Redbooks
ibm.com/redbooks

Db2 for z/OS Utilities in Practice

Craig Friske

Hendrik Mynhardt

IBM®

Redpaper

International Technical Support Organization

Db2 for z/OS Utilities in Practice

June 2018

REDP-5503-00

Note: Before using this information and the product it supports, read the information in “Notices” on page vii.

First Edition (June 2018)

This edition applies to Version 12 of IBM DB2 for z/OS.

© Copyright International Business Machines Corporation 2018. All rights reserved.
Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, MD-NC119, Armonk, NY 10504-1785, US

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at “Copyright and trademark information” at http://www.ibm.com/legal/copytrade.shtml

The following terms are trademarks or registered trademarks of International Business Machines Corporation, and might also be trademarks or registered trademarks in other countries.

Db2®	FlashCopy®	Redbooks (logo) ®
DB2®	IBM®	z Systems®
Distributed Relational Database Architecture™	Redbooks®	z/OS®
	Redpaper™	

The following terms are trademarks of other companies:

Evolution, are trademarks or registered trademarks of Kenexa, an IBM Company.

Other company, product, or service names may be trademarks or service marks of others.

Preface

As IBM® continues to enhance the functionality, performance, and availability of IBM Db2® for z/OS, the utilities have made significant strides towards self-management.

IBM Db2 for z/OS utilities is leading the trend towards autonomics. During the last couple of versions of Db2 for z/OS, and through the maintenance stream, new features and enhancements have been delivered to further improve the performance and functionality of the Db2 for z/OS utilities.

The intent of this IBM Redpaper™ publication is to help Db2 Database Administrators, Db2 System Programmers, and anyone who runs Db2 for z/OS utilities implement best practices. The intent of this paper is not to replicate the Db2 Utilities Reference Guide or the Db2 Installation Guide.

This paper describes and informs you how to apply real-life practical preferred practices for the IBM Db2 for z/OS Utilities Suite. The paper concentrates on the enhancements provided by Db2 utilities, regardless of the version, albeit some functions and features are available only in Db2 12 for IBM z/OS®.

Authors

This paper was produced by a team of specialists from around the world.

Craig Friske has over 25 years of development experience with IBM DB2® for z/OS at the Silicon Valley Lab. He has led or worked on numerous utility projects including statistics gathering enhancements, partition independence, data compression, and online utilities (**REORG**, **CHECK**, and **REBUILD**). Craig has also worked on many availability enhancements for Online Schema Evolution®.

Hendrik "Hennie" Mynhardt is an Executive IT Specialist based in the USA. He has lead and worked on various technical projects for database customers in the USA and overseas. His special interests are systems performance tuning and backup/recovery. He currently provides technical consulting, pre-sales support, and post-sales support for Db2 and related tooling for the Analytic space. Hennie has co-authored *Securing and Auditing Data on DB2 for z/OS*, SG24-7720, *Optimizing Restore and Recovery Solutions with DB2 Recovery Expert for z/OS V2.1*, SG24-7606, *DB2 Recovery Expert for z/OS User Scenarios*, SG24-7226, *DB2 9 for z/OS and Storage Management*, SG24-7823, and *Modernize Your IBM DB2 for IBM z/OS Maintenance with Utility Autonomics*, SG24-8304.

Thanks to Karen Wilkins for her contributions in reviewing this paper.

Now you can become a published author, too

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time. Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run 2- 6 weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us.

We want our papers to be as helpful as possible. Send us your comments about this paper or other IBM® Redbooks® publications in one of the following ways:

- Use the online **Contact us** review Redbooks form:

  ibm.com/redbooks

- Send your comments in an email:

  redbooks@us.ibm.com

- Mail your comments:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- Find us on Facebook:

  http://www.facebook.com/IBMRedbooks

- Follow us on Twitter:

  http://twitter.com/ibmredbooks

- Look for us on LinkedIn:

  http://www.linkedin.com/groups?home=&gid=2130806

- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

  https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm

- Stay current on recent Redbooks publications with RSS Feeds:

  http://www.redbooks.ibm.com/rss.html

1

Loading Db2 data

The **LOAD** utility is a fast way to bring data into Db2 tables from flat files, a delimited stream, or even directly from other systems through IBM Distributed Relational Database Architecture™ (DRDA) fast loading. In the last few years there have been numerous enhancements for improved functional usability, availability, and performance, so that is the focus of this section.

1.1 Usability functions

This section addresses considerations related to usability functions.

1.1.1 CONSTANT and CONSTANTIF

The `CONSTANT` and `CONSTANTIF` keywords of the **`LOAD`** utility allow columns to be populated with a specific constant values when loading a data row instead of populating the column from data values in the input stream.

Consider the following **`LOAD`** statement and input (Example 1-1).

Example 1-1 LOAD statement and input

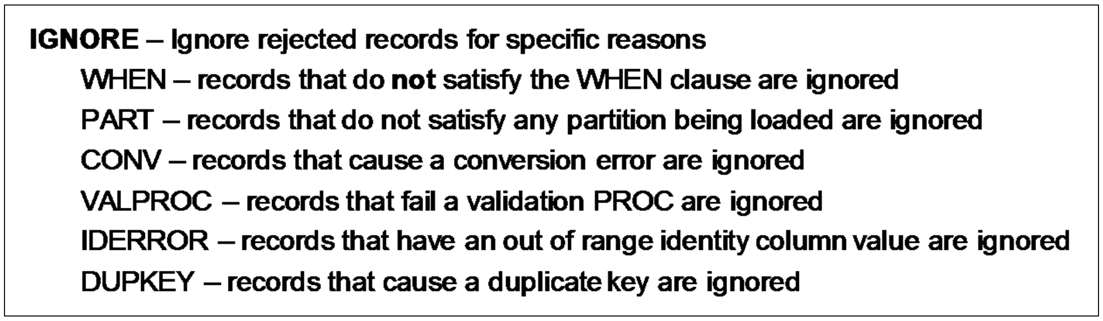
```
//SYSIN    DD *
LOAD DATA REPLACE COPYDDN(SCOPY1)
  INTO TABLE TB1(
    ID       POSITION(43:46) INTEGER EXTERNAL,
    NAME     POSITION(1:15),
    CITY     POSITION(17:28)
                CONSTANTIF(ZIP='95037')
                CONSTANT('Morgan Hill'),
    STATE    POSITION(30:31)
                CONSTANTIF(STATE='  ')
                CONSTANT('CA'),
    ZIP      POSITION(34:38),
    DEPT     POSITION(40:41) INTEGER EXTERNAL
                CONSTANTIF(DEPT=X'F4F0')
                CONSTANT(X'F3F0'),
    MGR      POSITION(48:57)
                CONSTANTIF(DEPT=X'F4F0')
                CONSTANT('DWIDAR'),
    EDATE    CONSTANT(CURRENT DATE))


//SYSREC DD *
Patrick Malone  Campbell               95008 50 2500 LEUNG
Ellen Zhao         Cupertino               94087 30 3000 DWIDAR
Craig Friske                          CA       95037 10 1000 HEGLAR
/*

+------------------------------------------------------------------------------------------------+
|      NAME       |      CITY       | STATE  |  ZIP  | DEPT |   MGR      |   EDATE      |
+------------------------------------------------------------------------------------------------+
| Patrick Malone  | Campbell        | CA     | 95008 | 30     | DWIDAR | 2018-02-27 |
| Ellen Zhao      | Cupertino       | CA     | 94087 | 30     | DWIDAR | 2018-02-27 |
| Craig Friske    | Morgan Hill     | CA     | 95037 | 10     | HEGLAR | 2018-02-27 |
+------------------------------------------------------------------------------------------------+
```

The CITY column was filled in with the appropriate city for the single zip code city of `95037`. In addition, we know that all the data is being loaded for the state of California, although the input isn’t consistently marked as such. Department 50 was eliminated and combined with Department 30, so those changes were handled as well. Finally, the effective date is set to the date on which the rows were loaded.

1.1.2 IGNORE keyword

LOAD processing generally expects all data rows to be good rows from the input stream that is loaded into the table, so there shouldn't be many errors. However, if errors occur, there is the ability to detect them and discard them into a data set for further handling after the **LOAD** job finishes. The `DISCARD N` keyword recognizes that there might be a limit as to how many errors can be examined and processed in the discard data set, so at some point with too many errors this parameter allows processing to stop.

`IGNORE` gives a lot more flexibility with the handling of different error situations that may occur with the input data. Now with `IGNORE` and `DISCARD`, there is even better control for handling errors and filtering out input data rows so they are ignored instead of treated as errors. Figure 1-1 shows a list of the options that can be used to ignore input records.



IGNORE – Ignore rejected records for specific reasons

- **WHEN – records that do not satisfy the WHEN clause are ignored**
- **PART – records that do not satisfy any partition being loaded are ignored**
- **CONV – records that cause a conversion error are ignored**
- **VALPROC – records that fail a validation PROC are ignored**
- **IDERROR – records that have an out of range identity column value are ignored**
- **DUPKEY – records that cause a duplicate key are ignored**

Figure 1-1 Options to ignore input records

The `WHEN` clause is one that can be used with `IGNORE` for filtering of input data. Suppose that there is a single data set or stream of input records, but we are only interested in loading a subset of rows (for example, entries for Californians). The following conditions could be specified (Example 1-2).

Example 1-2 WHEN clause

```
//SYSIN    DD *
LOAD DATA REPLACE COPYDDN(SCOPY1)
    IGNORE(WHEN)
    INTO TABLE TB1
     WHEN STATE = 'CA'          (
       (Same as previous example)
                                               )
/*

//SYSREC DD *
Patrick Malone   Campbell              95008 50 2500 LEUNG
Ellen Zhao       Cupertino             94087 30 3000 DWIDAR
Craig Friske                        CA 95037 10 1000 HEGLAR
/*
```

For this example, there is 1 entry for `'CA'`, but two don't satisfy the `WHEN` criteria with a value of blanks. Without `IGNORE`, the rows not satisfying the criteria would be discarded and written to the DISCARD data set (if specified). However, with `IGNORE(WHEN)`, the rows not satisfying the criteria will be bypassed. The messages during **LOAD** describing the actions are as follows:

```
DSNU304I  -  (RE)LOAD PHASE STATISTICS - NUMBER OF RECORDS=1 FOR TABLE SYSADM.TB1
DSNU1150I - (RE)LOAD PHASE STATISTICS -  NUMBER OF INPUT RECORDS NOT LOADED=2
```

1.1.3 DATE and TIMESTAMP formats

Until recently, the `LOAD` utility only supported four different formats: the International Standards Organization standard, the IBM United States standard, the IBM European standard, and the Japanese industrial standard. In addition, one could define a local format during installation.

The original supported formats are shown in Example 1-3.

Example 1-3 Original date and time formats

```
Date  Time
dd.mm.yyyy  hh.mm.ss   (ISO standard)
mm/dd/yyyy   hh.mm AM or PM  (US standard)
yyyy-mm-dd   hh.mm.ss  (EU standard)
dd.mm.yyyy  hh:mm:ss  (JIS standard)
Any local format defined when DB2 was installed
```

Today there are a lot more formats being specified by these standards, so it was incumbent on **`LOAD`** to expand the accepted list of formats so that data can be easily ingested into Db2 without first massaging it. As a result, starting in V11, the following formats are now supported (Example 1-4).

Example 1-4 Supported date and time formats

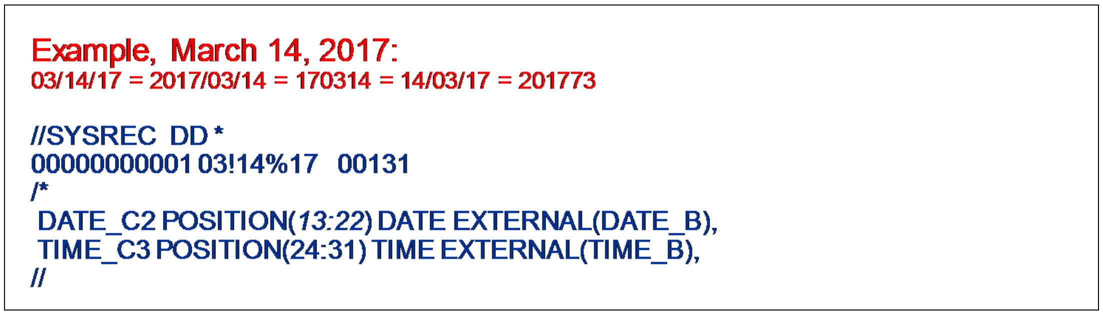
```
DATE_A  mm-dd-yyyy          DATE_I  mmddyyyy
DATE_B  mm-dd-yy            DATE_J  mmddyy
DATE_C  yyyy-mm-dd          DATE_K  yyyymmdd
DATE_D  my-mm-dd            DATE_L  yymmdd
DATE_E  dd-mm-yyyy          DATE_M  ddmmyyyy
DATE_F  dd-mm-yy            DATE_N  ddmmyy
DATE_G  yyyy-ddd            DATE_O  yyyyddd
DATE_H  yy-ddd              DATE_P  yyddd
```

In addition, there is an array of time and time-stamp formats that are accepted as well (Example 1-5).

Example 1-5 Other accepted date and time formats

```
TIME_A  hh.mm.ss            TIMESTAMP_A  yyyy-mm-dd-hh.mm.ss
TIME_B  hh.mm               TIMESTAMP_B  yyyy-mm-dd-hh.mm.ss.nnnnnn
TIME_C  hh.mm AM or          TIMESTAMP_C  yyyymmddhhmmss
         hh.mm PM
TIME_D  hhmmss              TIMESTAMP_D  yymmddhhmmss
TIME_E  hhmm                TIMESTAMP_E  yyyymmddhhmmssnnnnnn
                            TIMESTAMP_F  yymmddhhmmssnnnnnn
```

With all the different formats available, there are many representations that have the same semantic, as shown by the many ways to represent May 14th, 2017 (Figure 1-2 on page 5). Note that on the LOAD input statement, the column will have DATE, TIME, or TIMESTAMP specified as external, along with the specific type of external format that is being used.



Example, March 14, 2017:
03/14/17 = 2017/03/14 = 170314 = 14/03/17 = 201773

```
//SYSREC  DD *
00000000001 03!14%17   00131
/*
 DATE_C2 POSITION(13:22) DATE EXTERNAL(DATE_B),
 TIME_C3 POSITION(24:31) TIME EXTERNAL(TIME_B),
//
```

Figure 1-2   Example format usage

1.1.4 NUMRECS specification

For optimal space and memory allocation, it’s important to know how much work is being done, and this is a function of the number of input records. Because it’s not always easy to estimate from the source of the **LOAD** utility, NUMRECS is a way to provide a good “guesstimate”. This specification is especially critical if the input is from tape, or if the source SYSREC has varying length records. In other cases, **LOAD** can come up with a reasonable estimate based on the size of the input data set.

1.2 Availability considerations

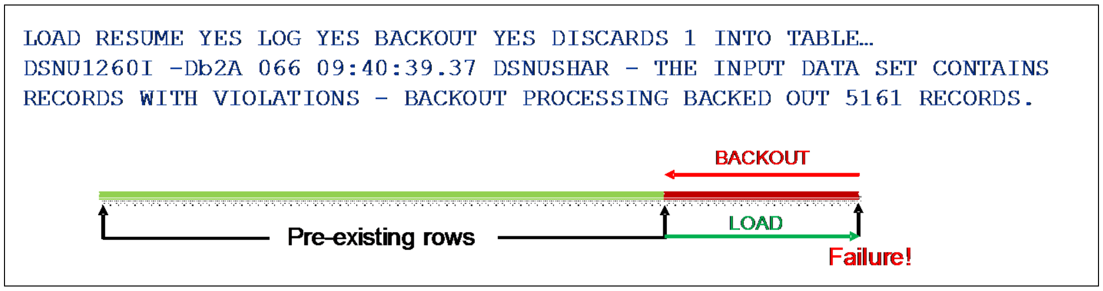
This section addresses considerations related to availability.

1.2.1 RESUME BACKOUT YES

A new keyword in Db2 V11 for LOAD RESUME YES is BACKOUT YES. The purpose is to improve availability if a LOAD SHRLEVEL NONE RESUME YES job fails with errors while loading in data rows.

Before this feature, errors during the loading of data rows would leave both the table space and indexes in a recovery pending and rebuild pending state, so there was unavailability for all the tables in the table space until after a RECOVER TABLESPACE and REBUILD INDEX were run.

Now upon failure, BACKOUT YES results in removing the data rows that were loaded, so upon completion, the table is in the same state as before the **LOAD** job was started. See an example in Figure 1-3.



```
LOAD RESUME YES LOG YES BACKOUT YES DISCARDS 1 INTO TABLE…
DSNU1260I -Db2A 066 09:40:39.37 DSNUSHAR - THE INPUT DATA SET CONTAINS
RECORDS WITH VIOLATIONS - BACKOUT PROCESSING BACKED OUT 5161 RECORDS.
```

Figure 1-3   RESUME BACKOUT YES example

1.2.2 REPLACE SHRLEVEL REFERENCE

For reference tables or read-only tables that are replaced with a new set of data rows on a periodic basis, specifying **LOAD** with SHRLEVEL REFERENCE is a good way to maximize availability, especially if there are errors for data that is loaded. That is because the target table remains available for read access while the **LOAD** processes and adds data rows and index keys to shadow data sets. The only unavailability period, similar to REORG SHRLEVEL REFERENCE or CHANGE, is during the SWITCH phase, when the shadow objects are switched over to become the active objects. See Figure 1-4.

LOAD SHRLEVEL REFERENCE has many of the same controls found with **REORG** to help control availability and the switching of data sets. This includes the DRAIN_WAIT, RETRY, RETRY_DELAY, and SWITCH keywords. The operation of those keywords and tuning is discussed in the **REORG** section on availability.

In addition, the NOCHECKPENDING keyword allows the **LOAD** to complete loading rows that are children within a referential constraint without going through the ENFORCE phase, and without leaving the table space in a check pending (CHKP) state. This option is only allowed when ENFORCE NONE is specified, and of course, care should be taken that this option should only be specified if it’s known that there are no referential integrity constraint violations with the loaded data.

For SHRLEVEL REFERENCE, the LOG NO option will always be enforced.

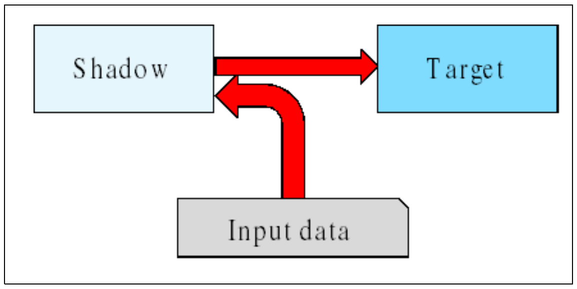


Figure 1-4 REPLACE SHRLEVEL REFERENCE

Clone Tables or SHRLEVEL REFERENCE?

