounting Facility

Once the list of data tables with heavy sequential processing is determined, you must determine if the current data row order has a negative effect on the sequential processing by the Native Key. The question typically requires additional research to answer questions such as the following:

■ Even though it is heavy sequential processing, is the processing as part of the total processing significant enough to worry about?

■ Is there reason to believe that the data row placement is negatively affecting sequential performance?

■ Is there reason to believe that a data row reorganization would significantly improve the performance of the native sequential access?

The following sections describe two different techniques that can be used to help answer these questions.

Using the Optional Accounting Facility

The CA Datacom/DB Accounting Facility allows you to define a persistent database that can be used to collect a wide variety of information. The Accounting Database supports up to 100 different Accounting tables. Each Accounting table can be defined to collect different types of information about the activity in the database region, that is, the CA Datacom/DB Multi-User Facility (MUF).

The CA Datacom/DB Database and System Administration Guide provides detailed information on the definition, implementation and use of the Accounting Facility. We recommend that you become familiar with the Accounting Facility and the information provided in the guide before attempting to implement the examples detailed below.

This section is focused on adding an additional set of Accounting tables that can be used to collect information on total activity and sequential activity for a given MUF. The Accounting table definitions demonstrated in this chapter are available from CA Support and are covered in the Information solution mentioned earlier in this guide. See Requirements (see page 9).

A key point to remember is that each Accounting table has a defined Master Key. The Accounting Facility creates a single row in the Accounting table for each unique Master Key value encountered while the Accounting table is open. Once a row is added for a given key value, subsequent requests with this same key value have their data values merged into the existing row. The CA Datacom/DB Database and System Administration Guide explains how each Accounting column type is handled during this merging.
Basically, what this means is that the more generic the key value, the less unique rows are created and the more data that is merged into the row. A more detailed key value will create more rows with less merged data. The more rows that are created and maintained, the higher the Accounting table overhead will be in the MUF.

The best situation is to select a key-value that gives you the needed level of detail in the least number of Accounting rows.

In addition to limiting the detail level of the Accounting Master Key values, you can also limit the focus of the collection. This can be done in the selection logic of the Accounting table. You can limit the focus by specifying selection criteria that limits the collection to certain databases, certain job types such as batch processing, certain time periods or days of the week, and so on.

The following samples demonstrate some of these selection criteria. In each pair of tables, the “all accesses” table includes all types of requests being processed while the “sequential accesses” table includes only requests that were made using one of the CA Datacom/DB commands specifically used for sequential access.

By creating the pair of Accounting tables, you can effectively measure the percentage of sequential activity to the overall or total activity. This is important information when considering if a table should be reorganized.

**High Level Accounting Tables (A90 and A91)**

The A90 and A91 pair of Accounting tables is focused on collecting the minimal amount of information on the user tables being processed. For each day, each user table that was accessed that day has one row created in the all accesses Accounting table (A90) and one row created in the all sequential access Accounting table (A91). By comparing the data for a selected user table in A90 to the same user table in A91, we can determine the amount of sequential processing and its relative performance to the total processing and its relative performance.

By relative performance comparison, we mean comparing the amount of logical data requests (LOGDT) to the following:

- The amount of physical data I/Os (EXCDT)
The elapsed time to do the requests (ETIME)

The MUF active or run time to do the requests (RTIME)

By comparing the sequential requests to the total requests we can easily determine the amount of sequential processing of the total. While this pair of Accounting tables collects the least amount of information (the most efficient Accounting process), they also combine the sequential access for all keys together. Since a data row reorganization only improves sequential access by the Native Key, having this combined access information where all keys are combined may not provide the level of granularity that we need. It will depend on how often non-Native Keys are used for sequential processing in the site.

**Table: A90 (all accesses)**

Table columns:
- BASE, ANAME, TNAME, DATE,
- REQS, ETIME, RTIME, LOGIX, LOGDT, EXCIX, EXCDT

Master Key columns:
- BASE, ANAME, TNAME, DATE

Selection criteria defined in the Datadictionary text:

None

**Table: A91 (all sequential accesses)**

Table columns:
- BASE, ANAME, TNAME, DATE,
- REQS, ETIME, RTIME, LOGIX, LOGDT, EXCIX, EXCDT

Master Key columns:
- BASE, ANAME, TNAME, DATE

Selection criteria defined in the Datadictionary text:

```sql
/(     (COMND = 'REDNX') OR (COMND = 'RDUNX')
     OR (COMND = 'REDBR') OR (COMND = 'RDUBR')
     OR (COMND = 'REDNR') OR (COMND = 'RDUNR')
     OR (COMND = 'REDNK') OR (COMND = 'RDUNK')
     OR (COMND = 'REDNE') OR (COMND = 'RDUNE')
     OR (COMND = 'GETIT') OR (COMND = 'SELNR')
   )  /
```
High Level Accounting Tables with Key Information (A92 and A93)

The A92 and A93 pair of Accounting tables is focused on collecting the minimal amount of information on the user tables processed by the key names (KNAME) being used to access the data. To do this, we add the KNAME column to the Accounting table Master Key definition. We will have multiple rows, one per key used, in the Accounting table for each user table accessed. By comparing the data for a selected table/key in A92 to the same table/key in A93, we can determine the amount of sequential processing and its relative performance to all access to the table by a specific key.

As noted previously, data row reorganization affects the performance of sequential access by Native Key. This Accounting table allows the user to determine the amount of sequential access by the Native Key as well as the other keys in the database. If multiple keys in the table have the same high-level columns as the Native Key, sequential processing by those keys may also see performance benefits by doing a data row reorganization.

Table: A92 (all accesses)

Table columns:

- BASE, ANAME, TNAME, DATE, KNAME,
- REQS, ETIME, RTIME, LOGIX, LOGDT, EXCIX, EXCDT

Master Key columns:

- BASE, ANAME, TNAME, DATE, KNAME

Selection criteria defined in the Datadictionary text:

None

Table: A93 (all sequential accesses)

Table columns:

- BASE, ANAME, TNAME, DATE, KNAME,
- REQS, ETIME, RTIME, LOGIX, LOGDT, EXCIX, EXCDT

Master Key columns:

- BASE, ANAME, TNAME, DATE, KNAME
Selection criteria defined in the Datadictionary text:

```
/ ( (COMND = 'REDNX') OR (COMND = 'RDUNX')
  OR (COMND = 'REDBR') OR (COMND = 'RDUBR')
  OR (COMND = 'REDNR') OR (COMND = 'RDUNR')
  OR (COMND = 'REDNK') OR (COMND = 'RDUNK')
  OR (COMND = 'REDNE') OR (COMND = 'RDUNE')
  OR (COMND = 'GETIT') OR (COMND = 'SELNR')
) /
```

High Level Access with Key Information for Non-CICS (A94 and A95)

The A94 and A95 pair of Accounting tables is focused on collecting the minimal amount of information on the user tables processed by the key names (KNAME) being used to access the data. However, we are excluding CICS requests from the collection. To do this, we add additional selection criteria to both Accounting tables. We still have multiple rows, one per key used, in the Accounting table for each user table accessed. By comparing the data for a selected table/key in A94 to the same table/key in A95, we can determine the amount of sequential processing and its relative performance to all access to the table by a specific key.

Since CICS applications typically do not do significant sequential processing, eliminating the CICS jobs from the Accounting data collection reduces the Accounting tables' overhead in MUF. We exclude CICS jobs with the JTYPE NE 02 specification. This is the internal job classification that the MUF assigns to a CICS region.

**Table: A94 (all accesses)**

Table columns:
- BASE, ANAME, TNAME, DATE, KNAME,
- REQS, ETIME, RTIME, LOGIX, LOGDT, EXCIX, EXCDT

Master Key columns:
- BASE, ANAME, TNAME, DATE, KNAME

Selection criteria defined in the Datadictionary text:
```
/ (JTYPE NE "02") /
```

**Table: A95 (all sequential accesses)**

Table columns:
- BASE, ANAME, TNAME, DATE, KNAME,
- REQS, ETIME, RTIME, LOGIX, LOGDT, EXCIX, EXCDT
Master Key columns:

BASE, ANAME, TNAME, DATE, KNAME,

Selection criteria defined in the Datadictionary text:

\[
\begin{align*}
&/ \ ( \text{COMND} = \text{'REDNX'} \) \ OR \ ( \text{COMND} = \text{'RDUNX'} \) \\
&\ OR \ ( \text{COMND} = \text{'REDKR'} \) \ OR \ ( \text{COMND} = \text{'RDURR'} \) \\
&\ OR \ ( \text{COMND} = \text{'REDBR'} \) \ OR \ ( \text{COMND} = \text{'RDURR'} \) \\
&\ OR \ ( \text{COMND} = \text{'REDNR'} \) \ OR \ ( \text{COMND} = \text{'RDURNR'} \) \\
&\ OR \ ( \text{COMND} = \text{'REDNK'} \) \ OR \ ( \text{COMND} = \text{'RDUNK'} \) \\
&\ OR \ ( \text{COMND} = \text{'REDNE'} \) \ OR \ ( \text{COMND} = \text{'RDUNE'} \) \\
&\ OR \ ( \text{COMND} = \text{'GETIT'} \) \ OR \ ( \text{COMND} = \text{'SELNR'} \) \\
&\ \AND\ (JYTPH \ NE \ "02") \\
&/ \nonumber
\end{align*}
\]

High Level Access with Key Information for Selected DBIDs (A96 and A97)

The A96 and A97 pair of Accounting tables is focused on collecting the minimal amount of information on the user tables processed by the key names (KNAME) being used to access the user data. We are limiting the data collection to only requests for selected user databases (DBIDs). To do this, we add selection criteria. We have multiple rows, one per key used, in the Accounting table for each user table accessed. By comparing the data for a selected user table/key in A96 to the same user table/key in A97, we can determine the amount of sequential processing and its relative performance to all access to the table by a specific key.

The Accounting table pair has the selection criteria that limit the collection to those databases with DBIDs greater 20 and less than 2000.

Table: A96 (all accesses)

Table columns:

BASE, ANAME, TNAME, DATE, KNAME,

REQS, ETIME, RTIME, LOGIX, LOGDT, EXCIX, EXCDT

Master Key columns:

BASE, ANAME, TNAME, DATE, KNAME

Selection criteria defined in the Datadictionary text:

\[
/ \ ( \text{BASE} > 20) \ \AND \ ( \text{BASE} < 2000) / \nonumber
\]
Using the Optional Accounting Facility

### Table: A97 (all sequential accesses)

**Table columns:**

- BASE, ANAME, TNAME, DATE, KNAME,
- REQS, ETIME, RTIME, LOGIX, LOGDT, EXCIX, EXCDT

**Master Key columns:**

- BASE, ANAME, TNAME, DATE, KNAME

**Selection criteria defined in the Datadictionary text:**

```sql
/ ( (COMND = 'REDNX') OR (COMND = 'RDUNX')
  OR (COMND = 'REDBR') OR (COMND = 'RDUBR')
  OR (COMND = 'REDNR') OR (COMND = 'RDUNR')
  OR (COMND = 'REDNK') OR (COMND = 'RDUNK')
  OR (COMND = 'REDNE') OR (COMND = 'RDUNE')
  OR (COMND = 'GETIT') OR (COMND = 'SELNR')
 )

AND
  ( (BASE > 20) AND (BASE < 2000) )
/```

### High Level Access with Key Information for Selected Time Periods (A98, A99)

The A98 and A99 pair of Accounting tables is focused on collecting the same amount of information as the previous Accounting tables, but we will limit the focus to just Monday through Friday and from 8:00:00 AM to 5:59:59 PM.

### Table: A98 (all accesses)

**Table columns:**

- BASE, ANAME, TNAME, DATE, KNAME,
- REQS, ETIME, RTIME, LOGIX, LOGDT, EXCIX, EXCDT

**Master Key columns:**

- BASE, ANAME, TNAME, DATE, KNAME

**Selection criteria defined in the Datadictionary text:**

```sql
/ ( (DAY NOT = 'S') AND
  (TIME GT '07' AND TIME LT '18')
 )
/```
Table: A99 (all sequential accesses)

Table columns:

BASE, ANAME, TNAME, DATE, KNAME,
REQS, ETIME, RTIME, LOGIX, LOGDT, EXCIX, EXCDT

Master Key columns:

BASE, ANAME, TNAME, DATE, KNAME

Selection criteria defined in the Datadictionary text:

```
/(     (COMND = 'REDNX') OR (COMND = 'RDUNX')
OR (COMND = 'REDBR') OR (COMND = 'RDUBR')
OR (COMND = 'REDNR') OR (COMND = 'RDUNR')
OR (COMND = 'REDNK') OR (COMND = 'RDUNK')
OR (COMND = 'REDNE') OR (COMND = 'RDUNE')
OR (COMND = 'GETIT') OR (COMND = 'SELNR')
)
AND    ( (DAY NOT = 'S') AND
(TIME GT '07' AND TIME LT '18') )
```

Greater Flexibility with the Accounting Facility

As you can see, there is a great amount of flexibility in the Accounting Facility on the types of information collected, when it is collected, and so forth. The Accounting information is stored in a persistent table so that it can be accessed when you need it. This also means that you must purge the collected data when it is no longer needed.

To obtain the greatest benefit from the preceding, you must be familiar with the Accounting Facility implementation and use as described in the CA Datacom/DB Database and System Administration Guide.

Sample Implementation and Use to Determine Sequential Processing

For this example, we are going to employ the A90, A91, A92, and A93 Accounting tables. For sites with SQL, the following two-step process could be accomplished by using more complex SQL syntax using sub-selects, groups by, and so forth. However, to keep the process simple and to allow for non-SQL access using CA Dataquery DQL, we use multiple steps to review the four Accounting tables selected.
We implement the Accounting tables and open them to collect information on tables accessed in the MUF over a period of time. We then process that data using queries to determine the amount of sequential access.

Using the Accounting table information collected in tables A90 and A91, you can create a simple SQL or CA Dataquery DQL query to locate data tables that have significant sequential access.

**Finding Data Tables with Significant Sequential Access**

For this example, we determine that we want to scan the Accounting tables’ data for the last seven days and locate any data tables that have had what we consider significant sequential access.

To qualify, a data table would be listed if it has at least 10,000 sequential accesses (REQS) in any one of the seven selected dates.

For qualifying tables:

- Join the information from the total access table (A90) with the information from the sequential access table (A91)
- List the data tables in order of highest number of sequential accesses to lowest
- List the tables multiple times if they hit the criteria on multiple days

The following SQL query provides this information. For a non-SQL site, you could use the CA Dataquery DQL to obtain the first set of information.
SELECT
   A91.BASE, A91.ANAME, A91.TNAME, A91.ACTDATE,
   A90.REQS AS TOTAL_REQS, A91.REQS AS SEQ_REQS,
   DECIMAL(A91.REQS / (A90.REQS / 100), 5, 2) AS PERCENT_SEQ
FROM
   SYSADM.SAMP_ACT_DB_A90 A90,
   SYSADM.SAMP_ACT_DB_A91 A91
WHERE
   ((A91.BASE = A90.BASE) AND
    (A91.ANAME = A90.ANAME) AND
    (A91.TNAME = A90.TNAME) AND
    (A91.ACTDATE = A90.ACTDATE))
AND
   (( DATE ( '20' ||SUBSTR(A91.ACTDATE,1,2)||'
               -'
               ||SUBSTR(A91.ACTDATE,3,2)||'
               -'
               ||SUBSTR(A91.ACTDATE,5,2)) ) > (CURRENT DATE - 7 DAYS))
   AND
   (A91.REQS > 100000)
ORDER BY
   A91.REQS DESC;

The output of the query follows. Note that in this system there were only three data areas with heavy sequential access as defined by the query in this MUF. The three data areas had multiple heavy use days during the last seven days.

<table>
<thead>
<tr>
<th>BASE</th>
<th>ANAME</th>
<th>TNAME</th>
<th>ACTDATE</th>
<th>TOTAL_REQS</th>
<th>SEQ_REQS</th>
<th>PERCENT_SEQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>4033</td>
<td>603</td>
<td>MET</td>
<td>101014</td>
<td>54,742,252</td>
<td>46,377,170</td>
<td>85%</td>
</tr>
<tr>
<td>4033</td>
<td>603</td>
<td>MET</td>
<td>101013</td>
<td>52,001,457</td>
<td>45,494,288</td>
<td>87%</td>
</tr>
<tr>
<td>4033</td>
<td>603</td>
<td>MET</td>
<td>101015</td>
<td>50,189,117</td>
<td>44,611,406</td>
<td>89%</td>
</tr>
<tr>
<td>1022</td>
<td>REO</td>
<td>REO</td>
<td>101009</td>
<td>3,578,676</td>
<td>3,131,184</td>
<td>87%</td>
</tr>
<tr>
<td>1023</td>
<td>REO</td>
<td>REO</td>
<td>101009</td>
<td>3,562,604</td>
<td>3,103,828</td>
<td>87%</td>
</tr>
<tr>
<td>1022</td>
<td>REO</td>
<td>REO</td>
<td>101013</td>
<td>1,789,338</td>
<td>1,565,592</td>
<td>87%</td>
</tr>
<tr>
<td>1022</td>
<td>REO</td>
<td>REO</td>
<td>101014</td>
<td>1,789,338</td>
<td>1,565,592</td>
<td>87%</td>
</tr>
<tr>
<td>1023</td>
<td>REO</td>
<td>REO</td>
<td>101013</td>
<td>1,796,352</td>
<td>1,565,592</td>
<td>87%</td>
</tr>
<tr>
<td>1023</td>
<td>REO</td>
<td>REO</td>
<td>101014</td>
<td>1,796,352</td>
<td>1,565,592</td>
<td>87%</td>
</tr>
<tr>
<td>1022</td>
<td>REO</td>
<td>REO</td>
<td>101015</td>
<td>1,789,091</td>
<td>1,563,572</td>
<td>87%</td>
</tr>
</tbody>
</table>

This is just an example of possible criteria a site can use to locate those data tables that have heavy sequential access. The selection criteria used is dependent on the site. One recommendation would be to adjust the WHERE clause (REQS > 10,000) so that you can return only the top 20 data tables by sequential access. As you become more familiar with the process you can adjust the WHERE clause to expand to view more tables as needed.
Breaking Down the Sequential Access by the Key Name

For the tables selected, break down the sequential access by the key name used to do the access.

Using the list of heavy sequential use tables that we found using the queries against Accounting tables A90 and A91, we can now look at the processing done by the key selected on each table. Remember that it is only sequential processing by the Native Key that will be improved by a data row reorganization. Sequential access by non-native keys may not be helped by the reorganization. Non-native keys that have the same high order columns as the Native Key’s high order columns have the best chance to be helped by a data organization.

For each of the selected tables, we look at the last 30 days of history of activity. This allows us to see if there are consistent access patterns.

For our second query, we list the tables from the A90/A91 query to focus our efforts on those tables. For a non-SQL site, you could use CA Dataquery DQL to get this second set of information.

```
FROM SYSADM.SAMP_ACT_DB_A92 A92,
     SYSADM.SAMP_ACT_DB_A93 A93
WHERE ((A92.BASE = 1022) AND (A92.TNAME = 'REO') OR
       (A92.BASE = 1023) AND (A92.TNAME = 'REO') OR
       (A92.BASE = 4033) AND (A92.TNAME = 'MET'))
    AND ((A92.BASE = A93.BASE) AND
         (A92.ANAME = A93.ANAME) AND
         (A92.TNAME = A93.TNAME) AND
         (A92.KNAME = A93.KNAME) AND (A92.KNAME > '     ') AND
         (A92.ACTDATE = A93.ACTDATE))
    AND (( DATE ( '20' ||SUBSTR(A93.ACTDATE,1,2)||'-'
                     ||SUBSTR(A93.ACTDATE,3,2)||'-'
                     ||SUBSTR(A93.ACTDATE,5,2)) )
         > (CURRENT DATE - 30 DAYS))
ORDER BY A93.BASE, A93.ANAME, A93.TNAME, A93.KNAME, A93.ACTDATE;
```
The section of the WHERE clause highlighted in bold font, would be edited as needed to place the tables identified with the 3-character table name, TNAME, found by the first query into the second query. The database ID column (BASE), is also required if the same table occurs in multiple databases. The information is sorted so that we can review the recent activity for each table as a group.

The following chart was generated by the second query. The chart has been truncated so that it only includes the activity for the last seven days. We have also only listed the access by the Native Key for the tables.

Using the Accounting table information to answer questions on sequential performance, we focused the first query on tables that had a recent history of high sequential access, that is, sequential access greater than 10,000. Once a table was selected, we then used more detailed Accounting in the second query to gather information on the selected tables’ activity over the last 30 days.

The query returns all of the keys used for sequential processing. You must locate the Native Key for the table within the chart and review its sequential access.

For ease of use we reduced the chart to show only the Native Key of each of the selected tables. The chart provides several key pieces of information:

**PERCENT_SEQ**

Shows the percentage of sequential access by the Native Key on the table. If this percentage is relatively small, then a data row reorganization may have limited effect on overall performance. This column also provides a history of sequential access. We can use this information to determine if the sequential access is constant or if it can be associated to specific processing period such as end of month.
TOTAL_EFFICIENCY

Shows the average number of requests, both sequential and non-sequential, that are processed using the Native Key and the generated physical I/O. We can use this information to determine if the total access efficiency is constant or if it can be associated to a specific processing period, such as end of month. We can also compare the total efficiency to sequential efficiency to see if they are both affected by data row reorganizations.

SEQUENCIAL_EFFICIENCY

 Shows the average number of sequential requests by Native Key and the generated physical I/O. We can use this information to determine if the sequential access efficiency is constant or if it can be associated to a specific processing period, such as end of month. We can also review the sequential efficiency to see if data row placement is affecting the sequential efficiency.

A basic rule of thumb is that the native sequential efficiency is always at its best just after a data row reorganization. From that point, the sequential efficiency will decline as data rows are added and deleted and the sequential order is negatively affected. This decline in sequential processing typically continues until the data is reorganized.

In some cases, this decline is so slow that there is no real need to have regularly scheduled data row reorganizations. In these cases, it may be necessary to record a high-level mark for sequential efficiency after the data reorganization and then just periodically check the current efficiency against the high-level performance mark.

In other cases, the decline is very measurable and you need to adopt a strategy to schedule regular data row reorganizations to keep the sequential efficiency at a high-level.

There are a few examples where the sequential processing efficiency declines so rapidly after a data row reorganization that a regularly scheduled reorganization does not really help. In these very rare cases, alternatives such as alternate data row ordering, data extraction and sorting, and so forth may need to be considered.

Looking at the data in the chart, we can determine that while the REO table in DBID 1022 and the REO table in DBID 1023 have heavy sequential processing, the historical information does not show a significant reduction in total efficiency or a reduction in sequential efficiency. Overall, these tables look to be in good shape.
The one piece of information that we might want to look at before deciding on what to do, would be to determine when the data rows were last reorganized. If the data rows had been reorganized within the period we were reviewing and there was no significant change in efficiency, then from all the information we have, we could eliminate them from immediate consideration.

The MET table in DBID 4033, on the other hand is showing a consistent decline in sequential processing efficiency. There is also an associated decline in overall efficiency. The MET 4033 table should be high on the list of data row reorganization candidates.

A final point to consider is whether the degradation could have something to do with way the applications are accessing the table or possibly the area of the table that is being accessed by the sequential processing. Once we have a list of candidates to reorganize, it is always beneficial to ask questions about how the data is being processed by the applications. Having that information could help us to plan data reorganizations better.

The next section provides an alternative to the Accounting Facility for collecting a similar set of sequential access information.

**Using the Optional History Database**

In CA Datacom/DB r11.0, a new optional statistical collection facility known as the History Database was introduced for the z/OS user. The initial delivery of the History Database contained only one table known as the ADS table. This table is used to store access information about the database data areas accessed in the MUF. Future deliveries of CA Datacom/DB will deliver additional statistical collection tables as part of the History Database.

If the History Database is implemented and enabled, the History data table ADS is populated once a day by the MUF (Multi-User Facility) with a snapshot of user access activity (for that 24 hour period) by data area.

The enablement of data collection in the History Database is controlled by a MUF start-up option. The default time of the History snapshot is midnight each day the MUF is operational. You can alter this snapshot time by supplying an additional MUF start-up option.
History Database Rows and Columns

Each row of the History Database table ADS contains the following columns:

- DBID
- AREA_NAME
- DATE_OF_INFO
- DATA_USED_TOTAL
- IO_READ_TOTAL
- DATA_USED_NAT_NEXT
- IO_READ_NAT_NEXT
- DATE_LOADED
- DATE_OLREORG

Most of these columns are self explanatory. The History Database ADS table and its columns are described in the CA Datacom/DB Database and System Administration Guide. The implementation of the History Database is optional for each MUF.

At the selected snapshot time, the MUF populates one new or updated row per day into the History Database ADS table for each database data area that has been opened since the MUF was enabled. A row is also created or updated if the data area is closed in MUF. The close includes normal termination of the MUF using the EOJ process.

To use the History Database ADS table to find significant sequential access, you can create a simple SQL or CA Dataquery DQL query to locate data areas (and their tables) that have significant sequential access by the Native Key.

For example, you determine that you want to scan the History Database for the last seven days and locate any data areas that have had what you consider significant sequential access.

To qualify, a data area would be listed if it has at least 10,000 accesses by the Native Key in any one of the seven selected 24-hour periods. One history row equals one 24-hour period.

For qualifying areas:

- List the areas in order of highest number of sequential accesses to lowest
- List areas multiple times if they hit the criteria multiple days
The following SQL query provides this information. A similar CA Dataquery query could be constructed for non-SQL sites.

```
SELECT DATE_OF_INFO, DBID, AREA_NAME, DATA_USED_TOTAL, DATA_USED_NAT_NEXT, 
     (DATA_USED_NAT_NEXT / DATA_USED_TOTAL) AS PERCENT_SEQ 
FROM SYSADM.CADTCM_HISTORY_ADS 
WHERE 
    (DATE_OF_INFO > (CURRENT DATE - 7 DAYS))   AND 
    (DATA_USED_NAT_NEXT > 100000) 
ORDER BY 
    DATA_USED_NAT_NEXT DESC;
```

The output of the query is shown in the following chart. Note that in this system there are three data areas with heavy sequential access as defined by the query in this (MUF). Each of the data areas had multiple heavy use days.

<table>
<thead>
<tr>
<th>DATE_OF_INFO</th>
<th>DBID</th>
<th>AREA_NAME</th>
<th>DATA_USED_TOTAL</th>
<th>DATA_USED_NAT_NEXT</th>
<th>PERCENT_SEQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010-10-18</td>
<td>4033</td>
<td>GO3</td>
<td>137,221,436</td>
<td>125,729,769</td>
<td>92%</td>
</tr>
<tr>
<td>2010-10-17</td>
<td>4033</td>
<td>GO3</td>
<td>72,654,741</td>
<td>64,991,710</td>
<td>85%</td>
</tr>
<tr>
<td>2010-10-22</td>
<td>4033</td>
<td>GO3</td>
<td>48,541,065</td>
<td>45,494,197</td>
<td>94%</td>
</tr>
<tr>
<td>2010-10-17</td>
<td>1022</td>
<td>REO</td>
<td>3,487,984</td>
<td>3,131,002</td>
<td>90%</td>
</tr>
<tr>
<td>2010-10-17</td>
<td>1023</td>
<td>REO</td>
<td>3,394,042</td>
<td>3,131,002</td>
<td>92%</td>
</tr>
<tr>
<td>2010-10-18</td>
<td>1022</td>
<td>REO</td>
<td>1,727,893</td>
<td>1,565,501</td>
<td>91%</td>
</tr>
<tr>
<td>2010-10-19</td>
<td>1022</td>
<td>REO</td>
<td>1,669,725</td>
<td>1,565,501</td>
<td>94%</td>
</tr>
<tr>
<td>2010-10-22</td>
<td>1022</td>
<td>REO</td>
<td>1,592,806</td>
<td>1,565,501</td>
<td>98%</td>
</tr>
<tr>
<td>2010-10-22</td>
<td>1023</td>
<td>REO</td>
<td>1,592,806</td>
<td>1,565,501</td>
<td>95%</td>
</tr>
<tr>
<td>2010-10-21</td>
<td>4033</td>
<td>GO3</td>
<td>4,571,987</td>
<td>928,430</td>
<td>20%</td>
</tr>
<tr>
<td>2010-10-19</td>
<td>4033</td>
<td>GO3</td>
<td>257,148</td>
<td>257,148</td>
<td>100%</td>
</tr>
</tbody>
</table>

This is just an example of possible criteria a site can use to locate those data sets that have heavy access by the Native Key. The selection criteria used is dependent on the site. One recommendation would be to start with a query that locates 100 or more data areas and then using the descending list of sequential activities focus in on those areas that have the highest sequential access.
Break Down the Sequential Access Efficiency and Compare it to Overall Access Efficiency

Using the list of data areas with the highest sequential access, we now want to determine if that access is efficient. In particular we want to determine two things. First has the sequential access efficiency declined and second, is the total access efficiency showing similar declines.

Using the list of heavy sequential access data areas located in the first query, we now want to list the selected data areas’ history information for the last 30 days. This can be done by simply picking an area name from the list above and inserting it into a new query of the History Database.
Using the Optional Accounting Facility

```sql
SELECT
    H1.DATE_OF_INFO, H1.DBID, H1.AREA_NAME, H1.DATA_USED_TOTAL,
    H1.IO_READ_TOTAL, H1.DATA_USED_NAT_NEXT, H1.IO_READ_NAT_NEXT,
    H1.DATA_USED_NAT_NEXT / H1.DATA_USED_TOTAL AS PERCENT_SEQ,
    H1.DATA_USED_TOTAL /
    CASE WHEN H1.IO_READ_TOTAL = 0 THEN 1
         ELSE H1.IO_READ_TOTAL END AS TOT_EFFICIENCY,
    H1.DATA_USED_NAT_NEXT /
    CASE WHEN H1.IO_READ_NAT_NEXT = 0 THEN 1
         ELSE H1.IO_READ_NAT_NEXT END AS SEQ_EFFICIENCY,
    H1.DATE_LOADED, H1.DATE_OLREORG
FROM
    SYSADM.CADTCM_HISTORY_ADS H1
WHERE
    H1.DATE_OF_INFO > CURRENT_DATE - 30 DAYS AND
    H1.DATA_USED_NAT_NEXT > 0 AND
    ((H1.AREA_NAME = 'G03' AND H1.DBID = 4033) OR
     (H1.AREA_NAME = 'REO' AND H1.DBID = 1022) OR
     (H1.AREA_NAME = 'RE0' AND H1.DBID = 1023))
ORDER BY
    H1.DBID, H1.AREA_NAME, H1.DATE_OF_INFO;
```

<table>
<thead>
<tr>
<th>DATE_OF_INFO</th>
<th>DBID</th>
<th>AREA_NAME</th>
<th>DATA_USED_TOTAL</th>
<th>IO_READ_TOTAL</th>
<th>DATA_USED_NAT_NEXT</th>
<th>IO_READ_NAT_NEXT</th>
<th>PERCENT_SEQ</th>
<th>TOT_EFFICIENCY</th>
<th>SEQ_EFFICIENCY</th>
<th>DATE_LOADED</th>
<th>DATE_OLREORG</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010-11-09</td>
<td>1022</td>
<td>REO</td>
<td>3,487,975</td>
<td>146,940</td>
<td>3,113,002</td>
<td>94,742</td>
<td>95%</td>
<td>25</td>
<td>10/9/2010</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2010-11-13</td>
<td>1022</td>
<td>REO</td>
<td>1,706,099</td>
<td>71,809</td>
<td>1,505,501</td>
<td>68,600</td>
<td>95%</td>
<td>25</td>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2010-11-14</td>
<td>1022</td>
<td>REO</td>
<td>1,706,099</td>
<td>71,809</td>
<td>1,505,501</td>
<td>68,600</td>
<td>95%</td>
<td>25</td>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2010-11-15</td>
<td>1022</td>
<td>REO</td>
<td>3,520,151</td>
<td>143,559</td>
<td>3,113,002</td>
<td>92,120</td>
<td>95%</td>
<td>25</td>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2010-11-12</td>
<td>1022</td>
<td>REO</td>
<td>3,487,984</td>
<td>140,178</td>
<td>3,113,002</td>
<td>94,742</td>
<td>95%</td>
<td>25</td>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2010-11-18</td>
<td>1022</td>
<td>REO</td>
<td>1,727,873</td>
<td>68,701</td>
<td>1,505,501</td>
<td>68,602</td>
<td>95%</td>
<td>25</td>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2010-11-19</td>
<td>1022</td>
<td>REO</td>
<td>3,382,077</td>
<td>156,399</td>
<td>3,009,051</td>
<td>121,177</td>
<td>95%</td>
<td>25</td>
<td>14</td>
<td>10/9/2010</td>
<td></td>
</tr>
<tr>
<td>2010-11-13</td>
<td>1022</td>
<td>REO</td>
<td>1,706,162</td>
<td>71,748</td>
<td>1,505,501</td>
<td>11,265</td>
<td>92%</td>
<td>96</td>
<td>119</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2010-11-14</td>
<td>1022</td>
<td>REO</td>
<td>1,706,163</td>
<td>71,748</td>
<td>1,505,501</td>
<td>11,265</td>
<td>92%</td>
<td>96</td>
<td>119</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2010-11-15</td>
<td>1022</td>
<td>REO</td>
<td>1,706,163</td>
<td>71,748</td>
<td>1,505,501</td>
<td>11,265</td>
<td>92%</td>
<td>96</td>
<td>119</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2010-11-17</td>
<td>1022</td>
<td>REO</td>
<td>3,386,042</td>
<td>156,399</td>
<td>3,113,002</td>
<td>131,151</td>
<td>92%</td>
<td>93</td>
<td>135</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2010-11-18</td>
<td>1022</td>
<td>REO</td>
<td>1,992,806</td>
<td>71,199</td>
<td>1,505,501</td>
<td>12,332</td>
<td>93%</td>
<td>93</td>
<td>127</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2010-11-01</td>
<td>4033</td>
<td>REO</td>
<td>100,009,399</td>
<td>1,882,000</td>
<td>90,988,394</td>
<td>475,233</td>
<td>91%</td>
<td>72</td>
<td>19</td>
<td>11/2/2010</td>
<td></td>
</tr>
<tr>
<td>2010-11-02</td>
<td>4033</td>
<td>REO</td>
<td>12,912,900</td>
<td>62,579</td>
<td>1,006,588</td>
<td>6,778</td>
<td>14%</td>
<td>207</td>
<td>211</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2010-11-03</td>
<td>4033</td>
<td>REO</td>
<td>49,712,498</td>
<td>999,673</td>
<td>45,494,197</td>
<td>901,796</td>
<td>92%</td>
<td>72</td>
<td>141</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2010-11-04</td>
<td>4033</td>
<td>REO</td>
<td>149,230,340</td>
<td>1,885,153</td>
<td>134,442,981</td>
<td>1,092,648</td>
<td>91%</td>
<td>79</td>
<td>81</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2010-11-05</td>
<td>4033</td>
<td>REO</td>
<td>50,210,967</td>
<td>695,501</td>
<td>45,494,197</td>
<td>827,675</td>
<td>93%</td>
<td>72</td>
<td>55</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2010-11-08</td>
<td>4033</td>
<td>REO</td>
<td>50,983,796</td>
<td>598,644</td>
<td>46,177,079</td>
<td>258,759</td>
<td>92%</td>
<td>85</td>
<td>179</td>
<td>11/8/2010</td>
<td></td>
</tr>
<tr>
<td>2010-11-10</td>
<td>4033</td>
<td>REO</td>
<td>149,513,648</td>
<td>1,572,902</td>
<td>135,589,729</td>
<td>678,431</td>
<td>92%</td>
<td>76</td>
<td>103</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2010-11-11</td>
<td>4033</td>
<td>REO</td>
<td>72,034,743</td>
<td>1,231,034</td>
<td>64,991,720</td>
<td>578,365</td>
<td>89%</td>
<td>59</td>
<td>112</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2010-11-12</td>
<td>4033</td>
<td>REO</td>
<td>137,211,436</td>
<td>1,812,223</td>
<td>126,729,729</td>
<td>1,625,520</td>
<td>92%</td>
<td>76</td>
<td>78</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2010-11-14</td>
<td>4033</td>
<td>REO</td>
<td>257,148</td>
<td>12,159</td>
<td>257,148</td>
<td>6,875</td>
<td>19%</td>
<td>56</td>
<td>29</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2010-11-15</td>
<td>4033</td>
<td>REO</td>
<td>4,672,907</td>
<td>47,672</td>
<td>928,490</td>
<td>43,344</td>
<td>22%</td>
<td>96</td>
<td>21</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2010-11-15</td>
<td>4033</td>
<td>REO</td>
<td>98,072,125</td>
<td>2,935,004</td>
<td>90,988,394</td>
<td>2,801,481</td>
<td>93%</td>
<td>33</td>
<td>12</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
From the chart, we can see similar information to what we discussed in the section on Accounting Facility use. While the History Database is limited compared to the Accounting Facility because it only provides access information by the Native Key, it does include some new information that is not available in the Accounting Facility. The History Database ADS table provides two columns that tell the user when the data area was last loaded, DATE_LOADED, and last online reorganized using DBUTLTY OLREORG, DATE_OLRERORG.

Using the ADS Table Information to Answer Questions on Sequential Performance

We focused the first query on data areas that had a recent history of high sequential access greater than 10,000. Once a data area was selected, we then used the second query to gather information on the data areas’ activity over the last 30 days.

The chart provides several key pieces of information:

**PERCENT_SEQ**

Shows the percentage of sequential access by Native Key. If this percentage is relatively small, then a data row reorganization may have limited effect on overall performance. This column also provides a history of sequential access. We can use this information to determine if the sequential access is constant or if it may be associated to specific processing period such as end of month.

**TOT_EFFICIENCY**

Shows the requests the average number of requests (both sequential and non-sequential) that are processed and the generated physical I/O. We can use this information to determine if the total access efficiency is constant or if it may be associated to specific processing period such as end of month. We can also compare the total efficiency to sequential efficiency to see if they are both affected by data row reorganizations.

**SEQ_EFFICIENCY**

Shows the requests the average number of sequential requests that are processed and the generated physical I/O. We can use this information to determine if the sequential access efficiency is constant or if it may be associated to specific processing period such as end of month. We can also review the sequential efficiency to see if data row placement is affecting the sequential efficiency.
A basic rule of thumb is that the native sequential efficiency will always be at its best just after a data row reorganization. From that point, the sequential efficiency will decline as data rows are added and deleted and the sequential order is negatively affected. This decline in sequential processing typically continues until the data is reorganized.

In some cases, this decline is so slow that there is no real need to have regularly scheduled data row reorganizations. In these cases, it may be necessary to record a high-level mark for sequential efficiency after the data reorganization and then just periodically check the current efficiency against the high-level mark.

In other cases, the decline is very measurable and the user will need to adopt a strategy to schedule regular data row reorganizations to keep the sequential efficiency at a high-level.

There are a few examples, where the sequential processing efficiency declines so rapidly after a data row reorganization that a regularly scheduled reorganization does not really help. In these very rare cases, alternatives such as alternate data row ordering, data extraction and sorting, and so forth may need to be considered.

Looking at the data in the chart, we can determine that while data area REO of DBID 1022 and data area REO of DBID 1023 have heavy sequential processing, the historical information does not show a significant reduction in total efficiency or a significant reduction in sequential efficiency. Overall, these tables look to be in good shape.

In both cases, the data areas show a recent DATE_LOADED date. If we were to research this activity and found that this was a LOAD that was part of a data row reorganization, we would be able to quickly determine that these data areas do not benefit at this time from a data row reorganization. If additional research shows that the reorganization process does not improve efficiency for this data area, the user should consider removing the reorganization process (as it is not helping).

Unfortunately, the DATE_LOADED date can also reflect DBUTLTY activity where a data area was loaded from a physical backup as part of some recovery operation and not as part of a data row reorganization. In this scenario that data load would not have re-ordered the data rows into native sequence and would not have affected performance. If additional research shows that this physical load is occurring often, you may want to review why it is being created using a physical backup. Data load for recovery processes should be the exception and not the norm.

Data area MET of DBID 4033, on the other hand is showing a consistent decline in sequential processing efficiency. There is also an association between the DATE_LOADED dates and the most efficient access. We can see that on the day following the DBUTLTY LOAD sequential efficiency was at its highest. We also see a considerable decline in efficiency each day following the LOAD.
By analyzing this information, it becomes relatively certain that the MET data area was positively affected by the data reorganization. In addition, it seems that the sequential efficiency declines rapidly requiring weekly data row reorganizations. This would indicate a significant amount of random add and delete activity. This table would most likely continue to be on the candidate list for data row reorganizations.

One final point to consider is whether the degradation could have something to do with the way the applications are accessing the table or possibly the area of the table that is being accessed by the sequential processing. Once we have a list of candidates to reorganize, it is always beneficial to ask questions about how the data is being processed by the applications. Having that information could help us to plan data row reorganizations better.

If you have reduced the data areas found by the first query to a reasonable number, you can also use that query as a sub-select to qualify the areas for the second query. This query combines the two queries into one.

```
SELECT
  H1.DATE_OF_INFO, H1.DBID, H1.AREA_NAME, H1.DATA_USED_TOTAL,
  H1.IO_READ_TOTAL, H1.DATA_USED_NAT_NEXT, H1.IO_READ_NAT_NEXT,
  H1.DATA_USED_NAT_NEXT / H1.DATA_USED_TOTAL AS PERCENT_SEQ,
  H1.DATA_USED_TOTAL / CASE WHEN H1.IO_READ_TOTAL = 0 THEN 1
  ELSE H1.IO_READ_TOTAL END AS TOT_EFFICIENCY,
  H1.DATA_USED_NAT_NEXT / CASE WHEN H1.IO_READ_NAT_NEXT = 0 THEN 1
  ELSE H1.IO_READ_NAT_NEXT END AS SEQ_EFFICIENCY,
  H1.DATE_LOADED, H1.DATE_OLREORG
FROM
SYSDM.CADTCM_HISTORY_ADS H1
WHERE
  H1.DATE_OF_INFO > CURRENT DATE - 30 DAYS AND
  H1.DATA_USED_NAT_NEXT > 0 AND
  H1.AREA_NAME || DIGITS(H1.DBID) IN(
    SELECT DISTINCT H2.AREA_NAME || DIGITS(H2.DBID)
    FROM SYSDM.CADTCM_HISTORY_ADS H2
    WHERE H2.DATE_OF_INFO > CURRENT DATE - 7 DAYS
    AND H2.DATA_USED_NAT_NEXT > 100000)
ORDER BY
  H1.DBID, H1.AREA_NAME, H1.DATE_OF_INFO;
```
Comparing the Accounting Facility with the ADS Table of the History Database

There are a few differences that we should note when we compare the sample Accounting Facility tables to the information in the ADS table of the History Database.

The Accounting Facility provides:

- Much greater flexibility in the content as well as schedule for data collection,
- More granular information as well as additional performance and use statistics,
- Up to 100 different collection tables to be created.

The Accounting Facility requires:

- The user to define, build and implement the Accounting Database and its data tables,
- A small amount of CPU and I/O overhead as long as the Accounting Database has a table open for processing,
- User defined process to delete rows from the Accounting table that are no longer needed.

The History Database provides:

- A simple pre-defined and installed database for collecting access information on a daily (24 hour basis),
- A fixed set of columns,
- The date of the last DBUTLTY load (DATE_LOADED) of this data area
- The date of the last DBUTLTY online reorganization (DATE OLREORG).
Recognizing the Effect on the Environment

The History Database requires:

- Very minimal MUF start-up option changes to implement,
- Very minimal CPU and I/O overhead as it is only active for a few seconds each day when it produces the daily information snapshot,
- User-defined process to delete rows from the ADS table that are no longer needed.

**Important!** The ADS table of the History Database records its information at the data area level. If there are multiple tables in an area, they are grouped together as one set of data area statistics. Also, the ADS table only provides sequential access information based on the Native Key. All other activity is included in the total activity only.

Recognizing the Effect on the Environment

Until now, the Accounting Facility or the History Database and their daily access efficiency ratings were the best way to determine if a data area or table needed a data row reorganization. The only drawback to these statistics is that they only report on the data that is being accessed.

Both total efficiency and sequential efficiency can be greatly affected by:

- Availability of data buffers. Multi-User Facilities with large data buffer pools have a better chance at fulfilling secondary requests from an existing buffer, therefore increasing efficiency ratings. This is especially true where sequential processing is occurring and the group of rows being accessed is spread across a small set of data blocks. Tuning the number of buffers provided in the MUF for processing can improve the efficiency of the access.

- For GSETL/GETIT processing the number of buffers allocated by the User Requirements Table (URT) can improve the possibility of a secondary request finding its data already in a buffer. Tuning the number of buffers provided in the URT for GSETL/GETIT processing can improve the efficiency of the access.
Recognizing the Effect of Larger Buffer Pools

In the following chart, we see the effect of having more buffers available for the sequential process. Remember that the more buffers that are available, the higher the possibility that a block accessed in the past can still be in a buffer in memory. Any block read from memory reduces the need for physical I/Os and will improve performance.

<table>
<thead>
<tr>
<th>GSETL Read Program</th>
<th>Buffers</th>
<th>MRDF</th>
<th>Data Logical</th>
<th>Data PhysI0</th>
<th>Logical Change</th>
<th>Percent Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unorganized</td>
<td>Read by native</td>
<td>4</td>
<td>No</td>
<td>31,953</td>
<td>3,509</td>
<td></td>
</tr>
<tr>
<td>Unorganized</td>
<td>Read by native</td>
<td>128</td>
<td>No</td>
<td>31,953</td>
<td>128</td>
<td>0%</td>
</tr>
<tr>
<td>Reorganized</td>
<td>Read by native</td>
<td>4</td>
<td>No</td>
<td>31,953</td>
<td>448</td>
<td>0%</td>
</tr>
<tr>
<td>Reorganized</td>
<td>Read by native</td>
<td>128</td>
<td>No</td>
<td>31,953</td>
<td>22</td>
<td>0%</td>
</tr>
<tr>
<td>Unorganized</td>
<td>Read by non-native</td>
<td>4</td>
<td>No</td>
<td>31,953</td>
<td>25,373</td>
<td></td>
</tr>
<tr>
<td>Unorganized</td>
<td>Read by non-native</td>
<td>128</td>
<td>No</td>
<td>31,953</td>
<td>529</td>
<td>0%</td>
</tr>
<tr>
<td>Reorganized</td>
<td>Read by non-native</td>
<td>4</td>
<td>No</td>
<td>31,953</td>
<td>30,783</td>
<td>0%</td>
</tr>
<tr>
<td>Reorganized</td>
<td>Read by non-native</td>
<td>128</td>
<td>No</td>
<td>31,953</td>
<td>94</td>
<td>0%</td>
</tr>
</tbody>
</table>

While the batch job using the Native Key was significantly improved by doing the offline reorganization when only 4 buffers were in use, we can see that changing buffers from 4 to 128 actually improved performance dramatically, and the offline reorganization could have been eliminated.

We also see that the non-native sequence job also improved dramatically by having the additional buffers. While this example is using the GSETL/GETIT commands and the SEQBUFs data buffer pool, similar savings can be achieved by jobs or transactions doing sequential processing using the general MUF data buffer pools.

Recognizing the Effect of Larger Buffer Pools

Recognizing that a program accessing CA Datacom/DB may acquire and use all the data buffers in the MUF, the possibility of that happening in a busy system serving many users is minimal. Typically, in these MUFs, individual processes would have only a portion of the available buffers available for their use. The larger the buffer pool compared with the frequency of other applications making requests will cause buffer availability to vary. However, processes running where lots of buffers are available will not be as negatively impacted by poor data row order.

To make our sample charts accurate, we have chosen to limit the buffers available to each process.
In the preceding charts, the number of data blocks in use was relatively small. For larger samples, the change from 4 to 128 buffers would have definitely helped, but may not have been this significant. It just depends on the number of blocks that are in memory while the program is running and the frequency in which one of those in-memory blocks has the next row that is needed.

Before scheduling reorganization, the user should consider whether there is enough activity to warrant increasing the SEQBUFS buffer pool for GSETL/GETIT jobs or adding additional general use buffers using the MUF DATAPool parameter or FLEXPOOL parameter. The FLEXPOOL buffer pool can be dynamically increased during high sequential use periods and then decreased when no longer needed.

See the CA Datacom/DB Systems and Database Administration Guide for more information on the MUF DATAPool and FLEXPOOL parameters and the SEQBUFS parameter in User Requirements Tables.

**Considering the Use of MRDF**

For small to medium data tables that have high access rates, you can also consider placing the data table in the Memory Resident Data Facility (MRDF). You define memory resident data or index areas to the MUF with the startup options COVERED and VIRTUAL. For information, see the CA Datacom/DB Database and System Administration Guide.

The MRDF allows data blocks to be saved in MRDF memory when they are initially read from DASD. Later, when the data block is needed again and it is not resident in a buffer, the data block is read from the MRDF memory and the physical I/O is avoided.

We have placed the example data area in MRDF covered storage. We accessed the data table using a simple locate index command which touched each data block and loaded them into a buffer and then MRDF storage.
Recognizing the Effect of the Application Access

Now repeating the same batch jobs we see that the physical cost of the jobs has now dropped to zero physical I/Os and there is no need for either more buffers or data reorganization.

<table>
<thead>
<tr>
<th>MRDF COVERED</th>
<th>GSETL Read Program</th>
<th>Buffers</th>
<th>MRDF</th>
<th>Data Logical</th>
<th>Data Physical I/O</th>
<th>Logical Change</th>
<th>Percent Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unorganized</td>
<td>Read by native</td>
<td>4</td>
<td>No</td>
<td>31,953</td>
<td>-</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>Unorganized</td>
<td>Read by native</td>
<td>128</td>
<td>No</td>
<td>31,953</td>
<td>-</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>Reorganized</td>
<td>Read by native</td>
<td>4</td>
<td>No</td>
<td>31,953</td>
<td>-</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>Reorganized</td>
<td>Read by native</td>
<td>128</td>
<td>No</td>
<td>31,953</td>
<td>-</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>Unorganized</td>
<td>Read by non-native</td>
<td>4</td>
<td>No</td>
<td>31,953</td>
<td>-</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>Unorganized</td>
<td>Read by non-native</td>
<td>128</td>
<td>No</td>
<td>31,953</td>
<td>-</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>Reorganized</td>
<td>Read by non-native</td>
<td>4</td>
<td>No</td>
<td>31,953</td>
<td>-</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>Reorganized</td>
<td>Read by non-native</td>
<td>128</td>
<td>No</td>
<td>31,953</td>
<td>-</td>
<td>0%</td>
<td>0%</td>
</tr>
</tbody>
</table>

Both of these recommendations could improve the access efficiency and if the improvements are significant, can delay or remove the need for the reorganization. Changes to the MRDF, the DATAPOOL parameter, and the FLEXPOOL parameter should be monitored for total system performance as well as being tracked in the history report for overall efficiency.

Recognizing the Effect of the Application Access

As we stated, the daily access efficiency ratings are the best way to determine if a data area needs to have data row reorganization. The only drawback to these statistics is that they only report on the data rows that are being accessed by the applications.

Certain applications may improve efficiency ratings by doing very heavy sequential processing requests where data rows are read from beginning to end. Other applications could lower efficiency rating by doing what we call skip sequential processing where a point in the index is located and a row or two is read sequentially, then the application skips to the next place in the index to repeat the process.
Another application function that could affect efficiency would be sequential access of recently added rows. Following a normal data pattern a user has rows added over a period of a few minutes or hours, then a series of sequential application requests are focused on these data rows.

Since RANDOM reuse is the most typical DSOP choice that users select for recovering deleted space, the added rows tend to be spread out across the data area. In the case where only 10% of the data is spread out this way, but that is the only data being processed by the sequential applications, the sequential efficiency ratings may paint a picture that the whole data area is out of sequence, when only a portion of the data is actually affecting the Native Key processing.

Chapter Summary

We have two tools available, the Accounting Facility and the History Database, that provide the best way to measure sequential access and sequential efficiency.

These tools provide relatively good measurement of data accesses that were done and the efficiency of those accesses. As we have noted, there can be some cases where the data accesses may focus on only a portion of a table or data area and could give a misleading impression about the complete table or data areas sequential organization.

Users can augment the data collected by these two tools with job execution time statistics for their largest or most critical sequential batch jobs. Correlating the job execution times with the sequential efficiency ratings could help to determine when a data row reorganization is required.
There is a new DBUTLTY function that reports on a data table in Native Key sequence and provides a native sequential access efficiency rating for the table. The DBUTLTY function is called the Native Efficiency Report (NER). It is generated by executing the DBUTLTY REPORT function with the TYPE=DATANE parameter.

The DBUTLTY REPORT TYPE=DATANE function is delivered in CA Datacom/DB Version 14.0 and can be added to CA Datacom/DB Version 12.0 by applying a set of PTFs. For information on the PTFs, see Requirements (see page 9).

Native Efficiency Report

The main function of the Native Efficiency Report (NER) is to build a simple report that shows the sequential access efficiency for a given data table. The NER requires the MUF to be up and the selected data table to be available for processing. The NER uses highly efficient read-only index commands to limit the effect on other processing in the MUF. That said, it should be understood that there is some overhead, and the NER should only be executed on an as needed basis.

The NER processes a selected KEYNAME for the selected data table using a series of LOCxx index commands. You should select the Native Sequence key name for the NER. The only reason to select a non-native key name would be to determine if the way data rows are being processed, both added and deleted, would indicate that a different key might be a better choice for the Native Key. Being able to compare the native efficiency on various keys of a table using the utility is a side benefit, but usually not a primary concern.
The NER process is based on a very simple metric. Data row organization efficiency can be determined by recording how many times a data block change occurs during a sequential pass (ascending) of the index values for the selected key name. In this case, we are defining a data block change as the movement that occurs when the next data row in the selected index sequence is on a data block which is not the same data block that the current data row is on. There is no value or weighting assigned to whether we are changing from block 7 to block 8 or from block 7 to block 77.

Since the program allows other users to be active while the report is being compiled, the results may be slightly affected by ongoing processing, the add and delete processing. The program reads every value for the selected index, the KEYNAME, in order to compile its report. It is for these reasons that we recommend that the report only be executed at times where the MUF is not under a heavy workload and the data table being accessed is not experiencing heavy maintenance activity.

**Native Efficiency Report and Single-Table Data Areas (Non-Compressed)**

The NER utility provides the most concise sequential efficiency information for tables that are not compressed and reside in single-table data areas. For these types of tables, the utility can easily determine the following:

**Maximum Rows per Block (MRB)**

The maximum number of data rows that can be stored on one data block.

**Perfect Block Count (PBC)**

The number of blocks, if filled to the MRB, that would be needed to house a selected number of data rows.

The consistency of these numbers makes the calculation of the detail level lines generated in the NER very accurate.

For information on the DBUTLTY control statements to generate the NER, see [Implementing and Using DBUTLTY REPORT=DATANE](#) (see page 139).
Native Efficiency Report and Single-Table Areas (Compressed)

The NER utility depends on its ability to calculate the MRB and PBC in order to generate accurate detail level lines found in the NER.

If the amount of compression found in data rows causes the actual number of data rows being stored on data blocks rows to vary widely and differ from the calculated MRB and PBC, the information in the detail rows of the NER is not as concise as needed and limits its ability to provide significant insight to the sequential efficiency of the data blocks represented by these detail lines.

If the amount of compression is reasonably consistent and the calculated value for MRB is close to what each of the data blocks actually has, then the detail lines will be a reasonably effective measurement of sequential access efficiency.

The NER’s FULL TABLE MRB efficiency line presents an assessment of the table’s actual efficiency compared to the “ideal” efficiency. It is calculated by comparing the actual data order and data row population in the blocks to the projected ideal scenario where each block has the maximum rows per block and those rows are in order.

The NER’s FULL TABLE AMRB (Adjusted Maximum Rows per Block) efficiency line presents an adjusted assessment of the table’s efficiency since it is calculated on actual blocks with data divided by the number of block changes found while reading the entire table. In this adjusted efficiency, the efficiency is solely based on the order of the rows and no penalty in efficiency is deducted for partially full data blocks.

Both FULL TABLE MRB and FULL TABLE AMRB are discussed in detail in Implementing and Using DBUTLTY REPORT=DATANE (see page 139).

You must be careful when comparing two separate executions of the NER on the same compressed table. If some activity, such as an OLREORG has improved the density of sequential rows in the blocks and the MRB has improved, then the NER will report sequential efficiency based the improved MRB values.

Since compressed tables will always be changing, you must remember to collect both the MRB values as well as the sequential efficiencies when you are preserving data for historical comparisons.
How to Compare Efficiency Ratings if the MRB Changes

The FULL TABLE MRB and FULL TABLE AMRB efficiency ratings are covered in the discussion of the NER in Implementing and Using DBUTLTY REPORT=DATANE (see page 139). These percentages are based on calculations using the calculated MRB for detail lines and the AMRB for the FULL TABLE lines.

If a change in the maximum rows per block calculations occurs between two NERs and you want to compare the sequential efficiency of the table in the two reports, you need to adjust the efficiency percentages for the differences in the maximum rows per block values.

Simply put, determine how to adjust the lines from the first report to compare to the lines from the second report. Since this is a manual calculation, we recommend that you focus on the FULL TABLE MRB and AMRB efficiency ratings at the bottom of the NER.

Native Efficiency Report and Multi-Table Areas

The NER was not intended for use with a multi-table area. While the report will function to the best of its ability, the variation in row sizes and row counts between the tables will greatly affect its ability to report on efficiency.

For these reasons, we do not recommend using the NER for multi-table areas.

Sequential Access and the Data Block

The whole basis of establishing the efficiency of sequential processing is that, if a set of data rows are read in sequence and those data rows reside on a few data blocks, the amount of physical processing, the physical I/O, needed to support the retrieval of that set of rows will be small. The more data blocks that are involved in the set of data rows, the higher the physical processing requirements.

Typically, the lower the physical processing to accomplish a task also results in a lower CPU consumption and lower elapsed times.

Following this rule, the more rows that can be processed from a single data block the more efficient the process.
Sequential Access Efficiency by the Native Sequence Key

In the case of a perfectly ordered table, such as just after an offline reorganization, the number of block changes when processing all of the data rows by Native Key would equal the number of data blocks with active rows in them minus 1. Or, put another way, if all data rows were in native sequence order in the data blocks and there were 5 data blocks in use, 1 thru 5, it would be expected that the sequential process would only see the block number change 4 times. In this case, we would say that the data table has 100% sequential access efficiency. The following chart shows an example of this proposed access.

In this example, we are taking the number of blocks with data minus 1 to get the perfect block change score and dividing that number by the actual number of block changes that occurred to get the sequential efficiency. Here we have 5 minus 1 divided by 4 which equals 100%.

In the next example, the data rows in the 5 data blocks recorded 12 block number changes during the process, that would mean that, on average, each data block was accessed three times in order to process the data rows in sequence.

In this example, we would say that the data area had a 33% sequential access efficiency, because 5 data blocks minus 1 divided by 12 block changes equals 33%.

The lower the % of sequential access efficiency, the more likely that sequential access of the table would take more I/Os and therefore take more CPU to complete the sequential access.
Buffered Sequential Access Efficiency

While the sequential efficiency is a valid indicator of projected performance, we also know that most processes, that is, tasks, that are accessing data in MUF will have a number of MUF data buffers associated with the task. The number of buffers available to a task will vary according to the number of other tasks accessing the data blocks at the same time.

Knowing this, the NER provides additional efficiency percentages that are based on the assumption that a set number of data buffers will be available to the task. By utilizing the theory of a set number of buffers being available, the NER does not count a block change if the new block number that is being accessed would have been in a previous buffer associated with this task.

For example, in the following chart, we show how the efficiency would improve with just two buffers, the current data buffer and the previous data buffer. Just like with the block change count, we do not count the first buffer loaded when the first block is read. The buffers, A and B, are being managed by an LRU (least recently used) algorithm.

<table>
<thead>
<tr>
<th>Data row</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Block</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Block Change</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Buffer Changes</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
</tr>
</tbody>
</table>

In the chart, this table has a basic sequential access efficiency of 33%, but the efficiency could be improved to 50% with just two buffers.
As we see in the following second buffer example, having just 4 buffers available improved the sequential efficiency back to 100%. We realize that most user data tables are going to include hundreds or thousands of data blocks with millions of rows and they cannot all be kept in a buffer. However, even with a reasonable number of buffers, the sequential performance will improve for these larger data block amounts.

<table>
<thead>
<tr>
<th>Buffer A</th>
<th>Buffer B</th>
<th>Buffer C</th>
<th>Buffer D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 1 1 1</td>
<td>2 2 2 2 2</td>
<td>3 3 3 3 3</td>
<td>4 4 4 4 4</td>
</tr>
</tbody>
</table>

The NER provides efficiency ratings for a basic setting of no buffers and for settings with buffers of 2, 4, 8, 12, 16, and so forth. When reading the report there will always be a figure for basic efficiency. There will only be a percentage listed under a specific buffer number if that amount of buffers would improve the efficiency percentage.

Applying the knowledge gained by analyzing both the Accounting Facility or History Database and the NER, you can decide if data row reorganization is necessary.

If an offline data row reorganization is done, the basic efficiency should return to 100%. Also, the Accounting Facility or History Database should show a significant improvement in the total efficiency and sequential efficiency metrics.

If an online reorganization is done, the basic efficiency should improve as well as the buffer efficiency ratings. Also, the Accounting Facility or History Database should show an improvement in the total efficiency and sequential efficiency metrics.

The key to the success of data reorganization is that it improves the data sequence enough to improve sequential access to the data rows. A goal of having the basic efficiency always at 100% is not the driving metric and should not be the only goal for this effort. The goal of the online reorganization is to increase the basic efficiency to be high enough to support the batch sequential processing performance requirements without requiring an offline organization. If the window for batch processing is shrinking, perhaps this goal for the NER basic percent moves up a couple percentage points to support that shrinking window. Maintaining a basic efficiency at or near 100% for a given table may not be achievable due to 24/7 processing. Yet, using the various reorganization techniques available, you could still meet the batch sequential processing needs.
By following a process of using the Accounting Facility or History Database to locate possible reorganization candidates and using the NER to validate those findings as well as documenting the improvement after the reorganization effort, you can take a proactive approach to determining what data tables or areas need to be reorganized and when.

Sample Native Efficiency Reports

The follow two sample Native Efficiency Reports are for a table that is not organized very well by the Native Key. We are calling this table’s data rows “disorganized”.

The low MRB efficiency combined with the low efficiency rates with buffers indicates that it would have poor sequential processing performance.

As we discussed previously, if this table was designated as COVERED, it would mitigate the poor sequential data row placement. If not COVERED, having a large data buffer pool could also help processing efficiency.

The first and second pages of the NER provide information on the table being processed.
The third page of this sample NER provides the detailed information on each of the reference groups within the table and their efficiency as well as the full table efficiency. The FULL TABLE has a 13% MRB efficiency and a 13% AMRB efficiency. The efficiencies do not get much better with buffering, so the table is in very poor shape for sequential access.

<table>
<thead>
<tr>
<th>GROUP</th>
<th>ROWS</th>
<th>BLOCKS</th>
<th>MAX ROW/BLK EFFICIENCY RATES WITH BUFFERS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3,600</td>
<td>365</td>
<td>26 1014</td>
</tr>
<tr>
<td>2</td>
<td>3,600</td>
<td>387</td>
<td>30 1014</td>
</tr>
<tr>
<td>3</td>
<td>3,600</td>
<td>448</td>
<td>36 1014</td>
</tr>
<tr>
<td>4</td>
<td>3,600</td>
<td>442</td>
<td>36 1014</td>
</tr>
<tr>
<td>5</td>
<td>3,600</td>
<td>711</td>
<td>16 1014</td>
</tr>
<tr>
<td>6</td>
<td>3,600</td>
<td>602</td>
<td>24 1014</td>
</tr>
<tr>
<td>7</td>
<td>3,600</td>
<td>545</td>
<td>28 1014</td>
</tr>
<tr>
<td>8</td>
<td>3,600</td>
<td>642</td>
<td>34 1014</td>
</tr>
<tr>
<td>9</td>
<td>3,150</td>
<td>557</td>
<td>33 1014</td>
</tr>
</tbody>
</table>

The following shows the NER for the same table after it has been reorganized using the standard BACKUP SEQ=NATIVE and LOAD process. The first and second pages have been skipped, as they are the same as shown in the previous sample.
Notice that the table is now 100% efficient because every block is only passed once when reading the table from beginning to end using the Native Key.

<table>
<thead>
<tr>
<th>GROUP</th>
<th>ROWS</th>
<th>BLOCKS</th>
<th>MAX</th>
<th>AVG</th>
<th>CHANGES</th>
<th>BLOCK EFF</th>
<th>2%</th>
<th>4%</th>
<th>8%</th>
<th>12%</th>
<th>20%</th>
<th>28%</th>
<th>32%</th>
<th>48%</th>
<th>64%</th>
<th>96%</th>
<th>128%</th>
<th>256%</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ</td>
<td>3,600</td>
<td>100</td>
<td>36</td>
<td>36</td>
<td>100</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>READ</td>
<td>3,600</td>
<td>100</td>
<td>36</td>
<td>36</td>
<td>100</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>READ</td>
<td>3,600</td>
<td>100</td>
<td>36</td>
<td>36</td>
<td>100</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>READ</td>
<td>3,600</td>
<td>100</td>
<td>36</td>
<td>36</td>
<td>100</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>READ</td>
<td>3,600</td>
<td>100</td>
<td>36</td>
<td>36</td>
<td>100</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>READ</td>
<td>3,600</td>
<td>100</td>
<td>36</td>
<td>36</td>
<td>100</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>READ</td>
<td>3,600</td>
<td>100</td>
<td>36</td>
<td>36</td>
<td>100</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>READ</td>
<td>3,150</td>
<td>88</td>
<td>36</td>
<td>18</td>
<td>36</td>
<td>88</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>READ</td>
<td>3,150</td>
<td>88</td>
<td>36</td>
<td>18</td>
<td>36</td>
<td>88</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

If you compare the two reports, you see that the actual number of data blocks in-use is the same for both the disorganized table and for the just reorganized table. This indicates that random space reuse is working well and that there is very little wasted space per data block in the disorganized table.