The function provided by LOAD SHRLEVEL REFERENCE is similar to using clone tables, then executing the EXCHANGE statement. Some of the differences between these two options include the following aspects:

- The input for a clone table is more flexible. It can be done using the **LOAD** utility or by SQL **INSERT**s, and the format of the incoming data may need that flexibility.
- Clone tables can be populated independently over time, and then the switch is done independent of the time it is populated, if needed. With LOAD REPLACE the loading of the data and switchover are tightly coupled within the same utility job.
- **LOAD** has more options to deal with contention by allowing the DRAIN, DRAIN_RETRY, RETRY, and SWITCHTIME specifications.
- Some Db2 functions, such as RENAME, are not allowed for Clone tables.

1.2.3 SHRLEVEL CHANGE

The SHRLEVEL CHANGE option of **LOAD** provides for superior availability while running concurrently with applications, but there is a tradeoff when compared with SHRLEVEL NONE or REFERENCE.

Here are some considerations:

- Advantages
  - Availability as a claim writer, which does inserts and serializes with other applications that are running concurrently.
  - Availability by not needing an image copy because `LOG YES` processing is recoverable. Without `SHRLEVEL CHANGE`, an inline image copy is an option for `LOAD REPLACE` and, more recently, `RESUME YES`.
  - Avoidance of space required for an external sort.
- Disadvantages
  - Performance won't be as good for loading lots of data. In addition to logging overhead, **`INSERT`**s are done instead of **`LOAD`**s of the data rows, and index key processing can be extremely slow if keys are out of order, because of searching for key insertions and index page splitting.
  - **`LOAD`** failures result in a rollback to the last commit point. These commit points are determined internally by the utility, so it may be difficult to determine what rows were successfully added, and which rows were missed.

1.2.4 RESUME YES copy support

`LOAD RESUME SHRLEVE NONE` will create inline image copy after **`LOAD`** processing if `COPYDDN` or `RECOVERYDDN` was specified.

This is supported if `BACKOUT YES` is specified and the **`LOAD`** fails as well. This improves availability by not leaving a `LOG NO` job in a copy pending (`COPY`) state.

The copies can be at the table space level, or at the partition level. In addition, the `FLASHCOPY` and `FLASHCOPY CONSISTENT` allows a flash copy to be taken at the end of **`LOAD`** for `SHRLEVEL NONE`, `SHRLEVEL REFERENCE`, or `SHRLEVEL CHANGE`.

1.3 Performance considerations

The best elapsed time for a **`LOAD`** job is achieved when parallelism is used. **`LOAD`** is designed to have parallelism for loading the data rows (also known as *partition parallelism*), and parallelism for building index keys, or *index parallelism*. Other parallel tasks for creating image copies and gathering statistics might be spawned as well. This section focuses on understanding how partition and index parallelism can be exploited.

1.3.1 Partition parallelism: Multiple input data sets

If the target table space is partitioned with limit keys, such as non-UTS or Partitioned By Range, the most optimal form of partition parallelism can occur. This occurs if there are separate data sets or input streams, one for each partition. **`LOAD`** can then allocate separate task sets for each data partition to read the input records, convert the data to Db2 internal format, and load the rows into the appropriate partitions. Figure 1-5 on page 8 shows an example with two data partitions.

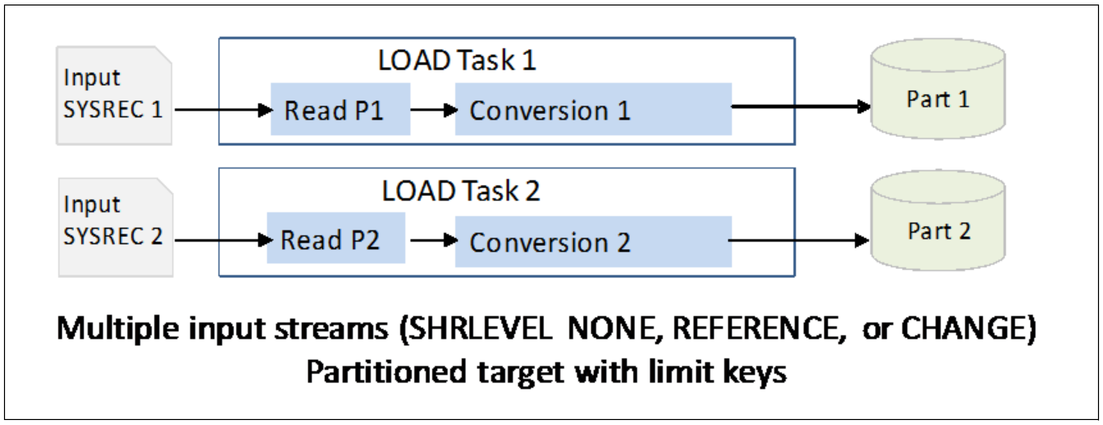


Figure 1-5 Two data partitions

When **LOAD** detects multiple input data sets or streams, partition parallelism is automatically activated. Details are shown with the following messages:

```
DSNU364I  - PARTITIONS WILL BE LOADED IN PARALLEL,  NUMBER OF TASKS = 2
DSNU3345I - MAXIMUM PARTITION PARALLELISM IS 2 BASED ON NUMBER OF INPUT FILES
```

1.3.2 SHRLEVEL NONE or REFERENCE Partition Parallelism: Single DS

`LOAD SHRLEVEL NONE` and `SHRLEVEL REFERENCE` support partition-level parallelism from a single data set or input stream. In this scenario, the records for all partitions are mixed together in no particular order. Parallelism involves a single read task passing rows to a number of multiple tasks converting to internal format, and then passing them back to a single task that loads the rows into the appropriate partitions, as shown in Figure 1-6.

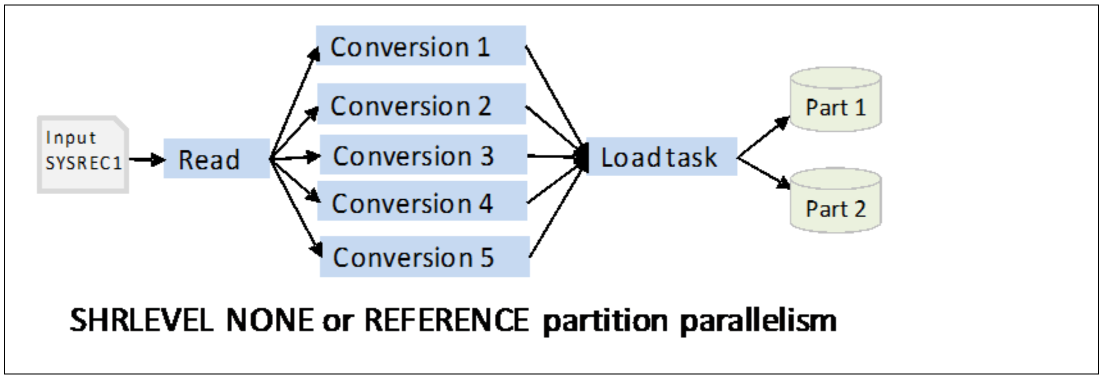


Figure 1-6 SHRLEVEL NONE or REFERENCE partition parallelism

For classic partitioned or partitioned by range table spaces, the rows are loaded in the appropriate partitions according to each row's partitioning key. For partitioned-by-growth table spaces, the rows are loaded in the current partition until filling it, and then move to the next partition.

Partition parallelism for a single data set is enabled by specifying the `PARALLEL n` keyword, where `n` is a constraint for the maximum number of tasks. Specify `PARALLEL 0` to request the maximum parallelism possible given the number of CPUs, memory, and DD statements specifications. The `DSNU1177I` message indicates that partition parallelism is activated with the number of conversion tasks shown.

```
DSNU1177I  -  TABLE SPACE WILL BE LOADED IN PARALLEL , NUMBER OF TASKS = 7
```

1.3.3 Index parallelism

For `LOAD SHRLEVEL NONE` or `REFERENCE` index parallelism is also supported when there are two or more indexes. This is supported for all table space types partitioned or non-partitioned.

Figure 1-7 shows the index parallelism, which ties in with the partition parallel diagram (above) with the single load task. In addition to the single **`LOAD`** task write data rows to each partition, it will also extract keys and send to the appropriate internal pipes for building parallel indexes.

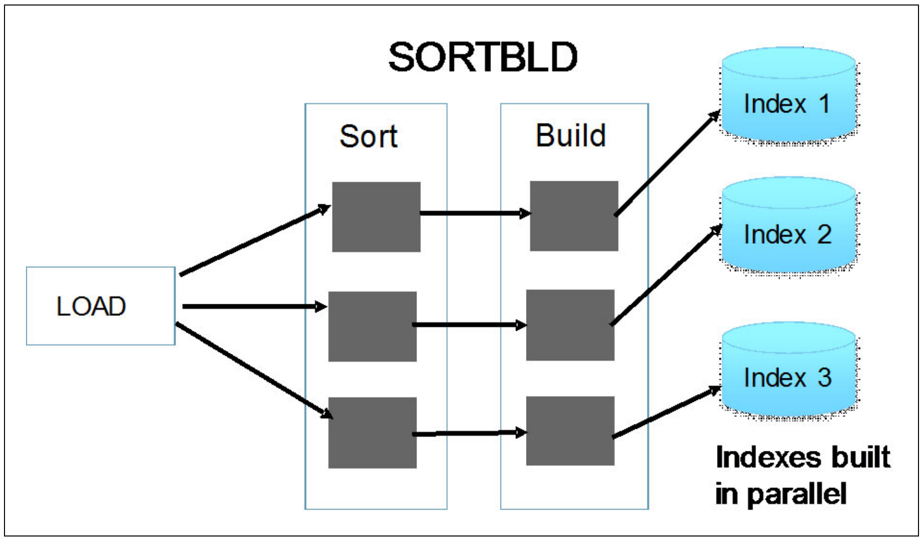


Figure 1-7 Index parallelism

The parallel index build is enabled if `SORTDEVT` is specified, and when **`LOAD`** can estimate the number of keys from the number of input records. If the source file is on disk, the number can be estimated. If the source is on tape, in the JCL stream, or remote via DRDA, `NUMRECS n` must be provided.

When activated, there will be a `SORTBLD` phase as shown, and the `DSNU3345I` message is issued showing the actual degree of parallelism used. In addition, `DSNU397I` messages will be issued indicating what resource is constraining parallelism, if any (Example 1-6). Sometimes it is possible to identify the constraint, and then make appropriate changes to exploit a higher degree of parallelism.

Example 1-6 Messages showing degree of parallelism and constraints

```
DSNU3345I - MAXIMUM UTILITY PARALLELISM IS 55 BASED ON NUMBER OF PARTITIONS AND
INDEXES
DSNU397I    - NUMBER OF TASKS CONSTRAINED BY MISSING  SORTDEVT TO 51
DSNU397I    - NUMBER OF TASKS CONSTRAINED BY CPUS TO  13
DSNU397I    340 15:52:23.82 DSNURPIB - NUMBER OF TASKS CONSTRAINED BY PARALLEL
KEYWORD TO 10
DSNU395I   - INDEXES WILL BE BUILT IN PARALLEL, NUMBER OF TASKS =
```

1.3.4 SHRLEVEL CHANGE Partition Parallelism: Single input data set

When `LOAD SHRLEVEL CHANGE` is used for partition-level parallelism with a single input data set or stream, there is a single read task that pipes the input records to a number of parallel conversion tasks. These conversion tasks will do **`INSERT`**s for the input rows directly into the appropriate partitions.

For classic partitioned and partitioned-by-growth table spaces, the partition key is used to directly insert the row into the proper partition according to the limit keys for each partition. For partition by growth table spaces, the inserts will all insert into the first partition until full, and then they will start inserting in parallel to the next partition.

1.3.5 FORMAT INTERNAL

The `FORMAT INTERNAL` option is a fast method of loading rows into a Db2 table. It's fast because it avoids the overhead of data conversion from external to internal format because the data is already in internal format. Typically, the source of internal format data is generated from a source table using the **`UNLOAD`** utility for `FORMAT INTERNAL`; however, it is possible to format the source data internally as well.

1.3.6 DRDA fast load

Db2 12 introduces DRDA fast load, which enables quick and easy loading of data from files residing on distributed clients. The Db2 Call Level Interface (CLI) APIs and Command Line Processor (CLP) have been enhanced to support remote loading of data to Db2 for z/OS. This new feature is supported in all Db2 client packages.

This new method of loading from distributed clients is much faster; however this option should be used very carefully, because error conditions during **`LOAD`** may be difficult to resolve.

1.3.7 zIIP offload

The **`LOAD`** utility supports zIIP offload for the `RELOAD`, `SORT`, `BUILD`, and `SORTBLD` phases. **`LOAD`** has supported zIIP offload for sorting and building indexes for quite a while, but with the more recent zIIP offload for loading the data rows in the `RELOAD` phase, an additional 50% or more of general CPU can be offloaded.

2

Reorganizing Db2 data

`REORG` is a utility for the maintenance of application data and indexes. There are a number of reasons for running this utility, but some of the most common include keeping data organized for application performance, applying changes to the table space schema, formatting data rows, or redistributing and discarding of data rows. These functions are all expected to be done with little disruption of application availability. Here’s a more detailed list:

- Materialization of schema changes:
  - Adding columns
  - Altering column data types or sizes
  - Converting to Universal Table Spaces
  - Converting to use Relative Page Numbering
  - Page size alteration
  - Resizing of page sets or partitions
  - Segment size alteration
  - Converting to or from hash page set format
  - Resizing hash space
- Data formatting:
  - BRF to RRF conversion
  - Inlining of existing LOB data
  - Extended page format conversion
  - Compressing data rows (or decompressing)
- Redistribution and removal of data rows:
  - Rebalancing rows across partitions
  - Changing partition limit keys
  - Discarding rows to a data set
- Maintaining application performance:
  - Re-establish data clustering
  - Re-establish free space
  - Consolidate pointer-overflow records
  - Remove deleted and pseudo-deleted records
  - Reorder index leaf pages and remove pseudo-deleted entries
  - Re-chunk LOB data

© Copyright IBM Corp. 2018. All rights reserved.

2.1 Availability considerations

The availability of applications while running reorganizations depends on how well the **REORG** can run concurrently with those applications. If there is an extended batch window when applications are not active, **REORG**s with any SHRLEVEL option may be able to start and finish within the window without impacting applications. However, that's not usually the case, so the suggested **REORG** is with SHRLEVEL CHANGE for maximum concurrency and application availability. The following sections describe some considerations for successful coexistence.

2.1.1 Batch windows

Many data centers have a small period of time every week or month when applications are stopped. If every Saturday night there is a window (for example, between 10 PM and 2 AM), any **REORG** running less than four hours could conceivably be run during that period and complete without any impact to applications. However, because of the large number of objects needing reorganization, shrinking batch windows, and the elapsed time of large **REORG** jobs, this has very limited use (Figure 2-1).

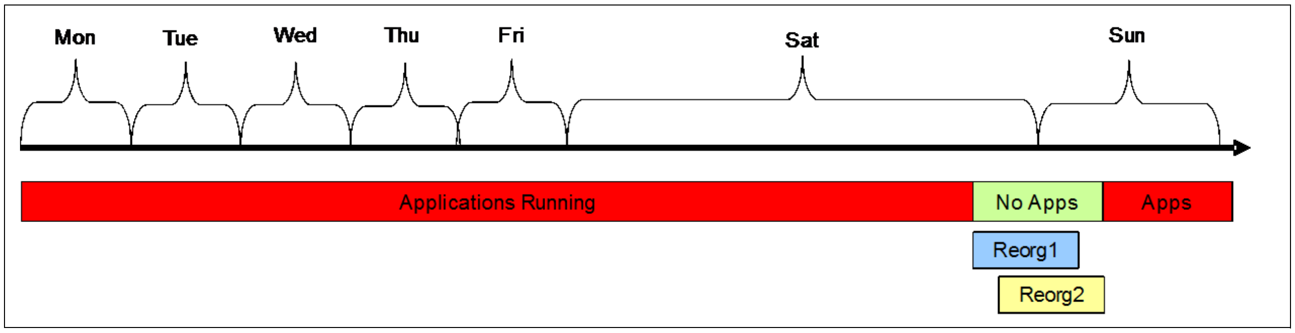


Figure 2-1 Batch windows

2.1.2 Switching phase during batch window

With SHRLEVEL CHANGE REORGs, there is only contention between reorganizations and applications at the end of the **REORG**, in a time spanning a bit longer than the SWITCH phase. Therefore, if the SWITCH outage can be scheduled to occur either in a batch window or in a way that doesn't disrupt the application, full availability can be achieved (Figure 2-2).

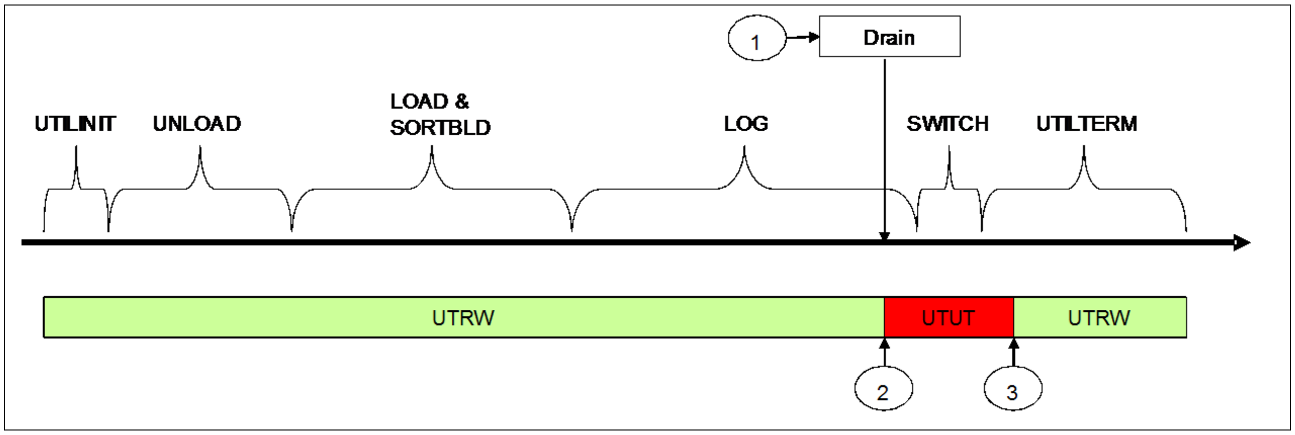
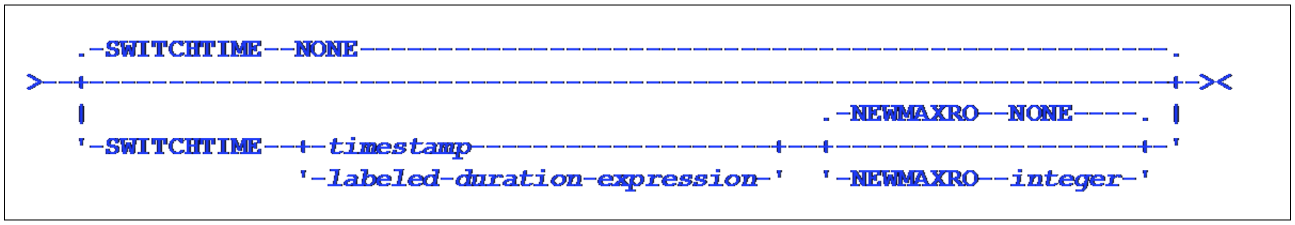


Figure 2-2 Switching phase during batch window

As illustrated in Figure 2-2 on page 12, availability is blocked when drains are acquired (1) and the objects are set into a `UTUT` state to block access in the case of any failures where the utility ABENDs. Unavailability continues until the objects are set back to `UTRW` and the drains are released (3). The object is available during most of the `UTILTERM` phase when statistics are updated in the catalog and data sets are deleted. Note that there is no quiesce or drain needed during `UTILINIT` phase.