For random retrieval processing, the table before and after reorganization would offer relatively the same performance levels. Only when sequential processing by the Native Key is significant would there be any concerns about data organization.
The following report shows the result of taking the previous reorganized table and running a program process that adds a new set of rows, deletes some of those rows, then adds more rows.

<table>
<thead>
<tr>
<th>GROUP</th>
<th>ROWS</th>
<th>BLOCKS</th>
<th>MAX</th>
<th>MIN</th>
<th>AVG</th>
<th>BLOCK EFF</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
<th>256</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3,600</td>
<td>100</td>
<td>36</td>
<td>36</td>
<td>36</td>
<td>100</td>
<td>100 100</td>
<td>100 100</td>
<td>100 100</td>
<td>100 100</td>
<td>100 100</td>
<td>100 100</td>
<td>100 100</td>
<td>100 100</td>
</tr>
<tr>
<td>2</td>
<td>3,600</td>
<td>100</td>
<td>36</td>
<td>36</td>
<td>36</td>
<td>100</td>
<td>100 100</td>
<td>100 100</td>
<td>100 100</td>
<td>100 100</td>
<td>100 100</td>
<td>100 100</td>
<td>100 100</td>
<td>100 100</td>
</tr>
<tr>
<td>3</td>
<td>3,600</td>
<td>100</td>
<td>36</td>
<td>36</td>
<td>36</td>
<td>100</td>
<td>100 100</td>
<td>100 100</td>
<td>100 100</td>
<td>100 100</td>
<td>100 100</td>
<td>100 100</td>
<td>100 100</td>
<td>100 100</td>
</tr>
<tr>
<td>4</td>
<td>3,600</td>
<td>100</td>
<td>36</td>
<td>36</td>
<td>36</td>
<td>100</td>
<td>100 100</td>
<td>100 100</td>
<td>100 100</td>
<td>100 100</td>
<td>100 100</td>
<td>100 100</td>
<td>100 100</td>
<td>100 100</td>
</tr>
<tr>
<td>5</td>
<td>3,600</td>
<td>100</td>
<td>36</td>
<td>36</td>
<td>36</td>
<td>100</td>
<td>100 100</td>
<td>100 100</td>
<td>100 100</td>
<td>100 100</td>
<td>100 100</td>
<td>100 100</td>
<td>100 100</td>
<td>100 100</td>
</tr>
<tr>
<td>6</td>
<td>3,600</td>
<td>100</td>
<td>36</td>
<td>36</td>
<td>36</td>
<td>100</td>
<td>100 100</td>
<td>100 100</td>
<td>100 100</td>
<td>100 100</td>
<td>100 100</td>
<td>100 100</td>
<td>100 100</td>
<td>100 100</td>
</tr>
<tr>
<td>7</td>
<td>3,600</td>
<td>100</td>
<td>36</td>
<td>36</td>
<td>36</td>
<td>100</td>
<td>100 100</td>
<td>100 100</td>
<td>100 100</td>
<td>100 100</td>
<td>100 100</td>
<td>100 100</td>
<td>100 100</td>
<td>100 100</td>
</tr>
<tr>
<td>8</td>
<td>3,600</td>
<td>306</td>
<td>36</td>
<td>1</td>
<td>12</td>
<td>386</td>
<td>26 28 33</td>
<td>17 18 19 21 51</td>
<td>15 16 58</td>
<td>17 23 24 80</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>3,600</td>
<td>297</td>
<td>36</td>
<td>1</td>
<td>12</td>
<td>620</td>
<td>16 17 18 19 21 51</td>
<td>15 16 58</td>
<td>17 23 24 80</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>3,600</td>
<td>279</td>
<td>35</td>
<td>1</td>
<td>13</td>
<td>739</td>
<td>14 15 16 58</td>
<td>17 23 24 80</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>1,275</td>
<td>154</td>
<td>26</td>
<td>1</td>
<td>8</td>
<td>173</td>
<td>28 23 24 80</td>
<td>17 23 24 80</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In this report you can see that the good organization remains through the first part of the table from the previous reorganization, while the new data at the end shows some data row disorganization.

As data is added and deleted, the effect on the data row organization will be seen as a lowering of the efficiency percentages. However, the effect of the data rows changes will vary greatly depending on the pattern of the adds and deletes and the DSOP reuse options.

Chapter Summary

The new Native Efficiency Report (NER) provides the additional information needed to allow you to make informed decisions about which database tables need to be reorganized and the effect of the selected reorganization process.

While the NER can be used standalone to locate database tables that may need reorganization, we recommend a process that includes both the sequential usage information from the Accounting Database and History Database combined with the sequential efficiency ratings from the NER.

This combined process insures that you are focusing your reorganization efforts on tables that have significant native sequential processing that would benefit from the data row reorganization.
Chapter 4: DBUTLTY Offline Reorganization Techniques

In CA Datacom/DB, we provide utilities to reorganize data tables into Native Key order. Since the very early releases, all database utility functions have been centralized under the main database utility program known as DBUTLTY.

DBUTLTY is organized in such a way that each of the DBUTLTY functions ties into a specific sub-program. This allows the CA Datacom/DB development team to add new functions to DBUTLTY without making significant modification to the existing set of utility functions.

For this section, we focus on offline reorganization, which is made up of three basic parts:

- The backup in native sequence order
- The load of the native sequence backup
- The updating of the index

Native Sequence Backup

The recommended method to create the native sequence backup is to use the DBUTLTY BACKUP or EXTRACT functions with the SEQ=NATIVE parameter specified. The SEQ=NATIVE parameter triggers two important things to happen. First, it insures that the database about to be backed up is locked against other update access. Second, it instructs the utility to retrieve the data rows and place them on the output data set using the Native Key. This creates a backup copy with the data rows ordered in ascending sequence by the Native Key.

The DBUTLTY BACKUP function is preferred over the EXTRACT format since it allows the utility to build an output data set that includes the various CA Datacom/DB control information. The BACKUP also allows multiple tables in one area or multiple areas in one database to be processed together, thereby improving your efficiency. The DBUTLTY EXTRACT format creates an output data set with row images for one table only. If EXTRACT is used, every table must be processed separately.
Maximizing Performance when Taking a Native Sequence Backup

The DBUTLTY program uses buffers when doing a native sequence backup. Since the native sequence index is being read to determine the order of the rows to be retrieved and copied to the output dataset, both IXX and DXX buffers will be used in the process. Similarly, data buffers will be used to retrieve the data blocks containing the data rows.

While having a reasonable number of IXX and DXX buffers will improve performance, it is the number of data buffers available that will have most effect on performance.

The DBMSTLST module provides DBUTLTY with the buffer allocations for IXX, DXX, and DATA buffers. The default DBMSTLST provided when you install CA Datacom/DB uses a very small default number of buffers. We recommend that you significantly increase the IXX, DXX, and DATA buffers specified in DBMSTLST to improve Native sequence backup performance. As a standard override, CA Datacom/DB in z/OS environments overrides the data buffer setting to 128, if the current DBMSTLST setting is less than 128.

As an additional override, you can specify the SEQBUFS=nnn parameter on native sequence backups. This parameter overrides the DBMSTLST data buffer setting.

One drawback in specifying larger buffer counts in DBMSTLST is that the region size, the 31-bit memory, needs to be larger for DBUTLTY executions. For z/OS sites, this is typically not a problem unless very large buffer pools are specified. For z/VSE sites, there is a much smaller limit on available 31-bit storage in the DBUTLTY regions. The z/VSE sites are typically much more conservative in DBMSTLST buffer settings.

One option is to have two DBMSTLST modules. The first one with reasonable buffer pools is stored in the normal CA Datacom/DB library concatenation and is used for most DBUTLTY executions. The second one has much larger buffer allocations and resides in a non-standard library. Only when a large table, area, or database is being processed for a native backup and a large region is available, should you include the library with the “large DBMSTLST” in the front of the standard library concatenation.

Loading the Native Sequence Backup

Once you have a backup that is in Native Key sequence, use the DBUTLTY LOAD function to load the backup data set back into the database. While the LOAD function rebuilds and reorganizes the indexes as needed, it always loads the data rows into the data blocks in the same order as presented on the backup data set. The LOAD function does not attempt to do anything to change the data row order.
Updating and Rebuilding the Index

As noted, the index entries for the reorganized data must be updated to reflect the new data row content and placement. This is done automatically when using the DBUTLTY LOAD process. In certain cases, there may be a requirement to initialize, that is, empty the index area before starting the LOAD function. Each of the following cases show the required index update processes.

Defragmenting the Index

In cases where the DBUTLTY LOAD process has been limited to just part of the database or where the DBUTLTY REORG (parallel reorganization) function is used, we recommend that you consider running a DBUTLTY DEFRAG function to defragment and compress the index after the LOAD or REORG function has completed.

The defragmentation process can run while the database is open for user access. Each of the following cases state when the index defragmentation is recommended.

Matching the Backup and the Load

For each of the various backup formats that are created, there is a matching load process that must be followed. The following sections highlight the most common forms of database backups and loads to perform data reorganization.

In the following discussions, we focus on the key DBUTLTY parameters that are used to control reorganization. We will not discuss the majority of the parameters that are available for each of the DBUTLTY functions.

To learn more about the parameters for each of the DBUTLTY functions, see the CA Datacom/DB
DBUTLTY Guide.
Reorganization by DBID

For CA Datacom/DB Version 12.0 users, we recommend use of the DBID level backup and load to perform a full database reorganization. The process includes every data table and data area in the database. It provides the fastest and surest way to reorganize the full database. The following is the sample process.

1. Close the database to user processing. This includes the following:
   a. Close the user URTs that are open for this database in the MUF.
   b. Close the database to MUF using the COMM CLOSE command.
   c. Change the database’s ACCESS status from WRITE to UTLTY or OFF.

2. Back up the database.
   Once the database has been closed to other processing, submit a DBUTLTY BACKUP function specifying SEQ=NATIVE and DBID=nnnn. Do not include the AREA=aaa parameter.
   When DBUTLTY sees that no AREA is specified, it knows this is a full database backup and uses the CXX information to determine all of the data areas associated with this DBID. It then backs up each of those data areas, and the tables inside them, to the single output data set using the Native Keys. There will only be one output data set, but the DBUTLTY will include header records in the output which will allow the LOAD process to know to which data area the backup information belongs.

3. Initialize the Index (IXX).
   The database level processing requires you to initialize the index prior to starting the LOAD process. Initializing the index is the quickest way to clear the existing index entries and makes the index ready to receive the new data. An index initialization is done by submitting the DBUTLTY INIT function specifying the AREA=IXX and DBID=nnnn parameters.

4. Load the backup with the SORT option.
   The database level processing requires you to use the SORT processing during the loading of the database level backup. With the SORT option, all index entries are built and written to sort temporary files while the data rows are written from the backup data set to the data areas. Once all rows are written, the sort function is invoked to sort the index entries in the best possible sequence before writing them into the IXX data set.
The full DBID reorganization provides multiple benefits to you that include the following:

- Simplicity and safety since you do not need to list every data area and data table
- 100% sequential access efficiency as documented in the preceding example
- Data blocks that are tightly packed that provide the lowest blocks in-use percentage
- A resetting of the Unique Record Identifier (URI) value for each row, see the discussion in Parallel Reorganization by Data Area (see page 69)
- A rebuild and restructure of the Index data set (IXX) for the best possible efficiency
- Reduced elapsed time when compared to the Reorganization by Data Area (see page 65) when all data areas are being processed
- If needed, a single data area could be reloaded from the full database backup

The following are the full database reorganization’s main drawbacks:

- Reorganizes all of the data areas and data tables even if certain tables were not out of native sequence
- The entire database must be off line
- Recovery operations have to be in sync with the load

**Reorganization by Data Area**

For CA Datacom/DB Version 12.0 users, the only time that a data area reorganization is recommended over a reorganization by database, is when you are only reorganizing a part of the database. In this case, one or two specific data areas can be reorganized quicker than reorganizing the entire database. In some cases, this can result in a shorter downtime. The following is a sample for this process.

1. Close the database to user processing. This would include:
   
   a. Close the user URTs that are open for this database in the MUF
   
   b. Close the database to MUF using the COMM CLOSE command
   
   c. Change the database’s ACCESS status from WRITE to UTLTY or OFF
2. **Backup the affected areas.**

   Once the database has been closed to other processing, submit a DBUTLTY BACKUP function specifying SEQ=NATIVE, DBID=nnnn and AREA=aaa. A backup function must be executed for each of the areas being backed up.

   Each backup function creates its own output data set. The DBUTLTY includes header records which allow the LOAD process to know to which data area the information belongs.

3. **Initialize the index (optional)**

   Initializing the index is optional in a data area reorganization. If the index is not initialized, then each area load process will pause while all index entries for the affected area are removed from the index before allowing the data rows to be loaded and new index entries created.

   If the index is initialized, the area LOAD process run faster, but after the load completes, you need to use the DBUTLTY RETIX function to rebuild the index entries for the data areas that were not loaded as part of this reorganization.

   The decision to do an index initialization is based on how much of the database is being reorganized and the time it takes to do the extra RETIX for the non-affected data areas.

4. **Load the data area backup with the SORT option**

   The data area level processing does not require you to use the SORT processing during the loading of the data area. However, we recommend that you do use the SORT option to improve load performance and to keep the index structure as efficient as possible.

5. **Rebuild the index (dependant on whether INIT IXX was done)**

   If you chose to initialize the index prior to loading the areas (optional), rebuild the index for the non-loaded areas using the RETIX function.

6. **DEFRAG the database index (optional)**

   If the index was not initialized (step 3 and step 5), once the loads have been completed, we recommend that you execute the DBUTLTY DEFRAG utility to compress the database index, since the area load process does not restructure the index for those areas not included in the reorganization process.

   The data area reorganization provides the following benefits for the affected data areas:
   
   - 100% sequential access efficiency as documented previously
   - Data blocks that are tightly packed to provide the lowest blocks in-use percentage
- With index initialization and RETIX:
  - Index entries are rebuilt and restructured in the index dataset (IXX) for the best possible efficiency
  - Unique Record Identifier (URI) value for each row is reset
- Without index initialization:
  - Index entries for the affected data area are rebuilt ‘in place’ at the logically required orderly placement
  - Unique Record Identifier (URI) value for each row is reset in the affected data areas only
  - Reduced elapsed time compared to a full database reorganization when only a small number of the data areas are reorganized.

The following are the data area reorganization’s main drawbacks:

- You must make sure that you are backing up and loading all the appropriate data areas.
- If the index is not initialized and completely rebuilt:
  - There can be some minor inefficiencies in the index when compared to the full index rebuild.
  - URIs are not reset for areas that do not have their index entries rebuilt.
- The entire database must be off line.
- Recovery operations have to be in sync with the load.
Reorganization by Data Area with MULTUSE=YES

The MULTUSE=YES parameter for several DBUTLTY functions executes at the data area level. It allows you to perform maintenance functions such as a BACKUP SEQ=NATIVE function and the LOAD function on one data area while the rest of the database is left open for processing.

Being able to leave the unaffected areas of the database open in the MUF has limited value unless you have structured your data so that applications can continue to run with a specific data area offline for reorganization. The following is an example process.

1. Close the affected areas to user processing. This includes the following:
   a. Close the user URTs that are open for the data area(s) in the MUF
   b. Change the data area's ACCESS status from WRITE to UTLTY (or OFF).
   c. Backup the affected areas

2. Once the database area(s) has been closed to other processing, execute a DBUTLTY BACKUP function specifying SEQ=NATIVE, DBID=nnnn, AREA=aaa, and MULTUSE=YES.

   A backup function must be executed for each of the areas being backed up. Each backup function will create its own output data set.

   The DBUTLTY includes header records which allow the load process to know to which data area the information belongs.

3. Do not initialize the index (IXX)

   Since some portion of the database has been left accessible, the index must not be initialized.

4. Load the data area backup with the MULTUSE=YES and SORT option

   The data area level processing with MULTUSE=YES requires you to use the SORT processing during the loading of the data area. The MULTUSE option allows the non-affected parts of the database to remain open for other processing.

5. DEFrag the database index (optional)

   Once the load steps have been completed, we recommend that you execute the DBUTLTY DEFrag utility to defragment and compress the database index. The area load process does not restructure the index for those areas not included in the reorganization process.
The data area reorganization with MULTUSE=YES provides benefits that include the following for the affected data areas:

- 100% sequential access efficiency as documented in the preceding example.
- Data blocks that are tightly packed to provide the lowest blocks in-use percentage.
- Without index initialization, the index entries for the affected data area are rebuilt in place at the logical index location. This accomplishes the following:
  - Resets the URI value for each row in the affected data area (see discussion in Parallel Reorganization by Data Area)
  - Reduces elapsed time when only a small number of the data areas are reorganized when compared to a full database reorganization
- Non-affected parts of the database remain open for processing.

The data area reorganization with MULTUSE=YES has the following main drawbacks:

- You must make sure that you are backing up and loading all the appropriate data areas.
- There are possible user application failures when part but not all of the database is available for processing.
- There may be some minor inefficiencies when compared to the full database reorganization.

Parallel Reorganization by Data Area

The CA Datacom/DB DBUTLTY function REORG performs a parallel reorganization of the selected data area. The REORG function combines the three main parts (backup, load, and index update) of data reorganization into one DBUTLTY function.

As with the reorganization by data area covered previously, the REORG function is data area specific but still requires the database to be offline to other users.
A significant difference for the REORG function is that the three parts of the reorganization process can be split up into multiple tasks so that the process runs in a much faster elapsed time. REORG uses the multi-tasking ability of the MUF to perform much of its parallel processing. Therefore, the REORG function can only be executed when MUF is available.

For the REORG function, you indicate the number of parallel tasks that you would like to use. You also provide JCL DD statements for a matching number of output data sets to use for the backup process, as well as for the load process.

**Important:** DO NOT USE TEMP DATA SETS for the backup portion of the REORG process. These backup data sets should be created as permanent data sets. If a failure occurs during the load portion of the REORG process these data sets need to be there to restart the load process. Once the REORG process is complete, the data sets should be retained as the most recent backup for this data area for recovery purposes, like any other area backup.

The REORG function determines the selected number of equal segments of the area to be reorganized. It starts the selected number of tasks to back up each of the segments in parallel to the output data sets.

Once the backups are complete, a second set of tasks are initiated to read the output data sets and load the data into the data area. All index values, with the associated row’s new location in the data area, are stored in a memory data space during the data load. Once the data load is finished, multiple tasks are started to read the memory data space and update the index to reflect the new location of the data rows for the reorganized data area.

The REORG load process introduces a concept of data striping. Since there will be multiple processes running in parallel and since data row sizes in a data area can fluctuate, there is no way to know exactly how much space on the data area will be needed for the segment of data on the first backup data set or the second backup data set, and so on.

To accommodate the load-size variance, the REORG utility allows you to specify a data stripe size that each load task will use. For large data areas, a stripe of 15 or 30 tracks, that is, 1 or 2 cylinders, works well; for small tables, a stripe of 1 or 2 tracks would be used.

Each load task acquires a stripe of space on the data area. The task then starts loading rows into the stripe as fast as possible. Once the current stripe is full, the task acquires another stripe of available space on the data area and the process continues until all data rows are loaded. Each load task has a set of stripes that it has loaded with data. There is no guaranteed order for the stripes, but as we have stated earlier, with today’s modern DASD there is very little benefit to trying to place tracks and cylinders in perfect sequential order.

In today’s world, what matters is that rows that are read in sequence are on the same data block. The REORG function accomplishes this goal.
The parallel reorganization process can be significantly faster than a normal data area reorganization when there are multiple CPUs available to allow the multiple tasks to execute in parallel.

For example, to reorganize the PAY data area of database 001, you could specify:

```
REORG OPTION2=BACKUPLOAD, AREA=PAY, DBID=001, INNUM=3, DDNAME=BACKUP, NUMBER=15
```

**OPTION2=BACKUPLOAD**

Indicates that all three reorganization steps are to be done.

**INNUM=3**

Indicates that 3 tasks are to be used for the backup and load of the data rows.

**NUMBER=15**

Indicates that during the load process, each task is to acquire stripes of 15 tracks of space at a time.

The process to run a REORG function with three parallel tasks should reduce the elapsed processing time by two-thirds over the normal area reorganization process.

**An example REORG process is**

1. Close the affected areas to user processing. This includes the following:
   a. Close the user URTs that are open for the data area(s) in the MUF.
   b. Close the DBID to MUF using the COMM CLOSE command.
   c. Change the database’s ACCESS status from WRITE to UTLTY.

2. REORG the affected areas.

   Once the database has been closed to other processing, a DBUTLTY REORG function is executed. A REORG function must be executed for each of the affected data areas. Each REORG function creates its own set of output data sets.

3. **DEFRAG the database index (optional).**

   Once the load steps have been completed, we recommend that you execute the DBUTLTY DEFrag utility to defragment and compress the database index. The parallel reorganization process does not restructure the index during the reorganization process.
The parallel data area reorganization provides benefits include the following for the affected data areas:

- 100% sequential access efficiency.
- Data blocks that are tightly packed to provide the lowest blocks in-use percentage.
  
  There will be slightly more blocks in-use due to the fact that the last stripe for each segment may end with a partially used block or set of blocks. This should be a minor amount.
- Reduced elapsed time compared to a data area reorganization.
- This includes significant savings since the index area is not rebuilt.

The main drawbacks of the parallel REORG function are:

- The URI value for each row is not reset since the index entries are only updated with the new data row placement, that is, they are not rebuilt. This is a concern because of the following:
  - A given table may only have a high URI value of 4 billion.
  - URI values are assigned sequentially, 1 to 4 billion, each time a row is added.
  - URI values are not reclaimed when a data row is deleted.
- For most data areas, it could be years or even decades before enough rows were added to cause a concern about reaching the URI maximum.
- For certain very large data areas, the URI value may be a concern. For these data areas, you may need to do an offline data area reorganization or a database reorganization once the URI value exceeds a certain threshold.
- CA Datacom/DB warns you when the URI exceeds 3 billion, and you can query the URI value whenever needed by running a dynamic system table query (DB02801).
- The index is left intact which means that no IX$ restructuring or compression is done.
- We recommend that you execute a DBUTLTY DEFRAG function against the database once the REORG is complete. The DEFRAG function performs an index area compression that improves the index structure and recovers fragmented index space.
- You must make sure that you are backing up and loading all the appropriate data areas.
- The amount of blocks in-use may be slightly higher.
- The entire database must be off-line.
Parallel Reorganization by Data Area with MULTUSE=YES

The parallel data area reorganization (REORG) function can be executed with the MULTUSE=YES parameter, which allows the other data areas of the database to remain open while the specified area is being reorganized. It adds the benefit of the database having some areas open for processing while still using the multi-tasking capabilities to reduce the elapsed time.

Miscellaneous Reorganization Processes

Various other methods exist that can be used to pull the data from the data area in native sequence, and then to load the data rows back to the data area to achieve a data area reorganization. The same can be true for reorganizing a data table.

Other processes can use the physical backup or extract process to create a non-native key sequence backup data set. This data set can be processed by a sort function to place the data rows in Native Key sequence for the load process.

The major concern with these miscellaneous processes is that the data area or data table must be protected during the backup to keep rows from being deleted, added, or moved. The data area/table must remain locked while the data is being loaded.

If these processes are not protected or they are not executed correctly, you could face data integrity issues such as missing rows and duplicate rows.

If a non-standard process is used, you may want to use the NER to measure the effectiveness of the miscellaneous process against the various standard reorganization processes.

Chapter Summary

The CA Datacom/DB DBUTLTY program provides a series of functions that can be used to do data row reorganizations to restore the data rows into Native Key sequence.

Each process involves the following:

- Taking part or all of the database offline
- Creating a backup of the data rows in Native Key sequence
Chapter Summary

- Reloading the data rows
- Updating or rebuilding the index entries for the affected data rows

Each of these processes uses the integration between DBUTLTY and the MUF to insure that the selected data rows are unavailable for access by users, so that the data remains stable during the process. This prevents certain data integrity issues that could occur if the data was available for other users for maintenance activities.

Each user site has a selected favorite for how they do data row reorganizations. As we have discussed, each offline reorganization process offers its own set of advantages and disadvantages.

By considering all of the choices, the CA Datacom/DB user can use the offline reorganization process that best fits the need for a given database or situation.
Chapter 5: DBUTLTY OLREORG Functionality for Release 11

The online reorganization function (OLREORG) was introduced as part of DBUTLTY in CA Datacom/DB r11. This function uses a patented process to move data rows within groups of data blocks to improve the order of the rows according to the Native Key. The process runs in the background of the MUF while allowing other user tasks to continue their processing against the same tables.

In addition to the OLREORG functionality, DBUTLTY also delivered the new reference group report that allows users to view the data population of a group of rows within a reference group of blocks.

The OLREORG function is only valid for data tables residing in a data area with DSOP of 0, 1, or 2. The DBUTLTY OLREORG function is described in the CA Datacom/DB DBUTLTY Reference Guide.

Determining if OLREORG is a Good Choice for Reorganization

Using the offline BACKUP SEQ=NATIVE and LOAD functions provide the optimum results in reorganizing the data rows in native sequence. However, as we have also discussed, the requirement to take the database offline during this type of reorganization has become problematic for sites where the data must be continuously available.

For these sites, we delivered the online reorganization capability (DBUTLTY OLREORG).

The OLREORG function works within the MUF to perform the following:

- Locate a number of data blocks specified by the user
  - This is called the reference group.
  - The blocks are located by reading data rows using the Native Key until the specified number of blocks are selected.
  - The rows in these blocks are called the reference set.

- Determine which data blocks of the group have the most number of rows from the reference set
  - These are called the target blocks.
OLREORG for Single-Table Data Areas

- Determine which data blocks of the group have the least number of rows from the reference set
  - These are called the source blocks.
- Move the rows from the source data blocks to the target data blocks, as space on the target blocks permits
  - The goal is to eliminate one or more of the source blocks from use in this reference set by moving its rows into the available space in the target blocks.
  - Each block that is no longer needed for the reference set reduces the resources required to process the rows sequentially.
- Commit the work and move onto the next group of rows when all source rows have been moved or there is no space left in the target blocks.

The idea behind OLREORG is that it runs as a passive background process that can improve the native sequence order of the data rows for a table without taking the table offline or causing application performance issues.

The effectiveness of the OLREORG is limited by the amount of free space that is available in the data blocks of the reference group. This free space provides the room to move rows from least populated blocks to the most populated blocks.

Experience with the OLREORG function in CA Datacom/DB Version 12.0 has shown that data areas that have naturally occurring deleted space in the blocks or have free space created by a LOAD with SLACK receive the most benefit from OLREORG.

For example, if there is a nightly purge process that deletes rows from a table, this would create free space in various data blocks in the data area. Running the OLREORG function before other processing can use up this free space provides the best results for OLREORG.

Most users running the OLREORG function have reported that the utility does improve sequential ordering and, therefore, can be a benefit to sequential processing. You can run the NER before and after the OLREORG processing to validate the processing improvement.

OLREORG for Single-Table Data Areas

OLREORG is architected for the single-table data area. It is designed to gather a group of data blocks and move the selected rows from one block to another to improve the sequential performance of that table.
OLREORG for Multi-Table Data Areas

OLREORG is not designed for the multi-table data area. Multi-table data areas are no longer generally recommended for typical databases in CA Datacom/DB. However, in certain cases, multi-table data areas can still provide some value, and some sites may still be using these data areas for user data. Multi-table data areas provided two main benefits:

- Reduced JCL and data sets since there is only one physical data set per data area, which can house up to 240 tables
- Special performance optimization for very specialized applications

The Datadictionary database is an example of such a data area. It is delivered as a multi-table data area to reduce the installation data set requirements and to ensure that all Datadictionary tables are backed up and loaded in-sync. Datadictionary tables are very integrated and should never be restored independently of each other. At most sites, the Datadictionary database will be a minimally accessed database and will not have significant sequential processing. It typically is not a target of data row reorganization. If needed, the Datadictionary database should be reorganized using the standard offline data row reorganization techniques.

There are two types of multi-table data areas as follows:

**Co-Located tables**

These tables are in the same data area but they do NOT share the same Native Key DATACOM-ID, the data rows are said to be co-located.

In a co-located data area, a BACKUP SEQ=NATIVE and LOAD reorganization processes each table separately, thus providing a data area where each table’s data rows are physically located together. This occurs because the load is loading the data as it was backed up, by the separate Native Key DATACOM-IDs. After the data is loaded, database maintenance activity, that is, deletes or adds, occur in a random fashion which will intermix the data rows from the different tables in the data blocks over a period of time.

Co-location is typically something that is done to reduce the number of data sets, reduce the required JCL, share available DASD extents, and so forth. The Datadictionary database area DD1 is an example of a co-located data area used to reduce JCL and share DASD allocations.
Co-Mingled tables

These tables are in the same data area and they have Native Keys that share the same DATACOM-ID, these data area’s rows are said to be co-mingled.

In a co-mingled data area, a BACKUP SEQ=NATIVE creates a backup file where each of the data tables in the area have their rows intermixed according to the shared Native Key value. When the load is done with this backup file, the data blocks are built with the data rows co-mingled in Native Key sequence. After the data is loaded, database maintenance activity occurs in a random fashion, which over a period of time will reduce the co-mingling by Native Key value.

Co-mingling is typically done with some forethought and planning. The placement of data rows with the same Native Key value on the same data block would provide some performance benefits when all of the tables are typically accessed together by the same Native Key value.

For example, in the sample ORDER-ENTRY database (DBID 010) provided when CA Datacom/DB is installed, there is an area called DETAIL (DTL) that houses the DETAIL(DTL) and ORDERS(ORD) tables. For each of the ORDERS rows, a number of DETAIL rows exist. The DETAIL table has its Native Key called DTLOR with the DATACOM-ID of 004. The ORDERS table has its key called ORDID with the DATACOM-ID of 004. Since both tables have a Native Key with the same DATACOM-ID value, their data rows will be intermixed according to the Native Key value during a SEQ=NATIVE backup followed by a load.

This placement of the data rows could improve performance when the ORDERS row is read along with its matching DETAIL rows; however, if only the ORDERS rows are read, performance could be hurt because the chance that multiple ORDERS would be in the same data block would be smaller.

The concept of co-mingling for performance has lost value over the years. In today’s environment with thousands of data buffers, the chance that data blocks from two different tables will be kept in storage is much greater thereby reducing the need for the significant effort to plan and maintain the co-mingled data area.

OLREORG will run to completion on multi-table data areas. But the value of the reorganization may be limited.

For multi-table areas with co-located tables, OLREORG uses the table’s Native Key values to locate the data blocks and move the data rows for efficiency. However, since data blocks can also contain rows from other tables that are not being processed by the OLREORG, the amount of movement allowed can be limited. In this case, while OLREORG will have limited affect, it would not hurt the overall efficiency of the design.
For multi-table areas with co-mingled tables, OLREORG works to move rows from the selected table into the same data block. This movement works to co-locate the data rows by table and remove the data row co-mingling created by the shared Native Key DATACOM-ID specification. In this case, OLREORG could actually negatively affect the performance of the co-mingled data area.

**Important!** Do not use OLREORG against data areas that have been architected for performance purposes to house co-mingled data tables. For these types of data areas, you should rely on the various offline reorganization techniques which will use the shared Native Key information.

Understanding the OLREORG Process and Determining the REFGROUP Parameter Value

The DBUTLTY guide defines the use of the reference group (REFGROUP) as input to OLREORG as the following:

The OLREORG function of DBUTLTY combines rows in a given key range into as few blocks as possible while reducing the effort, that is, the physical I/O, needed to retrieve the rows in native sequence, and it does this without making the data unavailable to other users. OLREORG moves rows from blocks with the fewest rows within a specified reference group (REFGROUP) to blocks with the most rows within the REFGROUP, thereby reducing the number of blocks used to contain the REFGROUP. Be aware that OLREORG does not provide a one hundred percent Native Key sequence.

A reference group is a specified number of data blocks that are to be considered as a group in reporting and analyzing, and in reorganizing using OLREORG data rows on a reference group by reference group basis.

For the OLREORG user, determining the right number to use for the REFGROUP parameter is part science and part experience. First, let’s consider the science part of the equation.
The Science Part

The OLREORG utility is made up of many small separate processes where a group of rows on selected blocks are analyzed and then moved to improve native sequential ordering. Each process is completed by a commit which saves the work done in the process.

The REFGROUP parameter selects the number of data blocks that will be used for a single OLREORG process. For each process, the data rows are read in native sequence until one more block is referenced than the REFGROUP value. For example, if we select REFGROUP=2, the process begins reading rows in Native Key sequence saving the block number where the rows reside. We cannot stop reading rows until we hit the third unique block number or the end of the table.

When the process hits the third unique block number, the process knows all the rows in Native Key sequence that are in the reference group. In this example, this could be just 2 rows (1 per data block) or it could many rows (2 times the maximum rows per block). We call this group of rows the reference set.

This OLREORG process is only concerned with the rows in this reference set. There may be other rows also on the selected data blocks, but they are ignored for this OLREORG process. These other rows are most likely involved in a different OLREORG process. A certain data block can be involved in many reference groups if the data rows on the block are spread across the Native Key range.