SWITCHTIME keyword

The `SWITCH` phase outage can be controlled so that it occurs within the batch window by specifying the `SWITCHTIME` keyword. For example, a **REORG** could be started well before our batch window Saturday evening. It would remain in the `LOG` phase applying logs until the batch begins, which in our example is 10 PM. This would be specified with "`MAXRO DEFER SWITCHTIME 22:00`". When that time is reached, **REORG** will `DRAIN` and apply the last bit of logs, then `SWITCH`. Here is the syntax of the keyword allowing a `NEWMAXRO` to be specified (Figure 2-3).



```
   .-SWITCHTIME--NONE----------------------------------------------------.
>--+---------------------------------------------------------------------+-><
   |                                               .-NEWMAXRO--NONE----.  |
   '-SWITCHTIME--+-timestamp--------------------+--+-------------------+-'
                 '-labeled-duration-expression-'  '-NEWMAXRO--integer-'
```

Figure 2-3 SWITCHTIME keyword

Because the time to complete the `SWITCH` phase is usually seconds to at most minutes, these techniques with `SWITCHTIME` should be very reliable for almost any **REORG** scenario.

2.1.3 Switch phase while applications are running

In the typical case, one can't depend on having a batch window, so completing the switch phase depends on applications being "well behaved", and **REORG**s using the proper settings for minimum interferences with applications.

"Well behaved" applications claiming and locking

For availability while **REORG**s are running, applications should be designed in such a way as to maximize the ability for concurrency. We call these applications *well behaved*. Here are some areas that should be considered to be well behaved:

- Commit often. Held locks will prohibit successful draining in the `SWITCH` phase, and even if locking isn't done, and is a held claim, will prohibit **REORG** from breaking in.
- Close an open `CURSOR WITH HOLD` ahead of commit.
- Implement retry logic, allowing a failed transaction due to timeout to be retried one or more times. If coded within an application, it might behave much like the `DRAIN_WAIT` and `RETRY` options for **REORG**.

In addition, **ZPARM** settings can help concurrency and allow **REORG** to break in a successful switch. The following **ZPARM**s can be adjusted for tuning.

Long-running reader (LRDRTHLD)

The **LRDRTHLD** parameter on the DSNTIPE1 panel controls help identify readers that don't commit within a period of time. It can be set from 0 - 1439 minutes, with the default being 10 minutes. Any uncommitted reader that exceeds the specified threshold will be identified with a DSNB260 message on the console, so it's possible to terminate them (if warranted). Example 2-1 shows an example of a rogue reader.

Example 2-1 Rouge reader

```
DSNB260I  -DB2A DSNB1PCK WARNING - A READER HAS BEEN RUNNING FOR 10 MINUTES
CORRELATION NAME=SEL01
   CONNECTION ID=BATCH
   LUWID=DB2A.SYEC1DB2.D37A899CF3A3=28
   PLAN NAME=DSNTEP3
   AUTHID=SYSADM
   END USER ID=SYSADM
   TRANSACTION NAME=SEL01
   WORKSTATION NAME=BATCH
```

Resource Timeout field (IRLMRWT)

The **IRLMRWT** parameter on the DSNTIPE panel controls the number of seconds to wait for an unavailable resource before "timing out" and failing. The default is 30 seconds. Essentially, the unavailability or *SWITCH outage* time must be less than this value, or the application transaction will fail.

Therefore, if the value is increased, more **REORG**s can complete their switch phases without application timeouts occurring. For example, if SEL01 couldn't acquire a claim on table space DBB.TS01 within the required timeout threshold, the following failure would be output to the console (Example 2-2).

Example 2-2 Correlation-ID of SEL01

```
CORRELATION-ID=SEL01
   CONNECTION-ID=BATCH
   LUW-ID=DB2A.SYEC1DB2.D37A96041AE9=48
   REASON 00C900BA
   TYPE 00002000
   NAME DBB     .TS01
```

The failing transaction would receive the following failure:

```
DSNT408I SQLCODE = -911, ERROR:  THE CURRENT UNIT OF WORK HAS BEEN ROLLED BACK DUE
TO DEADLOCK OR TIMEOUT. REASON 00C900BA, TYPE OF RESOURCE 00002000, AND RESOURCE
NAME DBB     .TS01
```

2.1.4 The DRAIN_WAIT, RETURN, and RETRY_DELAY keywords

The **REORG** for a table space object (or indexes) has various keywords that can help tune the utility for success without application availability. The DRAIN_WAIT, RETRY, and RETRY_DELAY keywords allow tuning of a particular **REORG** that is run on a specific table space or index.

DRAIN_WAIT integer

The keyword specifies the number of seconds **REORG** will wait trying to acquire drains before timeout (similar to IRLMWAIT for applications). The default is the IRLMWAIT value. If **REORG** can't break in and acquire the needed drains for the SWITCH within the specified number of seconds, it will fail with a DSNU1122I message on the console, and any drains held are released so the application has full availability.

RETRY integer

This is the number of times **REORG** with attempt to acquire the drains if there is a failure or timeout because the DRAIN_WAIT time was exceeded. The default is the UTIMOUT value (which defaults to 6).

RETRY_DELAY integer

This is the number of seconds **REORG** waits before attempting to retry acquiring drains on all objects again. The default is the smaller of the following two values: *DRAIN_WAIT* value x *RETRY* value, *DRAIN_WAIT* value x 10.

An example of an application (perhaps not well behaved) holding claims so that **REORG** can't break in follows (Figure 2-4).

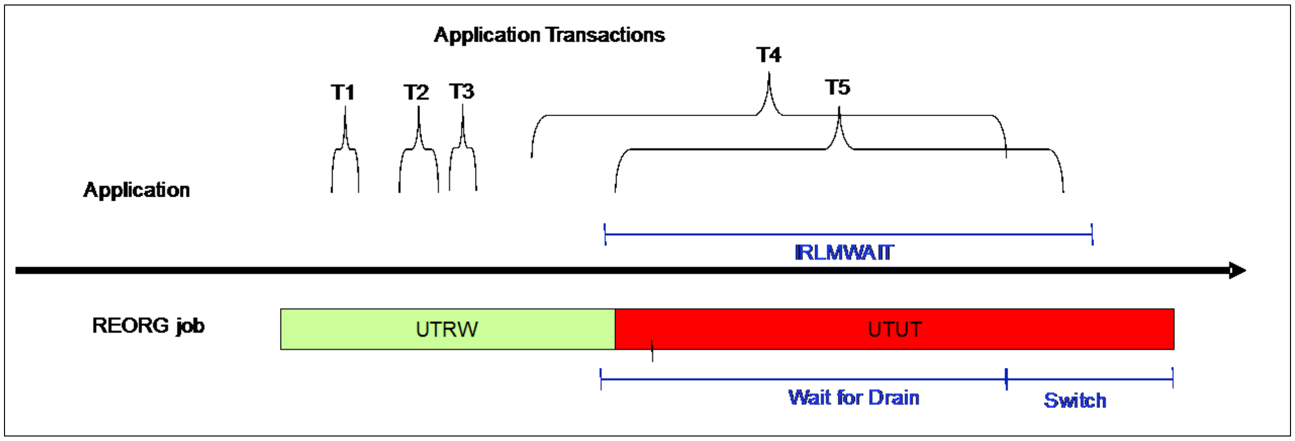


Figure 2-4 Application holding claims

In this case, T4 never commits, and **REORG** is trying to obtain drains, and T5 is waiting behind **REORG**'s request. With DRAIN_WAIT specified, **REORG** will back out of the way deferring to T4 and T5. The follow message is issued on the console (Example 2-3).

Example 2-3 Console message

```
DSNU1122I -DB2A DSNURSWD - JOB REORGJOB PERFORMING
REORG  WITH UTILID REORGJ1 UNABLE TO DRAIN DBB.TS01.
      RETRY 1 OF 6 (i.e. retry) WILL BE ATTEMPTED IN 30 (retry_delay) SECONDS
```

When this message is received on the console, the following message will be written to the job output indicating the holder of the resource that prohibited **REORG** from obtaining it (Example 2-4).

Example 2-4 Job output message

```
NAME     TYPE PART  STATUS            CONNID   CORRID      CLAIMINFO
-------- ---- ----- ----------------- -------- ------------ --------
TS01     TS         RW,UTRO           BATCH    SEL01         (CS,C)
```

The highest availability can be achieved when applications and **REORG**s are designed to work in concert. For applications, this means `RELEASE COMMIT` applications that are well behaved because commits are done immediately after transactions complete, generally within seconds, which allows **REORG** to more easily break in. On the **REORG** side, `DRAIN_WAIT` must be a small enough value so that **REORG** allows applications to wait for or hold locks without extending beyond the `IRLMWAIT` value, therefore timing out.

2.1.5 Minimizing application failures proper REORG settings

The total outage for the **REORG** job is approximately the time needed to 1) obtain drains, 2) complete the log apply, 3) complete the inline image copy, and 4) Update the catalog and directory I/J entries to switch over to the shadow data sets. Therefore, this total time should be less than the `ILRMWAIT` value. This is illustrated in Figure 2-5.

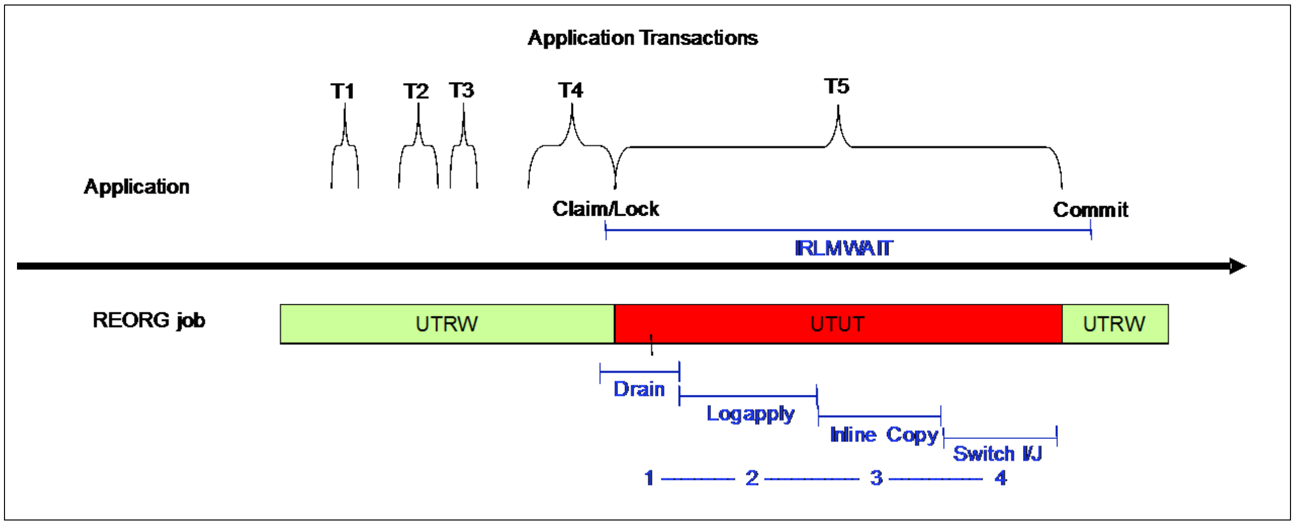


Figure 2-5 Application transactions with REORG job

As shown in Figure 2-5, a **REORG** is running currently with an application that has five transactions (T1 - T5). Transactions. T1, T2, and T3 run without any interference from the **REORG** because it is unloading, reloading, sorting, and building indexes without any drains. However, when at the end of the log apply phase, the decision is made to `DRAIN` objects for the switch. The drain request takes time because T4 hasn't completed and committed.

However, the time waiting is less than the `DRAIN_RETRY`, so the drain is acquired. After **REORG** obtains all the drains, T5 is blocked until **REORG** releases the drains again at the beginning of the `UTILTERM` phase. The Drain, Logapply, Inline Copy, Switch I/J, and de-DRAIN happens within the `IRLMWAIT` period, so there is no disruption in application availability.

The key to success is completing all four `SWITCH` phase components in less than the `IRLMWAIT` time. This can be measured for any **REORG** job by specifying `DIAGNOSE TYPE(101,102)` before the **REORG** statement. This will print out elapsed time measurements in very granular details during the critical `SWITCH` phase.

DRAIN time

This is controlled by the `DRAIN_WAIT` parameter. The smaller the value, the less time taken. However, with large numbers of concurrent transactions happening in parallel, the smaller this value, the less likely **REORG** can "break in" and acquire all of the drains. Instead of waiting for just T4 to commit, all active transactions on all partitions must commit for **REORG** to break in.

DRAIN depends on the number of objects needing drains, and how long it takes to acquire each drain. The advice is to always use DRAIN ALL (not DRAIN WRITERS), because experience has shown little value using DRAIN WRITERS when trying to maximize availability.

If a subset of partitions is reorganized for a table space that has non-partitioned secondary indexes (NPSIs), then it is suggested that DRAIN_ALLPARTS be specified. This helps avoid undetected deadlock conditions when an application can claim a logical partition of an NPSI before accessing the data while the **REORG** drains in a different order.

When all partitions are reorganized, one drain lock on all partitions is obtained. After the table space is drained, the indexes are easily drained with no contention, because applications with indexes always do data-first claiming of the table space, even if they are doing index-only access.

When a subset of partitions is reorganized, the drain requests are done in parallel, but no drains are needed.

This time can be obtained from DIAGNOSE TYPE(101,102) under this entry:

```
INTERVAL = LOG DRAIN ALL   ELAPSED TIME (SEC) = nnnn.nnnn
```

Logapply time

This is controlled by the MAXRO keyword parameter specification for **REORG**. MAXRO is a bit of a misnomer going back to the original implementation, technically standing for *Maximum RO* time. MAXRO really represents the amount of time that **REORG** estimates it will need to complete log processing. This estimate comes from the utility remembering the log apply rate during each log iteration. A MAXRO value of *n* indicates that **REORG** believes it can apply all existing log records left to apply within *n* seconds.

This value can be raised or lowered. A low value ensures that a shorter total SWITCH phase outage as seen by our diagram; however, the flip side is that too small a value results in never attempting to drain because there are always too many logs to apply to finish within the MAXRO time specified, so the LOG phase will continue forever.

This time can be obtained from DIAGNOSE TYPE(101,102) under the entries for the following intervals:

- INTERVAL = READ LOG RECORDS
- INTERVAL = SORT LOG RECORDS OLD
- INTERVAL = TRANSLATE LOG RECORDS
- INTERVAL = SORT LOG RECORDS NEW
- INTERVAL = DATALOGnn, where nn represents parallel log apply tasks

Final Inline Copy time

The inline copy is constantly being done (non-FlashCopy) during the reload phase and each log iteration. However, after the DRAIN is acquired and the last bit of logs are applied, the final set of changed pages that are part of the inline copy must be captured and written out.

This time can be obtained from DIAGNOSE TYPE(101,102) under the following entry:

```
INTERVAL = FINAL INCREMENTAL
```

SWITCH I/J time

This is the updating of the catalog and directory with all the new SYSTABLEPART PREFIX and SYSINDEXPART PREFIX values of I and J representing the shadow data sets becoming the active data sets. This will vary according to the number of data sets that are processed.

For a table space with a large number of objects processed (for example, 4096 data partitions and a partitioned index), this will be at least 8192 updates. The elapsed time can be reduced by breaking the **REORG** into multiple reorganizations, each handling a subset of the partitions.

This time can be obtained from `DIAGNOSE TYPE(101,102)` under the entry for the following interval:

```
INTERVAL = SWITCH   ELAPSED TIME (SEC) = n.nnn
```

In Db2 V11, attention was given to reducing switch duration. Internal measurements show that the `SWITCH I/J` duration in our environment to rename 100 objects is about 2 seconds, and for 1000 objects is about 16 seconds (Figure 2-6).

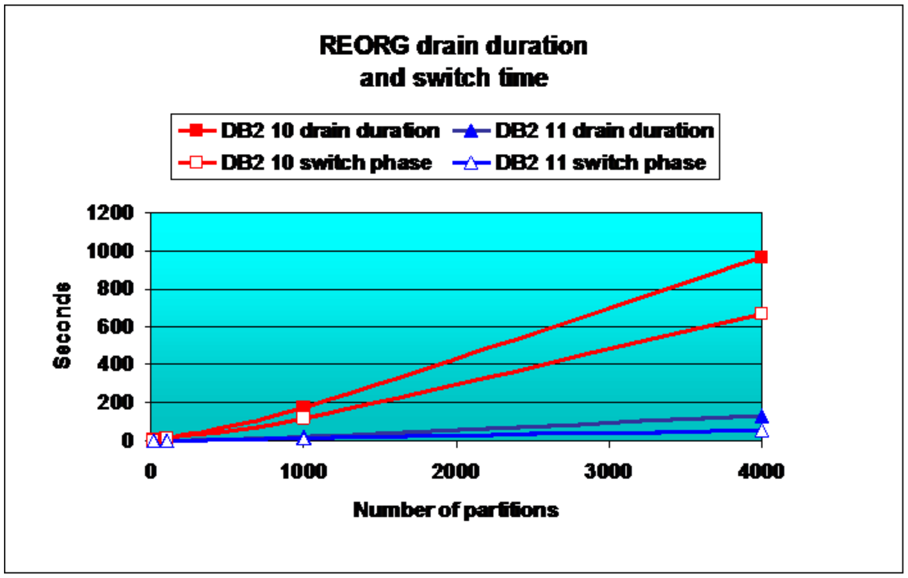


Figure 2-6 REORG drain duration and switch time

2.1.6 Recommendation for settings

In summary, if the components of the `REORG SWITCH` outage are less than the application timeout value, there should be full availability. It can be represented as follows:

```
DRAIN_WAIT + MAXRO + IC + SWITCH duration < IRLMRWT
```

The `IRLMRWT` has typically been tuned for applications and probably not easily changed, so the other variables can be measured and adjusted as needed, and as described previously. For the majority of objects and environments, full availability should be achieved when running applications on Db2 V11 and V12.

There can be certain applications where there are spikes of update and insert activity that are so heavy that the `DRAIN_WAIT` and `MAXRO` can't be set low enough to break in. In this case, there will be periodic lulls in activity, so a judicious setting of `RETRY` and `RETRY_DELAY` can usually break in.