Once the reference set is determined, each block in the reference group is weighted to determine which of the blocks have the most rows for the reference set and which of the blocks have the least rows of the reference set.

Starting with the data block with the least rows in the reference set, a move operation begins that attempts to move the least block rows to the block with the most rows in the reference set. Once the most block is full, the process attempts to move rows to the block that has the second most rows, and so. This continues until all the blocks with the least block rows have been move to the most blocks, or until the most blocks are full. The overall goal is to reduce the number of data blocks that are used to hold this reference set of rows. By reducing the number of data blocks holding the reference set, we improve processing by the Native Key.
The Experience Part

Since the OLREORG process needs to move the least block rows to the blocks with the most rows to improve performance, larger reference groups provide more possibility that there will be free space available to move rows.

However, for a large reference group, there is a higher possibility that moved rows will still not be close enough together to improve the sequential processing done by a user program.

Most user programs run with some number of attached data buffers. These buffers allow data blocks that are read in for previous requests to stay in memory and possibly be reused for future requests.

The user program issuing “read next” types of commands get their data buffers from the shared pool of MUF data buffers. The user program using the GSETL/GETIT set of commands can assign an additional pool of dedicated buffers for this program’s processing. Use your knowledge of the way your user programs process this table by the Native Key and the number of buffers typically available to the sequential process to determine the OLREORG REFGROUP value.

Basically, we want to do the following:

■ Choose a REFGROUP size that provides as many data blocks as possible to provide the best opportunity to combine rows from the source blocks to targets blocks,

■ But also a number of blocks that remains within the reasonable number of buffers that a batch sequential task may have for this table.

If you are unable to determine what a reasonable expectation for available buffers to sequential tasks will be, then we recommend that a relatively small REFGROUP be used, such as a value between 4 and 12.
The OLREORG Report

The DBUTLTY OLREORG utility generates a simple report showing the activity performed by the OLREORG function. For the remainder of this chapter, we will be extracting key information from the OLREORG report and including it as part of various figures.

The OLREORG report is documented in the DBUTLTY guide. The primary information columns are shown in the OLREORG report as the following:

- **Rows processed**: 928,454
- **Rows moved**: 358,691
- **Blocks freed in reference group**: 105,737
- **Exclusive control conflicts**: 0

For sites that have applied the new OLREORG PTFs discussed in the next chapter, the following two additional lines appear on the OLREORG report:

- No empty blocks used due to DSOP
- Block changes save by packing: 0

These additional lines can be ignored for CA Datacom/DB Version 12.0 users that are not using the new DSOP options (DSOP 4 and DSOP 5) discussed in the following sections.

**A Challenging Table for OLREORG**

The following set of charts shows the effect of using OLREORG with different REFGROUP sizes against two heavy sequential use tables.

The first table is the REO table of data area REO in database 1022. This data area and table can have heavy sequential access by the Native Key. At times the sequential efficiency of the table can be very low at around 13% MRB efficiency rating.

The REO table provides a significant challenge to the OLREORG process because it was created in such a way that there is very little free space in the data blocks. This limits the ability of OLREORG to move rows. However, even in this constrained environment, we can see that OLREORG did help improve efficiency slightly.

The example is typical of a data area that has not been reorganized very often and has very little free space, deleted rows, in the data blocks that are in-use.
The following chart contains various items of information collected from the OLREORG report (blue), the Native Efficiency Report (red), and the CXX data area and index area reports (purple). The NER is discussed throughout this document and is documented in Implementing and Using DBUTLTY REPORT=DATANE (see page 139).

For this example, we are resetting the data area back to the same exact image and then running the OLREORG utility with different REFGROUP values.

| DSOP 1 | DBID/Table | Rows | Rows Moved | Refers Freed | Empty Blocks Used | Blocks Moved | MRB Eff% | AMRB Eff% | MRB w/BER | MRB w/AMRB | MRB w/AMRB | MRB w/AMRB | MRB w/AMRB | Empty Blocks Freed | Index MRI Blocks |
|--------|------------|------|------------|--------------|-------------------|--------------|----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------------|-----------------|
| After Load | 2022/TEDO | 31,950 | | | | 0 | 12 | 12 | 12 | 12 | 12 | 0 | 0 | | |
| REFGROUP 2 | 31,950 | 3 | 3 | 0 | 0 | 12 | 12 | 12 | 12 | 12 | 12 | 0 | 1 | | |
| REFGROUP 4 | 31,950 | 11 | 10 | 0 | 0 | 12 | 12 | 12 | 12 | 12 | 12 | 0 | 1 | | |
| REFGROUP 8 | 31,950 | 31 | 20 | 0 | 0 | 12 | 12 | 12 | 12 | 12 | 12 | 0 | 1 | | |
| REFGROUP 12 | 31,950 | 35 | 27 | 0 | 0 | 12 | 12 | 12 | 12 | 12 | 12 | 0 | 3 | | |

In this example, you can see that each test starts with a table that has 31,950 rows stored on 888 data blocks. The NER has shown that the sequential MRB efficiency as 12% with no buffers and only 12% with 12 buffers.

Each line of the chart shows the reference group value followed by the number of data rows that were moved to improve the sequential order. Following the rows moved, we see the number of block references that were eliminated or freed. As described, a reference that is eliminated means that we have improved the sequential processing. In the chart the number of rows moved and the number of references freed is very small and had no significant affect on the data row order. This information is validated by the NER which shows that both the base efficiencies, MRB and AMRB, did not improve in any of the OLREORG runs in this chart.

Since the main purpose of the OLREORG is to move rows within the existing set of data blocks to improve sequential efficiency, there may be no effect on the total number of blocks in-use, as we can see in the chart. The final column in the chart is the Index MRI (moved record index) Blocks. When OLREORG moves a data row to improve its native sequence, an MRI index entry is created in the MRI index. The MRI index resides in the database index (IXX) area. These entries are only used in the case of a database recovery operation. These entries are not used by normal user processing and, as such, will not affect application performance. Typically, the amount of space occupied by the MRI in the IXX is very small.

**Note:** The purpose of the OLREORG function is to re-order data rows for better sequential access performance. While the moving of data rows allows some data blocks to be emptied there is no direct correlation or even guarantee that moving rows for better sequential performance will reduce blocks in use. It is just a side benefit. OLREORG with DSOP 1 or DSOP 2 will not use any additional data blocks.
A Good Table for OLREORG

For this second example, we will use a different table that has a much larger row count and has deleted row free space spread among its blocks. The free space allows OLREORG to move more rows and accomplish more re-sequencing of the data rows.

The MET table resides in data area G03 of database 4033. In Chapter 2, we saw that this data area and table had heavy sequential access by the Native Key.

The following chart contains various items of information collected from the OLREORG report (blue), the Native Efficiency Report (red), and the CXD data area and index area reports (purple). The NER is discussed throughout this document and is documented in Implementing and Using DBUTLTY REPORT=DATANE (see page 139).

For this example, we are resetting the data area back to the same exact image and then running the OLREORG utility with different REFGROUP values.

In this example, you can see that each test starts with a table that has 928,454 rows stored on 8,069 data blocks. The NER has show that the sequential MRB efficiency is 0% with no buffers and only 2% with 12 buffers. This table is currently in a very poor sequential access state.

Each line of the chart shows the number of data rows that were moved to improve the sequential order. Following the rows moved, we see the number of block references that were eliminated or freed. As described, a reference that is eliminated means that we have improved the sequential processing. This information is validated by the NER which shows that both the base efficiencies, MRB and AMRB, improved, while the efficiency with buffers significantly improved. The NER figures also show that the smaller REFGROUP values tend to focus on improving the efficiency for the smaller with buffer efficiencies while the larger REFGROUP values move more rows, but focus on improving the higher with buffer efficiencies.

It is significant to note that with a REFGROUP=12, the sequential efficiency for sequential access with 12 data buffers went from 2% to 51%. This would result in a dramatically improved sequential access process that had at least 12 data buffers available to it.
Since the main purpose of the OLREORG is to move rows to improve sequential efficiency within the existing REFGROUP of data blocks, there may be no effect on the total number of blocks in-use. However, when the data area has a significant number of blocks with space available that allows more rows to be moved, there can be a significant compaction of data rows into fewer data blocks which results in a number of data blocks being emptied and returned to the empty block pool.

In this example, we see that a number of blocks have been returned to the empty pool for this data area. The final column in the chart is the Index MRI (moved record index) Blocks. When OLREORG moves a data row to improve its native sequence an MRI index entry is created in the MRI index. The MRI index resides in the database index (IXX) area. These entries are only used in the case of a database recovery operation. These entries are not used by normal user processing and, as such, will not affect application performance. Typically, the amount of space occupied by the MRI in the IXX is very small. In this case more rows were moved, so more index blocks, 4K in size, were used for the MRI; however, the number of data blocks emptied, 27K in size, greatly offsets the space used in the IXX.

**Determining the NUMBER Value for OLREORG**

We can see that the REFGROUP parameter specification is the main consideration for running the OLREORG. However, the NUMBER parameter is provided as an optional parameter for OLREORG. The DBUTLTY guide contains the following definition:

**NUMBER parameter input to OLREORG**

This parameter can be specified as 1-99 or omitted. When you omit NUMBER=, all blocks qualify as source blocks and OLREORG moves as many rows as possible, so that as few blocks as possible contain the REFGROUP. Specifying a value for this parameter indicates you want to control which blocks qualify as move-from blocks. It can be used to limit row movement to those cases that provide the most benefit. For example, specifying NUMBER=2 indicates that only blocks with two or fewer rows may have their rows moved to other blocks in the REFGROUP.

For the OLREORG user, determining the right value to use for NUMBER is dependent on the size of the data table, quantity of the rows, and how much resource is available, CPU and I/O, to perform the OLREORG.

In general, the NUMBER specification limits which blocks can qualify as source blocks. If you specify a number such as 2, you tell OLREORG to only consider data blocks with 2 or less rows from the reference set as source blocks.
Determining the Need for Repeating an OLREORG

Using a low value for NUMBER can greatly reduce the amount of data blocks that are qualified as source blocks, and targets just the worst source data blocks for the OLREORG move process.

Using a high-value or omitting the NUMBER parameter allows OLREORG to qualify move blocks as source blocks enabling the OLREORG to move as many rows as it deems possible to improve the overall sequential order of the data table.

Since the OLREORG function can be canceled any time without endangering the data table or losing the OLREORG work done so far, we recommend that you use the NUMBER parameter only when you have experience with a given data table and you know that the resource to complete the full OLREORG is not available.

For other circumstances, we recommend that you omit the NUMBER parameter from the OLREORG function.

**Determining the Need for Repeating an OLREORG**

You can execute multiple OLREORG functions against the same table. Typically, each of the succeeding OLREORG function moves less rows than the previous OLREORG. At some point, the amount of rows moved and sequential performance gained may not be cost effective when comparing the cost of the multiple OLREORG executions to the improved efficiencies.

In some cases, you may find that the table is so densely packed that the first OLREORG will have minimal effect. So running additional OLREORG functions is just wasting resources.

In other cases, you may find that the table is already in good sequential order and there is no need for multiple OLREORGs.

While there is no technically reason that you could not run a set of serially executed OLREORG functions against the same table, we recommend that you review the output of one OLREORG before starting a second one for the same table.

Using the MET 4033 example, we executed a series of 15 OLREORG functions with REFGROUP=4 against the same table. Each of the OLREORG functions had a diminishing effect. Since the entire table is passed in each of the OLREORG executions, it is probable that the cost per row moved goes up with each succeeding execution.
If you have available processing cycles and there is no concern over the resources consumed, you might consider running repeated OLREORG functions where past experience has shown this to be beneficial.

The following chart contains various items of information collected from the OLREORG report (blue), the Native Efficiency Report (red), and the CXX data area and index area reports (purple).

For this example, we are going to run repeated OLREORG functions with the same selected REFGROUP value to show the way the reference set size works to improve sequential efficiency.

In this example, the number of rows moved continued to decline as the OLREORGs executed. Each OLREORG function still had some positive effect on the blocks in use as well as the sequential efficiency of the table.

Since each OLREORG function is moving rows to make the sequence better, it is possible that the same row could be moved multiple times in repeated OLREORG functions. Remember, each OLREORG run will be focused on moving rows to improve order using the reference group blocks. Subsequent runs, even with the same REFGROUP value, may provide slightly different reference sets which could result in a row being moved twice.

Using the DATANE Report to Verify the OLREORG Process

In the following example, we are using a very large compressed table that has widely varying data row size. Rows are added and deleted frequently. In this case, we run a NER before and after the OLREORG function to measure the impact.
Using the DATANE Report to Verify the OLREORG Process

Here is the NER before the OLREORG

| GROUP | ROWS | BLOCKS | MAX | MIN | AVG | BLOCK EFF | 2 | 4 | 8 | 12 | 16 | 20 | 24 | 28 | 32 | 48 | 64 | 96 | 128 | 256 |
|-------|------|--------|-----|-----|-----|-----------|---|---|---|----|----|----|----|----|----|----|----|----|----|
| READ  | RMDS | RMDS   | RMDS| RMDS| RMDS| RMDS      |  |   |   |    |    |    |    |    |    |    |    |    |    |    |
| 1     | 50,000 | 1,135 | 139 | 1  | 44 | 48,529 | 0  |   |   |    |    |   |    |    |    |    |    |    |    |
| 2     | 50,000 | 829   | 277 | 1  | 60 | 38,864 | 0  | 1  | 2  |    |    |   |    |    |    |    |    |    |    |    |
| 3     | 50,000 | 752   | 277 | 1  | 66 | 28,381 | 0  | 1  | 21 | 23 | 24 |   |    |    |    |    |    |    |    |
| 4     | 50,000 | 638   | 277 | 1  | 78 | 24,824 | 0  | 2  |    |    |    |   |    |    |    |    |    |    |    |
| 5     | 50,000 | 546   | 277 | 1  | 92 | 22,734 | 0  | 2  | 28 | 32 |    |   |    |    |    |    |    |    |    |
| 6     | 50,000 | 307   | 277 | 2  | 163| 7,469  | 0  | 4  | 55  | 58 |    |   |    |    |    |    |    |    |
| 7     | 50,000 | 286   | 277 | 1  | 179| 6,289  | 2  | 44 | 62  | 64 |    |   |    |    |    |    |    |
| 8     | 50,000 | 364   | 277 | 2  | 137| 12,620 | 1  | 2  | 41  | 48  | 49 |    |   |    |    |    |    |
| 9     | 50,000 | 402   | 277 | 1  | 124| 24,485 | 0  | 36 | 42  | 45 |    |   |    |    |    |    |
| 10    | 50,000 | 313   | 277 | 1  | 160| 12,817 | 1  | 2  | 3  | 57 |    |   |    |    |    |    |
| 11    | 50,000 | 352   | 277 | 2  | 142| 20,635 | 0  | 49 | 50  | 51 |    |   |    |    |    |    |
| 12    | 50,000 | 447   | 277 | 1  | 112| 31,053 | 0  |    |    |    |    |   |    |    |    |    |
| 13    | 50,000 | 518   | 139 | 2  | 97 | 47,061 | 0  |    |    |    |    |   |    |    |    |    |
| 14    | 50,000 | 516   | 277 | 1  | 97 | 23,876 | 0  |    |    |    |    |   |    |    |    |
| 15    | 50,000 | 413   | 139 | 3  | 121| 428   | 43 |    |    |    |    |   |    |    |    |
| 16    | 50,000 | 379   | 139 | 9  | 132| 6,172  | 2  | 11 | 47 |    |    |   |    |    |    |
| 17    | 50,000 | 424   | 139 | 1  | 118| 49,159 | 0  |    |    |    |    |   |    |    |    |
| 18    | 50,000 | 395   | 139 | 1  | 127| 35,067 | 0  |    |    |    |    |   |    |    |
| 19    | 28,454 | 241   | 139 | 1  | 118| 9,749  | 1  | 18 | 40 | 41 |    |   |    |    |

Note that before the OLREORG run, the MRB efficiency ratings are:

- Basic 0%
- 12 buffers 2%
- 20 buffers and above 36%

```
CONTROL CARD(S)
........1........2........3........4........5........6........7........8
OLREORG TABLE=MET, DBID=4033, REFGROUP=4

FUNCTION=OLREORG
DBID=4033
REFGROUP=00004
TABLE=MET

Rows processed - 928,454
Rows moved - 342,090
Blocks freed in reference group - 112,264
Exclusive control conflicts - 0
No empty blocks used due to DSOP
Block changes save by packing - 0
```
On a fully loaded system, the OLREORG ran for 3 minutes elapsed time and was able to move over 342,090 rows to improve performance.

<table>
<thead>
<tr>
<th>GROUP</th>
<th>ROWS</th>
<th>BLOCKS</th>
<th>MAX</th>
<th>MIN</th>
<th>AVG</th>
<th>BLOCK EFF</th>
<th>CHANGES</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ</td>
<td>W/DATA</td>
<td>ROWS</td>
<td>BLOCKS</td>
<td>MAX</td>
<td>MIN</td>
<td>AVG</td>
<td>BLOCK EFF</td>
</tr>
<tr>
<td>1</td>
<td>50,000</td>
<td>932</td>
<td>277</td>
<td>1</td>
<td>54</td>
<td>14,620</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>50,000</td>
<td>468</td>
<td>277</td>
<td>1</td>
<td>107</td>
<td>7,513</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>50,000</td>
<td>317</td>
<td>277</td>
<td>2</td>
<td>158</td>
<td>4,117</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>50,000</td>
<td>397</td>
<td>277</td>
<td>1</td>
<td>126</td>
<td>7,365</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>50,000</td>
<td>392</td>
<td>277</td>
<td>1</td>
<td>128</td>
<td>7,510</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>50,000</td>
<td>258</td>
<td>277</td>
<td>2</td>
<td>194</td>
<td>3,434</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>50,000</td>
<td>258</td>
<td>277</td>
<td>1</td>
<td>194</td>
<td>3,215</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>50,000</td>
<td>323</td>
<td>277</td>
<td>1</td>
<td>155</td>
<td>5,587</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>50,000</td>
<td>273</td>
<td>277</td>
<td>1</td>
<td>183</td>
<td>8,545</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>50,000</td>
<td>271</td>
<td>277</td>
<td>1</td>
<td>185</td>
<td>4,551</td>
<td>3</td>
</tr>
<tr>
<td>11</td>
<td>50,000</td>
<td>297</td>
<td>277</td>
<td>2</td>
<td>168</td>
<td>8,811</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>50,000</td>
<td>354</td>
<td>277</td>
<td>1</td>
<td>141</td>
<td>7,967</td>
<td>2</td>
</tr>
<tr>
<td>13</td>
<td>50,000</td>
<td>364</td>
<td>277</td>
<td>2</td>
<td>137</td>
<td>15,781</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>50,000</td>
<td>360</td>
<td>277</td>
<td>1</td>
<td>139</td>
<td>6,721</td>
<td>2</td>
</tr>
<tr>
<td>15</td>
<td>50,000</td>
<td>261</td>
<td>277</td>
<td>1</td>
<td>192</td>
<td>424</td>
<td>42</td>
</tr>
<tr>
<td>16</td>
<td>50,000</td>
<td>273</td>
<td>277</td>
<td>2</td>
<td>183</td>
<td>3,931</td>
<td>4</td>
</tr>
<tr>
<td>17</td>
<td>50,000</td>
<td>319</td>
<td>277</td>
<td>1</td>
<td>157</td>
<td>16,874</td>
<td>1</td>
</tr>
<tr>
<td>18</td>
<td>50,000</td>
<td>262</td>
<td>277</td>
<td>1</td>
<td>191</td>
<td>14,454</td>
<td>1</td>
</tr>
<tr>
<td>19</td>
<td>28,454</td>
<td>149</td>
<td>277</td>
<td>1</td>
<td>191</td>
<td>3,938</td>
<td>2</td>
</tr>
</tbody>
</table>

FULL TABLE
| MRB | 928,454 | 6,154 | 277 | 1 | 151 | 145,108 | 2 | 6 | 9 | 16 | 25 | 32 | 50 | 51 |
| AMRB| 928,454 | 6,154 | 277 | 1 | 151 | 145,108 | 4 | 12 | 16 | 30 | 45 | 58 | 92 | 94 | 95 |

Now we repeat the NER and see that the basic MRB efficiency improved from 0 to 2%, but that the 12 buffer efficiency improved from 2% to 25% making the table 10 times more efficient to read sequentially with just 12 buffers.

We have decided to run two additional OLREORG runs to improve the table further. Note that the number of rows we are able to move continues to decline.

CONTROL CARD(S)...........1...........2...........3

...4...........5...........6...........7...........8

OLREORG TABLE=MET,DBID=4033,REFGROUP=4
FUNCTION=OLREORG
DBID=04033
REFGROUP=00004
TABLE=MET

Rows processed - 928,454
Rows moved - 81,824
Blocks freed in reference group - 5,598
Exclusive control conflicts - 0
No empty blocks used due to DSOP
Block changes save by packing - 0

Rows processed - 928,454
Rows moved - 121,103
Blocks freed in reference group - 21,540
Exclusive control conflicts - 0
No empty blocks used due to DSOP
Block changes save by packing - 0
Each of the OLREORG functions ran in under 2 minutes. After the two OLREORG runs, we execute a NER.

| GROUP | ROWS | BLOCKS | MAX | AVG | CHANGES | BLOCK EFF | MRB | AMRB | 4 | 8 | 12 | 16 | 20 | 24 | 28 | 32 | 48 | 64 | 96 | 128 | 256 |
|-------|------|--------|-----|-----|---------|-----------|-----|------|---|---|---|---|----|----|----|----|----|----|----|----|----|----|
| 1     | 50,000 | 473 | 277 | 1 | 106 | 5,491 | 3 | 8 | 23 | 31 | 35 | 36 | 37 | 38 |
| 2     | 50,000 | 313 | 277 | 1 | 160 | 5,015 | 3 | 16 | 22 | 46 | 57 | 58 |    |
| 3     | 50,000 | 248 | 277 | 1 | 202 | 3,788 | 4 | 52 | 73 |    |    |    |    |    |
| 4     | 50,000 | 251 | 277 | 1 | 199 | 5,894 | 3 | 55 | 72 |    |    |    |    |    |
| 5     | 50,000 | 292 | 277 | 2 | 171 | 5,820 | 3 | 45 | 62 |    |    |    |    |    |
| 6     | 50,000 | 238 | 277 | 1 | 210 | 3,372 | 5 | 52 | 76 |    |    |    |    |    |
| 7     | 50,000 | 246 | 277 | 2 | 208 | 2,580 | 7 | 53 | 75 |    |    |    |    |    |
| 8     | 50,000 | 250 | 277 | 1 | 200 | 4,897 | 3 | 53 | 72 |    |    |    |    |    |
| 9     | 50,000 | 231 | 277 | 1 | 216 | 7,538 | 5 | 55 | 78 |    |    |    |    |    |
| 10    | 50,000 | 249 | 277 | 1 | 201 | 3,546 | 5 | 28 | 70 | 72 | 73 |    |    |    |
| 11    | 50,000 | 259 | 277 | 2 | 193 | 6,635 | 2 | 39 | 70 |    |    |    |    |    |
| 12    | 50,000 | 268 | 277 | 3 | 187 | 3,723 | 4 | 21 | 34 | 45 | 55 | 68 |    |    |
| 13    | 50,000 | 282 | 277 | 1 | 177 | 6,533 | 2 | 47 | 55 | 63 | 65 |    |    |    |
| 14    | 50,000 | 274 | 277 | 2 | 182 | 3,577 | 5 | 23 | 45 | 57 | 61 | 67 | 68 |    |
| 15    | 50,000 | 239 | 277 | 1 | 209 | 448 | 41 | 56 | 76 |    |    |    |    |    |
| 16    | 50,000 | 266 | 277 | 1 | 189 | 3,811 | 4 | 50 | 68 |    |    |    |    |    |
| 17    | 50,000 | 219 | 277 | 2 | 228 | 13,212 | 1 | 63 | 82 |    |    |    |    | 83 |
| 18    | 50,000 | 211 | 277 | 1 | 237 | 13,882 | 1 | 59 | 86 |    |    |    |    |    |
| 19    | 28,454 | 128 | 277 | 2 | 222 | 3,526 | 2 | 61 | 81 |    |    |    |    |    |

What we see is that while the MRB basic efficiency only improved from 2% to 3%, the efficiency with 12 buffers for sequential processing has jumped dramatically from 25% to 66%.

We executed the OLREORG function two more times and we see (NER not shown) that the MRB efficiency with 12 buffers has moved up to 77%. At this point, you could decide to do no further OLREORG runs as the table's sequential efficiency has now been improved enough to meet their needs.

**Using DATANE to Measure OLREORG in a Challenging Scenario**

In the first scenario, we stated that the REO 1022 table presented a challenging scenario. The data rows in the table are very disorganized and the rows are tightly packed into each data block. In this case, even though the OLREORG function recognizes that data rows could be moved to improve performance, there is no free space for it to work within the reference group. In these cases, OLREORG does what it can but it may not be very effective. Before the NER, we had to base our determination on whether the OLREORG function really helped by how many rows were moved.

Now with the NER, we can verify the table's sequential efficiency before and after the OLREORG function.
A simple way to demonstrate a tightly packed disorganized table would be to take a very
disorganized data area and table and, using the DBUTLTY BACKUP function, do a PHYSICAL
backup with RECID=NO specified followed by a LOAD function on the physical backup data. The
net effect is that the data remains out of order, but the data rows are densely packed by the load
into each data block.

<table>
<thead>
<tr>
<th>GROUP</th>
<th>ROWS</th>
<th>BLOCKS</th>
<th>MAX ROWS</th>
<th>MIN ROWS</th>
<th>AVG ROWS</th>
<th>MAX BLOCKS</th>
<th>MIN BLOCKS</th>
<th>AVG BLOCKS</th>
<th>RECID</th>
<th>PBC</th>
<th>MRB</th>
<th>MAX ROW/BLK EFFICIENCY RATES WITH BUFFERS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3,600</td>
<td>365</td>
<td>26</td>
<td>10</td>
<td></td>
<td>720</td>
<td>14</td>
<td></td>
<td></td>
<td>17</td>
<td></td>
<td><em>... BLOCK ...</em></td>
</tr>
<tr>
<td>2</td>
<td>3,600</td>
<td>387</td>
<td>30</td>
<td>9</td>
<td></td>
<td>857</td>
<td>12</td>
<td></td>
<td></td>
<td>13</td>
<td></td>
<td>PBC 100 MRB</td>
</tr>
<tr>
<td>3</td>
<td>3,600</td>
<td>448</td>
<td>36</td>
<td>8</td>
<td></td>
<td>724</td>
<td>14</td>
<td></td>
<td></td>
<td>18</td>
<td></td>
<td><em>... MAX ROW/BLK EFFICIENCY RATES WITH BUFFERS ....</em></td>
</tr>
<tr>
<td>4</td>
<td>3,600</td>
<td>442</td>
<td>36</td>
<td>8</td>
<td></td>
<td>677</td>
<td>15</td>
<td></td>
<td></td>
<td>19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>3,600</td>
<td>711</td>
<td>16</td>
<td>5</td>
<td></td>
<td>872</td>
<td>11</td>
<td></td>
<td></td>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>3,600</td>
<td>602</td>
<td>24</td>
<td>6</td>
<td></td>
<td>867</td>
<td>11</td>
<td></td>
<td></td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>3,600</td>
<td>545</td>
<td>28</td>
<td>7</td>
<td></td>
<td>780</td>
<td>13</td>
<td></td>
<td></td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>3,600</td>
<td>642</td>
<td>34</td>
<td>6</td>
<td></td>
<td>833</td>
<td>12</td>
<td></td>
<td></td>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>3,150</td>
<td>557</td>
<td>33</td>
<td>6</td>
<td></td>
<td>601</td>
<td>14</td>
<td></td>
<td></td>
<td>22</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This NER shows that the table has the minimum number of data blocks needed to hold the
number of data rows, 888 blocks times 36 rows per block equals 31,968 rows, and those data
rows are running only at 13% MRB efficiency. Even with 128 data buffers the efficiency only
improves to 16%. Running an OLREORG function with a reference group of 4 blocks against this
table still provides some processing gains. But as we can see, only 11 rows could be moved due
to the lack of free space in the reference group’s data blocks.

<table>
<thead>
<tr>
<th>CONTROL CARD(S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>...........1........2........3........4........5........6........7........8</td>
</tr>
<tr>
<td>OLREORG TABLE=REO,DBID=1022,REFGROUP=4</td>
</tr>
<tr>
<td>FUNCTION=OLREORG</td>
</tr>
<tr>
<td>DEID=001022</td>
</tr>
<tr>
<td>REFGROUP=000004</td>
</tr>
<tr>
<td>TABLE=REO</td>
</tr>
</tbody>
</table>

| Rows processed - | 31,950 |
| Blocks freed in reference group - | 10 |
| Exclusive control conflicts - | 0 |
| No empty blocks used due to DSOP | |
| Block changes save due to packing - | 0 |
Running the same OLREORG pass again had slightly better results because it could use some of the free space created by the first OLREORG run. But the amount of data row movement is still very limited.

CONTROL CARD(S)

Running the same OLREORG pass again had slightly better results because it could use some of the free space created by the first OLREORG run. But the amount of data row movement is still very limited.

Note: The OLREORG report has a new message at the bottom showing "No empty blocks used due to DSOP." This message indicates that the new OLREORG functionality PTFs have been applied, but the data set has not been altered to allow this new option for OLREORG. We discuss this further in the next chapter.

<table>
<thead>
<tr>
<th>GROUP</th>
<th>ROWS</th>
<th>BLOCKS</th>
<th>MAX</th>
<th>MIN</th>
<th>AVG</th>
<th>CHANGES</th>
<th>2%</th>
<th>4%</th>
<th>8%</th>
<th>12%</th>
<th>16%</th>
<th>20%</th>
<th>24%</th>
<th>28%</th>
<th>32%</th>
<th>48%</th>
<th>96%</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3,600</td>
<td>365</td>
<td>26</td>
<td>1</td>
<td>10</td>
<td>719</td>
<td>14</td>
<td>18</td>
<td>19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3,600</td>
<td>387</td>
<td>30</td>
<td>1</td>
<td>9</td>
<td>854</td>
<td>12</td>
<td>13</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3,600</td>
<td>448</td>
<td>36</td>
<td>1</td>
<td>8</td>
<td>722</td>
<td>14</td>
<td>19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3,600</td>
<td>442</td>
<td>36</td>
<td>1</td>
<td>8</td>
<td>673</td>
<td>15</td>
<td>19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>3,600</td>
<td>716</td>
<td>16</td>
<td>1</td>
<td>5</td>
<td>873</td>
<td>11</td>
<td>12</td>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>3,600</td>
<td>598</td>
<td>24</td>
<td>1</td>
<td>6</td>
<td>860</td>
<td>11</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>3,600</td>
<td>542</td>
<td>28</td>
<td>1</td>
<td>7</td>
<td>778</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>3,600</td>
<td>637</td>
<td>36</td>
<td>1</td>
<td>6</td>
<td>833</td>
<td>12</td>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>3,150</td>
<td>551</td>
<td>33</td>
<td>1</td>
<td>6</td>
<td>597</td>
<td>15</td>
<td>23</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Executing the NER after the two OLREORG runs, confirms that the number of block changes needed for a sequential process has declined by a small amount, but the overall MRB efficiency is still at 13%. You are faced with making a choice:

- Keep running OLREORG and getting the minimum benefits.
- Wait for a program to delete rows or other processing to occur that may create some free space to allow OLREORG to be more efficient.
- Schedule a database outage to allow for offline reorganization.
While this seems like a bleak scenario, you can do the following things until a scheduled outage can be put in place:

- Verify that sequential performance is really an issue.
- Consider making the data area or part of data area MRDF covered.
- Consider adding more data buffers for the current MUF execution using FLEXPOOL or for later MUF executions increasing DATAPool in the startup options.
- Consider using GSETL and GETIT processing for the heavy sequential processes.

The History Database or Accounting Facility along with the NER can be used as needed to monitor the data table efficiency and, if it remains reasonable, you may not need to do anything.

Chapter Summary

The CA Datacom/DB DBUTLTY OLREORG functionality provided a major enhancement for users. It provides a simple low-resource utility that could improve the sequential ordering of the data rows in a table without impacting user access to the table.

Using the new Native Efficiency Report (NER) before and after the OLREORG run, we can determine the need for the OLREORG run and the improvement in efficiency after the run.

The NER gives us the information needed to determine if an online reorganization is needed and in some cases whether multiple OLREORGs should be considered.

However, as we have discussed, the tightly-packed data area with a highly disorganized table may still be a challenge. We have shown in these cases that OLREORG runs may have only a minimal effect on improving efficiency.

In the next chapter, we discuss an enhancement to the OLREORG process that significantly improves the OLREORG function’s ability to process disorganized tables in these tightly packed data areas.
Chapter 6: New DBUTLTY OLREORG Functionality for Version 12.0

New OLREORG functionality is delivered with PTFs for DBUTLTY in CA Datacom/DB Version 12.0. This functionality combines the patented process to move data rows within groups of data blocks with the ability to allocate an empty block to improve the OLREORG function’s ability to increase sequential efficiency.

There are several key points that you must consider before using the new OLREORG functionality.

New Data Area Space Management Options (DSOP)

To implement the new OLREORG functionality with empty block use, we have created two new Data Area Space Management Option (DSOP) settings.

DSOP 4 (RANDOM FOR OLREORG)

This space management (reuse) option works exactly the same for user processing as DSOP 1 (RANDOM REUSE). In addition, this option creates a small space map that tracks empty blocks that can be used for the new OLREORG functionality. The space map is built in the index dataset (IXX) and is very small, using only a few blocks of space.

DSOP 5 (SEQUENTIAL FOR OLREORG)

This space management (reuse) option works exactly the same for the user processing as DSOP 2 (SEQUENTIAL REUSE). In addition, this option creates a small space map that tracks empty blocks that can be used for the new OLREORG functionality. The space map is built in the index dataset (IXX) and is very small, using only a few blocks of space.
Using RETIX with KEYNAME=*DATA to Rebuild Empty Block Index

Using CXXMAINT to select DSOP 4 or DSOP 5

DSOP 4 and DSOP 5 are fully implemented in the CA Datacom/DB Version 14 delivery as standard Datadictionary data area level attributes. However, for early users of the new OLREORG functionality in CA Datacom/DB Version 12.0, the primary way to select DSOP 4 or DSOP 5 is to use the DBUTLTY CXXMAINT function to alter the DSOP setting for the data area, such as in the following example:

```
CXXMAINT OPTION=ALTER, AREA=REO, DBID=1023, DSOP=4
```

This function can be submitted while the database is open for processing. As soon as the option is changed, the MUF begins collecting information about empty blocks for OLREORG use.

However, this information is not fully complete until the area has been loaded with the DSOP set at option 4 or 5, or a RETIX KEYNAME=*DATA has been completed.

Testing with only the CXXMAINT step and not the LOAD and RETIX functions has still proven to be effective; however, without the updated empty block index, the OLREORG function does not know about any empty blocks that are imbedded in the data area and focuses its attention only on the empty blocks at the end of the data area.

Using RETIX with KEYNAME=*DATA to Rebuild Empty Block Index

Use the DBUTLTY RETIX KEYNAME=*DATA function to rebuild the data space index. The MUF must be active to run this DBUTLTY function. The database can remain open for user processing during the execution of this function.

When a data area is converted from DSOP 1 to DSOP 4 (or DSOP 2 to DSOP 5) and the CXXMAINT ALTER statement was not followed with a LOAD of the data area, you should do a DBUTLTY RETIX KEYNAME=*DATA for the affected data area or database. This recreates the space index and the empty block index which allows OLREORG with DSOP 4 or 5 to work more efficiently.

For more information on this special RETIX function, see the CA Datacom/DB DBUTLTY Reference Guide.

Datadictionary Catalog Process Reverts the DSOP Setting

Until the full CA Datacom/DB Version 14.0 environment has been implemented, a Datadictionary catalog of the database or of a specific data area resets the DSOP setting back to the value that was stored in the Datadictionary.
You must re-execute the CXXMAINT ALTER DSOP command to reset the DSOP to option 4 or 5.

If you perform a catalog and do not reset the DSOP option to 4 or 5, no adverse affects will occur except that the OLREORG processing will revert back to the OLREORG function without the empty blocks.

Understanding the OLREORG DSOP Process and Determining the REFGROUP Parameter Value

The REFGROUP parameter in the OLREORG function with DSOP 4 or 5 works the same as documented for the OLREORG function with DSOP 1 or 2. As we discussed in the previous chapter, the REFGROUP is used to determine the number of blocks and their data rows that will be processed as a reference set in the OLREORG process.

With DSOP 4 or 5, one extra step been added before the moving of data rows begins. OLREORG determines whether all the rows on the least block can be redistributed to the target blocks in the reference group. If this is possible, then the OLREORG process is done for this reference group the same as is documented for an OLREORG with DSOP 1 or 2. The goal of the OLREORG function is to reduce the number of data blocks that are used to house the reference set.

However, if the OLREORG function determines both of the following, the OLREORG function with DSOP 4 or 5 allocates an empty block from the pool of empty blocks and moves all of the rows in the two least blocks of the reference set to the empty block.

- The least block rows cannot be distributed to the target blocks in the reference group.
- The reference set rows on the two least blocks could fit into an empty block.
This merging of the two least blocks into one empty block reduces the number of blocks housing the reference set by one block. If the newly acquired reference block still has space, the OLREORG function with DSOP 4 or 5 continues to move rows from the next least block to the newly acquired block until it is full.

Once the newly acquired block is full of rows from the least blocks, the OLREORG process reverts back to the process of moving the remaining least block rows to the most blocks. This secondary process could merge additional data blocks and reduce the number of blocks to house the reference set even further.

If all of the rows in this reference set are completely moved into the new block, and the new block still has available space, the native sequence index is used to locate the next sequentially occurring rows and to move them into this new block. Rows continue to be moved until the new block is full. Once this has been achieved, the OLREORG function uses the native sequence index to select the next reference group of blocks and continues the process.

The goal of this new OLREORG process is to use empty blocks when needed to reduce the number of data blocks housing the reference set. In cases where all data rows in the reference set can be moved to the new block, we see the extra benefit of a block that has been built with all the rows in sequence.

The process for determining the value to use for REFGROUP is still based on the user experience of determining how many buffers would be normally available for sequential processing.

However, one additional fact should be considered while making the choice of the REFGROUP value. The new OLREORG with DSOP 4 or 5 process will only allocate one empty block per reference group being processed. If you have significant space available, that is, empty blocks, in the data area and want the best possible reorganization, you should use a smaller REFGROUP size (2 or 4). Using a small REFGROUP size increases the possibility of the creation of the combined blocks with all rows in sequence.

If you have limited empty blocks, then you should consider using a higher REFGROUP size (8 or 12). While this higher REFGROUP value is not as efficient in creating blocks with all rows in sequence, it still provides significant improvement while using less empty blocks.
Using DATANE to Measure OLREORG in the Challenging Scenario

The following chart shows the effect of using different REFGROUP sizes against the “challenging” table covered in the previous chapter. The top half of the chart represents the REO table with various OLREORG REFGROUPs and DSOP 1. The bottom half shows the same table with OLREORG and DSOP 4.

The chart contains various items of information collected from the OLREORG report (blue), the Native Efficiency Report (red), and the CXX data area and index area reports (purple).

For this example, we load the data in a compressed form and then run an OLREORG function with a selected REFGROUP value. The test (LOAD and OLREORG functions) is repeated for each REFGROUP value to show the different improvements to the sequential efficiency.

This is only a demonstration sample, but you can see that in the bottom half of the chart where OLREORG with DSOP 4 was executed with the lower REFGROUP values, more rows are moved because of the availability of an empty block per reference group. We also see that the sequential efficiency of the data is higher with REFGROUP=2 where the most empty blocks were used.

Note: The OLREORG function’s purpose is to re-order data rows for better sequential access performance. While many times the moving of data rows allows some data blocks to be freed, there is no direct correlation or even guarantee that moving rows for better sequential performance reduces blocks in use. It is just a side benefit.
Determining the NUMBER Value for OLREORG with DSOP 4 or 5

From the previous discussions, we can see that the REFGROUP parameter specification is the main consideration for running the OLREORG function. However, the NUMBER parameter is provided as an optional parameter. The DBUTLTY guide defines the use of the NUMBER parameter input to OLREORG as follows:

This parameter can be specified as 1-99 or omitted. When you omit NUMBER=, all blocks qualify as source blocks and OLREORG moves as many rows as possible, so that as few blocks as possible contain the REFGROUP. Specifying a value for this parameter indicates you want to control which blocks qualify as move-from blocks. It can be used to limit row movement to those cases which provide the most benefit. For example, specifying NUMBER=2 indicates that only blocks with two or fewer rows may have their rows moved to other blocks in the REFGROUP.

OLREORG with DSOP 4 or 5 is specifically constructed to take an empty block, combine the data rows in Native Key sequence from the worst blocks in the reference set, and then, as available, add in additional rows. If, after the process is complete, the empty block is still not full, the Native Key is accessed to find the next set rows that should be added to the block.

Using the NUMBER parameter limits the effectiveness of this merging of the least row blocks and, in certain cases, can actually create an empty block with just a few rows from each reference group. This basically defeats the whole purpose of the OLREORG function.

Important! We strongly recommended that NUMBER= not be specified for an OLREORG statement with DSOP 4 or 5.

Determining the Need for Repeating an OLREORG with DSOP 4 or 5

You can execute multiple OLREORG processes against the same table. Typically, each of the succeeding OLREORG process moves less rows that the previous OLREORG process. At some point, the amount of rows moved and sequential performance gained is not cost effective when comparing the cost in CPU and I/O of the multiple OLREORG submissions to the improved efficiencies.

For OLREORG with DSOP 4 or 5, the need for repeating submissions to improve sequential order is less evident. This occurs because the use of empty blocks during each OLREORG run allows the utility to move more rows and create better sequenced data blocks.
While there is no technically reason that you cannot run a set of serially executed OLREORG processes against the same table, we recommend that you review the output of one OLREORG run before starting a second one for the same table.

Using the REO 1022 example, we executed a serious of 10 OLREORG processes against the same table. As you can see, each of the OLREORG runs had a diminishing effect. Since the entire table is passed for each of the OLREORG executions, it is probable that the cost per row moved goes up with each succeeding execution.

If you have available processing cycles and there is no concern over the resources consumed, you can consider running repeated OLREORG executions when past experience has shown this to be beneficial.

The following chart contains various items of information collected from the OLREORG report (blue), the Native Efficiency Report (red), and the CXX data area and index area reports (purple).

<table>
<thead>
<tr>
<th>DSOP #</th>
<th>DBD/Table</th>
<th>Rows Moved</th>
<th>Columns Freed/Used</th>
<th>Empty Blocks used</th>
<th>Blocks in use</th>
<th>MRB</th>
<th>AMRB</th>
<th>MRB w/2BFK</th>
<th>MRB w/4BFK</th>
<th>MRB w/8BFK</th>
<th>Empty Blocks Created</th>
<th>Index MRI Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>After Load</td>
<td>1022/REO</td>
<td>31,950</td>
<td></td>
<td>888</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>After OLREORG 1</td>
<td>31,950</td>
<td>31,950</td>
<td>11,057</td>
<td>2,088</td>
<td>378</td>
<td>1,229</td>
<td>27</td>
<td>37</td>
<td>30</td>
<td>30</td>
<td>27</td>
<td>54</td>
</tr>
<tr>
<td>After OLREORG 2</td>
<td>31,950</td>
<td>5,750</td>
<td>1,423</td>
<td>252</td>
<td>316</td>
<td>39</td>
<td>56</td>
<td>48</td>
<td>48</td>
<td>48</td>
<td>17</td>
<td>71</td>
</tr>
<tr>
<td>After OLREORG 3</td>
<td>31,950</td>
<td>1,950</td>
<td>541</td>
<td>101</td>
<td>1,294</td>
<td>45</td>
<td>64</td>
<td>60</td>
<td>60</td>
<td>60</td>
<td>60</td>
<td>124</td>
</tr>
<tr>
<td>After OLREORG 4</td>
<td>31,950</td>
<td>1,246</td>
<td>185</td>
<td>37</td>
<td>1,261</td>
<td>40</td>
<td>64</td>
<td>85</td>
<td>85</td>
<td>85</td>
<td>85</td>
<td>76</td>
</tr>
<tr>
<td>After OLREORG 5</td>
<td>31,950</td>
<td>896</td>
<td>63</td>
<td>12</td>
<td>1,252</td>
<td>46</td>
<td>64</td>
<td>86</td>
<td>86</td>
<td>86</td>
<td>86</td>
<td>21</td>
</tr>
<tr>
<td>After OLREORG 6</td>
<td>31,950</td>
<td>332</td>
<td>11</td>
<td>9</td>
<td>1,255</td>
<td>47</td>
<td>64</td>
<td>87</td>
<td>87</td>
<td>87</td>
<td>87</td>
<td>6</td>
</tr>
<tr>
<td>After OLREORG 7</td>
<td>31,950</td>
<td>182</td>
<td>12</td>
<td>3</td>
<td>1,258</td>
<td>47</td>
<td>64</td>
<td>87</td>
<td>87</td>
<td>87</td>
<td>87</td>
<td>5</td>
</tr>
<tr>
<td>After OLREORG 8</td>
<td>31,950</td>
<td>59</td>
<td>4</td>
<td>0</td>
<td>1,252</td>
<td>47</td>
<td>64</td>
<td>87</td>
<td>87</td>
<td>87</td>
<td>87</td>
<td>1</td>
</tr>
<tr>
<td>After OLREORG 9</td>
<td>31,950</td>
<td>21</td>
<td>1</td>
<td>0</td>
<td>1,251</td>
<td>47</td>
<td>64</td>
<td>87</td>
<td>87</td>
<td>87</td>
<td>87</td>
<td>1</td>
</tr>
<tr>
<td>After OLREORG 10</td>
<td>31,950</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

In this example, the number of rows moved and empty blocks used diminishes quickly. The best result is after the sixth OLREORG execution. However, the majority of the gain could have been achieved after just three executions. The sequential efficiency of this table is now about four times better with no buffers and five times better with just two buffers. Remember that this table was unable to be improved by the OLREORG function when running DSOP 1.

OLREORG executions 8 through 10 no longer used any empty blocks and had little effect on the overall sequential efficiency. To use an empty block, the OLREORG must be able to compress at least the two worst blocks of the reference group into one “new” empty block. What we see in this case is that we have compressed and re-ordered most data blocks to the point that they are all reasonable well ordered.
Since each OLREORG execution is moving rows to make the sequence better, it is possible that the same row could be moved multiple times in repeated OLREORG executions. This will be less evident when using an OLREORG function with DSOP 4 or 5, but could still happen.

Future plans for OLREORG include an option that would allow you to further reorganize this challenging table to improve efficiency to 100%.

A Good Table for OLREORG with DSOP 4 and 5

Continuing on with the same table that we demonstrated as a good working example in Chapter 5, we are now going to convert the example table into a DSOP 4 table and repeat the online reorganization process.

CONTROL CARD(S)
.........1.........2.........3.........4.........5.........6.........7.........8
CXXMAINT OPTION=ALTER, AREA=G03, DBID=4033, DSOP=4
FUNCTION=CXXMAINT
AREA=G03
DBID=04033
DSOP=4
OPTION=ALTER
*                                        REQUEST COMPLETE

After converting the data area to DSOP 4, we will rebuild the data space index and do a DEFRAG to get the index in the best possible shape.

CONTROL CARD(S)
.........1.........2.........3.........4.........5.........6.........7.........8
RETIX DBID=4033, AREA=G03, KEYNAME=*DATA
FUNCTION=RETIX
AREA=G03
DBID=04033
KEYNAME=*DATA
*                                        REQUEST COMPLETE
The DEFrag report indicates the two special internal indexes. The empty block index takes 3 index blocks and the moved record index (MRI) takes 497 blocks. The MRI is populated since this is a compressed table and rows can move as they expand and compress. OLREORG also uses this MRI index, so we want to measure its use.

<table>
<thead>
<tr>
<th>KEY ID</th>
<th>BLOCKS BEFORE</th>
<th>BLOCKS COMBINED</th>
<th>BLOCKS DELETED</th>
<th>BLOCKS AFTER</th>
<th>PERCENT DELETED</th>
<th>NOTES</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>9</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>71</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>72</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>73</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>81</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>91</td>
<td>3,132</td>
<td>0</td>
<td>0</td>
<td>3,132</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>92</td>
<td>299</td>
<td>0</td>
<td>0</td>
<td>299</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>111</td>
<td>5,590</td>
<td>0</td>
<td>0</td>
<td>5,590</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>39324</td>
<td>497</td>
<td>0</td>
<td>0</td>
<td>497</td>
<td>0.00</td>
<td>Moved Record</td>
</tr>
<tr>
<td>39327</td>
<td>15</td>
<td>12</td>
<td>12</td>
<td>3</td>
<td>80.00</td>
<td>Data Free Space</td>
</tr>
<tr>
<td>TOTALS</td>
<td>9,556</td>
<td>12</td>
<td>12</td>
<td>9,544</td>
<td>0.12</td>
<td></td>
</tr>
</tbody>
</table>
The following is the NER for this table before the OLREORG function with DSOP 4:

<table>
<thead>
<tr>
<th>GROUP</th>
<th>ROWS</th>
<th>BLOCKS</th>
<th>PBC</th>
<th>MRB</th>
<th>AVG</th>
<th>BLOCK EFF</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>12</th>
<th>16</th>
<th>20</th>
<th>24</th>
<th>28</th>
<th>32</th>
<th>48</th>
<th>64</th>
<th>96</th>
<th>128</th>
<th>256</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>50,000</td>
<td>1,135</td>
<td>139</td>
<td>1</td>
<td>44</td>
<td>48,529</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>50,000</td>
<td>829</td>
<td>277</td>
<td>1</td>
<td>60</td>
<td>30,864</td>
<td>21</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>50,000</td>
<td>752</td>
<td>277</td>
<td>1</td>
<td>66</td>
<td>28,381</td>
<td>28</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>50,000</td>
<td>638</td>
<td>229</td>
<td>1</td>
<td>78</td>
<td>24,024</td>
<td>24</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>50,000</td>
<td>546</td>
<td>277</td>
<td>1</td>
<td>92</td>
<td>22,734</td>
<td>33</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>50,000</td>
<td>307</td>
<td>277</td>
<td>2</td>
<td>163</td>
<td>7,469</td>
<td>59</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>50,000</td>
<td>286</td>
<td>179</td>
<td>1</td>
<td>69</td>
<td>6,289</td>
<td>65</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>50,000</td>
<td>364</td>
<td>277</td>
<td>2</td>
<td>137</td>
<td>12,620</td>
<td>50</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>50,000</td>
<td>402</td>
<td>277</td>
<td>2</td>
<td>124</td>
<td>24,485</td>
<td>58</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>50,000</td>
<td>313</td>
<td>277</td>
<td>1</td>
<td>160</td>
<td>12,817</td>
<td>51</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>50,000</td>
<td>352</td>
<td>277</td>
<td>2</td>
<td>142</td>
<td>20,635</td>
<td>40</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>50,000</td>
<td>447</td>
<td>277</td>
<td>1</td>
<td>112</td>
<td>31,653</td>
<td>36</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>50,000</td>
<td>518</td>
<td>139</td>
<td>2</td>
<td>97</td>
<td>47,061</td>
<td>34</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>50,000</td>
<td>516</td>
<td>139</td>
<td>1</td>
<td>97</td>
<td>23,876</td>
<td>36</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>50,000</td>
<td>413</td>
<td>139</td>
<td>3</td>
<td>121</td>
<td>428,43</td>
<td>42</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>50,000</td>
<td>379</td>
<td>139</td>
<td>9</td>
<td>132</td>
<td>6,172</td>
<td>43</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>50,000</td>
<td>424</td>
<td>139</td>
<td>1</td>
<td>118</td>
<td>49,159</td>
<td>42</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>50,000</td>
<td>395</td>
<td>139</td>
<td>1</td>
<td>127</td>
<td>35,067</td>
<td>43</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>28,454</td>
<td>241</td>
<td>139</td>
<td>1</td>
<td>118</td>
<td>9,749</td>
<td>43</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note that the table has 8,059 data blocks in use with an MRB efficiency rating of 0% and an adjusted MRB (AMRB) efficiency rating of 1% with 4 buffer efficiency only at 2%.

Following the NER, we will run our first OLREORG process with DSOP 4.

On a fully loaded system, the OLREORG process ran for 3 minutes elapsed time and was able to move over 386K rows to improve performance.
We now repeat the NER to get the new efficiency ratings.

<table>
<thead>
<tr>
<th>GROUP</th>
<th>ROWS</th>
<th>BLOCKS</th>
<th>MAX</th>
<th>MIN</th>
<th>AVG</th>
<th>BLOCK EFF</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>12</th>
<th>16</th>
<th>20</th>
<th>24</th>
<th>28</th>
<th>32</th>
<th>48</th>
<th>64</th>
<th>96</th>
<th>128</th>
<th>256</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>50,000</td>
<td>932</td>
<td>277</td>
<td>1</td>
<td>54</td>
<td>14,620</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>17</td>
<td>19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>50,000</td>
<td>473</td>
<td>277</td>
<td>1</td>
<td>106</td>
<td>7,413</td>
<td>2</td>
<td>5</td>
<td>6</td>
<td>13</td>
<td>37</td>
<td>38</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>50,000</td>
<td>317</td>
<td>277</td>
<td>2</td>
<td>158</td>
<td>4,117</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>55</td>
<td>57</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>50,000</td>
<td>397</td>
<td>277</td>
<td>1</td>
<td>126</td>
<td>7,365</td>
<td>2</td>
<td>10</td>
<td>11</td>
<td>24</td>
<td>35</td>
<td>39</td>
<td>42</td>
<td>44</td>
<td>45</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>50,000</td>
<td>383</td>
<td>277</td>
<td>1</td>
<td>131</td>
<td>7,490</td>
<td>2</td>
<td>39</td>
<td>47</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>50,000</td>
<td>246</td>
<td>277</td>
<td>2</td>
<td>201</td>
<td>3,454</td>
<td>5</td>
<td>58</td>
<td>73</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>50,000</td>
<td>234</td>
<td>277</td>
<td>1</td>
<td>214</td>
<td>3,248</td>
<td>5</td>
<td>63</td>
<td>78</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>50,000</td>
<td>307</td>
<td>277</td>
<td>1</td>
<td>163</td>
<td>5,441</td>
<td>3</td>
<td>45</td>
<td>57</td>
<td>58</td>
<td>59</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>50,000</td>
<td>269</td>
<td>277</td>
<td>1</td>
<td>186</td>
<td>8,548</td>
<td>2</td>
<td>57</td>
<td>67</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>50,000</td>
<td>250</td>
<td>277</td>
<td>2</td>
<td>200</td>
<td>5,447</td>
<td>3</td>
<td>17</td>
<td>72</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>50,000</td>
<td>273</td>
<td>277</td>
<td>5</td>
<td>183</td>
<td>8,831</td>
<td>2</td>
<td>28</td>
<td>64</td>
<td>65</td>
<td>66</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>50,000</td>
<td>354</td>
<td>277</td>
<td>1</td>
<td>141</td>
<td>7,952</td>
<td>2</td>
<td>3</td>
<td>9</td>
<td>47</td>
<td>50</td>
<td>51</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>50,000</td>
<td>364</td>
<td>277</td>
<td>2</td>
<td>137</td>
<td>15,761</td>
<td>1</td>
<td>8</td>
<td>19</td>
<td>49</td>
<td>51</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>50,000</td>
<td>359</td>
<td>277</td>
<td>1</td>
<td>139</td>
<td>6,718</td>
<td>2</td>
<td>4</td>
<td>7</td>
<td>17</td>
<td>48</td>
<td>50</td>
<td>52</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>50,000</td>
<td>260</td>
<td>277</td>
<td>1</td>
<td>192</td>
<td>422</td>
<td>4</td>
<td>50</td>
<td>70</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>50,000</td>
<td>273</td>
<td>277</td>
<td>2</td>
<td>183</td>
<td>3,927</td>
<td>4</td>
<td>51</td>
<td>66</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>50,000</td>
<td>319</td>
<td>277</td>
<td>1</td>
<td>157</td>
<td>16,870</td>
<td>1</td>
<td>55</td>
<td>56</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>50,000</td>
<td>262</td>
<td>277</td>
<td>1</td>
<td>191</td>
<td>14,454</td>
<td>1</td>
<td>65</td>
<td>69</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>28,454</td>
<td>149</td>
<td>277</td>
<td>1</td>
<td>191</td>
<td>3,808</td>
<td>2</td>
<td>62</td>
<td>70</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The table now has an MRB efficiency rating of 2%, an AMRB rating of 4% and a 4 buffer efficiency at 9%, which is approximately 4.5 times more efficient.

Important! Even though the OLREORG process used 162 empty blocks, it actually reduced the total number of blocks in use from 8,059 to 6,148. That is 1,911 less. The amount of blocks added or emptied will vary depending on the rows that can be moved and blocks that can be emptied.
Running Multiple OLREORGs to Continue to Improve Efficiency

We have decided to run two additional OLREORG processes to improve the table further. The elapsed time for this run was under 4 minutes. The movement statistics are in the following report. Note that the number of rows we are able to move continues to be significant. Each move improves efficiency.

| Rows processed | 928,454 | Rows moved | 142,518 | Blocks freed in reference group | 5,312 | Exclusive control conflicts | 0 | Empty blocks used | 145 | Block changes save by packing | 129 |

What we see in the NER is that, while the MRB efficiency only improved from 2% to 3%, the efficiency with 4 buffers for sequential processing has jumped dramatically from 9% to 60%, and with just 8 buffers the table would be at 66% efficiency.

At this point, you could decide to do no further OLREORG processes because the table’s sequential efficiency has now been improved enough to meet your needs.
Stacking OLREORGs in a Single DBUTLTY Execution

To demonstrate the continuing improvement that can be achieved, we have decided to run a DBUTLTY run with 6 more OLREORG requests. The requests are stacked in a single DBUTLTY execution. The run takes 12 minutes elapsed time.

<table>
<thead>
<tr>
<th>Base: 4033</th>
<th>******************************************</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rows processed -</td>
<td>928,454</td>
</tr>
<tr>
<td>Rows moved -</td>
<td>106,952</td>
</tr>
<tr>
<td>Blocks freed in reference group -</td>
<td>1,959</td>
</tr>
<tr>
<td>Exclusive control conflicts -</td>
<td>0</td>
</tr>
<tr>
<td>Empty blocks used -</td>
<td>167</td>
</tr>
<tr>
<td>Block changes save by packing -</td>
<td>65</td>
</tr>
<tr>
<td>Rows processed -</td>
<td>928,454</td>
</tr>
<tr>
<td>Rows moved -</td>
<td>76,102</td>
</tr>
<tr>
<td>Blocks freed in reference group -</td>
<td>1,175</td>
</tr>
<tr>
<td>Exclusive control conflicts -</td>
<td>0</td>
</tr>
<tr>
<td>Empty blocks used -</td>
<td>124</td>
</tr>
<tr>
<td>Block changes save by packing -</td>
<td>0</td>
</tr>
<tr>
<td>Rows processed -</td>
<td>928,454</td>
</tr>
<tr>
<td>Rows moved -</td>
<td>64,571</td>
</tr>
<tr>
<td>Blocks freed in reference group -</td>
<td>946</td>
</tr>
<tr>
<td>Exclusive control conflicts -</td>
<td>0</td>
</tr>
<tr>
<td>Empty blocks used -</td>
<td>124</td>
</tr>
<tr>
<td>Block changes save by packing -</td>
<td>9</td>
</tr>
<tr>
<td>Rows processed -</td>
<td>928,454</td>
</tr>
<tr>
<td>Rows moved -</td>
<td>59,906</td>
</tr>
<tr>
<td>Blocks freed in reference group -</td>
<td>846</td>
</tr>
<tr>
<td>Exclusive control conflicts -</td>
<td>0</td>
</tr>
<tr>
<td>Empty blocks used -</td>
<td>127</td>
</tr>
<tr>
<td>Block changes save by packing -</td>
<td>0</td>
</tr>
<tr>
<td>Rows processed -</td>
<td>928,454</td>
</tr>
<tr>
<td>Rows moved -</td>
<td>52,432</td>
</tr>
<tr>
<td>Blocks freed in reference group -</td>
<td>774</td>
</tr>
<tr>
<td>Exclusive control conflicts -</td>
<td>0</td>
</tr>
<tr>
<td>Empty blocks used -</td>
<td>95</td>
</tr>
<tr>
<td>Block changes save by packing -</td>
<td>0</td>
</tr>
<tr>
<td>Rows processed -</td>
<td>928,454</td>
</tr>
<tr>
<td>Rows moved -</td>
<td>46,402</td>
</tr>
<tr>
<td>Blocks freed in reference group -</td>
<td>724</td>
</tr>
<tr>
<td>Exclusive control conflicts -</td>
<td>0</td>
</tr>
<tr>
<td>Empty blocks used -</td>
<td>85</td>
</tr>
<tr>
<td>Block changes save by packing -</td>
<td>0</td>
</tr>
</tbody>
</table>
We then run the NER to see the efficiency after these six OLREORG runs. We see a slight improvement in the MRB efficiencies, but we see the 4 buffer efficiency improved to 78% and the Adjusted MRB efficiency with 4 buffers is now 99%.

<table>
<thead>
<tr>
<th>GROUP</th>
<th>ROWS</th>
<th>BLOCKS</th>
<th>MAX</th>
<th>MIN</th>
<th>AVG</th>
<th>BLOCK EFF</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>12</th>
<th>16</th>
<th>20</th>
<th>24</th>
<th>28</th>
<th>32</th>
<th>48</th>
<th>64</th>
<th>96</th>
<th>128</th>
<th>256</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>50,000</td>
<td>342</td>
<td>277</td>
<td>2</td>
<td>146</td>
<td>4,153</td>
<td>4</td>
<td>16</td>
<td>52</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>50,000</td>
<td>286</td>
<td>277</td>
<td>3</td>
<td>179</td>
<td>3,849</td>
<td>4</td>
<td>42</td>
<td>64</td>
<td>65</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>50,000</td>
<td>240</td>
<td>277</td>
<td>8</td>
<td>208</td>
<td>3,632</td>
<td>5</td>
<td>45</td>
<td>75</td>
<td>76</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>50,000</td>
<td>231</td>
<td>277</td>
<td>3</td>
<td>216</td>
<td>4,231</td>
<td>4</td>
<td>48</td>
<td>78</td>
<td>79</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>50,000</td>
<td>268</td>
<td>277</td>
<td>1</td>
<td>187</td>
<td>5,365</td>
<td>3</td>
<td>42</td>
<td>67</td>
<td>68</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>50,000</td>
<td>220</td>
<td>277</td>
<td>1</td>
<td>227</td>
<td>3,348</td>
<td>5</td>
<td>51</td>
<td>83</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>50,000</td>
<td>208</td>
<td>277</td>
<td>3</td>
<td>240</td>
<td>2,281</td>
<td>8</td>
<td>56</td>
<td>88</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>50,000</td>
<td>225</td>
<td>277</td>
<td>2</td>
<td>222</td>
<td>4,053</td>
<td>4</td>
<td>52</td>
<td>80</td>
<td>81</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>50,000</td>
<td>236</td>
<td>277</td>
<td>1</td>
<td>217</td>
<td>5,806</td>
<td>3</td>
<td>52</td>
<td>79</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>50,000</td>
<td>227</td>
<td>277</td>
<td>1</td>
<td>230</td>
<td>2,886</td>
<td>6</td>
<td>29</td>
<td>84</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>50,000</td>
<td>225</td>
<td>277</td>
<td>2</td>
<td>222</td>
<td>6,089</td>
<td>2</td>
<td>50</td>
<td>80</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>50,000</td>
<td>221</td>
<td>277</td>
<td>1</td>
<td>226</td>
<td>2,112</td>
<td>8</td>
<td>50</td>
<td>81</td>
<td>83</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>50,000</td>
<td>224</td>
<td>277</td>
<td>3</td>
<td>223</td>
<td>3,923</td>
<td>4</td>
<td>12</td>
<td>79</td>
<td>80</td>
<td>81</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>50,000</td>
<td>213</td>
<td>277</td>
<td>2</td>
<td>235</td>
<td>2,799</td>
<td>7</td>
<td>39</td>
<td>84</td>
<td>85</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>50,000</td>
<td>221</td>
<td>277</td>
<td>1</td>
<td>226</td>
<td>517</td>
<td>35</td>
<td>56</td>
<td>82</td>
<td>83</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>50,000</td>
<td>216</td>
<td>277</td>
<td>2</td>
<td>231</td>
<td>3,714</td>
<td>4</td>
<td>53</td>
<td>84</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>50,000</td>
<td>204</td>
<td>277</td>
<td>1</td>
<td>245</td>
<td>11,246</td>
<td>1</td>
<td>53</td>
<td>88</td>
<td>89</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>50,000</td>
<td>207</td>
<td>277</td>
<td>2</td>
<td>242</td>
<td>12,583</td>
<td>1</td>
<td>54</td>
<td>87</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>28,454</td>
<td>118</td>
<td>277</td>
<td>2</td>
<td>241</td>
<td>2,668</td>
<td>3</td>
<td>59</td>
<td>88</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

What is also important to note is that even though the six OLREORG runs allocated and used an additional 722 empty blocks, the row movements actually emptied 1,282 blocks, providing an overall reduction in blocks in-use by 560 blocks.

In the normal environment, we would not recommend the stacking of OLREORG functions without checking the sequential efficiency improvements. The improvement may have been achieved with just a few OLREORG runs.

REFGROUP Selections

As we stated, when using empty blocks and DSOP 4 and DSOP 5, you need to consider how many empty blocks are available for use. The lower the REFGROUP size, the higher the possibility of empty block use. The advantage of this is that when there is enough empty block space available, the OLREORG function has a better chance of creating well organized blocks, and it tends to free blocks back to the empty pool.
In the following example, we took the challenging table REO 1022 that did not perform well under OLREORG with DSOP 1 and changed it to DSOP 4. We then ran the OLREORG process four times with various REFGROUP settings.