2.1.7 Mapping table control (locking contention)

REORG SHRLEVEL CHANGE requires the use of a mapping table. As of Db2 V11, this can automatically be created and dropped for usability. However, this feature does implicit DDL operations, and the locks acquired on the Data Base Descriptor may result in contention.This is especially true with **REORG** jobs running concurrently, unless attention is given to where the mapping table is created.

The default is to create the table in the same data base as the object being reorganized. To provide more flexibility in placement of this table, the following specifications are available:

- Zparm REORG_MAPPING_DATABASE: Specifies the database where the table is to be created for any **REORG** job if MAPPING_DATABASE is not specified.
- **REORG** keyword MAPPINGDATABASE: Specifies the database where the mapping table will be created.
- **REORG** keyword MAPPINGTABLE: Specifies a predefined mapping table to be used. This is less usable, but it does allow specific parameters if object placement is important.

2.2 Performance considerations

This section discusses the following performance considerations:

- Parallelism
- SORTNPSI
- Inline copies and flash copies
- CPU and zIIP offload
- DISCARDING rows during REORG
- Sort products

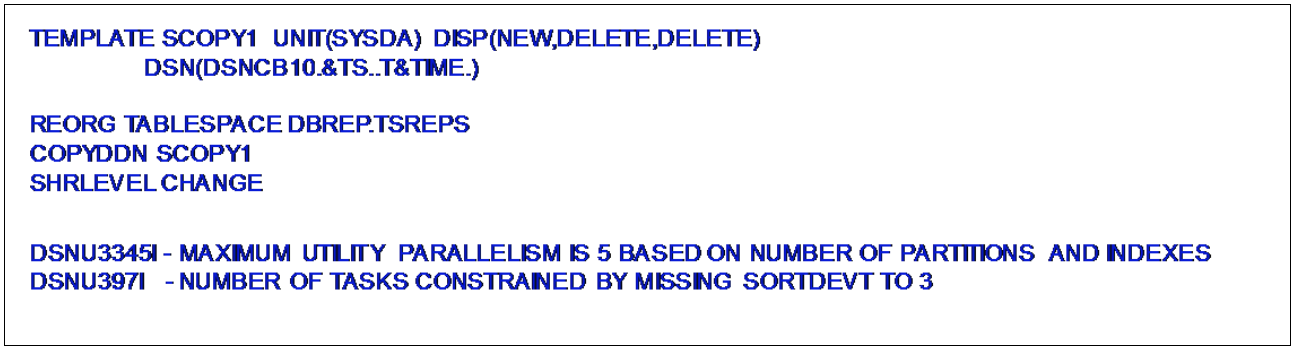
2.2.1 Parallelism

The REORG utility uses parallelism of tasks to reduce the elapsed time for completion of the job. For Partitioned by Range table spaces, unloading, sorting, and reloading of data rows uses partition parallelism. Building indexes uses parallelism for sorting and building indexes. The log apply process also has parallelism, as well as inline statistics collection.

Each task used for parallelism requires space in memory, so there are limits to the amount of parallelism when reorganizing an object. The unload, sorting, and loading of data rows is restricted to a task per partition. The sorting and building of index keys is restricted to a task per index. The number of available CPUs and memory are other factors that limit the amount of parallelism possible. Note that partition parallelism is only supported for classic and PBR table spaces.

The amount of parallelism for a **REORG** is shown with the DSNU397I message, and this message also identifies the restricting resource, whether it is memory, CPUS, JCL DD specifications, and so on. The DSNU3345I message shows the maximum parallelism possible if there weren't any restricting resources.

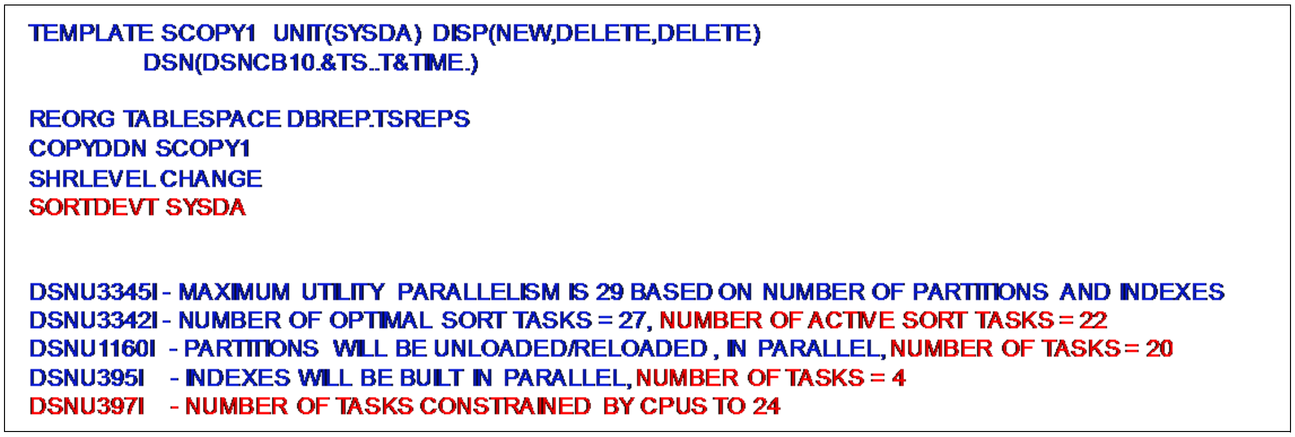
Parallelism can also be capped with a high value using the PARALLEL keyword, or by specifying the zparm PARAMDEG_UTIL. This restriction increases the elapsed time for the **REORG** to complete, but it helps govern the amount of CPU taken by a single **REORG** utility so that applications don't become resource starved for CPUs. Figure 2-7 on page 20 shows a sample of a **REORG** job with 25 partitions and a single partitioned index.



```
TEMPLATE SCOPY1  UNIT(SYSDA)  DISP(NEW,DELETE,DELETE)
            DSN(DSNCB10.&TS..T&TIME.)

REORG TABLESPACE DBREP.TSREPS
COPYDDN SCOPY1
SHRLEVEL CHANGE


DSNU3345I - MAXIMUM  UTILITY  PARALLELISM IS 5 BASED ON NUMBER OF PARTITIONS  AND INDEXES
DSNU397I   - NUMBER OF TASKS CONSTRAINED BY MISSING  SORTDEVT TO 3
```

Figure 2-7   REORG join with 25 partitions and a single partition index

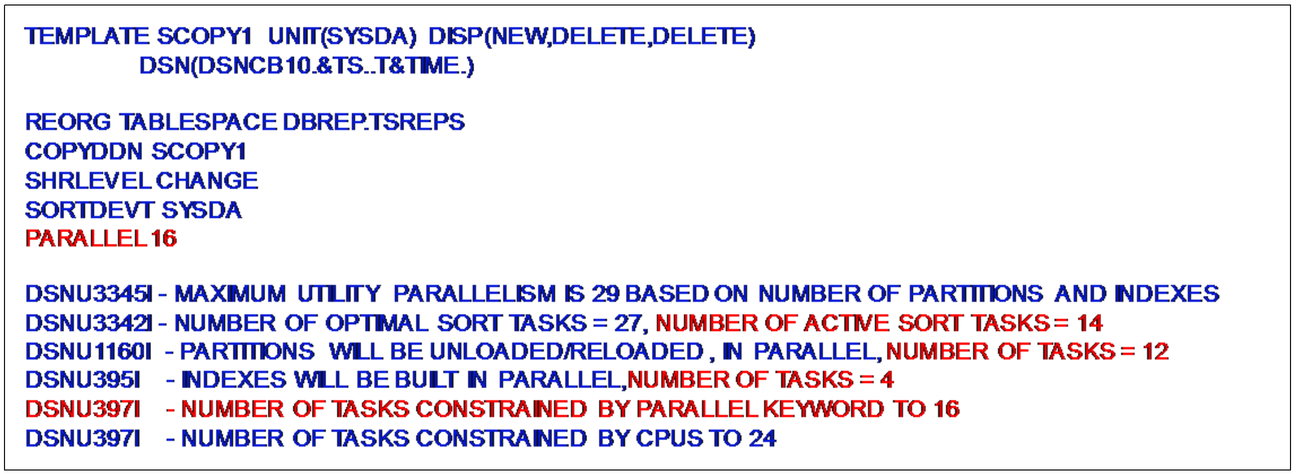
`SORTDEVT` is missing, so parallelism really isn't activated. The total number of tasks specified with the `PARALLEL` specifications are approximations, because sometimes a set of tasks must be started together in pairs or in triplets to function properly. The problem of no parallelism is remedied by adding the missing keyword `SORTDEVT`, and now the job and output looks like Figure 2-8.



```
TEMPLATE SCOPY1  UNIT(SYSDA)  DISP(NEW,DELETE,DELETE)
            DSN(DSNCB10.&TS..T&TIME.)

REORG TABLESPACE DBREP.TSREPS
COPYDDN SCOPY1
SHRLEVEL CHANGE
SORTDEVT SYSDA


DSNU3345I - MAXIMUM  UTILITY  PARALLELISM IS 29 BASED ON  NUMBER OF PARTITIONS  AND INDEXES
DSNU3342I - NUMBER OF OPTIMAL  SORT TASKS = 27, NUMBER OF ACTIVE SORT TASKS = 22
DSNU1160I  - PARTITIONS  WILL BE UNLOADED/RELOADED , IN  PARALLEL, NUMBER OF TASKS = 20
DSNU395I   - INDEXES WILL BE BUILT IN  PARALLEL, NUMBER OF TASKS = 4
DSNU397I   - NUMBER OF TASKS CONSTRAINED  BY CPUS TO 24
```

Figure 2-8   Adding SORT DEVT

The number of tasks has now substantially increased, and the parallelism is constrained from the maximum possible of 29 for this table space and indexes because of the number of CPUs. However, because we don't want **REORG** to use too much of the CPU resources, we can choose to limit the job using the `PARALLEL` keyword, as follows (Figure 2-9).



```
TEMPLATE SCOPY1  UNIT(SYSDA)  DISP(NEW,DELETE,DELETE)
            DSN(DSNCB10.&TS..T&TIME.)

REORG TABLESPACE DBREP.TSREPS
COPYDDN SCOPY1
SHRLEVEL CHANGE
SORTDEVT SYSDA
PARALLEL 16

DSNU3345I - MAXIMUM  UTILITY  PARALLELISM IS 29 BASED ON  NUMBER OF PARTITIONS  AND INDEXES
DSNU3342I - NUMBER OF OPTIMAL  SORT TASKS = 27, NUMBER OF ACTIVE SORT TASKS = 14
DSNU1160I  - PARTITIONS  WILL BE UNLOADED/RELOADED , IN  PARALLEL, NUMBER OF TASKS = 12
DSNU395I   - INDEXES WILL BE BUILT IN  PARALLEL, NUMBER OF TASKS = 4
DSNU397I   - NUMBER OF TASKS CONSTRAINED  BY PARALLEL KEYWORD  TO 16
DSNU397I   - NUMBER OF TASKS CONSTRAINED  BY CPUS TO 24
```

Figure 2-9   PARALLEL keyword

Parallelism is now exploited to improve elapsed time, but it's also constrained with PARALLEL 16 to not spawn too many concurrent tasks.

2.2.2 SORTNPSI

If a subset of partitions is specified for the REORG TABLESPACE with the PART(n:m) specification, and if the table has any non-partitioned indexes, there is the option to consider using the SORTNPSI specification:

- SORTNPSI YES merges the keys extracted for reorganized data partitions with the keys for the data partitions not reorganized into a single sort to build the non-partitioned secondary index. This involves sorting more keys (that is, all NPSI keys), but it can be faster if a large enough percentage of the entire table space is being reorganized.
- SORTNPSI NO unloads and builds the NPSI keys for the partitions not participating in the **REORG**, and then insert the sorted keys for the partitions being reorganized into that NPSI. This results in a smaller number of keys to sort. It can be faster when the original NPSI is not disorganized and the percentage of the table space data rows being reorganized is small compared with the entire table space.

The default is that the SORTNPSI option is determined by zparm REORG_PART_SORTNPSI, which can be set to YES, NO, or AUTO. The initial value is AUTO, which means that **REORG** uses Real Time Statistics to choose the option (YES or NO) that is estimated to take the least amount of elapsed time.

Before Db2 V11, **REORG** always used to use the SORTNPSI NO technique. With the implementation of SORTNPSI YES, some scenarios reorganizing 40% of a table space's partitions have showed an elapsed time improvement of 55%, with an additional CPU cost of 22%.

2.2.3 Inline copies and flash copies

The recommended practice using SHRLEVEL CHANGE requires that the **REORG** also produce an image copy while reorganizing. This can take the form of an inline copy, where pages of the reorganized table space and log apply are written out when changed, or a FLASHCOPY can be taken at the end. The IBM FlashCopy® technique is suggested as the better option for reducing elapsed time.

2.2.4 CPU and zIIP offload

The CPU usage can be a "performance" concern because of cost issues, especially if **REORG** is run during prime time. This CPU factor is substantially reduced because **REORG** supports zIIP offload for the UNLOAD, RELOAD, SORT, BUILD, and INLINE Statistics phases, and these phases comprise the bulk of **REORG** CPU processing.

2.2.5 DISCARDING rows during REORG

For the best performance, use the NOPAD keyword when discarding rows.

2.2.6 Sort products

The Db2 utilities come with the DFSORT product, but IBM Db2 Sort is an alternative. While DFSORT improvements to zIIP offload, elapsed time, and memory usage have been made over time, the best **REORG** performance is usually obtained by using Db2 Sort with the Db2 utilities.

2.3 Disk space usage and considerations

One of the limiting factors for reorganizations can be the amount of disk space needed during reorganization. On one hand, piping of data and keys from one task to another in memory has reduced the requirements for temporary data sets during the execution of **REORG** compared with earlier releases.

However, the reliance of building shadow table spaces and indexes for availability with the SHRLEVEL CHANGE or SHRELVEL REFERENCE options double the space used for those objects during **REORG** compared with SHRLEVEL NONE. This is another area that can consume much temporary disk space when external sort products like DFSORT or Db2 Sort are used.

2.3.1 Avoiding sorting by unloading by the cluster index

One method to save space used by sorting is to not sort. **REORG** with the SORTDATA NO option unloads using the clustering index to get data in clustering order without sorting. This is an option to save disk space during reorganization, but it's also typically much slower than a table space scan followed by a sort used for SORTDATA YES, especially if the clustering index is disorganized.

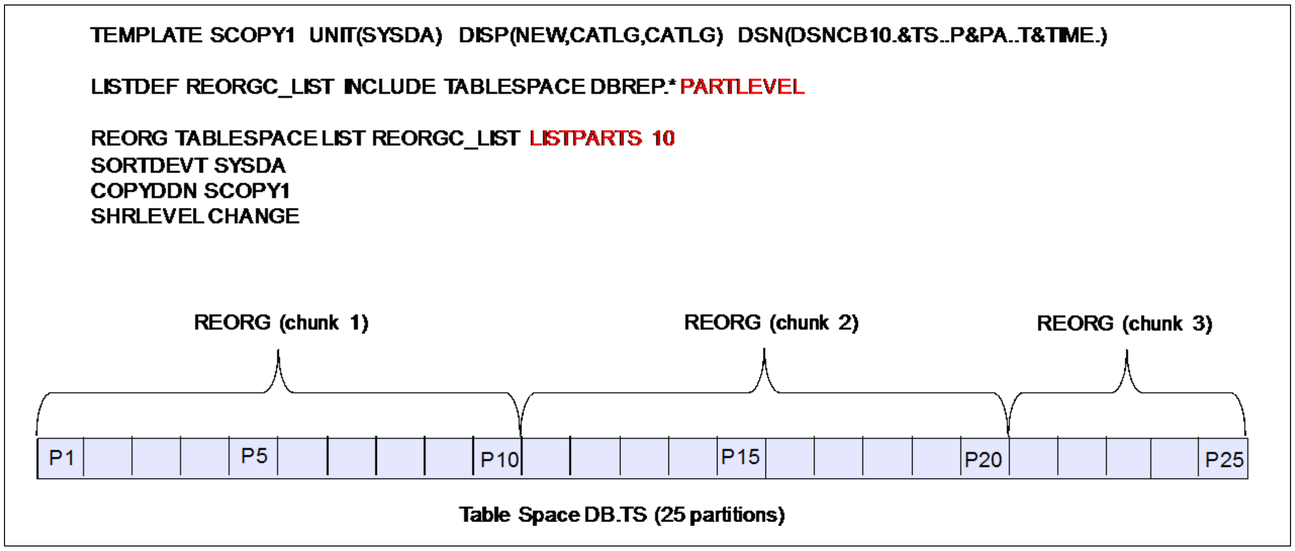
2.3.2 Avoiding sorting with RECLUSTER NO

The keyword RECLUSTER NO indicates that the ordering of data rows in clustering order doesn't matter when reorganizing. When combined with SORTDATA NO, **REORG** will simply unload the data and reload it without respecting the clustering order, and this will avoid sorting. This makes sense for data that might already be well clustered, or in the case where the reason for a **REORG TABLESPACE** is only to apply schema changes rather than to recluster data rows for application performance reasons.

2.3.3 Smaller sorts by reorganizing subsets of partitions

Another method to use less disk space during **REORG** is to divide and conquer by reorganizing a classic or PBR table space with multiple steps, each specifying a subset range of partitions. For example, a table space with 25 partitions can be reorganized with three **REORG**s, each covering up to 10 partitions. This reduces the amount of data and keys sorted significantly for each **REORG**.

However, if there are non-partitioned secondary indexes (NPSIs), there will be more overall work done than when a single **REORG** is done for all partitions. With this case, the SORTNPSI option is relevant for managing the tradeoffs with space requirements and performance (Figure 2-10 on page 23).



```
TEMPLATE SCOPY1 UNIT(SYSDA) DISP(NEW,CATLG,CATLG) DSN(DSNCB10.&TS..P&PA..T&TIME.)

LISTDEF REORGC_LIST INCLUDE TABLESPACE DBREP.* PARTLEVEL

REORG TABLESPACE LIST REORGC_LIST LISTPARTS 10
SORTDEVT SYSDA
COPYDDN SCOPY1
SHRLEVEL CHANGE
```

Figure 2-10 Reorganizing with three REORGs

2.4 Partition by growth table space considerations

This section introduces two considerations for partition by growth (PBG) table spaces.

2.4.1 REORG of a part range

When a subset of partitions for a PBG is reorganized, it’s possible that the reloaded data no longer fits into the reorganized partition or partitions. With Db2 V12, when **REORG** is run, instead of failing, a new PBG partition is added, and the overflow data rows are put into this newly added partition.

In Figure 2-11, a PBG table space has *n* partitions, but only partition 2 is being reorganized. When the data can’t be reloaded fully into partition 2, a new partition is added (if `MAXPARTITIONS` is not exceeded) to receive the extra rows.

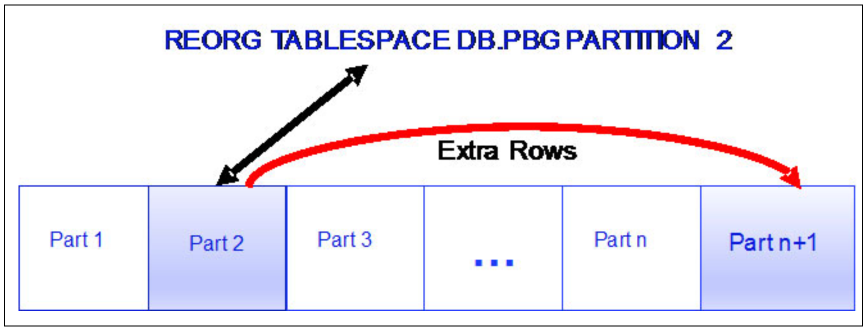


Figure 2-11 REORG of a part range

2.4.2 Dropping partitions

If a PBG table space has much less data than in the past, there might be many empty partitions left after a **REORG** of the entire table space. To eliminate these extra partitions, the keyword `DROP_PART YES` can be added to the **REORG** specification in Db2 V12. To set this behavior as the default for any **REORG** of a PBG table space, the zparm `DROP_PBG_PARTS` can be set in Db2 V11 and later releases.

2.5 Partition by range table space considerations

One of the common issues with Partitioned By Range table spaces (or non-UTS partitioned) is that partitions populate at different rates and can tend to fill and run out of space. In order to alleviate that problem and avoid outages because a partition has no more room, data can be moved to other partitions that aren't as full. There are two techniques available in Db2 V11 to do this:

- `REORG REBALANCE` allows **REORG** to redistribute data rows across multiple partitions. A range of logical partitions must be specified, and **REORG** tries to balance the data rows evenly, then update the new partition limit keys according to the data in each partition.
- **REORG** of a logical partition range redistributes data according to `ALTER TABLE ALTER PARTITION ENDING AT` statements are specified as pending DDL changes. With this technique, the new limit key is pre-specified, and **REORG** honors the new specification.

In cases where the limit key is meaningful so that it cannot be changed, increasing the partition size should be considered instead of repartitioning. For a Partition By Range table space, this can be done with the `ALTER TABLESPACE ALTER DSSIZE` specification.

2.6 Recovery considerations

REORG requires that an image copy be taken as a base for recovery. This can be taken for the entire object being reorganized, or an inline copy can be taken for each partition involved during the **REORG**. If a `TEMPLATE` is specified for the inline copy, an `&PA` or `&PART` specification as part of the data set name results in a separate data set for each partition. Example 2-5 shows a sample.

Example 2-5 TEMPLATE specification

```
  TEMPLATE SCOPY1  UNIT(SYSDA) DISP(MOD,CATLG,CATLG)
DSN(KB.&SN..D&JDATE..T&TIME..P&PART.)
  COPY TABLESPACE DBREP.TSREPS
    COPYDDN(SCOPY1)
    SHRLEVEL CHANGE
```

The advantage of partition-level copies is improved performance if a `RECOVER TABLESPACE` of a single partition or subset of partitions is ever needed. During the `RESTORE` phase of **RECOVER**, the data set containing pages for the partition is immediately located, which gives a performance advantage over scanning an image copy containing all partitions to find the partition requested.

Performance measurements (Figure 2-12 on page 25) have shown a 28% elapsed time improvement and 49% CPU improvement can be attained with the recovery of a single partition within a twenty partition table space.

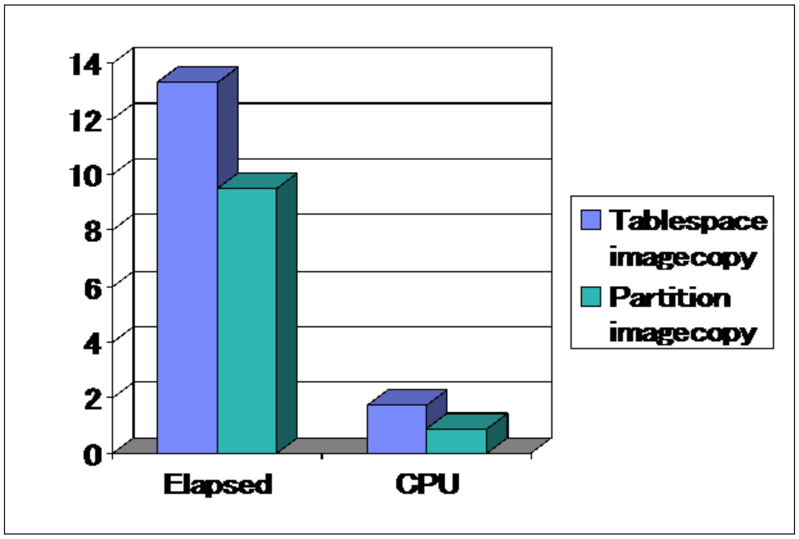


Figure 2-12 Improved elapsed and CPU times

If tape is used for inline copies, the part level `TEMPLATES` should be avoided unless there are as many available tape drives as there are partitions being reorganized.

2.7 Materialization of pending schema changes

One of the powerful functions of the **`REORG`** utility is the ability to apply or materialize various schema changes with minimal outage to applications. In the past, many of these schema changes required long periods of unavailability, and that was because the process to make these changes was to unload all data, drop all existing objects, create new objects with the appropriate schema definitions, and reload the data.

However, many changes can now be implemented using what is called deferred changes or pending changes. This comprises a `DDL ALTER` statement being done on an object to define the schema change, remembering this in the `SYSPENDINGDDL` table in the catalog, and then having it picked up and actually applied when the next applicable online **`REORG`** for that object is done.

2.7.1 Converting to universal table spaces

Universal Table Spaces are strategic and have advantages over non-UTS, so many users are running reorganizations to convert classic partitioned or segmented table spaces to UTS. Converting to UTS requires that a **`REORG`** for the entire table space be specified. Here are some considerations when doing these one-time conversions.

Classic to PBR or PBR with Relative Page Numbering

A classic partitioned table space can be converted to Partitioned By Range by adding segments, and in Db2 V12, it can also be changed to specify that PBRs use Relative Page Numbering (RPN). PBR RPN gives superior handling for active partitions that can become full, so it’s advised for PBR table spaces. Here are the steps taken for conversion from classic partitioned to PBR RPN:

```
ALTER TABLESPACE DB1.TSPBR SEGSIZE n          - for PBR
ALTER TABLESPACE DB1.TSPBR PAGENUM RELATIVE   - for PBR RPN
```

Segmented to Partitioned By Growth

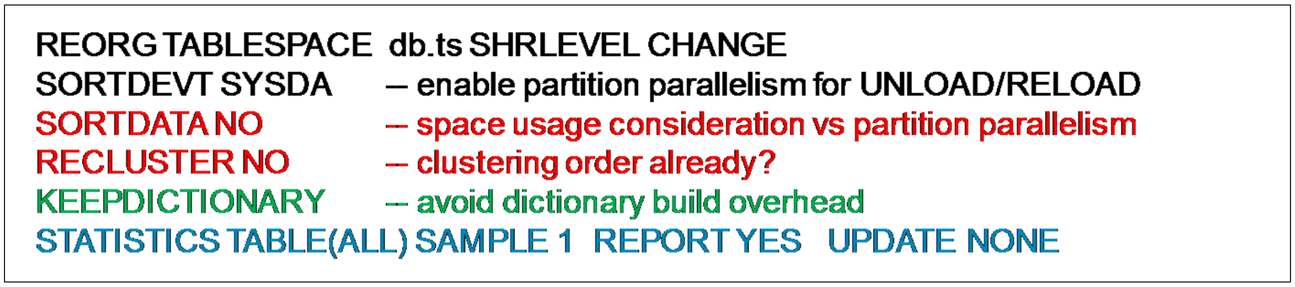
A segmented table space can be converted to Partition By Growth by adding the maximum number of partitions as follows:

```
ALTER TABLESPACE DB1.TSPBR MAXPARTITIONS n       - for PBG
```

REORG Materialization

After the **ALTER** statements have been issued, the table space is in an `Advisory REORG` state (AREO state). The changes will be applied when the entire table space is reorganized. Here is a sample **REORG** statement that might provide for minimal elapsed time and total work.

As specified, the **REORG** points out different options that can be used (Figure 2-13). `SORTDEVT` ensures not restricting partition parallelism for conversion to PBR. Note that this partition parallelism isn't supported for conversion to PBG.



```
REORG TABLESPACE  db.ts SHRLEVEL CHANGE
SORTDEVT SYSDA      – enable partition parallelism for UNLOAD/RELOAD
SORTDATA NO         – space usage consideration vs partition parallelism
RECLUSTER NO        – clustering order already?
KEEPDICTIONARY      – avoid dictionary build overhead
STATISTICS TABLE(ALL) SAMPLE 1  REPORT YES   UPDATE NONE
```

Figure 2-13   REORG options

`SORTDATA NO` avoids sorting the data rows, thus avoiding the resource usage of a sort. When that is combined with `RECLUSTER NO`, the ordering doesn't matter, so a fast (partition parallelism for PBR conversion) unload is done instead of unloading from the clustering index.

When using `RECLUSTER NO`, beware of the Real Time Statistics (RTS) settings changing. `REORGUNCLUSTINS`, `REORGINSERTS`, and `REORGUNCLUSTINS` will be set to `0` even though no data rows have been re-clustered. Therefore, you should only use `RECLUSTER NO` on table spaces that are well clustered, or else capture the RTS values and reset them after the **REORG**.

`KEEPDICTIONARY` avoids the extra resources required to rebuild new dictionaries.

`STATISTICS` actually does less than leaving it out. For conversions to UTS, `STATISTICS` for the table space and indexes is the default with `UPDATE YES`, so include it here with a minimum sample size.

2.8 Changing a partition boundary

For a classic partitioned table space or for a Partitioned By Growth table space, rows may be inserted and updated unevenly across the table space such that certain partitions begin to fill to maximum capacity. To avoid an outage because of a full partition, re-balancing of data rows by moving them from a full partition to an adjacent with empty space is one option.

For example, in a three-partition table space, partition two has almost filled. The limit key boundaries for the partitions are `10,000`, `20,000`, and `30,000`. Partition three is less than half filled, so rows can be moved from partition 2 to partition 3 by lowering the limit key for partition 2, and then the rows no longer fitting within the boundary will be moved to partition 3. This is done by moving the limit key for partition two to a value of `15,000`.

After the **`ALTER`**, partitions two and three are in an `Advisory REORG` state, and the **`ALTER`** doesn't take effect until those partitions are reorganized (Figure 2-14).

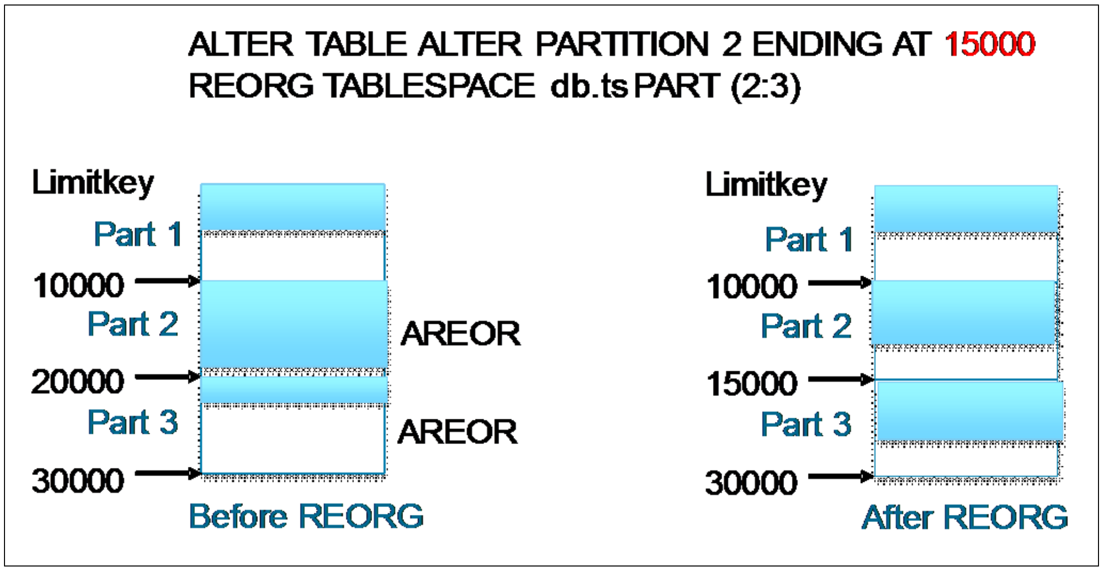


Figure 2-14 Before and after REORG

After the **`REORG`** of partition range of 2 - 3, rows have been moved so that neither partition is in immediate danger of filling to capacity (Figure 2-15).

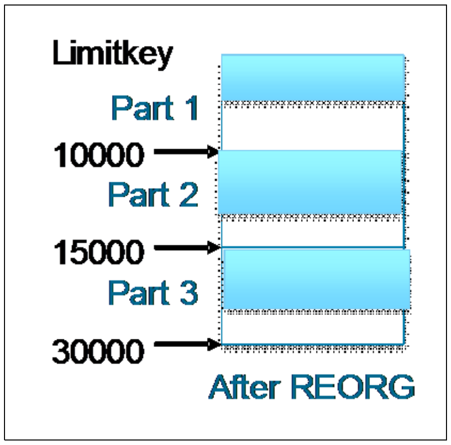


Figure 2-15 After REORG

2.9 Inserting a new partition

The solution to redistribute the data between partitions 2 and 3 works in our example, but notice that partition three is filled much closer to capacity. What could have been done if partition 3 didn't have enough extra space to take rows from partition 2? In Db2 V12, a new feature to split that data from partition 2 into two half full partitions without affecting partitions 1 or 3 is possible by inserting a new partition.

With our original table space with a full partition 2, a new partition can be "inserted" by adding it before partition 2. In the following example, note that the range of values from `10,000` - `20,000` can be split in half with a new partition having a limit key of `15,000`.

This is specified with the following **ALTER** statement (Figure 2-16).

```
ALTER TABLE ADD PARTITION ENDING AT 15000
REORG TABLESPACE db.ts PART 2
```

Figure 2-16 ALTER statement

After the **ALTER**, partition 2 is in `advisory REORG pending` because the new partition added has an ending limit key that puts it within the range of partition 2. A **REORG** of partition 2 smaterializes the new partition. Notice that the new partition added is physical partition 4, but because of its limit key, it is logical partition 2. With the extra space, the rows from partition two are redistributed between two partitions (logical partitions 2 and 3), so each will have plenty of room (Figure 2-17).

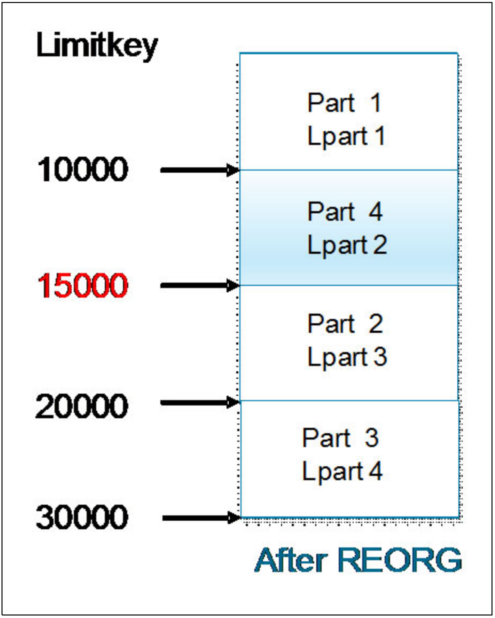


Figure 2-17 New partition

2.10 Altering column data types of lengths

Before Db2 V12, the **ALTER** of a column to a different data type, or the extending of a length for a column, would be considered an “immediate” **ALTER** because the next data row inserted or updated would have the new data row format immediately upon being modified. While this works well in isolation, sometimes many **ALTER**s to a table and table space need to be done together, and there are restrictions in mixing immediate and pending **ALTER**s.

In Db2 V12, a new zparm, `DDL_MATERIALIZATION` specifies that an **ALTER** to a column to change the column data type or length will be treated as a pending change, so the data won’t assume the new format until the new **REORG**.

A table space can then stack a number of **ALTER** statements to make all of the changes within one reorganization.

3

Db2 backup and recovery

This section describes how to apply best practices related to backup and recovery. The intent is not to repeat what is covered in the *Db2 for z/OS Utility Guide and Reference manual*, SC27-8860.

We first describe and show the best backup options using the **COPY** utility with all its versions and facets to meet your needs. We will include sequential, track-based, IBM FlashCopy® based and volume-based methods. After that we show and describe the best way to use the **RECOVER** utility.

We look first at the **COPY** utility and the methods used by the **COPY** utility. There are many options available to you to copy (or back up) your Db2 assets. The **COPY** utility provides many options to meet your needs. In addition to the traditional methods, using fast replication (as an example) can save you a lot of host I/O and CPU usage, should that be an option that you can use.

In Figure 3-1 on page 30 we show the different ways that the utility driver program, **DSNUTILB**, can drive the different backup programs. We then describe the options depicted.

© Copyright IBM Corp. 2018. All rights reserved.

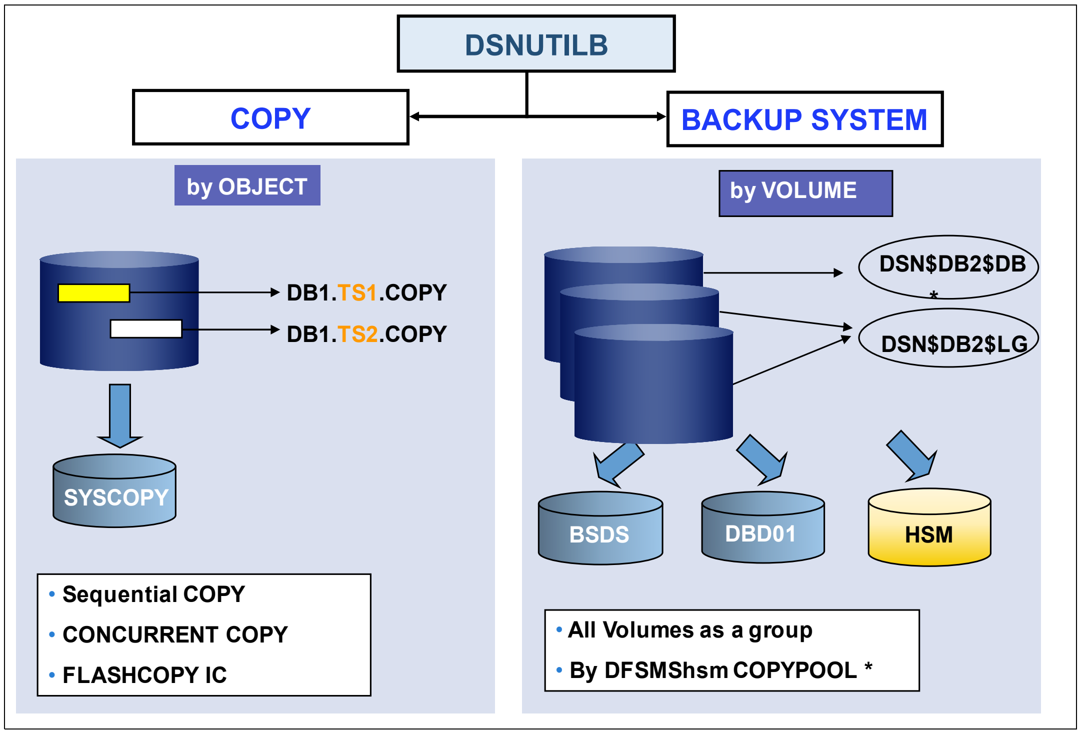


Figure 3-1 Using DSNUTILB to drive utility programs

3.1 Using the COPY utility

The **`COPY`** utility can be used to drive different types of backups. We briefly describe each method with best use cases to meet your needs.