<table>
<thead>
<tr>
<th>REFGROUP</th>
<th>ROWS</th>
<th>BLOCKS</th>
<th>MAX</th>
<th>MIN</th>
<th>AVG</th>
<th>BLOCK EFF</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>12</th>
<th>16</th>
<th>24</th>
<th>32</th>
<th>48</th>
<th>64</th>
<th>128</th>
<th>256</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRB</td>
<td>31,950</td>
<td>888</td>
<td>36</td>
<td>18</td>
<td>36</td>
<td>6,931</td>
<td>13</td>
<td>14</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AMRB</td>
<td>31,950</td>
<td>888</td>
<td>36</td>
<td>18</td>
<td>36</td>
<td>6,931</td>
<td>13</td>
<td>14</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REFGROUP=2 x4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MRB</td>
<td>31,950</td>
<td>1,261</td>
<td>36</td>
<td>1</td>
<td>25</td>
<td>1,965</td>
<td>46</td>
<td>65</td>
<td>66</td>
<td>68</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AMRB</td>
<td>31,950</td>
<td>1,261</td>
<td>36</td>
<td>1</td>
<td>25</td>
<td>1,965</td>
<td>46</td>
<td>65</td>
<td>66</td>
<td>68</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REFGROUP=4 x4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MRB</td>
<td>31,950</td>
<td>1,036</td>
<td>36</td>
<td>1</td>
<td>31</td>
<td>2,493</td>
<td>36</td>
<td>68</td>
<td>83</td>
<td>85</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AMRB</td>
<td>31,950</td>
<td>1,036</td>
<td>36</td>
<td>1</td>
<td>31</td>
<td>2,493</td>
<td>36</td>
<td>68</td>
<td>83</td>
<td>85</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REFGROUP=8 x4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MRB</td>
<td>31,950</td>
<td>965</td>
<td>36</td>
<td>1</td>
<td>33</td>
<td>4,889</td>
<td>18</td>
<td>30</td>
<td>58</td>
<td>69</td>
<td>70</td>
<td>71</td>
<td>72</td>
<td>79</td>
<td>74</td>
<td>75</td>
<td>76</td>
</tr>
<tr>
<td>AMRB</td>
<td>31,950</td>
<td>965</td>
<td>36</td>
<td>1</td>
<td>33</td>
<td>4,889</td>
<td>18</td>
<td>30</td>
<td>58</td>
<td>69</td>
<td>70</td>
<td>71</td>
<td>72</td>
<td>79</td>
<td>74</td>
<td>75</td>
<td>76</td>
</tr>
<tr>
<td>REFGROUP=12x4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MRB</td>
<td>31,950</td>
<td>956</td>
<td>36</td>
<td>2</td>
<td>33</td>
<td>6,214</td>
<td>14</td>
<td>20</td>
<td>36</td>
<td>57</td>
<td>59</td>
<td>60</td>
<td>61</td>
<td>62</td>
<td>63</td>
<td>64</td>
<td>70</td>
</tr>
<tr>
<td>AMRB</td>
<td>31,950</td>
<td>956</td>
<td>36</td>
<td>2</td>
<td>33</td>
<td>6,214</td>
<td>14</td>
<td>20</td>
<td>36</td>
<td>57</td>
<td>59</td>
<td>60</td>
<td>61</td>
<td>62</td>
<td>63</td>
<td>64</td>
<td>70</td>
</tr>
</tbody>
</table>

As we can see, the OLREORG processes with DSOP 4 greatly improved the sequential efficiency. The REFGROUP=2 setting used the most additional blocks, but it also got the best improvement. The REFGROUP=12 setting used the least empty blocks but its improvement was also less.
REFGROUP Selections

We repeat the same test with the MET 4033 “good” table. Remember that MET already has free space mixed in its data blocks, so the need to allocate empty blocks will be less.

| REFGROUP | GROUP | ROWS | BLOCKS | MAX | MIN | AVG | BLOCK EFF | 2 | 4 | 8 | 12 | 16 | 20 | 24 | 28 | 32 | 48 | 64 | 96 | 128 | 256 |
|----------|-------|------|--------|-----|-----|-----|-----------|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|          | READ  | 928,454 | 8,059 | 1 | 115 | 441,404 | 0 | 1 | 2 | 8 | 36 |
|          | AMRB  | 928,454 | 8,059 | 1 | 115 | 441,404 | 1 | 2 | 5 | 6 | 20 | 86 | 87 |
| x4       |       |       |       |   |     |       |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |
|          | REFGROUP=2 | 928,454 | 6,253 | 1 | 148 | 19,491 | 17 | 30 | 45 | 49 | 50 | 52 | 53 |
|          | AMRB  | 928,454 | 6,253 | 1 | 148 | 19,491 | 32 | 56 | 84 | 92 | 93 | 94 | 97 | 98 |
| x4       |       |       |       |   |     |       |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |
|          | REFGROUP=4 | 928,454 | 4,490 | 1 | 207 | 90,812 | 3 | 37 | 72 | 73 | 74 |
|          | AMRB  | 928,454 | 4,490 | 1 | 207 | 90,812 | 4 | 49 | 96 | 98 | 99 |
| x4       |       |       |       |   |     |       |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |
|          | REFGROUP=8 | 928,454 | 4,035 | 1 | 230 | 217,598 | 1 | 8 | 58 | 82 | 83 |
|          | AMRB  | 928,454 | 4,035 | 1 | 230 | 217,598 | 1 | 9 | 70 | 99 |
| x4       |       |       |       |   |     |       |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |
|          | REFGROUP=12 | 928,454 | 3,924 | 1 | 237 | 258,424 | 1 | 4 | 37 | 79 | 85 |
|          | AMRB  | 928,454 | 3,924 | 1 | 237 | 258,424 | 1 | 5 | 44 | 92 | 99 |

In this example, we see that each REFGROUP used empty blocks, but enough data blocks were emptied during the process that there was a “gain” in empty blocks after the OLREORG processes completed.

What is particular to note is that while REFGROUP=2 still provided the best MRB and AMRB basic efficiency, the larger REFGROUP settings actually created better efficiencies with buffers 8 and 12. This is a processing occurrence that is due to the way data blocks are processed within the reference group.

Unfortunately, there is not a hard rule for a REFGROUP value. We know that REFGROUP=2 will be the best choice when attempting to create basic efficiency, but depending on data block content, that is the amount of free space, the higher REFGROUP values may provide better efficiencies when buffers are available.

You should take some time to experiment with your data tables and locate the best REFGROUP choices.
Space Consumed in the Index for the MRI

As we discussed, the MRI (Moved Record Index) must keep track of any rows that are moved. However, if a row is moved by an OLREORG execution and then moved again by another OLREORG execution, the MRI entry is updated. Therefore, we will only have one MRI entry per row that has been moved.

<table>
<thead>
<tr>
<th>KEY ID</th>
<th>BLOCKS BEFORE</th>
<th>BLOCKS COMBINED</th>
<th>BLOCKS DELETED</th>
<th>BLOCKS AFTER</th>
<th>PERCENT DELETED</th>
<th>NOTES</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>9</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>71</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>72</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>81</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>91</td>
<td>3,132</td>
<td>0</td>
<td>0</td>
<td>3,132</td>
<td>0.00</td>
<td>INDEX FREE SPACE</td>
</tr>
<tr>
<td>92</td>
<td>299</td>
<td>0</td>
<td>0</td>
<td>299</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>111</td>
<td>5,590</td>
<td>0</td>
<td>0</td>
<td>5,590</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>39321</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.00</td>
<td>MOVED RECORD</td>
</tr>
<tr>
<td>39324</td>
<td>3,040</td>
<td>878</td>
<td>878</td>
<td>2,162</td>
<td>28.88</td>
<td>DATA FREE SPACE</td>
</tr>
<tr>
<td>39327</td>
<td>16</td>
<td>2</td>
<td>2</td>
<td>14</td>
<td>12.50</td>
<td></td>
</tr>
<tr>
<td>TOTALS</td>
<td>12,101</td>
<td>880</td>
<td>880</td>
<td>11,221</td>
<td>7.27</td>
<td></td>
</tr>
</tbody>
</table>

To see the overall affect on the index from the four OLREORG processes with REFGROUP=2 on the MET 4033 table executed in this section, we executed a DBUTLTY DEFRAG function before and after the OLREORG executions. The four OLREORG executions moved 772,519 rows.

We started with 497 blocks in the MRI. After the OLREORG processes, we had 2,167 blocks. So the growth for these four OLREORG processes resulted in 1,605 additional index blocks, or about 160 tracks of space. This amounted to about 28% of the index space used for the indexes on the 4033 database. While this is space that is taken up in the index, it does not affect the overall index processing. The MRI is only used during a restart or a forward recovery. With the ability of CA Datacom/DB Version 12.0 to separate the index into multiple data sets by key ID, a site with a large MRI could use the multi-data set index as a way to separate the MRI from the user key IDs.

The MRI is reset back to zero index blocks after a DBUTLTY LOAD function for the data area or database.
Recommend DEFRAG of Index Before and After OLREORG

As we can see in the previous examples, the OLREORG will affect the MRI index. Since the MRI index can get fragmented as rows are moved, it makes sense to run an index DEFRAG after the OLREORG processes, especially if a lot of rows were moved.

In general, since the DEFRAG function usually takes only a small amount of time to run and the DEFRAG and OLREORG processing would normally be scheduled for light processing periods, it is beneficial to do a DEFRAG execution before the OLREORG processing to improve the processing speed and after the OLREORG processing to recompress the MRI index.

OLREORG with DSOP 4 or 5 Does Not Trigger a Dynamic Extend

The OLREORG function with DSOP 4 or 5 has been designed so that it cannot trigger a dynamic extend. If the OLREORG processing tries to allocate an empty block and none are available, it switches back to the original OLREORG process and continues on just moving rows between blocks of the reference group to improve performance. If, as the OLREORG processing continues, a data block is emptied and put in the empty block list, the following REFGROUP processes are able to utilize the empty block as needed.

If user processing is also accessing the table and adding rows and it causes the data area to fill, the OLREORG process pauses just like all other processing to allow the dynamic extend to complete, and then continues on.

Once empty data blocks become available again, the OLREORG process with DSOP 4 or 5 switches back to the mode where empty blocks can be used.

OLREORG with DSOP 4 or 5 and Blocks In-Use Change

As we have documented, the OLREORG with DSOP 4 or 5 allows the process to use empty data blocks. The OLREORG report shows the amount of empty blocks used as the last line of the DBUTLTY OLREORG report.

In some cases, the amount of empty blocks used stated by OLREORG will correlate to the increase in the amount of BLOCKS IN-USE in the CXX report before and after the OLREORG execution.
In certain others cases, the number of empty blocks used by the OLREORG process is actually offset by the blocks that are emptied during the row moving process so that BLOCKS IN-USE actually stays the same or in some cases, may actually be less than before it started.

Regardless of the BLOCKS IN-USE change, the number of data rows remains consistent. Therefore, the ability of the data area to hold new data rows has not changed.

**Suppressing the DSOP 4 and DSOP 5 Empty Block Usage**

Typically a user will only change a data area from DSOP 1 to DSOP 4 or from DSOP 2 to DSOP 5 in order to activate the DBUTLTY OLREORG function’s ability to use empty blocks during its online reorganization process.

If for some reason, you want to suppress the empty block usage by an OLREORG after changing that data area’s space management option to DSOP 4 or DSOP 5, you can do one of the following:

- Use CXXMAINT to permanently change the DSOP option back to DSOP 1 or DSOP 2 which will disable the empty block index and stop the use of empty blocks is OLREORG.

- Use a new keyword parameter for DBUTLTY’s OLREORG of DSOP=1 (when the data area has been changed to DSOP 4) or DSOP=2 (when the data area has been changed to DSOP 5). Adding this keyword to the OLREORG execution will tell the DBIUTLTY program to suppress the use of empty blocks for this execution of OLREORG function. Specifying DSOP=4 and DSOP=5 as the keyword parameter is permitted for consistency, but it will require the data area to be at DSOP 4 or at DSOP 5.

**Chapter Summary**

The CA Datacom/DB DBUTLTY OLREORG function with DSOP 4 or 5 adds a major enhancement for users.

The use of empty blocks allows you to perform online reorganization on tables that may not have been suitable for the OLREORG function in the past. In addition, we continued to show the value of the NER in validated the native sequential efficiency of the table before and after the reorganization.
Users of CA Datacom/DB Version 12.0 have to use the CXXMAINT function of DBUTLTY to activate the OLREORG function with empty block use by changing the DSOP option from 1 to 4 or from 2 to 5. DSOP options 0 and 3 are not supported for OLREORG processing with empty blocks.

**Note:** In CA Datacom/DB Version 12.0, if the CXXMAINT function has been used to change the DSOP option to 4 or 5, then the utility function must be repeated if the database definition is catalogued to the CXX using the Datadictionary CATALOG function.
Chapter 7: New Data Area Space Usage Report

There is a new DBUTLTY function that is delivered with PTFs for DBUTLTY in CA Datacom/DB Version 12.0. This function processes the data area using a high speed I/O process and provides a space map of the data area. The space map provides detailed insight into the data block space usage and available free space.

The DBUTLTY functionality is called the Data Area Space Report and is generated by executing the DBUTLTY REPORT function with the TYPE=DATASP parameter.

Data Area Space Report (DATASP)

The main function of the Data Area Space Report (DATASP) is to build a simple report that shows the detailed space usage in a selected data area. The DATASP runs outside of the MUF and does not require that the MUF be available. The DATASP report uses a highly efficient I/O process that has no effect on the MUF processing. The DATASP function reads the data area in place and does not see activity currently occurring in the MUF that has not yet been posted to the data area. As such, the DATASP report should be executed when there is a minimal amount of update activity against the selected data area.

DATASP and Single-Table Data Areas (Non-Compressed)

The DATASP function provides the most concise space usage and space availability information for tables that are not compressed and reside in single-table data areas. For these types of tables, the utility can easily determine the following:

- The space in use in the block and active (not deleted) rows in the block
- Available free space in the block
- Available space from deleted rows
- The number of data rows that could be added to the block
The consistency of these numbers makes the calculation of the detail level lines generated in the SPACEMAP report very accurate.

For more information on the utility to generate the DATASP report, see Implementing and Using DBUTLTY REPORT=DATASP (see page 153).

DATASP and Single-Table Areas (Compressed)

The DATASP function depends on its ability to calculate the average row size, the total length, on each data block to generate accurate detail level lines found in the DATASP report.

If the amount of compression found in data rows causes the actual number of data rows to be stored on data blocks rows to vary widely and differ from the calculated average row size, the calculation of available rows that could be added will not be as concise.

If the amount of compression is reasonably consistent and the calculated value for average row size is close to what each of the compressed data rows actually is, then the detail lines will be reasonably effective measurement of available space to add more rows.

Using the New DBUTLTY TYPE=DATASP Utility Function

Since we have new situations where we are using OLREORG processes that are moving rows around and may use empty blocks to improve data row ordering, we can now see more cases where the percentage of data blocks in-use increases. In these cases, a block in-use can be only partially full and can have room to add more new rows.

Since many clients base the decision to manually extend the data area on the blocks in-use percentage, we need another way for users to determine the true fullness of those in-use data blocks. In short, we need a way to determine how many additional rows of average size could be added to the data area before all blocks in the data area are full.

To assist you in determining the amount of data row space that is available in the data area, the new DBUTLTY TYPE=DATASP function has been provided.
Measuring Data Row Available Space in a Data Area Before and After an OLREORG with DSOP 4/5

To demonstrate how the Data Area Space Report (DATASP) measurement works, we start with the challenging table scenario covered in the OLREORG with DSOP 4 or 5 in Chapter 6.

We start by loading the data area with a physical backup that has an unorganized data row order.

The CXX report shows the details about the area and table:

Data area REO (of database 1022)

- 1800 total blocks
- 890 blocks in use
- 910 blocks unused
- 49 percent full (really means 49% blocks in use)
- Dynamic extend is turned off for this test
- DSOP MODE - RANDOM FOR OLREORG (DSOP 4)
Table REO is in area REO

- Row length 112 bytes (with RCE)
- 31,950 rows in the data area

The Native Efficiency Report shows that the data table has poor efficiency of 13%. We also notice that the row count is 31,950, which matches the CXX report.

| GROUP | ROWS | BLOCKS | MAX | MIN | AVG | BLOCK EFF | 2 | 4 | 8 | 12 | 16 | 20 | 24 | 28 | 32 | 48 | 64 | 96 | 128 | 256 |
|-------|------|--------|-----|-----|-----|-----------|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|
| READ  | 1,900| 365    | 26  | 1   | 10  | 720       | 17|19 |
|       | 2,300| 367    | 30  | 1   | 9   | 857       | 13|15 |
|       | 3,300| 448    | 36  | 1   | 8   | 724       | 18|
|       | 4,300| 442    | 36  | 1   | 8   | 677       | 19|
|       | 5,300| 711    | 16  | 1   | 5   | 872       | 12|14 |
|       | 6,300| 622    | 24  | 1   | 6   | 867       | 12|
|       | 7,300| 417    | 28  | 1   | 7   | 780       | 14|
|       | 8,300| 542    | 34  | 1   | 6   | 833       | 13|
|       | 9,300| 557    | 33  | 1   | 6   | 681       | 14|

The DATASP report shows that the data area has 890 blocks in use and 910 blocks empty. The report shows us that there are 31,950 data rows, and that there is estimated room for another 32,778 rows.

<table>
<thead>
<tr>
<th>AREA</th>
<th>BUSIZE</th>
<th>TRACKS</th>
<th>BLOCKS</th>
<th>RECORD</th>
<th>1,800 UR</th>
<th>YES</th>
<th>DSOP</th>
<th>RANDOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>REO</td>
<td>4,096</td>
<td>150</td>
<td>1,800</td>
<td>31,950</td>
<td>31,950</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The DATA SP report shows that the data area has 890 blocks in use and 910 blocks empty. The report shows us that there are 31,950 data rows, and that there is estimated room for another 32,778 rows.
Measuring Data Row Available Space in a Data Area Before and After an OLREORG with DSOP 4/5

Now we run an OLREORG with DSOP 4. We see that the OLREORG used, at most, 378 free pace blocks.

<table>
<thead>
<tr>
<th>Table</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rows processed</td>
<td>31,950</td>
</tr>
<tr>
<td>Rows moved</td>
<td>11,057</td>
</tr>
<tr>
<td>Blocks freed in reference group</td>
<td>2,888</td>
</tr>
<tr>
<td>Empty blocks used</td>
<td>378</td>
</tr>
<tr>
<td>Block changes save by packing</td>
<td>1,805</td>
</tr>
</tbody>
</table>

The CXX report shows that we now have 1241 blocks in use which is 251 more blocks than before the OLREORG and the data area now has 68% (it was 49%) blocks full. The reason that the blocks in-use is not 378 more (empty blocks used by OLREORG function) is that the actual OLREORG process emptied some existing in-use blocks (127 blocks).
A new Data Area Space Report (DATASP) now shows that, while there are more blocks in use, the number of rows in the area has remained the same (31,950) and the number of rows that could be added (available) remained the same (32778).

To verify this, we executed an add program that adds rows until the data area is full. The program has a very simple display showing the rows being added. As you can see, the program was able to add 32778 rows before failing on an insert because of a data area full condition.

**Note:** This area full condition occurred because we purposely set this data area to not allow dynamic extents so that the test could demonstrate the ability of the DATASP report.

The DATASP report will be accurate for data areas that have a single table that is not compressed. For multi-table and compressed data areas, the rows available figure will be an estimate based on average row sizes per block and the free space in each block.
Chapter Summary

The new Data Area Space Report (DATASP) provides the additional information needed to allow users to make informed decisions about which database areas need to be extended.

While the DATASP report can be used while the data area is open to the MUF for processing we recommend that you run the report when there is low maintenance (MUF) activity to get the most accurate statistics.

When you begin using the new OLREORG function with DSOP 4 or 5, you have the ability to determine data row available space with the DATASP report.

Existing CA Datacom/DB users that use compressed tables or even multi-tables areas will also be able to benefit from the use of the DATASP report.
Chapter 8: DBUTLTY OLREORG and the Moved Record Index

Whether you are using the OLREORG function with DSOP 1 or DSOP 2 or the OLREORG function with DSOP 4 or DSOP 5, the database must keep track of moved records.

The Moved Records Index is Built in the IXA Area

The moved records index (MRI), also known as the URI index, is used when a program tries to locate a row that has been moved. This can occur in normal day-to-day operations if you are updating data tables that are compressed with either CA Datacom/DB or with user compression.

The MRI is also used by the OLREORG function. When the OLREORG function moves a row, it must record that moved row information in the moved record index. This information is required so that, if a recovery operation is attempted using a physical backup that was created before the OLREORG execution, all forward processing using the recovery files could be completed. The MRI is also used by some RAAT and SAAT commands. For example, a CBS command can cause a temporary index to be built for the set of rows that meet a set of criteria. The temporary index has the identity of each row using the URI and its current DASD placement location. If the OLREORG function moves the row, the use of this temporary index will fail to find the row at the before location and uses the MRI to find its current location.

The MRI entries created by the OLREORG function are not needed or referenced by normal user processing. Therefore, while it is an overhead in index space, it is not an overhead for normal user processing.

If the amount of space used by the MRI becomes a concern, typically only for very large tables with very active OLREORG processing, you should consider implementing the multi-data set index capabilities of CA Datacom/DB Version 12.0 for this database. For more information on multi-data set indexes, see the CA Datacom/DB Database and System Administration Guide.

In the chapter on the OLREORG with DSOP 4 or 5, we saw an example of the MRI index growth. The CA Datacom/DB Version 12.0 multi-data set index support allows you to segregate the overhead indexes in the IXA area from those indexes you create by allowing you to move the user indexes to the IO1, IO2, and so forth index data sets.

This would be an option for the extreme case where the MRI size became an issue.
A Recommendation for All OLREORG Users

All OLREORG users should run DBUTLTY index defragmentation (DEFRAG) function following any OLREORG process. This is true whether the process is the OLREORG function with DSOP 1 or 2 or the OLREORG function with DSOP 4 or 5. Anything that is going to update the MRI index can cause index fragmentation that the DEFRAG function will clean up.

As mentioned previously, for tables that have a lot of rows, running the index DEFRAG before the OLREORG process can also help to make the OLREORG run faster.

Chapter 6 provided a sample of a DEFRAG execution that was run after the multiple OLREORG functions with DSOP 4.

The DEFRAG not only recovers index space, it also gives us a simple way to track the amount of index space used for the MRI.

Chapter Summary

The moved record index (MRI) is an internal processing requirement of CA Datacom/DB and is not something the user can regulate. CA Datacom/DB creates MRI pointers when existing data rows are moved to support recovery processes. The MRI entries are maintained until the index has been rebuilt using the DBUTLTY LOAD or INIT/RETIIX functions.

In this chapter, we reviewed the use of the (MRI) entries in the OLREORG process and recommended best practices for keeping the index space compressed using the DBUTLTY DEFRAG function.
Chapter 9: Utilizing Multiple Data Block Sizes

The various table examples shown throughout this book used two different block sizes.

- The REO table is in the data area REO 1022 that uses a 4096 blocksize.
- The MET table is in the G03 4033 data area that uses a much larger blocksize of 27,998.

Two Data Buffer Sizes – Two Data Buffer Pools

The CA Datacom/DB Version 12.0 MUF supports the use of two different data buffer sizes. These buffer sizes are assigned in two data buffer pools on the MUF startup option DATAPool.

```
DATAPool dataln,datano,dataln2,datano2
```

The size of the first data buffer pool (dataln) must be smaller than the length of the second data buffer pool (dataln2).

When a data area is opened for processing in the MUF, the data area’s data blocks use the DATALN buffer pool if it is large enough, otherwise if the dataln2 buffer pool is available, the area will be processed using the dataln2 buffer pool. If the data block is larger than the largest available buffer size, the data area open will fail.

In the case noted, we have allocated our MUF DATAPool parameters as follows

```
DATAPool 8K, 8000, 28K, 1000
```

The first data buffer pool (8K) is used to handle the REO1022 data area, while the second buffer pool (28K) is used to handle the MET4033 data area. Using multiple buffer pools and different data block sizes is a common performance tuning process for CA Datacom/DB site.
Typically, using larger data buffer sizes benefits sequential processing by the Native Key. With larger buffer sizes, it takes a smaller number of I/Os to read through the physical data blocks and there is a better chance that the next data row in sequence will be on the same block as the current data row. With today’s DASD hardware, the difference in the time it takes to read an 8K block versus a 4K block is miniscule.

The main concern with a larger data block size is having the memory to support the larger buffers. In sites that use only a single data buffer pool, it is relatively easy to add a second larger pool to support the data areas that are enlarged to a new block size. However, if a site is already using two buffer pool sizes, some study is needed to determine which data areas should be enlarged and to what data block size.

In our example, the REO data area is currently set at 4K and the G03 data area for the MET table is set to 27,998. Since the large buffer pool size is already set to 28K to handle the G03 area, it would not make a lot of difference to enlarge the G03 (MET table) data area. In fact, if we make the G03 area any larger than 27,998, we will have a significant negative effect on DASD usage. When we look at the REO data area, we see that it is 4K and our small data pool size is 8K. We could change the REO data block size to 8K which would not affect the buffer processing but may improve sequential processing.

Reblocking REO 1022 from 4096 to 8192

To reblock a data area, the following must be done in sequence:

1. Close the data area or database
2. Execute a backup
3. Use Datadictionary to change the data area blocksize and recatalog it to the CXX
4. Initialize the data area using DBUTLTY
5. Reload the backup
As expected the data area now has 50% of the data blocks in-use compared to the 4096 block size.
It is interesting to note that, when the data was loaded to the larger 8K blocksize, the sequential efficiency actually dropped by a few percent. The 8K DATANE report has the 4K Full Table information added to the bottom.

<table>
<thead>
<tr>
<th>4096 GROUP</th>
<th>ROWS</th>
<th>BLOCKS</th>
<th>MAX</th>
<th>MIN</th>
<th>AVG</th>
<th>BLOCK EFF</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>12</th>
<th>16</th>
<th>20</th>
<th>24</th>
<th>28</th>
<th>32</th>
<th>48</th>
<th>64</th>
<th>96</th>
<th>128</th>
<th>256</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRB</td>
<td>31,950</td>
<td>888</td>
<td>36</td>
<td>18</td>
<td>36</td>
<td>6,931</td>
<td>13</td>
<td>14</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AMRB</td>
<td>31,950</td>
<td>888</td>
<td>36</td>
<td>18</td>
<td>36</td>
<td>6,931</td>
<td>12</td>
<td></td>
<td></td>
<td>13</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>REFGROUP=2 x4</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRB</td>
</tr>
<tr>
<td>AMRB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>REFGROUP=4 x4</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRB</td>
</tr>
<tr>
<td>AMRB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>REFGROUP=8 x4</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRB</td>
</tr>
<tr>
<td>AMRB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>REFGROUP=12 x4</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRB</td>
</tr>
<tr>
<td>AMRB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>8192 GROUP</th>
<th>ROWS</th>
<th>BLOCKS</th>
<th>MAX</th>
<th>MIN</th>
<th>AVG</th>
<th>BLOCK EFF</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>12</th>
<th>16</th>
<th>20</th>
<th>24</th>
<th>28</th>
<th>32</th>
<th>48</th>
<th>64</th>
<th>96</th>
<th>128</th>
<th>256</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRB</td>
<td>31,950</td>
<td>438</td>
<td>73</td>
<td>49</td>
<td>73</td>
<td>4,131</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AMRB</td>
<td>31,950</td>
<td>438</td>
<td>73</td>
<td>49</td>
<td>73</td>
<td>4,131</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>REFGROUP=2 x4</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRB</td>
</tr>
<tr>
<td>AMRB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>REFGROUP=4 x4</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRB</td>
</tr>
<tr>
<td>AMRB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>REFGROUP=8 x4</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRB</td>
</tr>
<tr>
<td>AMRB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>REFGROUP=12 x4</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRB</td>
</tr>
<tr>
<td>AMRB</td>
</tr>
</tbody>
</table>

The same holds true when we do the OLRERGx4 runs with the various REFGROUP sizes. The larger blocksize does not help or hurt the sequential access efficiency percentages.

So why would moving to a larger blocksize help sequential processing? Because it is not just the efficiency percentage, it is also the number of I/Os required to process the same set of rows in sequential order. If the entire table is being retrieved, the 8K blocksize is able to retrieve the 31,950 rows with less I/Os than the 4K blocksize.
In the figure below, the REO table has been reset back to the same starting point for three different executions, each with a different data area blocksize. In each case, the data row content and sequential efficiency was relatively the same (as shown above).

A series of sequential processing programs (using the native sequence key) were executed and the logical and physical accesses to the table’s data area were captured.

As we can see, the physical I/O was significantly lower for the larger blocks sizes even though the same number of sequential reads (logical requests) was processed in each case.

<table>
<thead>
<tr>
<th>BLOCKSIZE</th>
<th>DBID</th>
<th>AREA_NAME</th>
<th>PHYSICAL_READS</th>
<th>PHYSICAL_WRITES</th>
<th>LOGICAL_READS</th>
<th>LOGICAL_WRITES</th>
</tr>
</thead>
<tbody>
<tr>
<td>4096</td>
<td>1022</td>
<td>REO</td>
<td>8158</td>
<td>0</td>
<td>223666</td>
<td>0</td>
</tr>
<tr>
<td>8192</td>
<td>1022</td>
<td>REO</td>
<td>4947</td>
<td>0</td>
<td>223666</td>
<td>0</td>
</tr>
<tr>
<td>27998</td>
<td>1022</td>
<td>REO</td>
<td>1929</td>
<td>0</td>
<td>223666</td>
<td>0</td>
</tr>
</tbody>
</table>

While changing the data area blocksize can have a dramatic effect on the sequential processing physical I/O which will reduce the cost of doing sequential processing, the user must balance the sequential access savings against the requirements for larger data buffer sizes.

In the example above, we already had our DATALN buffer pool at 8K. The change from 4K to 8K would not have significantly affected the processing in MUF or the usage of the DATALN buffer pool.

When we changed the blocksize to a size larger than 8K, we moved the processing of this data area (REO 1022) from the DATALN buffer pool to the DATALN2 buffer pool. This change could possibly affect the overall usage of the DATALN and DATALN2 buffer pools. The user should monitor both buffer pools before and after the change to make sure performance of the buffer pools have not been adversely affected.

**Chapter Summary**

Changing a data area’s blocksize does not significantly affect the sequential access efficiency of the data area. However, when you consider that in today’s architecture, an 8K I/O takes about the same resource as a 4K I/O, and having a table that can be read sequentially from start to end with less I/Os could definitely improve performance.

The user needs to balance the improvement in sequential processing with any additional requirements for memory to manage the larger buffer pool sizes.
Chapter 10: Table Partitioning and Data Row Reorganizations

Most CA Datacom/DB tables reside in a single data area which means that all the rows for the table are stored together and are reorganized using the Native Key. In some cases, the number of data rows in very large data tables has raised concerns regarding:

- The number and size of the DASD extents needed to house the single data area (data set)
- The time it takes to backup the data area
- The time required for offline reorganizations
- The time it takes to recreate the data area in disaster recovery situations.

For these cases, CA Datacom/DB table partitioning provides the ability for the single data table’s rows to be stored on multiple data areas. Up to 239 data areas are permitted. The determination deciding which data area will house a specific data row is based on the data row’s partition key value and the data area’s partition key range.

For information on the implementation and use of table partitioning, see the CA Datacom/DB Database and System Administration Guide and the CA Datacom/DB Datadictionary User Guide.

The Partition Key

The partition key is defined using the Datadictionary facilities. The partition key is build by selecting one or more columns within the data row. You can select the high-order portion of a column when necessary.

For example, a user decides to partition the CATTLE_RANCH (CRT) data table, which holds one row for each type of cattle raised on each ranch. The user decides to partition that data table based on the region where the ranch is located. The user knows that each RANCH_ID is assigned a REGION_NO according to the current mapping of the US. The first digit of the region number represents the section of the country, where 1 is North, 2 is South, 3 is East, and 4 is West. In this case, the user could partition the table by selecting the high-order portion of the REGION_NO, the first digit, as the partition key value. In this example, the user would need four data areas to hold the four partitions of the data.
Data Area Partition Key Range

For each data area, you specify the partition key range. Since CA Datacom/DB does not allow the possibility that a data row could be created with a value outside of the assigned partition ranges, the four data areas would be created with the following partition key ranges:

- Low-values to 1 = North (CRN data area)
- 2 = South (CRS data area)
- 3 = East (CRE data area)
- 4 to high-values = West (CRW data area)

With the partitioned table concept, the full table, known as the parent, would still logically exist so that application programs will be able to continue to function unchanged. However, each of the member data areas would have a physical table, known as a child, that represents the rows in that data area. The physical child tables together make up the logical parent table.