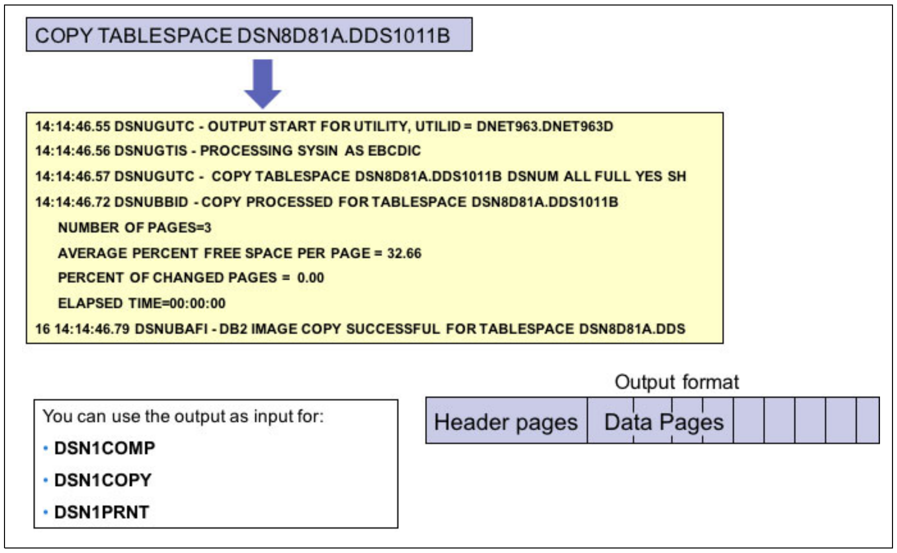
3.1.1 Sequential copies

This tried and tested way of backing up your table space data has been used since Db2 v1. It copies all of the pages in a table space: header pages, data pages, and so on. The output is sequential in nature, and offline utilities like **`DSN1PRNT`** can read it. It can also be used as input to the **`UNLOAD`** utility.

There are different ways to create sequential copies:

- **`COPY TABLESPACE`** with options:
  - `FULL`
  - `INCREMENTAL`
  - `SHRLEVEL REFERENCE` and `CHANGE`
- **`DSN1COPY`**
- **`CONCURRENT COPY`**

COPY TABLESPACE is shown in Figure 3-2.



```
14:14:46.55 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = DNET963.DNET963D
14:14:46.56 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
14:14:46.57 DSNUGUTC - COPY TABLESPACE DSN8D81A.DDS1011B DSNUM ALL FULL YES SH
14:14:46.72 DSNUBBID - COPY PROCESSED FOR TABLESPACE DSN8D81A.DDS1011B
   NUMBER OF PAGES=3
   AVERAGE PERCENT FREE SPACE PER PAGE = 32.66
   PERCENT OF CHANGED PAGES = 0.00
   ELAPSED TIME=00:00:00
16 14:14:46.79 DSNUBAFI - DB2 IMAGE COPY SUCCESSFUL FOR TABLESPACE DSN8D81A.DDS
```

You can use the output as input for:
- **DSN1COMP**
- **DSN1COPY**
- **DSN1PRNT**

Figure 3-2 COPY TABLESPACE

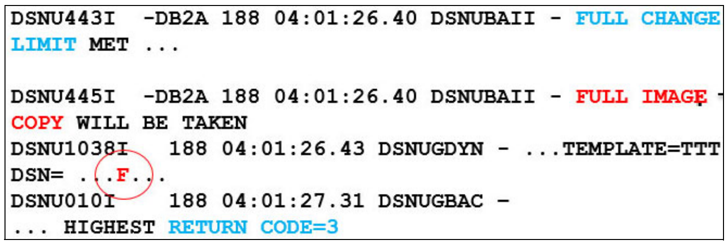
You can use the output as input to offline utilities such as **DSN1COMP.-COPY** and **-PRNT**. The sequential output is in a format that represents the header and data pages per the object copied.

The following list includes newer enhancements to the **COPY** utility, as far as sequential copies are concerned:

- Changes how &ICTYPE functions, when using TEMPLATE on the **COPY** utility. Now, starting with Db2 12, you can identify the actual type of the image copy that was taken. Db2 11 and prior used to set the type to 'C' when the data set name was allocated TEMPLATE with CHANGELIMIT. However, users were not able to differentiate if it was a Full or Incremental copy. In Db2 12, when &ICTYPE usage on the TEMPLATE will reflect the actual type of the image copy, when CHANGELIMIT is specified:
  - &ICTYPE = 'F', when a full image copy will be generated
  - &ICTYPE = 'I', when an incremental image copy will be generated

  Tip: For TEMPLATE GDGLIMIT, The default value is 99. The minimum value is 0, and for z/OS V2R1 it has a maximum value of 255. With z/OS V2R2 and later, the maximum value is 999. If you want to use a value greater than 255, verify that GDGEXTENDED is set to YES in member IGGCATxx in SYS1.PARMLIB.

  An example of a FULL copy in the DSN in the output (Figure 3-3).

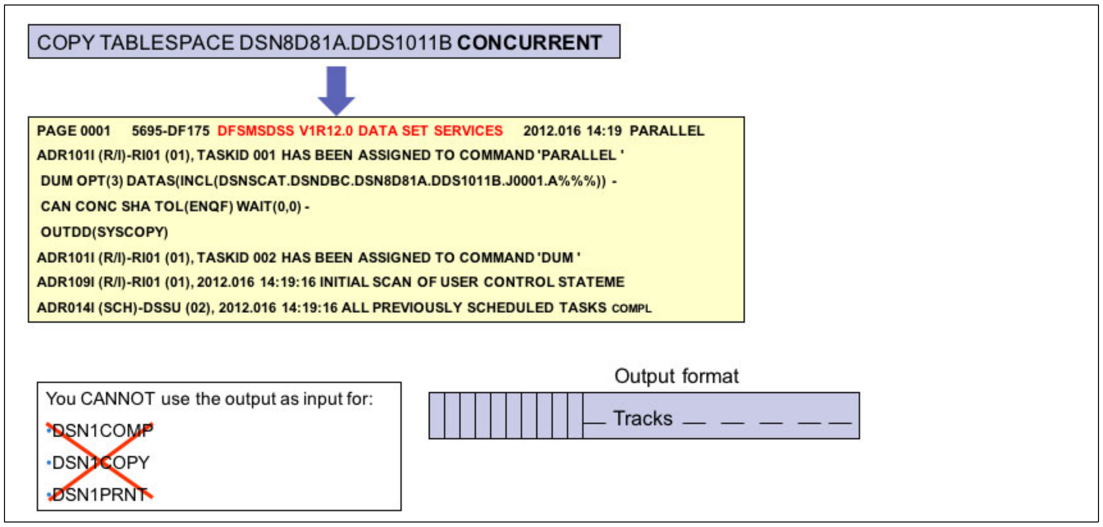


```
DSNU443I  -DB2A 188 04:01:26.40 DSNUBAII - FULL CHANGE
LIMIT MET ...

DSNU445I  -DB2A 188 04:01:26.40 DSNUBAII - FULL IMAGE
COPY WILL BE TAKEN
DSNU1038I   188 04:01:26.43 DSNUGDYN - ...TEMPLATE=TTT
DSN= ...F...
DSNU010I    188 04:01:27.31 DSNUGBAC -
... HIGHEST RETURN CODE=3
```

Figure 3-3 FULL copy

- Another way to use the **COPY** utility for a sequential type output is using the CONCURRENT method (Figure 3-4). This option of the **COPY** utility enables you to make full image copies using DFSMSdss (using the utility **ADRDSSU**) **Concurrent COPY**.

  The copy process is initiated by the Db2 **COPY** utility when you specify the CONCURRENT keyword on the **COPY** statement. The image copy is recorded in SYSCOPY with ICTYPE = F and STYPE = C (for example, node I00012 and STYPE = J for instance node J0001). When a recoverable point that indicates that a DFSMSdss **Concurrent COPY** is found in SYSCOPY, the **RECOVER** utility invokes a DFSMSdss **RESTORE** command to restore from the copy.



```
COPY TABLESPACE DSN8D81A.DDS1011B CONCURRENT
```

```
PAGE 0001     5695-DF175  DFSMSDSS V1R12.0 DATA SET SERVICES     2012.016 14:19  PARALLEL
ADR101I (R/I)-RI01 (01), TASKID 001 HAS BEEN ASSIGNED TO COMMAND 'PARALLEL '
 DUM OPT(3) DATAS(INCL(DSNSCAT.DSNDBC.DSN8D81A.DDS1011B.J0001.A%%%)) -
 CAN CONC SHA TOL(ENQF) WAIT(0,0) -
 OUTDD(SYSCOPY)
ADR101I (R/I)-RI01 (01), TASKID 002 HAS BEEN ASSIGNED TO COMMAND 'DUM '
ADR109I (R/I)-RI01 (01), 2012.016 14:19:16 INITIAL SCAN OF USER CONTROL STATEME
ADR014I (SCH)-DSSU (02), 2012.016 14:19:16 ALL PREVIOUSLY SCHEDULED TASKS COMPL
```

You CANNOT use the output as input for:
- ~~DSN1COMP~~
- ~~DSN1COPY~~
- ~~DSN1PRNT~~

Figure 3-4 CONCURRENT method

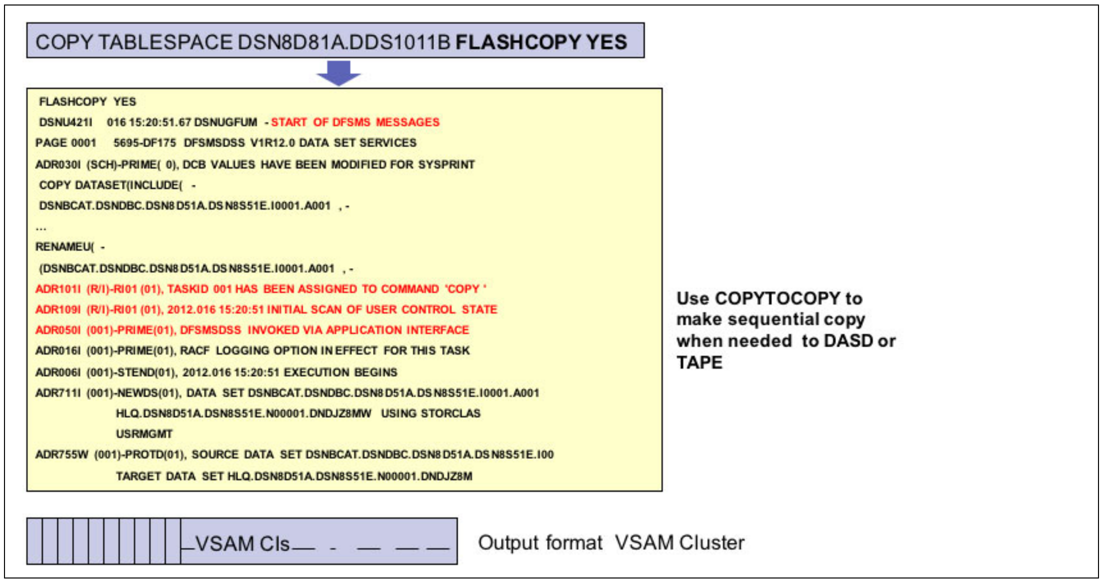
- With **SHRLEVEL REFERENCE**, the objects are not available for update until the copy operation is logically completed:
  – All writes are drained, and the objects being copied have a restrictive state of UTRO.
  – The objects are quiesced to ensure that all updated buffers are written to disk before DFSMSdss **Concurrent COPY** is invoked. The SYSCOPY records with ICTYPE=Q are inserted to indicate that the objects are successfully quiesced.
  – As soon as the DFSMSdss **Concurrent COPY** is logically complete, the objects are drained and the restrictive state is changed from UTRO to UTRW.

  A good use or business case for using this kind of copy is should you desire better improvement as far as availability is concerned (FlashCopy IC as described in the next section is still the fastest method). Because this is a track-based copy, but still sequential in nature, it does not process the header pages and space page map pages the way that DB2 image copy processes it. It is also typically less “expensive” to run.

3.1.2 FlashCopy Image Copy – non-sequential

FlashCopy image copies are output to VSAM data sets (a VSAM ESDS Cluster type. The traditional copy methods that are used by the utilities output to a non-VSAM sequential format data set. FlashCopy creates a separate VSAM data set for each partition or piece of the object that is being copied. the FlashCopy function that is provided by z/OS DFSMS and the IBM DS8880 storage subsystems.

FlashCopy can reduce both the unavailability of data during the copy operation and the amount of time that is required for backup and recovery operations. the FlashCopy image copy is allocated by DFSMSdss and is always cataloged. The I/O “cost” is offloaded to the storage arrays, saving you resources on the processor. See Figure 3-5.



```
COPY TABLESPACE DSN8D81A.DDS1011B FLASHCOPY YES
```

```
 FLASHCOPY  YES
 DSNU421I    016 15:20:51.67 DSNUGFUM  - START  OF DFSMS  MESSAGES
PAGE 0001     5695-DF175  DFSMSDSS  V1R12.0 DATA  SET SERVICES
ADR030I  (SCH)-PRIME(  0), DCB  VALUES  HAVE BEEN  MODIFIED FOR  SYSPRINT
 COPY DATASET(INCLUDE(   -
 DSNBCAT.DSNDBC.DSN8 D51A.DS N8S51E.I0001.A001   , -
...
RENAMEU(  -
 (DSNBCAT.DSNDBC.DSN8 D51A.DS N8S51E.I0001.A001   , -
ADR101I  (R/I)-RI01 (01), TASKID  001 HAS  BEEN ASSIGNED  TO COMMAND  'COPY '
ADR109I  (R/I)-RI01 (01), 2012.016 15:20:51 INITIAL  SCAN  OF USER  CONTROL  STATE
ADR050I  (001)-PRIME(01), DFSMSDSS  INVOKED VIA APPLICATION  INTERFACE
ADR016I  (001)-PRIME(01), RACF  LOGGING OPTION IN EFFECT  FOR THIS  TASK
ADR006I  (001)-STEND(01), 2012.016 15:20:51 EXECUTION  BEGINS
ADR711I  (001)-NEWDS(01), DATA  SET DSNBCAT.DSNDBC.DSN8 D51A.DS N8S51E.I0001.A001
                 HLQ.DSN8D51A.DSN8S51E.N00001.DNDJZ8MW   USING STORCLAS
                 USRMGMT
ADR755W  (001)-PROTD(01), SOURCE  DATA  SET DSNBCAT.DSNDBC.DSN8 D51A.DS N8S51E.I00
                 TARGET  DATA  SET HLQ.DSN8D51A.DSN8S51E.N00001.DNDJZ8M
```

Use COPYTOCOPY to make sequential copy when needed to DASD or TAPE
VSAM CIs
Output format VSAM Cluster

Figure 3-5   FLASHCOPY YES

In this example, we see how DFSMSdss (through the `ADRDSSU` messages `ADR101x`) is used to perform the actual copy. FlashCopy image copies are output to VSAM data sets. The traditional copy methods that are used by the utilities output to a non-VSAM sequential format data set. FlashCopy creates a separate VSAM data set for each partition or piece of the object that is being copied.

When creating a FlashCopy image copy, the following utilities can also create 1 - 4 additional sequential format image copies in a single execution:

- **`COPY`**
- **`LOAD`** with the `REPLACE` option specified
- **`REORG TABLESPACE`**

The **`COPYTOCOPY`** utility can create sequential format image copies by using an existing FlashCopy image copy as input.

Restrictions to note

A data-set-level FlashCopy has certain operational restrictions that can cause a utility to resort to traditional I/O methods to complete a copy operation. This behavior can occur even when you explicitly request FlashCopy support in either the subsystem parameter or the utility control statement. In some cases, the utility aborts the copy operation.

The circumstances in which the utilities might not be able to complete a copy operation by using FlashCopy include the following situations. In these situations, the term *data set* refers to a Db2 table space or index space, or a FlashCopy image copy:

- FlashCopy Version 2 disk volumes are not available.
- The source data set is already the target of a FlashCopy relationship.
- The target data set is already the source of a FlashCopy relationship.

- The source data set is already participating in the maximum number of FlashCopy relationships.
- The `CISIZE`, `CASIZE`, physical record size, or physical block size of the target data set is different from that of the source data set. The `CASIZE` of the target data set can be different from the source data set if the source data set is less than one cylinder.
- The source data set and the target data set are not both fully contained on the same physical control unit (controller).

Note: Use the storage class attribute `ACCESSIBILITY=REQUIRED` or `ACCESSIBILITY=PREFERRED` for the source data set and for the target data set. If the storage class that is associated with a data set has this attribute, DFSMS attempts to select volumes such that the data set is contained on volumes within a single physical control unit.

3.2 Using BACKUP SYSTEM

As mentioned previously, FlashCopy image is an instantaneous copy of a DASD volume taken at a particular point in time. It is possible to keep several versions if resources are available and the data can be maintained on disk or tape.

To take a FlashCopy, you must first establish a relationship between the Copy Pool (source) and the backup Copy Pool (target). Volumes are logically associated so that a physical copy of each volume can be made. At the point that the relationship is established, the **`BACKUP SYSTEM`** or **`RESTORE SYSTEM`** is considered logically complete.

For example, when doing a **`BACKUP SYSTEM`**, the unavailability period for access to the system only lasts until the **`BACKUP SYSTEM`** is logically complete, so it is very fast. When it is logically complete, a background copy is started so that the target will look like the source did at the time the operation was logically complete.

If any applications cause changes to tables so that tracks on the source volume must be updated before they are copied to the target, the source track with the change is first copied to the target before it is updated. When the copy of each volume is complete, the logical relationship can be terminated.

The previous DB2 releases can use up to only two copy pools, one for the database and one for logs. These copy pools define the storage groups to copy and the backup storage groups to store the copies. In the DB2 12, the system level backup supports multiple copy pools in which you can keep extra system level backups on disk during upgrades. Also, an alternate copy pool includes the same defined set of storage groups as the standard copy pool, however different backup storage groups are specified.

To use an alternate copy pool, specify the `ALTERNATE_CP` option and the related backup storage group options (`DBBSG` and `LGBSG`) on the **`BACKUP SYSTEM`** utility control statement:

- `DBBSG` refers to the backup storage group name for the database copy pool. It can be up to eight characters and must be defined to DFSMS with the `COPY POOL BACKUP` attribute.
- `LGBSG` refers to the backup storage group name for the log copy pool. It can be up to eight characters and must be defined to DFSMS with the `COPY POOL BACKUP` attribute.

3.2.1 FLASHCOPY_PPRCP option

In DB2 12, the FLASHCOPY_PPRCP keyword is added to the **RESTORE SYSTEM** and **RECOVER** utilities, enabling you to control the preserve mirror option for the DB2 production volumes during FlashCopy operations when the recovery base is a system-level backup. FLASHCOPY_PPRCP also applies to the **RECOVER** utility that uses a FlashCopy image copy as its recovery base.

Table 3-1, Table 3-2, and Table 3-3 are some reference tables that you can refer to which pertain to **RECOVER** in a fast replication setup (FlashCopy).

Table 3-1 Options for the RECOVER utility from FCIC

	Preserve mirror behavior	**FlashCopy usage**
ZPARM	FLASHCOPY_PPRC	REC_FASTREPLICATION
Default	REQUIRED	PREFERRED
RECOVER keyword	FLASHCOPY_PPRCP	n/a
ADRDSSU keyword	FCTOPPRCP	FASTREPLICATION

Table 3-2 Options for the RECOVER utility from SLB

	Preserve mirror behavior	**FlashCopy usage**
ZPARM	FLASHCOPY_PPRC	REC_FASTREPLICATION
Default	REQUIRED	PREFERRED
RECOVER keyword	FLASHCOPY_PPRCP	n/a
FRRECOV	ALLOWPPRCP	FR(=FASTREPLICATION)
ADRDSSU keyword	FCTOPPRCP	FASTREPLICATION
Control options before Db2 12	ISMF COPYPOOL ISPF panels	SETSYS FASTREPLICATION (DATASETRECOVERY(..))

Table 3-3 Options for the RECOVER utility from RESTORE SYSTEM

	Preserve mirror behavior	**FlashCopy usage**
ZPARM	FLASHCOPY_PPRC	
Default	REQUIRED	
RESTORE SYSTEM keyword	FLASHCOPY_PPRCP	
FRRECOV	ALLOWPPRCP	
ADRDSSU keyword	FCTOPPRCP	FASTREPLICATION
Control options before Db2 12	ISMF COPYPOOL ISPF panels	

3.3 Db2 Recovery

In this section, we describe some best practices and new enhancements that pertain to data set and volume-based recovery. This is summarized in Figure 3-6.

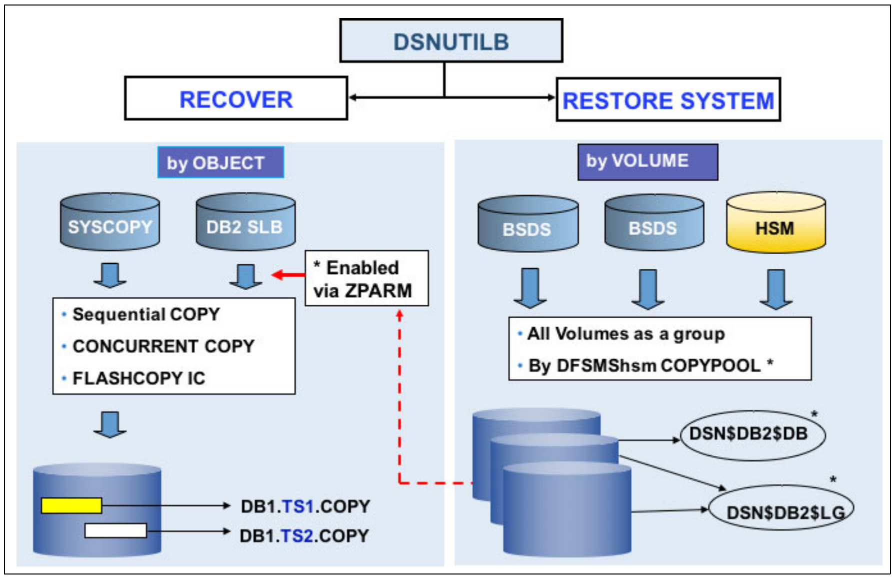


Figure 3-6 Data set and volume-based recovery

3.3.1 RECOVER using the SCOPE keyword

To meet your recovery time objectives (RTO), you need to have a sound backup strategy. When you have a decent backup process in place, Db2 provides some newer options to help you speed up your object recovery. These options are for planned and unplanned recoveries.

Db2 12 introduced the `SCOPE` keyword. It is applied when the **`RECOVER`** utility uses the `TORBA` option or the `TOLOGPOINT` option. `SCOPE` has two variations:

- `SCOPE UPDATED`
- `SCOPE ALL`

The `SCOPE UPDATED` option can potentially improve recovery time because it indicates which objects in the specified `LISTDEF` list are to be recovered. The objects in the list that have not changed since the recovery point are skipped by the **`RECOVER`** utility. In this way, it does not waste time processing objects that have not changed and therefore do not need to be recovered.

The exception is for the following objects that are recovered even if they have not changed since the specified recovery point:

- Indexes in information `COPY-pending` status
- Table spaces in `COPY-pending` status
- Any objects in `RECOVER-pending` status

When DB2 skips the objects in which the recovery is not required, a new message is issued:

```
DSNU1322I PROCESSING SKIPPED FOR dbname.tsname DSNUM n BECAUSE THE OBJECT DOES NOT
NEED TO BE RECOVERED.
```

The SCOPE ALL option indicates that all objects in the list are recovered, even if they have not been updated.

3.3.2 MODIFY RECOVERY enhancements

Starting in Db2 12, two new options were added to the **MODIFY RECOVERY** utility:

- DELETEDS
- NOCOPYPEND

DELETEDS option

The DELETEDS option drives the deletion of z/OS cataloged image copy data sets on disk, or those migrated by DFSMShsm tape when corresponding SYSCOPY records are deleted. The **IDCAMS** program is invoked to perform the deletion with **DELETE** commands.

This is an optional feature because it increases the elapsed time, and some users keep the image copy data sets even when no longer recorded in SYSCOPY.

The restart restrictions for DELETEDS are as follows:

- MODIFY abends after the deletion of the SYSCOPY records have been committed in the MODIFY phase and the job is restarted. In this case, the DELETEDS phase will be skipped and no image copy data sets will be deleted.
- MODIFY abends in the DELETEDS phase and the job is restarted. In this case, the phase will be changed to UTILTERM, and the image copy data sets that were not deleted in the first invocation will not be deleted.

NOCOPYEND option

The NOCOPYPEND option is added. It instructs **MODIFY RECOVERY** to not set COPY pending restricted status even if all backups were deleted from SYSCOPY. This feature was developed because **MODIFY RECOVERY** places objects in COPY-pending when all backups have been deleted from SYSCOPY. In this case, up to Db2 11, you were not able to update the data because of the restricted status.

3.4 BACKUP considerations and tips

In this section, we share some best practices that you can use in your environment. It is our intention to share how you can apply these suggestions, or to consider implementing them as and when it makes sense for you.

3.4.1 COPY INDEXES

Whether you recover or restore your data (both unplanned and planned), getting your index spaces back can be very costly. There are many ways to determine whether indexes are used or how active they are. See Chapter 5, “Db2 Real Time Statistics (RTS)” on page 45 to determine index activity and usage.

Consider **COPY**ing your indexes. Db2 has the Fast Log Apply (FLA) to assist here. Use RTS and RTS history to determine which indexes (you can copy all of them if you have enough space) are candidates for **COPY**. If you have the IBM DB2 Sort tool available, it can also help you.

You can also consider index candidates for backup based upon size or update activity: use RTS history.

3.4.2 FlashCopy IC

If you have the DASD storage and you need to back up your table spaces quickly without incurring resource use on the host, FlashCopy IC (FCIC) is something you should consider. To identify which table spaces are prime candidates for using FCIC, use RTS and RTS History. Customers who have started to use data set FCIC, started by copying the largest objects based upon size.

Newer advice is to back up the most active objects, regardless of size. The rationale here is to speed recovery to meet your RTO. Again, using RTS and RTS History, you can consider the following actions:

- Backing up the most active objects (to reduce log processing at recover time) using FCIC
- Backing up the most active objects using **`Incremental Copy`** and then **`Merge Copy`**.

3.4.3 CONCURRENT COPY

This track-based copy is very useful for speeding up your backups because it is typically cheaper and faster to run. However, this backup type is for recovery purposes only, and not available for use for unload processing and so on. Nevertheless, it is a consideration.

3.4.4 Split off active versus non-active objects

Revise your `LISTDEF`s or grouping. Also consider using the `SCOPE` function to not recover unchanged objects.

3.4.5 To QUIESCE or not to QUIESCE

In earlier times, customers used **`QUIESCE`** to establish a common point of recovery for a set or sets of objects, From a recovery point of view and with the introduction of using the `BACKOUT` function of the **`RECOVER`** utility, this might not be necessary anymore. If you still need to have a relative byte address (RBA) or log record sequence number (LRSN) created in `SYSCOPY`, you can run a **`QUIESCE`** on `DSNDB06.SYSEBCDC`. In addition, use `WRITE NO` unless you absolutely must have pages written out.

3.4.6 OPTIONS EVENT(ITEMERROR,SKIP)

Revise your grouping of objects based upon your requirements (for example, size versus activity). If you have many objects in your `LISTDEF`, and availability is a requirement, use this option. It may increase some resource use. However, it helps with setting your objects in `UTRW` only for the duration of the copy for that object. the read claim class is held only while the object is being copied.

If you do not specify `OPTIONS EVENT(ITEMERROR,SKIP)`, all of the objects in the list are placed in `UTRW` status and the read claim class is held on all objects for the entire duration of the **`COPY`**.

4

Statistics collection (RUNSTATS)

The **RUNSTATS** utility has had many enhancements for new functionality, usability, and availability over the years. Statistics can be collected by running the **RUNSTATS** utility, or they can be collected with inline statistics while running either a **REORG**, **LOAD**, or **REBUILD INDEX** utility and specifying STATISTICS.

4.1 Functional improvements

This section discusses the functional improvements of inline statistics parity and the `RESET` keyword.

4.1.1 Inline statistics parity

Enhancements to statistics gathering over the years tended to be first implemented in the **RUNSTATS** utility, but they weren't always also implemented with inline statistics for **REORG**, **LOAD**, or **REBUILD INDEX** at the same time.

As of V11, the functional capability (parity) for inline statistics gathering is available so that a separate **RUNSTATS** utility doesn't have to be run after a utility with inline statistics to fill in the statistics gap. The following list describes more recent improvements:

- The ability to collect inline histogram statistics with `HISTOGRAM`
- Distribution statistics with `COLGROUP` and `FREQVAL`
- NPI statistics when `SORTNPSI` is specified

Some keywords are still limited to being used with the **RUNSTATS** utility (for example, `REGISTER NO` and `RESET`), but these don't restrict the actual statistics gathering capability.

It's now possible to eliminate the extra overhead of a separate **RUNSTATS** jobs run by **REORG**, **LOAD**, or **REBUILD INDEX**.

4.1.2 RESET keyword

Many of the gathered statistics update existing columns in catalog tables, while other gathered statistics (for example, distribution stats and histograms) insert new rows into the catalog. In many cases, the inserted statistics can become obsolete and clutter up the catalog. With the `RESET` keyword, it's possible to clear out all of these extra unused statistics for an object, and then start over again collecting only what is needed.

`RESET` clears all statistics, not just those with timestamps earlier than a certain point in time. Therefore, it's important to follow a **RUNSTATS** that does a `RESET` with a **RUNSTATS** or inline statistics that collects the needed statistics, including those that are deleted with `RESET`.

4.2 Availability

This section discusses considerations related to availability.

4.2.1 INVALIDATECACHE

In the past, when gathering statistics, the dynamic cache was invalidated for the objects for which statistics were collected. However, this could prove disruptive to applications that depend on that cache in memory. By specifying `INVALIDATECACHE NO`, you can collect statistics without being disruptive. New statistics can then be picked up later when it is more convenient to `PREPARE` and possibly get a new access path.

It should be pointed out that in Db2 V12, the enhancement for dynamic plan stability (DPS) also helps manage plans to enable more stable, predictable performance when using cached access path.

DPS protects against access path changes and disruptions across these types of events:

- Release migration
- Function level activation
- System maintenance and DB2 bouncing
- System parameter changes
- Statistics gathering (when not using `INVALIDATECACHE`)

4.3 Performance

There have been various enhancements to improve performance by cutting the code path length, and CPU resource usage has been reduced with IBM z® Systems® Integrated Information Processor (zIIP) offload. In addition, if there are duplicate distribution statistics specified because a `COLGROUP` specified is the same as for the column of an index, the duplicate `COLGROUP` is ignored while the index stats are gathered once (Db2 V12).

Sampling is the recommended practice when there are millions or billions of rows. There are two methods for sampling: specifying either `SAMPLE SYSTEM` or `TABLESAMPLE SYSTEM`. `SAMPLE` results in looking at only every nth row, then extrapolating to get statistics. `TABLESAMPLE` randomly accesses rows when doing sampling. The TABLESPACE `AUTO` option is advised because it adapts to the amount of data in the table, so it balances the performance benefits with the accuracy without user intervention.

For Data Sharing environments when using **`RUNSTATS SHRLVEL CHANGE`**, use `REGISTER NO` to avoid registration of the accessed data in the coupling facility. This saves resources because the statistics are an approximation, so the serialization and blocking from updates on other members can be avoided without the differences in the statistics varying to any significant degree.

The **`RUNSTATS`** utility is zIIP off-loadable for the standard `TABLESPACE` scan and `INDEX` scan processing. As of Db2 V11, distribution statistics are 80% off-loadable to zIIP.

4.4 Usability

Statistics are classified into the two categories of *access path* statistics and *space* statistics.

In general terms, space statistics can track trends and point out characteristics which help determine when to schedule various maintenance activities:

- **`REORG`**s
- Backups with **`COPY`**s
- Gathering new statistics with **`RUNSTATS`**
- Determining when to build new dictionaries
- A host of other uses

These types of statistics are relatively inexpensive to gather because Real Time Statistics (RTS) gathers many of these in real-time, so it relieves the need for **`RUNSTATS`** or inline stats to collect these space statistics.

Alternatively, access path statistics are used by the optimizer to pick the best access path. These statistics include column cardinality and frequencies, histogram statistics, and information about available indexes. These can be very time-consuming and costly to gather, so there is a tradeoff.

Collecting statistics that aren't needed, or collecting them too often, is a waste of time and resources. However, if you do not collect enough statistics, or they become “stale,” the optimal access path might be missed, and application performance suffers. As a solution to this delicate balance, the PROFILE keyword was introduced in V11, and in V12 it can make the decision process with this statistics gathering dilemma dramatically easier.

4.4.1 PROFILE keyword

The suggested practice for statistics gathering is to use the PROFILE keyword. This enables **RUNSTATS** or inline statistics to easily keep the proper set of keywords and options for gathering, table space, table, and index statistics in the catalog rather than explicitly specified with the SYSIN statement of JCL for the **RUNSTATS** job.

As of Db2 V11, a PROFILE can be created for a given table and indexes, as shown in Example 4-1.

Example 4-1 Creating a PROFILE

```
RUNSTATS TABLESPACE DB1.TS1
   TABLE(TB1) SET PROFILE
   COLGROUP(CITY,ZIPCODE) FREQVAL COUNT 3
                          HISTOGRAM NUMQUANTILES 5
   INDEX(IX2) KEYCARD FREQVAL NUMCOLS 1 COUNT 5
```

The keyword options specified for TB1 and its indexes are stored into the catalog table SYSIBM.SYSTABLES_PROFILE. They are then picked up and used so that the following statement is equivalent to the original specification if SET PROFILE had not be specified:

```
RUNSTATS TABLESPACE DB1.TS1 TABLE(TB1)
USE PROFILE
```

In addition to just supporting profiles, whenever the optimizer determines the access path for an SQL action, it uses the available statistics available in the catalog, but it also looks to see if additional statistics might enable it to choose a better access path. If this is the case, and if the ZPARMS value STATFDBK_SCOPE is set up appropriately to give feedback, the optimizer populates the SYSSTATFEEDBACK table in the catalog table describing the additional statistics it desires to examine.

This SYSSTATFEEDBACK table identifies the object for gathering more statistics, and then the TYPE column identifies the type of statistics to be gathered, as shown in Table 4-1.

Table 4-1 SYSSTATFEEDBACK table

Type	Distribution
‘C’	Cardinality values
‘F’	Frequency values
‘H’	Histogram values
‘I’	Index statistics
‘T’	Table statistics

In V12, PROFILE usage was extended for support with INLINE statistics, and LISTDEF processing was improved to add defaults.

With V12, the profiles stored in the catalog are automatically maintained during `BIND` or `PREPARE` processing with direct updates by the Db2 optimizer. These updates are made to the profiles to ask for statistics that it thinks could be used to possibly pick a better access path.

For example, if an index exists on a column, but no statistics are found for that index, it will add the `INDEX(isname)` specification to the profile. Likewise, if distribution or histogram statistics are determined to be needed, the appropriate keywords (for example, `HISTOGRAM`) and parameters are added to the profile in the catalog.

If the following DDL operations are run, the appropriate updates are made to the profile specification so that it stays consistent:

- Drop Index
- Rename Index
- Drop Column
- Rename Column
- Rename Table

It also can removed unneeded distribution statistics

The feedback mechanism between the optimizer and **RUNSTATS** (or inline stats) can be represented as shown in Figure 4-1.

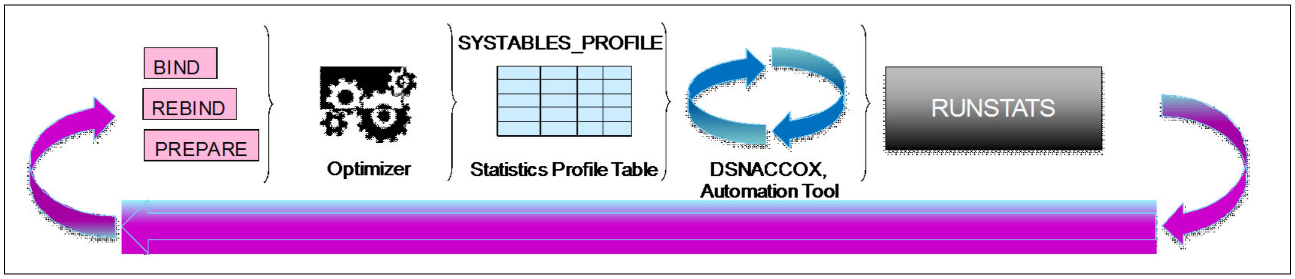


Figure 4-1 Optimizer and RUNSTATS feedback mechanism

Essentially, when profiles are set up with the `SET PROFILE` request during **RUNSTATS**, the profile table in the catalog table is used to gather statistics. The statistics gathered by **RUNSTATS** are used by `BIND`, `REBIND`, and `PREPARE` operations where the optimizer determines the access path.