For ease of reference in this example, the parent is CATTLE_RANCH (CRT) and the child tables are the following:

- CATTLE_RANCH_NORTH (CRN) stored in the CRN data area
- CATTLE_RANCH_SOUTH (CRS) stored in the CRS data area
- CATTLE_RANCH_EAST (CRE) stored in the CRE data area
- CATTLE_RANCH_WEST (CRW) stored in the CRW data area

Offline Data Row Reorganization

If the partition key is the high-order portion of the Native Key, then the concepts for data row reorganization remain relatively the same as we have discussed in the earlier chapters. The offline database reorganization (DBUTLTY BACKUP and LOAD functions) and the offline area reorganization (DBUTLTY BACKUP and LOAD or REORG functions) continue to work as documented. The only real difference is that, with a partitioned table, there are more data areas involved in the reorganization.
The following chart shows the data rows as they would be placed in the data areas after a database reorganization when the partition key is the high-order part of the Native Key.

<table>
<thead>
<tr>
<th>AREA</th>
<th>CHILD</th>
<th>REGION_NO</th>
<th>RANCH_ID</th>
<th>Rest of data</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRN</td>
<td>CRN</td>
<td>1001</td>
<td>CASTW01</td>
<td>abcdedefghi</td>
</tr>
<tr>
<td>CRN</td>
<td>CRN</td>
<td>1001</td>
<td>DENGA03</td>
<td>abcdedefghi</td>
</tr>
<tr>
<td>CRN</td>
<td>CRN</td>
<td>1003</td>
<td>DENGA01</td>
<td>abcdedefghi</td>
</tr>
<tr>
<td>CRN</td>
<td>CRN</td>
<td>1003</td>
<td>SMITH19</td>
<td>abcdedefghi</td>
</tr>
<tr>
<td>CRN</td>
<td>CRN</td>
<td>1009</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CRS</td>
<td>CRS</td>
<td>2001</td>
<td>ABBOT01</td>
<td>abcdedefghi</td>
</tr>
<tr>
<td>CRS</td>
<td>CRS</td>
<td>2001</td>
<td>CARSW01</td>
<td>abcdedefghi</td>
</tr>
<tr>
<td>CRS</td>
<td>CRS</td>
<td>2007</td>
<td>JONES01</td>
<td>abcdedefghi</td>
</tr>
<tr>
<td>CRS</td>
<td>CRS</td>
<td>2027</td>
<td></td>
<td>abcdedefghi</td>
</tr>
<tr>
<td>CRS</td>
<td>CRS</td>
<td>2027</td>
<td>SMITH01</td>
<td>abcdedefghi</td>
</tr>
<tr>
<td>CRS</td>
<td>CRS</td>
<td>2027</td>
<td>SMITH96</td>
<td>abcdedefghi</td>
</tr>
<tr>
<td>CRE</td>
<td>CRE</td>
<td>3003</td>
<td>BAIRD06</td>
<td>abcdedefghi</td>
</tr>
<tr>
<td>CRE</td>
<td>CRE</td>
<td>3003</td>
<td>FINLE01</td>
<td>abcdedefghi</td>
</tr>
<tr>
<td>CRE</td>
<td>CRE</td>
<td>3058</td>
<td>ALLEN01</td>
<td>abcdedefghi</td>
</tr>
<tr>
<td>CRE</td>
<td>CRE</td>
<td>3058</td>
<td>FREED01</td>
<td>abcdedefghi</td>
</tr>
<tr>
<td>CRW</td>
<td>CRW</td>
<td>4010</td>
<td>GARDA03</td>
<td>abcdedefghi</td>
</tr>
<tr>
<td>CRW</td>
<td>CRW</td>
<td>4012</td>
<td>GARDA01</td>
<td>abcdedefghi</td>
</tr>
<tr>
<td>CRW</td>
<td>CRW</td>
<td>4012</td>
<td>SMITH07</td>
<td>abcdedefghi</td>
</tr>
</tbody>
</table>

Data Row Placement

With partition tables, the concepts for data row placement, that is, adding data rows, remain relatively the same as described in the earlier chapters of this book. The main difference is that with partitioned tables, each data area has its own free space pool consisting of deleted space and empty blocks that is used to place new rows. The data row must be placed in the data area that has a partition key range that matches the partition key value of the row. Data rows cannot be placed in the “wrong” data area. If you allow the partition key value to be changed on an existing row and that value does not match the existing data area’s partition key range, then the row is actually deleted from the current data area and added to the matching data area.
In the following chart, we can see that some rows were deleted and new rows were added, depending on the partition value and the available deleted space. The new data rows were added according to where space was available in the data area.

<table>
<thead>
<tr>
<th>AREA</th>
<th>NAME</th>
<th>CHILD</th>
<th>REGION_NO</th>
<th>RANCH_ID</th>
<th>Rest of data</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRN</td>
<td>CRN</td>
<td>1001</td>
<td>CASTW01</td>
<td>abcdefghi ...</td>
<td></td>
</tr>
<tr>
<td>CRN</td>
<td>CRN</td>
<td>1001</td>
<td>DENG03</td>
<td>abcdefghi ...</td>
<td></td>
</tr>
<tr>
<td>CRN</td>
<td>CRN</td>
<td>1003</td>
<td>DENG01</td>
<td>abcdefghi ...</td>
<td></td>
</tr>
<tr>
<td>CRN</td>
<td>CRN</td>
<td>1009</td>
<td>SMITH02</td>
<td>abcdefghi ...</td>
<td></td>
</tr>
<tr>
<td>CRN</td>
<td>CRN</td>
<td>1006</td>
<td>ABBOTT02</td>
<td>new ranch ...</td>
<td></td>
</tr>
<tr>
<td>CRN</td>
<td>CRN</td>
<td>1006</td>
<td>GARDA04</td>
<td>new ranch ...</td>
<td></td>
</tr>
<tr>
<td>CRS</td>
<td>CRS</td>
<td>2001</td>
<td>ABBOTT01</td>
<td>abcdefghi ...</td>
<td></td>
</tr>
<tr>
<td>CRS</td>
<td>CRS</td>
<td>2001</td>
<td>CARSW01</td>
<td>abcdefghi ...</td>
<td></td>
</tr>
<tr>
<td>CRS</td>
<td>CRS</td>
<td>2113</td>
<td>SALLY01</td>
<td>new ranch</td>
<td></td>
</tr>
<tr>
<td>CRS</td>
<td>CRS</td>
<td>2027</td>
<td>SMITH01</td>
<td>abcdefghi ...</td>
<td></td>
</tr>
<tr>
<td>CRS</td>
<td>CRS</td>
<td>2027</td>
<td>SMITH06</td>
<td>abcdefghi ...</td>
<td></td>
</tr>
<tr>
<td>CRE</td>
<td>CRE</td>
<td>3003</td>
<td>BAIRD06</td>
<td>abcdefghi ...</td>
<td></td>
</tr>
<tr>
<td>CRE</td>
<td>CRE</td>
<td>3003</td>
<td>FINLE01</td>
<td>abcdefghi ...</td>
<td></td>
</tr>
<tr>
<td>CRE</td>
<td>CRE</td>
<td>3058</td>
<td>ALLEN01</td>
<td>abcdefghi ...</td>
<td></td>
</tr>
<tr>
<td>CRE</td>
<td>CRE</td>
<td>3058</td>
<td>FREED01</td>
<td>abcdefghi ...</td>
<td></td>
</tr>
<tr>
<td>CRW</td>
<td>CRW</td>
<td>4010</td>
<td>GARDA03</td>
<td>abcdefghi ...</td>
<td></td>
</tr>
<tr>
<td>CRW</td>
<td>CRW</td>
<td>4012</td>
<td>GARDA01</td>
<td>abcdefghi ...</td>
<td></td>
</tr>
<tr>
<td>CRW</td>
<td>CRW</td>
<td>4012</td>
<td>SMITH07</td>
<td>abcdefghi ...</td>
<td></td>
</tr>
</tbody>
</table>

Sequential Data Row Access for the Parent Table

If the partition key is the high-order portion of the Native Key, then the concepts for sequential access remain relatively the same as we have discussed in the earlier chapters. There are some slight changes in the way data rows are placed in RANDOM (DSOP 1, 4) and WRAP (DSOP 2, 5) and this could slightly affect the sequential access performance, but it should not be significant.

If the partition key is not the high-order portion of the Native Key, then the concepts of sequential data access become seriously jeopardized.
For example, suppose we use the same CATTLE_RANCH table with the same partition key (first byte of REGION_NO), but the Native Key is the RANCH_ID. After an offline reorganization, the data row placement would be as seen as the following:

<table>
<thead>
<tr>
<th>AREA NAME</th>
<th>CHILD TABLE</th>
<th>REGION_NO</th>
<th>RANCH_ID</th>
<th>Rest of data</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRN</td>
<td>CRN</td>
<td>1006</td>
<td>ABBOT02</td>
<td>NEW RANCH</td>
</tr>
<tr>
<td>CRN</td>
<td>CRN</td>
<td>1001</td>
<td>CASTH01</td>
<td>ABCDEFGHI</td>
</tr>
<tr>
<td>CRN</td>
<td>CRN</td>
<td>1003</td>
<td>DENGA01</td>
<td>ABCDEFGHI</td>
</tr>
<tr>
<td>CRN</td>
<td>CRN</td>
<td>1001</td>
<td>DENGA03</td>
<td>ABCDEFGHI</td>
</tr>
<tr>
<td>CRN</td>
<td>CRN</td>
<td>1006</td>
<td>GARDA04</td>
<td>NEW RANCH</td>
</tr>
<tr>
<td>CRN</td>
<td>CRN</td>
<td>1009</td>
<td>SMITH02</td>
<td>ABCDEFGHI</td>
</tr>
<tr>
<td>CRN</td>
<td>CRN</td>
<td>1003</td>
<td>SMITH09</td>
<td>ABCDEFGHI</td>
</tr>
<tr>
<td>CRS</td>
<td>CRS</td>
<td>2001</td>
<td>ABBOT01</td>
<td>ABCDEFGHI</td>
</tr>
<tr>
<td>CRS</td>
<td>CRS</td>
<td>2001</td>
<td>CARSW01</td>
<td>ABCDEFGHI</td>
</tr>
<tr>
<td>CRS</td>
<td>CRS</td>
<td>2113</td>
<td>SALLY01</td>
<td>NEW RANCH</td>
</tr>
<tr>
<td>CRS</td>
<td>CRS</td>
<td>2027</td>
<td>SMITH01</td>
<td>ABCDEFGHI</td>
</tr>
<tr>
<td>CRS</td>
<td>CRS</td>
<td>2027</td>
<td>SMITH01</td>
<td>ABCDEFGHI</td>
</tr>
<tr>
<td>CRE</td>
<td>CRE</td>
<td>3058</td>
<td>ALLEN01</td>
<td>ABCDEFGHI</td>
</tr>
<tr>
<td>CRE</td>
<td>CRE</td>
<td>3003</td>
<td>BAIRD06</td>
<td>ABCDEFGHI</td>
</tr>
<tr>
<td>CRE</td>
<td>CRE</td>
<td>3003</td>
<td>FINLE01</td>
<td>ABCDEFGHI</td>
</tr>
<tr>
<td>CRE</td>
<td>CRE</td>
<td>3058</td>
<td>FREED01</td>
<td>ABCDEFGHI</td>
</tr>
<tr>
<td>CRW</td>
<td>CRW</td>
<td>4012</td>
<td>GARDA04</td>
<td>ABCDEFGHI</td>
</tr>
<tr>
<td>CRW</td>
<td>CRW</td>
<td>4010</td>
<td>GARDA03</td>
<td>ABCDEFGHI</td>
</tr>
<tr>
<td>CRW</td>
<td>CRW</td>
<td>4012</td>
<td>SMITH07</td>
<td>ABCDEFGHI</td>
</tr>
</tbody>
</table>

The number at the far right in the chart shows the sequential data access sequence. As we can see from this very simple example, having a partition key that is not the same as the high-order portion of the Native Key, can seriously affect the sequential order of the data rows which in turn can seriously reduce the sequential access efficiency and performance.

Online Data Row Reorganization

The DBUTLTY online reorganization function (OLREORG) can be executed only against a single data area at a time. Therefore OLREORG is only permitted against the physical child tables. In the example, OLREORG would be executed against the CRN, CRS, CRE, and CRW tables. Any attempt to execute OLREORG against the logical parent table CRT will fail with a CA Datacom/DB return code.

The processing that OLREORG does at the physical child level is focused on improving the sequential order of the data rows within the data area where the child table is housed. OLREORG does not attempt to adjust the sequence of any data rows outside of the data area.

One advantage is that users with partitioned tables can run multiple OLREORGs in parallel, one per child table, with no adverse effects.
Native Efficiency Report (DATANE)

Like the DBUTLTY OLREORG function, the Native Efficiency Report (NER), DBUTLTY REPORT TYPE=DATANE, can be executed only on a single data area at a time. In the example, the NER would be executed on the CRN, CRS, CRE and CRW tables. Any attempt to execute the NER on the logical parent table (CRT) will fail with a CA Datacom/DB return code.

Running the NER on a child table provides insight into the sequential efficiency of the rows stored in that child table. This information can be used to determine if the data area containing the child table should be reorganized.

However, any implementation that specifies a partition key that is not the same as the high-order portion of the Native Key can seriously affect the sequential efficiency when processing the logical parent table.

Chapter Summary

Special consideration must be paid to partitioned tables with regards to data access efficiency and data row reorganization techniques. This is particularly evident in cases where the partition key is not the same as the high-order portion of the Native (sequence) Key.

Sites should be very careful when considering the partition key criteria. The amount of sequential access and the key that is used to do that access needs to be part of the partition key consideration.
Chapter 11: Data Reorganization

Green Book Summary

This Green Book has been written to provide a how to or cookbook approach to understanding how data row organization affects sequential processing by the Native Key in the CA Datacom/DB environment. It addresses the various utilities provided by CA Datacom/DB to perform a data row reorganization and presents each of the processes strengths and weaknesses.

The book introduces new functionality scheduled for delivery in CA Datacom/DB Version 14.0 that is also delivered to CA Datacom/DB Version 12.0 sites with PTFs.

It is the purpose of this Green Book to assist the CA Datacom/DB user in creating a step by step process to perform the following:

■ Determine when a data row reorganization is needed using a combination of existing and newly delivered product functions, History Database, Accounting Facility, and the Native Efficiency Report (NER).

■ Determine the best process for performing the data row reorganization such as the following:
  
  – Offline reorganization
    
    ▪ Database reorganization, a backup in native sequence followed by index initialization and load
    
    ▪ Data area reorganization, a backup in native sequence of the data area followed by an optional index initialization and load of the data area
    
    ▪ Parallel data area reorganization, a REORG followed by an optional index defragmentation
  
  – Partial offline reorganization
    
    ▪ Data area reorganization while rest of database remains available
    
    ▪ Parallel data area reorganization while rest of database remains available
  
  – Online reorganization
    
    ▪ OLREORG within existing data blocks using DSOP 1 or DSOP 2
    
    ▪ OLREORG with empty blocks using DSOP 4 and DSOP 5
Chapter Summary

- Determine the effectiveness of the process using the new DBUTLTY TYPE=DATANE report (NER).
- Determine if additional reorganization activities are needed.
- Repeat the process as needed.
- Implement a consistent data reorganization management practice with the following:
  - Eliminating unneeded data reorganizations
  - Save CPU and I/O
- Increases data availability
- Prevent potential sequential performance problems by performing the following:
  - Monitor sequential processing effectiveness of critical processes.
  - Schedule on-demand data row reorganizations to improve sequential efficiencies.
  - Successfully use the new online reorganization functionality to improve data availability and 24x7 access.
Appendix A: Implementing and Using DBUTLTY REPORT=DATANE

This appendix provides the basic operating instructions for the new DBUTLTY REPORT=DATANE function. The REPORT=DATANE function is only available in CA Datacom/DB Version 12.0 after applying the PTFs that add the functionality.

REPORT TYPE=DATANE (Native Sequence Efficiency Report)

A DBUTLTY DATANE report provides a detailed native sequence efficiency report for the selected data table. The DATANE report provides detailed insight into the physical order of the data rows in comparison to the order of the Native (sequence) Key.

The information provided by the report is one of the key factors when considering whether a data table is a good candidate for reorganization. The CA Datacom/DB Data Reorganization Green Book provides a detailed best-practice process that includes these key information points:

- Native Efficiency Report (DATANE)
- Sequential accesses by the Native Key
- Sequential accesses by the non-Native Key
- Sequential accesses as a portion of the total table accesses.

The DATANE report uses a highly efficient index level process that has minimal effect on MUF processing. However, the DATANE function reads through the index for the table using one key and can do some limited data row retrievals. The DATANE report must be executed while the MUF is active and the database table is available at the least for read processing.

The DATANE report can process information while the table is open for update by other users. Update tasks in the MUF that add, delete, or move data rows affect the accuracy of the DATANE report. We recommend that the DATANE report is executed when there is a minimal amount of update activity against the selected data table. The DATANE report cannot be executed against a table that is locked for single-user processes such as a DBUTLTY LOAD.

The DATANE report builds a simple report that shows the detailed data row sequential efficiency for a selected data table. The report can help you determine whether to reorganize the table.
In most cases, run the DATANE using the Native Key. You can also use a key other than the Native Key. This ability allows you to compare the data row efficiency with various keys. In some cases, you want to see if a different key would better match the data row order. In other cases, you have several keys that are similar to the Native Key and you want to measure the efficiency of those keys.

Before the DATANE report, most users relied on various sources of information to determine whether to reorganize the table. The sources were the number of overflows in the data area, batch job run times, and so on. In most cases, this information was limited in its effectiveness.

**Restrictions**

The MUF must be active and operational and the data table must be available at least for read processing.

The table selected for the report must be a valid CA Datacom/DB data table. The table cannot be open by a single user function such as a DBUTLTY LOAD function.

The data area must be loaded with URI=YES.

**Security Setting:** The DBUTLTY external resource name for the DATANE report is REPORT.DATANE.

**When to Use**

Use the DATANE report when you want to determine the sequential efficiency of the data table when compared to the Native (sequence) Key. The DATANE report also provides detailed information about the current data row sizes for compressed tables. The report also provides information about the rows and data blocks in use.

**How to Use**

To obtain a Native Efficiency report, execute the REPORT function with the following command format:

```
REPORT TYPE=DATANE,DBID=dbid,TABLE=table[,KEYNAME=keyname] [,NUMBER=nnnn][,RECORDS=nnnnnnnn]
```

**REPORT**

Invokes the report function.
TYPE=

(Required) Requests the Native Efficiency Report.

Valid Entries:

DATANE

Default Value:

(No default)

, DBID=

(Required) Specifies the name of the DBID that you want to use.

Valid Entries:

a valid DBID

Default Value:

(No default)

, TABLE=

(Required) Identifies the table on which you want a report.

Valid Entries:

3-character DATACOM-NAME of the table in the database specified

Default Value:

(No default)
How to Use

,KEYNAME=

(Optional) Specifies the key name to use to process the report. The data row sequential efficiency is compared to the order provided by this index.

If a value is not specified, the report uses the Native Key specified for the table.

Valid Entries:

5-character DATACOM-NAME of the key in the table specified

Default Value:

The Native Key of the selected table

,NUMBER=

(Optional) Specifies the number of data blocks that are read using physical processing (at the beginning of the report execution) to determine the maximum rows per block (MRB). The NUMBER parameter is only honored for data tables that are using compression. For non-compressed tables, the MRB is calculated using the row size and data block size.

Note: Specifying a low value limits the ability to calculate a valid MRB for this table. When the NUMBER parameter is not specified and the data table is compressed, the maximum value (9999 data blocks) is scanned to provide the MRB calculation.

Valid Entries:

0 to 9999

Default Value:

9999 (but only used if table is compressed)
How to Use CA Datacom®/DB Data Reorganization

,RECORDS=

(Optional) Specifies the rows to process (and include) in a printed group detail line.

If a value is not specified, no detail lines are produced and only the summary lines for the full table are produced.

**Note:** Specifying a low number for the RECORDS= parameter with a table with many rows generates many detail lines (one per record group). If the number of groups generated exceeds 99,999, the report allows the group number to roll over to zero and begin again. This rollover typically means that over 2000 pages of detail group lines are generated. We recommend that you specify a RECORDS= value that keeps the number of detail groups to 1000 groups or less.

**Valid Entries:**

0 to 99999999

**Default Value:**

(No default)

**Example JCL: Summary DATANE Report ("RECORDS=" not specified)**

The following example shows the command to obtain a DATANE summary report when the optional parameters KEYNAME, NUMBER, and RECORDS are not specified. These omissions tell the utility to produce a report using the Native Key and only produce the summary lines for the full table.

**Note:** Use the following as a guide to prepare your JCL. The JCL statements are examples only. Lowercase letters in a statement indicate a value you must supply. Code all statements to your site and installation standards.

```
//jobname   Per site standard             Job card
//          EXEC PGM=DBUTLTY,REGION=2M    Execute DBUTLTY
//STEPLIB   Per site standard             CA Datacom Execution Lib
//SYSIN     DD *                          Command Input
REPORT TYPE=DATANE,TABLE=REO,DBID=1022
/*
Sample Report: Summary DATANE Report ("RECORDS=" not specified)

The first page of the report shows the following:

- The command being processed, exactly as entered.
- An analysis of keywords encountered and expected. Any errors found are flagged with a note in the left margin.
- Any messages related to syntax processing.

The second page of the report provides the following information about the table that is being processed:

<table>
<thead>
<tr>
<th>TBL AREA KEY</th>
<th>CMP</th>
<th>USER COMPRESSION</th>
<th>RECLN</th>
<th>RECORDS</th>
<th>BLKSIZE</th>
<th>TRACKS</th>
<th>TOT BLKS</th>
<th>INUSE BLKS</th>
<th>PART BLKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>REO</td>
<td>REO</td>
<td>REOK1 N</td>
<td>112</td>
<td>31,950</td>
<td>4,096</td>
<td>150</td>
<td>1,800</td>
<td>890</td>
<td>1</td>
</tr>
</tbody>
</table>

CALC MAX ROWS/BLOCK 36

Top Heading and Lines

Displays the heading for the columns containing information describing the selected data table.

CALC MAX ROWS/BLOCK

Displays the maximum row per block (MRB) which was calculated by dividing the expanded row size plus RCE (record control element) into the data block size.

The third page of the summary report provides the following efficiency ratings for the FULL TABLE:
PBC in the top heading

Displays the PERFECT BLOCK CHANGE (PBC) count of 887. The PBC is the number of block changes that would occur when a table was loaded with all rows in sequential order and all blocks had the maximum number of rows per block. The next lines are the headings for the detail line area of the report. Because RECORDS was not specified, no detail lines were generated.

FULL TABLE section

Displays the summary section of the report. Two lines are created for the table.

The MRB line provides the sequential efficiency rating of this table compared to an ideal or perfect situation where each block is loaded to the maximum (rows) and all the rows are in sequential order.

The AMRB (Adjusted MRB) line provides the sequential efficiency rating of this table using only the sequential order of the rows as the ideal. This rating does not penalize the efficiency rating for available free space that can be in the data blocks.

Note: Having large free space amounts could substantially reduce sequential performance by limiting the rows returned in a data block.

31,950

Indicates the number of rows in the table.

888

Indicates the number of data blocks that contained at least one data row for this table. The value varies slightly from the BLOCKS INUSE figure on page 2. The difference is data blocks that are used as data area control blocks.

36 18 36

Indicates the high-level mark (MAX. ROWS) and the low-level mark (MIN. ROWS) of each block having a row for this table. The report also divides the total rows of the table by the blocks in-use to calculate the average rows (AVG. ROWS).
Indicates the BLOCK CHANGES that occur when the data rows are processed using the selected key sequence. A BLOCK CHANGE is counted each time the next row (in the sequence given by the index chosen with the KEYNAME=) is not within the same block as the current row.

The BLOCK CHANGES is the key factor in measuring sequential efficiency. When a table is in sequential order, all data rows are processed in a given data block before moving on to the next block. When rows are not in order, the sequential process bounces back and forth between data blocks to retrieve all data rows in the selected key sequence. The bouncing back and forth adds overhead and delays to any process that is reading the data in sequential order.

Indicates the MRB efficiency which is calculated by taking the ideal situation where all data blocks are only processed once as data rows are read sequentially (PBC 887) and dividing that by the actual number of block changes (6,931) that would have occurred. In other words, it is the efficiency of reading the data with a single buffer.

Displays the calculated MRB efficiency with buffers. Because buffering can significantly reduce the physical I/O used when a previously read data block is already in storage (a buffer), the MRB efficiency percentage is calculated using two buffers, four buffers, and so on.

A table that shows a high MRB efficiency percent (more than 80 percent) with buffers of 8 or less still provides a reasonable good sequential processing. There is processing overhead to find data blocks in the buffers, but it is significantly less than having to reread the data blocks from the DASD.

Some sites can have higher buffer availability for batch processing. For these sites, additional efficiency ratings are generated for 12 buffers, 16 buffers, and so on.

By monitoring the amount of sequential activity and the sequential efficiency of a table, you are able to make more informed decisions on when to do a data reorganization.

**Note:** The FULL TABLE or summary lines look at the table as a whole and their statistical calculations are based on the whole table.
Example JCL: Detail DATANE Report ("RECORDS=01000" specified)

The following shows the command to obtain a DATANE detail report when the RECORDS= parameter contains a value. In this detail report, we are requesting a detailed efficiency line for every 1000 records. By generating detail lines, you are able to determine how the efficiency ratings vary within the table. The variations can occur when a data table has been reorganized into sequential order, but follow-up processing has deleted and added rows in a random fashion in one or more areas of the table.

The online reorganization process allows you to limit the data rows that are being reorganized (FIRSTKEY and LASTKEY). A DATANE detail report with a reasonable number of detail lines can assist you in determining if such a limited OLREORG is helpful.

Note: Use the following as a guide to prepare your JCL. The JCL statements are examples only. Lowercase letters in a statement indicate a value you must supply. Code all statements to your site and installation standards.

```
//jobname   Per site standard             Job card
//          EXEC PGM=DBUTLTY,REGION=2M    Execute DBUTLTY
//STEPLIB   Per site standard             CA Datacom Execution Lib
//SYSIN     DD *                          Command Input
REPORT TYPE=DATANE,TABLE=REO,DBID=1022,RECORDS=01000
/*
```

Sample Report: Detail DATANE Report ("RECORDS=01000" specified)

The first and second pages of the report are described earlier in this section. The only difference in this report is that includes the RECORDS=1000 specification.
The third and following pages of the detail report provide the efficiency ratings for each group of records. At the end of the report, the FULL TABLE summary lines are generated.

<table>
<thead>
<tr>
<th>GROUP</th>
<th>ROWS</th>
<th>BLOCKS</th>
<th>MAX R</th>
<th>AVG</th>
<th>BLOCK EFF</th>
<th>CHANGES</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1,000</td>
<td>122</td>
<td>16</td>
<td></td>
<td>122</td>
<td>23</td>
</tr>
<tr>
<td>2</td>
<td>1,000</td>
<td>178</td>
<td>16</td>
<td></td>
<td>177</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>1,000</td>
<td>266</td>
<td>17</td>
<td>4</td>
<td>272</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>1,000</td>
<td>256</td>
<td>19</td>
<td>4</td>
<td>261</td>
<td>11</td>
</tr>
<tr>
<td>5</td>
<td>1,000</td>
<td>293</td>
<td>14</td>
<td>3</td>
<td>292</td>
<td>9</td>
</tr>
<tr>
<td>6</td>
<td>1,000</td>
<td>193</td>
<td>26</td>
<td>1</td>
<td>192</td>
<td>15</td>
</tr>
<tr>
<td>7</td>
<td>1,000</td>
<td>185</td>
<td>13</td>
<td></td>
<td>184</td>
<td>15</td>
</tr>
<tr>
<td>8</td>
<td>1,000</td>
<td>194</td>
<td>36</td>
<td>1</td>
<td>193</td>
<td>15</td>
</tr>
<tr>
<td>9</td>
<td>1,000</td>
<td>138</td>
<td>36</td>
<td>1</td>
<td>137</td>
<td>21</td>
</tr>
<tr>
<td>10</td>
<td>1,000</td>
<td>241</td>
<td>15</td>
<td></td>
<td>236</td>
<td>11</td>
</tr>
<tr>
<td>11</td>
<td>1,000</td>
<td>209</td>
<td>16</td>
<td>5</td>
<td>202</td>
<td>11</td>
</tr>
<tr>
<td>12</td>
<td>1,000</td>
<td>264</td>
<td>10</td>
<td>1</td>
<td>263</td>
<td>14</td>
</tr>
<tr>
<td>13</td>
<td>1,000</td>
<td>234</td>
<td>36</td>
<td>1</td>
<td>233</td>
<td>19</td>
</tr>
<tr>
<td>14</td>
<td>1,000</td>
<td>47</td>
<td>36</td>
<td>4</td>
<td>47</td>
<td>9</td>
</tr>
<tr>
<td>15</td>
<td>1,000</td>
<td>267</td>
<td>19</td>
<td>4</td>
<td>267</td>
<td>9</td>
</tr>
<tr>
<td>16</td>
<td>1,000</td>
<td>221</td>
<td>15</td>
<td></td>
<td>221</td>
<td>11</td>
</tr>
<tr>
<td>17</td>
<td>1,000</td>
<td>304</td>
<td>14</td>
<td>1</td>
<td>304</td>
<td>9</td>
</tr>
<tr>
<td>18</td>
<td>1,000</td>
<td>190</td>
<td>11</td>
<td>5</td>
<td>189</td>
<td>15</td>
</tr>
<tr>
<td>19</td>
<td>1,000</td>
<td>299</td>
<td>8</td>
<td>1</td>
<td>299</td>
<td>9</td>
</tr>
<tr>
<td>20</td>
<td>1,000</td>
<td>226</td>
<td>16</td>
<td>4</td>
<td>223</td>
<td>12</td>
</tr>
<tr>
<td>21</td>
<td>1,000</td>
<td>165</td>
<td>15</td>
<td>6</td>
<td>174</td>
<td>16</td>
</tr>
<tr>
<td>22</td>
<td>1,000</td>
<td>240</td>
<td>25</td>
<td>1</td>
<td>239</td>
<td>10</td>
</tr>
<tr>
<td>23</td>
<td>1,000</td>
<td>188</td>
<td>12</td>
<td>5</td>
<td>187</td>
<td>15</td>
</tr>
<tr>
<td>24</td>
<td>1,000</td>
<td>243</td>
<td>20</td>
<td>1</td>
<td>245</td>
<td>11</td>
</tr>
<tr>
<td>25</td>
<td>1,000</td>
<td>232</td>
<td>17</td>
<td>4</td>
<td>240</td>
<td>12</td>
</tr>
<tr>
<td>26</td>
<td>1,000</td>
<td>230</td>
<td>34</td>
<td>1</td>
<td>231</td>
<td>11</td>
</tr>
<tr>
<td>27</td>
<td>1,000</td>
<td>231</td>
<td>13</td>
<td>4</td>
<td>231</td>
<td>12</td>
</tr>
<tr>
<td>28</td>
<td>1,000</td>
<td>137</td>
<td>24</td>
<td>1</td>
<td>178</td>
<td>16</td>
</tr>
<tr>
<td>29</td>
<td>1,000</td>
<td>154</td>
<td>16</td>
<td>6</td>
<td>159</td>
<td>18</td>
</tr>
<tr>
<td>30</td>
<td>1,000</td>
<td>262</td>
<td>13</td>
<td>1</td>
<td>268</td>
<td>10</td>
</tr>
<tr>
<td>31</td>
<td>950</td>
<td>122</td>
<td>32</td>
<td>1</td>
<td>122</td>
<td>22</td>
</tr>
<tr>
<td>32</td>
<td>950</td>
<td>122</td>
<td>32</td>
<td>1</td>
<td>122</td>
<td>22</td>
</tr>
</tbody>
</table>

**PBC 27**

Displays the PERFECT BLOCK CHANGE (PBC) count for the selected group of records (1000). The PBC is the number of block changes that would occur if a table was loaded with a group (1000) of rows in sequential order and all blocks had the maximum number of rows per block.

The following lines display the detail line area of the report. Because RECORDS=1000 is specified, there is one line of output for each group of 1000 rows.
GROUP
Displays the group number being processed. The group number begins at 1 and increments until it reaches 99,999 groups. If this number of groups is reached, the report continues with the next group number being with zero.

**Note:** Each group is processed autonomously from the other rows in the table. The goal is to establish efficiency for this group of rows. When a data block has 30 rows but only 15 rows are from this group, only those 15 rows are recognized and the rest of the rows in the block are ignored. This fact must be remembered when you are evaluating the statistical values in the columns. You cannot assume that the detail group line values add up to match the FULL TABLE summary values.

ROWS READ
Displays the number of rows in the group.

BLOCKS W/DATA
Displays the number of data blocks that contained at least one data row for this group.

BLOCK: MAX ROWS, MIN ROWS and AVG ROWS
Displays an evaluation for each group. Identifies the high-level mark (MAX ROWS) and the low-level mark (MIN ROWS). The report also divides the total rows of the group by the blocks with data rows for the group to calculate the average rows (AVG. ROWS).

BLOCK CHANGES
Displays the number of changes that would occur when the groups data rows were processed using the selected key sequence (chosen using KEYNAME=). A BLOCK CHANGE is counted each time the next row in the sequence is not within the same block as the current row.

The BLOCK CHANGES is the key factor in measuring sequential efficiency. When a table is in sequential order, all data rows are processed in a given data block before moving on to the next block. When rows are not in order, the sequential process bounces back and forth between data blocks to retrieve all data rows in the selected key sequence. This bouncing back and forth adds overhead and delays to any process that is reading the data in sequential order.

EFF %
Indicates the MRB efficiency which is calculated by taking the ideal situation where all data blocks are only processed once as data rows are read sequentially in the group (PBC 27) and dividing that by the actual number of block changes that actually occurred.
MAX ROW/BLK EFFICIENCY RATES WITH BUFFERS

Displays the calculated MRB efficiency with buffers. Because buffering can significantly reduce the physical I/O used to read a previously read data block back into storage, the MRB efficiency percentage is recalculated using two buffers, four buffers, and so on.

FULL TABLE section

Summarizes the report. Two lines are created for the table. This information was described earlier in this section.

Note: The detail group lines treat each group autonomously and generate their statistical calculations based on that group. The total or summary lines look at the table as a whole and their statistical calculations are based on the whole table. You cannot assume that the detail group line values add up to match the FULL TABLE summary values.

Example JCL: Detail DATANE Report for a compressed table (“NUMBER=0200” specified)

The following example shows the command to obtain a DATANE detail report when the table is compressed and the NUMBER= parameter contains a value. For this report, we specified to use the first 200 physical blocks (NUMBER=0200) to determine the MRB value.

We also specified a detailed efficiency line for every 1000 records.

Note: Use the following as a guide to prepare your JCL. The JCL statements are examples only. Lowercase letters in a statement indicate a value you must supply. Code all statements to your site and installation standards.

```
//jobname   Per site standard   Job card
// EXEC PGM=DBUTLTY,REGION=2M   Execute DBUTLTY
//STEPLIB   Per site standard   CA Datacom Execution Lib
//SYSIN     DD *                  Command Input
REPORT TYPE=DATANE,TABLE=REO,DBID=1023,RECORDS=01000,NUMBER=0200
/*
Sample Report: Detail DATANE Report for a compressed table ("NUMBER=0200" specified)

The first page of the report is described earlier in this section. The only difference is in this report we have included the NUMBER=0200 parameter to limit the physical blocks scanned to determine the MRB value. The scan is only done when a table has compression.

```sql
CONTROL CARD(S)........1........2.........3.........4.........5.........6.........7.........8
REPORT TYPE=DATANE, TABLE=REO, DBID=01023, RECORDS=00001000,
       NUMBER=0200
FUNCTION=REPORT
DBID=01023
NUMBER=0200
RECORDS=00001000
TABLE=REO
TYPE=DATANE

The second page of the report is similar to what you have seen earlier in this section. However, note the following specific differences.

<table>
<thead>
<tr>
<th>TBL AREA KEY</th>
<th>CMP USER COMPRESSION</th>
<th>RECLN</th>
<th>RECORDS BLKSIZE</th>
<th>TRACKS</th>
<th>TOT BLKS</th>
<th>INUSE BLKS</th>
<th>PART BLKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>REO</td>
<td>REO</td>
<td>100</td>
<td>31,950</td>
<td>4,096</td>
<td>150</td>
<td>1,800</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

SCAN MAX ROWS/BLOCKS 108  #ROWS READ 18,260

CMP Y

Indicates that this table is compressed. Because the USER COMPRESSION field is blank, we know that this table is compressed using CA Datacom/DB standard compression.

SCAN MAX ROWS/BLOCKS

Indicates that a physical scan of the data blocks was used to determine the MRB value. The number of blocks scanned was specified in the NUMBER=0200 parameter. In scanning the first 200 data blocks, the MRB value was determined to be 108.

#ROWS READ

Indicates that, in scanning the first the 200 data blocks, the report encountered 18,260 data rows.
The third and following pages of the detail report provide the efficiency ratings for each group of records. At the end of the report, the FULL TABLE summary lines are generated.

<table>
<thead>
<tr>
<th>GROUP</th>
<th>ROWS</th>
<th>BLOCKS</th>
<th>MAX W/DATA CHANGES</th>
<th>MRB</th>
<th>BLOCK EFF</th>
<th>2 4 8 12 16 20 24 28 32 48 64 96 128 256</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ</td>
<td>w/DATA</td>
<td>PBC</td>
<td>ROWS</td>
<td>BLOCKS</td>
<td>MIN</td>
<td>AVG</td>
</tr>
<tr>
<td>1</td>
<td>1,000</td>
<td>113 38 2 9</td>
<td>113 7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1,000</td>
<td>152 18 1 7</td>
<td>151 5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1,000</td>
<td>93 30 2 11</td>
<td>183 8</td>
<td>9</td>
<td>81 2 11</td>
<td>1 14</td>
</tr>
<tr>
<td>4</td>
<td>1,000</td>
<td>85 34 1 12</td>
<td>106 8</td>
<td>10 14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1,000</td>
<td>62 48 1 16</td>
<td>61 14</td>
<td>61</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1,000</td>
<td>81 74 2 12</td>
<td>80 11</td>
<td></td>
<td>15 56</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1,000</td>
<td>155 21 1 6</td>
<td>154 5</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1,000</td>
<td>116 25 1 9</td>
<td>123 7</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1,000</td>
<td>90 39 1 11</td>
<td>119 8</td>
<td>9</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1,000</td>
<td>56 43 5 18</td>
<td>56 16</td>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>1,000</td>
<td>66 26 1 15</td>
<td>65 13</td>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>1,000</td>
<td>135 22 1 7</td>
<td>128 6</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>1,000</td>
<td>170 22 1 6</td>
<td>169 5</td>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>1,000</td>
<td>99 42 1 10</td>
<td>100 9</td>
<td>9</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>1,000</td>
<td>54 60 1 19</td>
<td>53 16</td>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>1,000</td>
<td>59 26 1 17</td>
<td>58 15</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>1,000</td>
<td>111 32 1 9</td>
<td>111 8</td>
<td>8</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>1,000</td>
<td>138 40 1 7</td>
<td>137 6</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>1,000</td>
<td>86 43 2 12</td>
<td>93 9</td>
<td>9</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>1,000</td>
<td>86 33 1 12</td>
<td>113 7</td>
<td>8</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>1,000</td>
<td>66 49 1 15</td>
<td>65 13</td>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>1,000</td>
<td>108 27 1 9</td>
<td>107 8</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>1,000</td>
<td>126 18 1 8</td>
<td>125 7</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>1,000</td>
<td>110 54 1 9</td>
<td>109 8</td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>1,000</td>
<td>93 57 1 11</td>
<td>94 9</td>
<td>9</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>1,000</td>
<td>79 31 1 13</td>
<td>78 11</td>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>1,000</td>
<td>91 24 1 11</td>
<td>90 10</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>1,000</td>
<td>119 21 1 8</td>
<td>119 7</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>1,000</td>
<td>124 36 1 8</td>
<td>123 7</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>1,000</td>
<td>90 33 1 11</td>
<td>90 10</td>
<td>10</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>1,000</td>
<td>94 31 1 11</td>
<td>93 9</td>
<td>9</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>950</td>
<td>56 27 5 17</td>
<td>55 16</td>
<td>16</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

PBC 9

Displays the PERFECT BLOCK CHANGE (PBC) count for the selected group of records (1000). The PBC is the number of block changes that would occur if a table was loaded with a group (1000) of rows in sequential order and all blocks had the maximum number of rows per block. A compressed table allows more rows to fit per block as evidenced by the increased MRB (from 36 to 108), which in turn lowers the PBC value.

This rest of the entries on the report were described in the previous samples.
Appendix B: Implementing and Using DBUTLTY REPORT=DATASP

This appendix provides the basic operating instructions for the new DBUTLTY REPORT=DATASP function. The REPORT=DATASP function is only available in CA Datacom/DB Version 14.0 and in CA Datacom/DB Version 12.0 only after applying the PTFs that add the functionality.

REPORT TYPE=DATASP (Data Area Space Report)

The Data Area Space (DATASP) Report provides a detailed space usage map for a selected data area. This function processes the data area using a high speed I/O process analyzing each data block in the data area. The DATASP report provides detailed insight into the data block space usage and available free space.

The DATASP report uses a highly efficient I/O process that has no effect on the MUF processing. The DATASP function reads the data area in place and does not see activity currently occurring in the MUF that is not yet posted to the data area. The DATASP report can be executed when there is only a minimal amount of update activity against the selected data area.

The purpose of the DATASP report is to build a simple report that shows the detailed space usage for a selected data area that can help you determine, among other things, the amount of rows that could be added in the space available in the data area.

Before the DATASP report, most users relied on the information in the CXX report or Dynamic System Tables to determine the “fullness” of a data area. This information is provided in the following CXX report columns:

- BLOCKS IN USE - 87
- BLOCKS UNUSED - 12
- TOTAL BLOCKS - 99
- PERCENT FULL - 88
Restrictions

While this information provides a reasonably accurate accounting of the data blocks usage, it may not accurately reflect the data row “fullness.” For example, if a table was loaded with 900 data rows and those rows filled 90 of 100 available data blocks, both the data block and data row “fullness” in the data area would be approximately 90% full. If following the load, an application process deleted 90 data rows where each data row just happened to be in the next data block, the data block fullness would still be at 90%, but the data row fullness would only be 81%.

In many cases, the difference between data block fullness and data row fullness will be minimal. A few examples of where this would occur are the following:

- Data areas with DSOP 0
- Data areas with DSOP 1 where row sizes in the data area are similar and more rows are added than are deleted
- DSOP 3 without a cluster key where row sizes in the data area are similar and more rows are added than are deleted.

In certain other cases, the difference between data block fullness and data row fullness may be significant. A few examples of where this could occur are the following:

- Data areas with DSOP 1 where more rows are deleted than are added
- Data areas with DSOP 2
- Data areas with DSOP 3 where a cluster key has been selected
- Data areas with DSOP 4 after an OLREORG function
- Data areas with DSOP 5

The DATASP report gives you a simple and accurate way to access the data row fullness for a data area. The report also provides other information on the data rows that is beneficial for you to know. For example, it can provide the maximum, minimum, and average compressed row size in a compressed table.

Restrictions

The area selected for the report must be a valid CA Datacom/DB data area. The area cannot be open by a DBUTLTY INIT or DBUTLTY LOAD function. The data area must be loaded with URI=YES.

Security Considerations: The DBUTLTY external resource name for the DATASP report is REPORT.DATASP.
When to Use

Use the DATASP report when you want to determine how full the current data area is and the available row space in a data area. The DATASP report also provides detailed information about the available free space, the number and size of data rows, and various other data row information.

How to Use

To obtain a Space Usage (DATASP) report, execute the REPORT function with the following command format:

```
REPORT TYPE=DATASP, DBID=dbid, AREA=area[, BLOCKS=nnnnnn]
```

- **REPORT**
  - Invokes the report function.

- **TYPE=DATASP**
  - *(Required)* Requests the data area space usage report.
  - Valid Entries: DATASP
  - Default Value: (No default)

- **DBID=**
  - *(Required)* Specifies the number of the DBID that you want to use.
  - Valid Entries: A valid DBID
  - Default Value: (No default)

- **AREA=**
  - *(Required)* Identifies the area you want to report on.
  - Valid Entries: DATACOM-NAME of the area in the database specified
  - Default Value: (No default)
How to Use

,BLOCKS=

(Optional) Specifies the blocks to process (and include) in each printed detail line output. If a value is not specified, no detail lines are produced and only the summary lines for the data area are produced.

Note: Specifying a low number for BLOCKS= parameter with a data area with a large amount of data blocks generates a large number of detail lines (one per block group). If the number of groups generated exceeds 99,999 groups, the report allows the group number to “roll” over to zero and begin again. This typically would also mean that over 2000 pages of detail group lines would be generated. We recommend that a BLOCKS= value be specified that keeps the number of detail groups to a 1000 groups or less.

Valid Entries: 0 to 999999

Default Value: (No default)

Example JCL: Summary DATASP Report (“BLOCKS=” not specified)

The following shows the command to obtain a DATASP report when BLOCKS= is not specified.

Note: Use the following as a guide to prepare your JCL. The JCL statements are for example only. Lowercase letters in a statement indicate a value you must supply. Code all statements to your site and installation standards.

```
//jobname     Per site standard              Job card
//       EXEC PGM=DBUTLTY,REGION=2M          Execute DBUTLTY
//STEPLIB     Per site standard
//CXX    DD DSN=cxx.data.set,DISP=SHR        Directory data set
//SYSIN  DD *                                Command Input
   REPORT TYPE=DATASP,DBID=1,AREA=PAY
/*
```
Sample Report: Summary DATASP Report ("BLOCKS=" not specified)

The first page of the report shows the following information about the command processed by the utility.

<table>
<thead>
<tr>
<th>CONTROL CARD(S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>........1.......2........3........4........5........6........7.......8</td>
</tr>
</tbody>
</table>

```
REPORT TYPE=DATASP,DBID=1,AREA=PAY  
FUNCTION=REPORT  
AREA=PAY  
DBID=00001  
TYPE=DATASP
```

- The command exactly as entered.
- An analysis of keywords encountered and expected. Any errors found are flagged with a note in the left margin.
- Any messages related to syntax processing.

The second page of the report displays the following.

```
<table>
<thead>
<tr>
<th>AREA PAY</th>
<th>BLKSIZE</th>
<th>TRACKS</th>
<th>2 BLOCKS</th>
<th>66 URI YES</th>
<th>DSOP 0</th>
<th>NO RECLAMATION</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>AREA PAY</th>
<th>PAY</th>
<th>BLKSIZE</th>
<th>1,024</th>
<th>TRACKS</th>
<th>2 BLOCKS</th>
<th>66 URI YES</th>
<th>DSOP 0</th>
<th>NO RECLAMATION</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>GROUP#</th>
<th>BLOCS</th>
<th>KEYTES</th>
<th>IN-USE %</th>
<th>IN-USE %</th>
<th>IN-USE</th>
<th>MAX</th>
<th>MIN</th>
<th>MAX</th>
<th>MIN</th>
<th>MAX</th>
<th>MIN</th>
<th>MAX</th>
<th>MIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOTALS</td>
<td>13</td>
<td>17</td>
<td>10</td>
<td>16</td>
<td>200</td>
<td>1,022</td>
<td>19</td>
<td>52</td>
<td>10</td>
<td>1</td>
<td>53</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>53</td>
<td>83</td>
<td>54</td>
<td>84</td>
<td>0</td>
<td>34</td>
<td>0</td>
<td>52</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1,016</td>
<td>866</td>
<td>3</td>
<td>52</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

AREA

Displays the area DATACOM-NAME, PAY.

BLKSIZE

Displays the blocksize in the area, 1,024.

TRACKS

Displays the number of tracks used, 2.

BLOCKS

Displays the number of blocks, 66.
How to Use

URI
Displays whether URI is assigned, YES.

DSOP
Identifies that the data space option is 0 with no reclamation.

The next lines are created for each table in the area.

TBL
Displays the table DATACOM-NAME, PAY.

CMP
Displays whether this is a compressed table.

USER COMPRESSION
Contains the user exit name when this table has a user compression routine specified or blanks when CA Datacom/DB standard compression is used.

RECLN
Displays record length including the RCE bytes, 52.

RECORDS
Displays the number of records, 200.

DSOP
Identifies that the data space option is 0 with no reclamation.

BLOCKS IN-USE % EMPTY %
Displays the number of blocks in use followed by the percentage in use over the number of blocks empty followed by the percentage of blocks empty.

KBYTES IN-USE % EMPTY %
Displays the number of kilobytes in use followed by the percentage in use over the number of kilobytes empty followed by the percentage of kilobytes empty.
ROWS IN-USE DELETED *AVAIL

Displays the number of rows in use, rows that are currently marked logically deleted, and number of rows could be added. Available rows are a best estimate based on average row size, space available in data blocks, and so on.

FREESP MAX MIN AVG

Displays the size of the largest segment of free-space bytes (MAX), the smallest segment of free-space bytes (MIN), and the average segment size of free-space bytes (AVG).

ROWS MAX MIN AVG

Displays the largest number of rows found in any data block (MAX), smallest number of rows found in a data block (MIN), and average number of rows per data block (AVG).

ROWLEN MAX MIN AVG

Displays the largest row length found in any data block (MAX), smallest row length found in a data block (MIN), and average row length per data block (AVG).

FREESPACE IN BLOCKS

Displays a map of free-space segments in each size grouping.

Example JCL: Detail DATASP Report ("BLOCKS=10" specified)

The following shows the command to obtain a detail DATASP report when "BLOCKS=10" is specified.

Note: Use the following as a guide to prepare your JCL. The JCL statements are for example only. Lowercase letters in a statement indicate a value you must supply. Code all statements to your site and installation standards.

```jcl
//jobname     Per site standard              Job card
//       EXEC PGM=DBUTLTY,REGION=2M          Execute DBUTLTY
//STEPLIB     Per site standard              CA Datacom/DB Execution Library
//CXX    DD DSN=cxx.data.set,DISP=SHR        Directory data set
//SYSIN  DD *                                Command Input
      REPORT TYPE=DATASP,DBID=1,AREA=PAY,BLOCKS=10
/*
```
Sample Report: Detail DATASP Report ("BLOCKS=10" specified)

The first page of the report shows the following information about the command processed by the utility.

<table>
<thead>
<tr>
<th>CONTROL CARD(S)</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>REPORT TYPE=DATASP, DBID=1, AREA=PAY, BLOCKS=10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FUNCTION=REPORT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AREA=PAY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BLOCKS=000010</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DBID=000003</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TYPE=DATASP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- The command exactly as entered.
- An analysis of keywords encountered and expected. Any errors found are flagged with a note in the left margin.
- Any messages related to syntax processing.
The second page displays the information describing the selected data area followed by information describing the table or tables in the area. One line is created for each table in the area.

<table>
<thead>
<tr>
<th>AREA</th>
<th>PAY</th>
<th>BLKSIZE</th>
<th>1,024</th>
<th>TRACKS</th>
<th>2</th>
<th>BLOCKS</th>
<th>66</th>
<th>URI</th>
<th>YES</th>
<th>DSOP</th>
<th>0</th>
<th>NO RECLAMATION</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The columns under the following headings are filled in for each reference group as specified with the REFGROUP parameter on the DBUTLTY OLREORG function.

**BLOCKS IN-USE % EMPTY %**

Displays the number of blocks in use followed by the percentage in use over the number of blocks empty followed by the percentage of blocks empty.

**KBYTES IN-USE % EMPTY %**

Displays the number of kilobytes in use followed by the percentage in use over the number of kilobytes empty followed by the percentage of kilobytes empty.

**ROWS IN-USE DELETED *AVAIL**

Displays the number of rows in use, rows that are currently marked logically deleted, and number of rows could be added. Available rows are a best estimate based on average row size, space available in data blocks, and so on.
FREESP MAX MIN AVG

Displays the size of the largest segment of free-space bytes (MAX), the smallest segment of free-space bytes (MIN), and the average segment size of free-space bytes (AVG).

ROWS MAX MIN AVG

Displays the largest number of rows found in any data block (MAX), the smallest number of rows found in a data block (MIN), and the average number of rows per data block (AVG).

ROWLEN MAX MIN AVG

Displays the largest row length found in any data block (MAX), smallest row length found in a data block (MIN), and average row length per data block (AVG).

FREESPACe IN BLOCKS

Displays a map of freespace segments in each size grouping.

Note: Additional groups of blocks are listed with their information.

TOTALS

Displays a summary of all group information.
Glossary

**adjusted maximum rows per block (AMRB)**

The *Adjusted Maximum Rows per Block (AMRB)* is a DBUTLTY TYPE=DATANE report metric for measuring the efficiency of sequential processing by comparing the actual number of blocks in use to the number of block changes encountered when processing the data rows in sequence.

**cluster key**

*Cluster key (value)* is used to assign data blocks in DSOP 3.

**compressed data table**

A *compressed data table* is a CA Datacom/DB data table that uses a compression routine to compress data rows before they are stored on a data block. Similarly, the compressed data table uses the same routine to uncompress the data row when it is being retrieved from the data block.

**dark window**

A *dark window* is a time period when a database is unavailable for user access. Typically occurs during the weekend when database maintenance is being performed.

**data row reorganization**

*Data row reorganization* is the process used to re-order data rows within a table into native key sequence.

**data space management options (DSOPs)**

*Data space management options (DSOPs)* are options selected at the data area level that determine how new rows are added and how deleted row space will be re-used.

**DATANE Report (NER)**

The *DATANE Report (NER)* is a new DBUTLTY function delivered in PTFs for CA Datacom/DB Version 12.0 and in CA Datacom/DB Version 14.0. It provides sequential efficiency ratings based on the Native Key.

**DATASP Report**

The *DATASP Report* is a new DBUTLTY function delivered in PTFs for CA Datacom/DB Version 12.0 and in CA Datacom/DB Version 14.0. The DATASP report provides detailed information on block and row space usage in a data area.
**full table adjusted maximum rows per block (AMRB)**

The *full table adjusted maximum rows per block (AMRB)* is a DBUTLTY TYPE=DATANE report metric for measuring the efficiency of sequential processing by comparing the actual number of blocks in use to the number of block changes encountered when processing the data rows in sequence.

**full table maximum rows per block (MRB)**

The *full table maximum rows per block (MRB)* is a DBUTLTY TYPE=DATANE report metric for measuring the efficiency of sequential processing by comparing a calculated minimum number of data blocks to hold a set of data rows compared to the number of block changes encountered when processing the data rows in sequence.

**least recently used**

The *least recently used* (LRU) is an algorithm used to maintain a list of items in order by the last access or use.

**master key**

The *master key* is a unique key (index) within a CA Datacom table where there is one index value per data row.

**maximum rows per block (MRB)**

The *maximum rows per block (MRB)* is a DBUTLTY TYPE=DATANE report metric for measuring the maximum number of rows that will fit in a data block. For uncompressed (single table) areas this is calculated by dividing the blocksize minus 2 by the row size. For compressed and multi-table data areas, the MRB is the actual “high water” count of rows in a set of blocks that are scanned by the DATANE report.

**multi-table data area**

The *multi-table data area* is a CA Datacom/DB data area (physical) that contains more than one data table (logical).

**native (sequence) key**

The *native key* or *native sequence key* is a key within a CA Datacom/DB table that defines the targeted sequence of the data rows. The Native Key is used when the backup process specifies the SEQ=PHYSICAL parameter. To reorganize database or area into native sequence, the SEQ=PHYSICAL backup function is followed by a database or area load function.

**non-compressed data table**

The *non-compressed data table* is a CA Datacom/DB data table that does not use a compression routine. All data rows are stored as an exact image on the data block.
**perfect block count (PBC)**

The *perfect block count (PBC)* is used in the DATANE report. PBC represents the number of data blocks required to hold a set of data rows when each data block is filled to its maximum. PBC is typically calculated by taking the row count and dividing it by the maximum rows per block (MRB).

**reference group (REFGROUP)**

The *reference group (REFGROUP)* is used in the DBUTLTY OLREORG function. The reference group is the number of blocks that will be selected (using the native sequence key) for the OLREORG process.

**reference set**

The *reference set* is used in the DBUTLTY OLREORG function. The reference set consists of the data rows that are processed as a set within the OLREORG REFGROUP process.

**sequential efficiency**

The *sequential efficiency* is a simple metric for measuring the efficiency of sequential requests against a set of data rows (table). The most common form is the number of data row requests (logical) divided by the number of I/Os generated to fulfill the requests (physical).

**sequential processing**

The *sequential processing* is performed when retrieving data rows from a table in an index or key sequence. Usually done by the Native Key.

**single-table data area**

The *single-table data area* is a CA Datacom/DB data area (physical) that contains only one data table (logical).

**skip-sequential processing**

The *skip-sequential processing* is performed when retrieving data rows by first locating a place in the index and then retrieving a set of rows in the index sequence. Once the required rows are retrieved, the process skips to the next location in the index to begin reading rows.
## Index

### A

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A Challenging Table for OLREORG</td>
<td>82</td>
</tr>
<tr>
<td>A Good Table for OLREORG</td>
<td>84</td>
</tr>
<tr>
<td>A Good Table for OLREORG with DSOP 4 and 5</td>
<td>102</td>
</tr>
<tr>
<td>A Recommendation for All OLREORG Users</td>
<td>124</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>iii</td>
</tr>
<tr>
<td>adjusted maximum rows per block (AMRB)</td>
<td>163</td>
</tr>
<tr>
<td>Audience</td>
<td>9</td>
</tr>
</tbody>
</table>

### B

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Break Down the Sequential Access Efficiency and Compare it to Overall</td>
<td>38</td>
</tr>
<tr>
<td>Access Efficiency</td>
<td></td>
</tr>
<tr>
<td>Breaking Down the Sequential Access by the Key Name</td>
<td>32</td>
</tr>
<tr>
<td>Buffered Sequential Access Efficiency</td>
<td>54</td>
</tr>
</tbody>
</table>

### C

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CA Datacom/DB and Data Row Access through Indexes</td>
<td>10</td>
</tr>
<tr>
<td>CA Datacom/DB and Data Row Storage and Ordering</td>
<td>12</td>
</tr>
<tr>
<td>CA Datacom/DB and Data Row Storage Methods (DSOPs)</td>
<td>13</td>
</tr>
<tr>
<td>CA Datacom/DB and the Effects of Data Row Placement</td>
<td>16</td>
</tr>
<tr>
<td>CA Datacom/DB Table Partitioning Considerations</td>
<td>17</td>
</tr>
<tr>
<td>CA TECHNOLOGIES PRODUCT REFERENCES</td>
<td>iv</td>
</tr>
<tr>
<td>Chapter Summary</td>
<td>18, 48, 59, 73, 93, 113, 121, 124, 129, 136</td>
</tr>
<tr>
<td>cluster key</td>
<td>163</td>
</tr>
<tr>
<td>Comparing the Accounting Facility with the ADS Table of the History</td>
<td>43</td>
</tr>
<tr>
<td>Database</td>
<td></td>
</tr>
<tr>
<td>compressed data table</td>
<td>163</td>
</tr>
<tr>
<td>Considering the Use of MRDF</td>
<td>46</td>
</tr>
</tbody>
</table>

### D

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>dark window</td>
<td>163</td>
</tr>
<tr>
<td>Data Area Partition Key Range</td>
<td>132</td>
</tr>
<tr>
<td>Data Area Space Report (DATASP)</td>
<td>115</td>
</tr>
<tr>
<td>Data Reorganization Green Book Summary</td>
<td>137</td>
</tr>
<tr>
<td>Data Row Placement</td>
<td>133</td>
</tr>
<tr>
<td>data row reorganization</td>
<td>163</td>
</tr>
<tr>
<td>data space management options (DSOPs)</td>
<td>163</td>
</tr>
<tr>
<td>Datadictionary Catalog Process Reverts the DSOP Setting</td>
<td>96</td>
</tr>
<tr>
<td>DATANE Report (NER)</td>
<td>163</td>
</tr>
<tr>
<td>DATASP and Single-Table Areas (Compressed)</td>
<td>116</td>
</tr>
<tr>
<td>DATASP and Single-Table Data Areas (Non-Compressed)</td>
<td>115</td>
</tr>
<tr>
<td>DATASP Report</td>
<td>163</td>
</tr>
<tr>
<td>DBUTLY Offline Reorganization Techniques</td>
<td>61</td>
</tr>
<tr>
<td>DBUTLY OLREORG and the Moved Record Index</td>
<td>123</td>
</tr>
<tr>
<td>DBUTLY OLREORG Functionality for Release 11</td>
<td>75</td>
</tr>
<tr>
<td>Defragmenting the Index</td>
<td>63</td>
</tr>
<tr>
<td>Determining if a Data Row Reorganization Is Needed</td>
<td>19</td>
</tr>
<tr>
<td>Determining if OLREORG is a Good Choice for Reorganization</td>
<td>75</td>
</tr>
<tr>
<td>Determining the Need for Repeating an OLREORG</td>
<td>86</td>
</tr>
<tr>
<td>Determining the Need for Repeating an OLREORG with DSOP 4 or 5</td>
<td>100</td>
</tr>
<tr>
<td>Determining the NUMBER Value for OLREORG</td>
<td>85</td>
</tr>
<tr>
<td>Determining the NUMBER Value for OLREORG with DSOP 4 or 5</td>
<td>100</td>
</tr>
</tbody>
</table>

### F

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>FEEDBACK</td>
<td>iv</td>
</tr>
<tr>
<td>Finding Data Tables with Significant Sequential Access</td>
<td>30</td>
</tr>
<tr>
<td>full table adjusted maximum rows per block (AMRB)</td>
<td>164</td>
</tr>
</tbody>
</table>
full table maximum rows per block (MRB) • 164

G
Greater Flexibility with the Accounting Facility • 29

H
High Level Access with Key Information for Non-CICS (A94 and A95) • 26
High Level Access with Key Information for Selected DBIDs (A96 and A97) • 27
High Level Access with Key Information for Selected Time Periods (A98, A99) • 28
High Level Accounting Tables (A90 and A91) • 23
High Level Accounting Tables with Key Information (A92 and A93) • 25
History Database Rows and Columns • 36
How to Compare Efficiency Ratings if the MRB Changes • 52
How to Use • 140, 155

I
Implementing and Using DBUTLTY REPORT=DATANE • 139
Implementing and Using DBUTLTY REPORT=DATASP • 153
Introduction • 9

L
least recently used • 164
LEGAL NOTICE • ii
Loading the Native Sequence Backup • 62

M
master key • 164
Matching the Backup and the Load • 63
Maximizing Performance when Taking a Native Sequence Backup • 62
maximum rows per block (MRB) • 164
Measuring Data Row Available Space in a Data Area Before and After an OLREORG with DSOP 4/5 • 117

Miscellaneous Reorganization Processes • 73
Modern DASD and the Effects of Data Row Placement • 17
multi-table data area • 164

N
native (sequence) key • 164
Native Efficiency Report • 49
Native Efficiency Report (DATANE) • 136
Native Efficiency Report and Multi-Table Areas • 52
Native Efficiency Report and Single-Table Areas (Compressed) • 51
Native Efficiency Report and Single-Table Data Areas (Non-Compressed) • 50
Native Sequence Backup • 61
New Data Area Space Management Options (DSOP) • 95
New Data Area Space Usage Report • 115
New Data Table Native Efficiency Report for DBUTLTY • 49
New DBUTLTY OLREORG Functionality for Version 12.0 • 95
non-compressed data table • 164

O
Offline Data Row Reorganization • 132
OLREORG for Multi-Table Data Areas • 77
OLREORG for Single-Table Data Areas • 76
OLREORG with DSOP 4 or 5 and Blocks In-Use Change • 112
OLREORG with DSOP 4 or 5 Does Not Trigger a Dynamic Extend • 112
Online Data Row Reorganization • 135

P
Parallel Reorganization by Data Area • 69
Parallel Reorganization by Data Area with MULTUSE=YES • 73
perfect block count (PBC) • 165
Purpose of this Document • 9

R
Reblocking REO 1022 from 4096 to 8192 • 126
Recognizing the Effect of Larger Buffer Pools • 45
Recognizing the Effect of the Application Access • 47
Recognizing the Effect on the Environment • 44
Recommend DEFRAG of Index Before and After OLREORG • 112
reference group (REFGROUP) • 165
reference set • 165
REFGROUP Selections • 108
Reorganization by Data Area • 65
Reorganization by Data Area with MULTUSE=YES • 68
Reorganization by DBID • 64
REPORT TYPE=DATANE (Native Sequence Efficiency Report) • 139
REPORT TYPE=DATASP (Data Area Space Report) • 153
Requirements • 9
Restrictions • 140, 154
Running Multiple OLREORGs to Continue to Improve Efficiency • 106

S
Sample Implementation and Use to Determine Sequential Processing • 29
Sample Native Efficiency Reports • 56
Sequential Access and the Data Block • 52
Sequential Access Efficiency by the Native Sequence Key • 53
Sequential Data Row Access for the Parent Table • 134
sequential efficiency • 165
sequential processing • 165
single-table data area • 165
skip-sequential processing • 165
Space Consumed in the Index for the MRI • 111
Stacking OLREORGs in a Single DBUTLTY Execution • 107
Suppressing the DSOP 4 and DSOP 5 Empty Block Usage • 113

T
Table Partitioning and Data Row Reorganizations • 131
The Moved Records Index is Built in the IXX Area • 123

The OLREORG Report • 82
The Partition Key • 131
Two Data Buffer Sizes – Two Data Buffer Pools • 125

U
Understanding the OLREORG DSOP Process and Determining the REFGROUP Parameter Value • 97
Understanding the OLREORG Process and Determining the REFGROUP Parameter Value • 79
Updating and Rebuilding the Index • 63
Using DATANE to Measure OLREORG in a Challenging Scenario • 90
Using DATANE to Measure OLREORG in the Challenging Scenario • 99
Using RETIX with KEYNAME=*DATA to Rebuild Empty Block Index • 96
Using the ADS Table Information to Answer Questions on Sequential Performance • 40
Using the DATANE Report to Verify the OLREORG Process • 87
Using the New DBUTLTY TYPE=DATASP Utility Function • 116
Using the Optional Accounting Facility • 22
Using the Optional History Database • 35
Utilizing Multiple Data Block Sizes • 125

W
What is Sequential Processing • 19
When to Use • 140, 155