The optimizer (or DDL statements), can make changes to an existing profile in the catalog. An automation tool examining the column `PROFILE_UPDATE`, so see if any profiles have changed, therefore requiring **RUNSTATS** to be run to gather the new statistics requested in the profile.

4.4.2 PROFILE keyword and LISTDEF

As of V12, a `LISTDEF` of utilities such as **REORG** or **RUNSTATS** can also use profiles for gathering statistics. If there is an object that doesn't have a profile in the catalog, then default statistics are gathered.

5

Db2 Real Time Statistics (RTS)

To maintain an efficient Db2 can require continuous and periodic monitoring of Db2 objects. Db2 object monitoring is an essential component due to the dynamic changes (growth, access), application behavior due to fluctuations in business activity, changes in data access patterns, and so on. Db2 deep technical skills are also becoming scarcer as experienced DBAs retire. Db2s running ERP-based applications may have more than 100,000 page sets to manage.

It is nearly impossible to manually monitor these objects for maintenance, such as **REORG**s. A lack of a proper maintenance strategy can indirectly cause performance problems, including running utilities without getting any real benefits.

Db2 Real Time Statistics (RTS) is an integral part of Db2's journey towards self-managing utilities. By having real-time statistics available, the Db2 utilities can automatically size work and temporary data set allocations. This benefits all tasks related to sort as far as the utilities are concerned.

RTS is used more extensively to determine which Db2 objects require maintenance for utilities, such as **REORG**, **RUNSTATS**, or **COPY**. The Db2 Stored Procedure, **DSNACCOX**, helps you to obtain recommendations for when to reorganize, copy, or update statistics for both table and index spaces.

You can define your own threshold to let **DSNACCOX** identify object maintenance (**REORG**s, **RUNSTATS**), but also use it to revisit your backup (**COPY**) strategy vis-à-vis object update activity. In addition, there are many other supplied thresholds to identify maintenance. **DSNACCOX** can be scheduled to run using the Db2 Administration Console, or your own Job Scheduler.

© Copyright IBM Corp. 2018. All rights reserved.

5.1 RTS overview

This section contains the following sections:

- How RTS is collected
- How RTS is externalized
- Where RTS information is stored

5.1.1 How RTS is collected

When Db2 is started, RTS is collected by the `DBM1` address space. The component of DB2 responsible for RTS functions is the RTS manager. The RTS manager runs as a subtask in the `DBM1` address space. For each object there is only one row in the relevant Catalog table. A size of approximately 140 bytes per pageset or partition is kept above the bar for each block. The RTS manager ensures that its active statistics blocks are kept in clustering order, and all inserts and updates to the rows in the RTS tables are accomplished through the clustering index.

In a data sharing environment, each member starts collecting statistics. Db2 generates in-memory statistics for each table space and index space, including catalog objects.

A control block is allocated in memory for each table and index page set as follows:

- For a tablespace, this control block is allocated the first time that the object is first updated.
- For an indexspace, it is different. A control block is allocated at the time of opening the index.

5.1.2 How RTS is externalized

Db2 periodically writes these real-time statistics to the two RTS catalog tables. It is done via an interval which is specified in **`DSNPARM`**. The system parameter, **`STATSINT`**, controls this with a default of 30 minutes. You can change this system parameter dynamically without recycling Db2.

The following list describes other ways that RTS is externalized:

- During the **`INIT`** phase of the Db2 utilities: **`DSNUTILB`**
- Using the Db2 command **`-ACCESS DB(*) SP(*) MODE(STATS)`**
- Stopping the table and indexspace
- Stopping RTS database **`-STOP DATABASE(DSNDB06) SPACENAM(SYSTSTSS)`**
- Stopping the Db2 subsystem

Note: offline utilities (those that are named `DSN1*`) do not have an **`INIT`** phase and subsequently do not flush the RTS information.

Important: DB2 does not maintain RTS for RTS objects. If you run a utility that includes RTS objects, RTS is not externalized for any of the objects in the utility list. DB2 does not maintain incremental counters for RTS against `SYSLGRNX` and its indexes during utility run. DB2 only maintains statistics for these objects during non-utility operations.

5.1.3 Where RTS information is stored

Db2 stores the RTS information in two Db2 Catalog tables:

- `SYSIBM.SYSTABLESPACESTATS`: Contains RTS statistics on table spaces and table space partitions
- `SYSIBM.SYSINDEXSPACESTATS`: Contains RTS statistics on index spaces and index space partitions

Capturing RTS history can provide you with many benefits, such as capacity planning (see how your objects are growing), partition strategy planning (where are my partitions growing), redefining your maintenance strategy (when do reorgs take place, and why do you run them), and so on. You can also use it for object activity for your backup strategy.

Db2 12 provides RTS history in two new Catalog tables. They have the same columns and same data types. These are temporal tables and you need to enable the relationship yourself, should you want to implement RTS history. The two RTS history tables are:

- `SYSIBM.SYSTABLESPACESTATS_H`
- `SYSIBM.SYSTABLEIDEXSPACE_H`

To enable the temporal relationship between the RTS Catalog table and the history table, you need to issue a Db2 **ALTER** statement. Example 5-1 shows the DDL to achieve that.

Example 5-1 ALTER statement

```
ALTER TABLE SYSIBM.SYSINDEXSPACESTATS
  ADD VERSIONING
  USE HISTORY TABLE SYSIBM.SYSIXSPACESTATS_H;
ALTER TABLE SYSIBM.SYSTABLESPACESTATS
  ADD VERSIONING
  USE HISTORY TABLE SYSIBM.SYSTABSPACESTATS_H;
```

If you want to remove RTS history, or you disconnect the temporal relationship as follows, remember that you can always turn off historical RTS collection by severing the temporal relationship. This is accomplished by issuing **ALTER TABLE** with the `DROP VERSIONING` clause (Example 5-2).

Example 5-2 ALTER TABLE

```
ALTER TABLE SYSIBM.SYSTABLESPACESTATS
  DROP VERSIONING;
```

By turning *on* RTS history, you should plan for some additional capacity (depending on how much history you want to maintain) and plan your maintenance and a purging (archiving) of historical information. Note that at each externalization (`STATSINT` 30-minute default), “old” information will be written to the system temporal tables. As the history table grows over time, plan to manage that growth. Customers typically aggregate this data on a weekly or monthly basis for up to a year and then go to a yearly basis when greater than a year.

It is always advised to establish a baseline for RTS. You might find missing statistics and it is typically due to some objects that never had a **REORG** utility run. Very old objects (before RTS) can also have missing statistics. For newly created objects, there is no need to establish a base value, because Db2 maintains the counters from the beginning.

5.2 RTS table SYSIBM.SYSTABLESPACESTATS

The RTS table `SYSIBM.SYSTABLESPACESTATS` contains real time statistics for table spaces. It resides in database DSNDB06 in tablespace `SYSTSTSS`. It contains 48 columns in Db2 12. See the *Db2 12 for z/OS SQL Reference*, SC27-8859, for a complete description of each column and related data type.

In the following tables, we show the columns and provide information to use them for your maintenance tasks.

Table 5-1 depicts incremental statistics which are written to the RTS tables. This occurs when `STATSINT` (default 30 minutes) is reached. After the base values are established, the delta is recorded by DB2 and kept in memory. The following table has some practical use case examples in the “Usage” column. The usage does not contain an exhaustive list of examples.

5.2.1 SYSTABLESPACESTATS: Incremental statistics

Incremental statistics are added or updated during the operation against the table.

Table 5-1 and Table 5-2 on page 49 denote how DML activities affect the RTS counters.

Table 5-1 DML effects against SYSIBM.SYSTABLESPACESTATS

Operation	**End Result**
INSERT	Increment REORGINSERTS, STATSINSERTS, TOTALROWS, and COPYCHANGES. Can update NACTIVE, SPACE, EXTENTS, REORGUNCLUSTERINS, distinct PAGES, COPYUPDATELRSN, UPDATESTATSTIME, and DATASIZE.
UPDATE	Increment REORGUPDATES, STATSUPDATES, COPYUPDATEDPAGES, and COPYCHANGES counters. Can update REORGNEARINDREF, REORGFARINDREF, NACTIVE, SPACE, EXTENTS for VARCHAR tables, distinct NPAGES, COPYUPDATELRSN, UPDATESTATSTIME, and DATASIZE.
DELETE	Increment REORGDELETEs, STATSDELETES, COPYCHANGES, and DATASIZE.
DELETE without WHERE clause	Increment the MASSDELETE/DROPs counter.
ROLLBACK – after INSERT	Decrement the INSERT counters.
ROLLBACK – after DELETE	Decrement the DELETES counter.
ROLLBACK – after UPDATE	Decrement the UPDATE counters.
ROLLBACK – after MASS DELETE	Will not decrement the MASSDELETES counter.
ROLLBACK – after DROP TABLE	Will not decrement the DROPs counter.
TRIGGERS	Statistics counters will not be updated.
Db2 RESTART	May cause statistics to be updated for other objects.

Table 5-2 SYSIBMSYSTABLESPACESTATS: Incremental

Column name	Short description	Usage example
DBID	Internal identifier of the database	Use when you need to perform DBID translations.
PSID	Internal identifier of the page set	Use when you need to perform PSID translations.
PARTITION	The partition number	Use together with SIZE and SPACE to determine partition strategy.
INSTANCE	Data set instance of the object	Use with PSID and DBID for reporting.
DBNAME	Name of database	
NACTIVE	Number of actual active pages	Use to see how PAGEFREE affects size. Can also be used when using MAXROWS per page and sizing.
NPAGES	Number of distinct pages containing row data	Use to calculate estimate of LOB data. Also, use to see how PBR is affecting part sizes.
EXTENTS	Number of extents for the space	If a data set is striped this value denotes the logical extents. The column is typically used for REORG planning. Can be looked at when using EA (Extended Addressability – pagesets > 4GB) effects.
SPACE	In KB, the amount of space	Use to revise your LISTDEF groupings for COPY utility and many more.
TOTALROWS	Total # of rows or LOBS	Use to determine PBR efficiency.
DATASIZE	Size in byes for row data	Use to estimate FREEPAGE and/or compression considerations.
UNCOMPRESSED	Not used – always zero	
HASHLASTUSED	A date column – when last used by DML	
DRIVETYPE	Device type – HDD or SSD	Use with your PBG strategy – Can be used to determine where to place most volatile / most active pagesets – SSD or HDD.
LPFACILITY	1 Byte (Y/N) indicator disk control unit has high performance list prefetch	Use with DRIVETYPE – stale pagesets on slower disks and highly reference pagsesets on SSD with high performance prefetch.
UPDATESIZE	# of bytes updated since last creation, last REORG or LOAD REPLACE	Can be used to revise your backup strategy based on activity and not just size.
LASTDATACHANGE	Time RTS was last updated due to data modification	Consider using this counter to determine of a TS is opened for read only or update. Can be used with your BP planning too.
SYS_START	System-period versioning start time	
SYS_END	System-period versioning end time	
TRANSEND	Transaction-id-column for period versioning	
GETPAGES	# of GETPAGE requests since creation of last REORG	Use this data together with REORGUPDATES and COPYUPDATEDPAGES to determine object activity. Use for BP object placement when using together with size.

5.2.2 SYSTABLESPACESTATS – Columns affected by the REORG utility

SYSTTABLESPACESTATS columns affected by the `REORG` utility are shown in Table 5-3.

Table 5-3 SYSTTABLESPACESTATS columns affected by the REORG utility

Column name	**Short description**	**Usage example**
REORGLASTIME	Timestamp of last REORG, LOAD REPLACE and TS was created	Use as date to determine next REORGs. In RTS history use to determine how often REORG is run.
REORGINSERTS	# of inserts since last REORG, LOAD REPLACE or creation	Use together with PBG size to see where rows are inserted. Use this data in RTS history to see insert activity between reorgs.
REORGDELETES	# of deletes since last REORG, LOAD REPLACE and creation	Use together with REORGINSERTS, -DELETES and –UPDATES to see activity between REORGS.
REORGUPDATES	# of updates since last REORG, LOAD REPLACE and creation	Use together with REORGINSERTS, -DELETES and –UPDATES to see activity between REORGS.
REORGUNCLUSTINS	# of rows inserted that were not clustered well since last REORG, LOAD REPLACE of creation. "Well-clustered" denotes a record within 16 pages of candidate page.	Use as an exception-based column to determine REORGs and RUNSTATS. Also, potentially use to REORG the clustering index.
REORGDISORGLOB	# of LOBS that are not 100% chunked since last REORG, LOAD REPLACE or creation.	Use for exception-based REORG.
REORGMASSDELETE	# of mass deletes since last REORG, LOAD REPLACE or object creation.	Use for exception-based REORG when counter is high.
REORGNEARINDREF	# of overflow rows near the pointer record	Use for exception-based REORG planning. Use with DSNACCOX or IBM Db2 tooling.
REORGFARINDREF	# of overflow far from the pointer record	Counter can be used for exception based REORGs: see DSNACCOX.
REORGSCANACCESS	# of times data is accessed using DML since last REORG, LOAD REPLACE or creation	Use this counter together with GETPAGES to determine BP placement, backup strategy.
REORGHASHACCESS	# of times hash access is used since last REORG, LOAD REPLACE or creation.	Use together with HASHLASTUSED to determine complete hash access.
REORGCLUSTERSENS	# of times SQL requires data in cluster order	If this counter is low, consider not using REORGUNCLUSTINS as the main indicator for REORG.

5.2.3 SYSTABLESPACESTATS – Columns affected by the LOAD utility

SYSTTABLESPACESTATS columns affected by the LOAD utility are shown in Table 5-4.

Table 5-4 SYSTTABLESPACESTATS columns affected by the LOAD utility

Column	Short Description	Use case examples
LOADLASTTIME	Timestamp of the last LOAD	Use with RTS history to determine LOAD frequency which in turn can indicate how many times you have to REBUILD IXs. Use this counter in RTS history together with INDEX ACCESS in SYSINSPACESTATS to determine IX usage.

5.2.4 SYSTABLESPACESTATS – Columns affected by the COPY utility

SYSTTABLESPACESTATS columns affected by the COPY utility are shown in Table 5-5.

Table 5-5 SYSTTABLESPACESTATS columns affected by the COPY utility

Column	Short Description	Use case examples
COPYLASTTIME	Time of the last time COPY was run – both FULL and Incremental	Use to determine when the last COPY was run. This includes in-line copies by REORG and LOAD. Use this counter in RTS history to see how often you COPY and when.
COPYUPDATEDPAGES	A counter to denote the actual pages updated since the last COPY	Use this counter to see the update activity between COPY runs. If this value is low, determine of IIC is better. If very high, use for potential FCIC or more regular runs.
COPYCHANGES	Number of rows loaded since last IC or update, delete and inserts since last IC	Use this counter with COPYUPDATEDPAGES to see update activity between copies. Especially in RTS history. See more details in best practices
COPYUPDATELRSN	The RBA/LRSN associated with the latest IC	Can be used together with current RBA/LRSN in a calculation – to determine when to IC next
COPYDATETIME	Timestamp of first update since last IC	Can be used for exception-based backups based on time

5.2.5 SYSTABLESPACESTATS – columns affected by RUNSTATS utility

SYSTTABLESPACESTATS columns affected by the RUNSTATS utility are shown in Table 5-6.

Table 5-6 SYSTTABLESPACESTATS columns affected by the RUNSTATS utility

Column name	Short description	Use case examples
STATSLASTTIME	Timestamp of the last execution of RUNSTATS	Use to potentially determine how often should RUNSTATS run. Use RTS history equivalent to see when and how often RUNSTATS is actually run.
STATSINSERT	# of rows or LOBs that were inserted since LOAD without REPLACE and since last RUNSTATS	For your most active objects use the following 3 counters to set a threshold to run RUNSTATS by exception.
STATSDELETES	# of rows or LOBs that were deleted since LOAD without REPLACE and since last RUNSTATS	See above
STATSUPDATES	# of rows or LOBs that were updated since the last RUNSTATS or since object creation.	See above
STATSMASSDELETE	# of mass deletes from a segment or LOB table space, or the number of tables dropped from a segmented table space since RUNSTATS was last run.	See above

5.3 RTS table SYSIBM.SYSINDEXSPACESTATS

Db2 records RTS information for indexes into SYSIBM.SYSINDEXSPACESTATS and externalizes this information the same as for tablespace statistics. The only difference is that for indexspace statistics, information is recorded when the space is first opened versus for tablespaces it is the first time the space is updated. In the following table we show the columns in `SYSINDEXSPACESTATS` and how you can potentially use these counters for object maintenance.

For DML operations, the incremental statistics are effected as follows in Table 5-7 and Table 5-8 on page 53.

Table 5-7 Operational effects for SYSINDEXSPACESTATS

Operation	End result
INSERT for a COPY YES INDEX	In addition to INSERT, update COPYUPDATELRSN, COPYUPDATEDPAGES, COPYCHANGES, and UPDATESTATSTIME
DELETE for a COPY YES INDEX	In addition to INSERT, update COPYUPDATELRSN, OPYUPDATEDPAGES, COPYCHANGES, and PDATESTATSTIME
DROP without a WHERE clause	Increment REORGMASSDELETE
DROP TABLE	Increment REORGMASSDELETE
ROLLBACK – after INSERT	Decrement the INSERT counter
ROLLBACK – after DELETE	Decrement the DELETE counter

ROLLBACK – after UPDATE	Decrement the UPDATE counter
ROLLBACK – after MASS DELETE	Will not decrement the MASSDELETES counter
ROLLBACK – after DROP table	Will not decrement the DROPs counter.
Db2 RESTART	Statistics counters will not be updated.

Table 5-8 SYSINDEXSPACESTATS – Incremental counters and usage

Column name	Short description	Use case examples
UPDATESTATSTIME	Timestamp of the latest insert or update	Potentially use this counter to determine updates. If the time is older, use RTS history to determine the trends.
NLEVELS	# of levels in the index tree	This counter is useful in RTS history. It will show how the index tree levels increase over time. You can also look at the history to see how the levels may increase after REORG INDEX.
NPAGES	The number of completely empty pseudo-deleted pages	
NLEAF	# of leaf pages on the index	Using a value of LEAFFAR / NLEAF > 10% for LOB auxiliary indexes for REORG INDEX
NACTIVE	# of active pages in the index space	Can be used as one indicator to determine of you want to IC the IX.
SPACE	KB space size of the index	As above – if the index spaceset is very large, consider doing an IC. Modern advice is to use update activity and not always size.
EXTENTS	# of extents. For striping it is the # of logical extents	Use for exception—based REORG of INDEX space.
IBMREQD	Value of Y denotes row comes from tape.	
DBID	Internal ID of the DB	
ISOBID	Internal identifier of the index space set	
PSID	Internal identifier of the space set for the index	
PARTITION	The data set number of the index	
INSTANCE	Indicates the object association with data set 1 or 2	
TOTALENTRIES	# of entries in the space – including duplicate entries	
DBNAME	Database name	
NAME	Index name	
CREATOR	Index creator	
INDEXSPACE	Index space name	
LASTUSED	Date the index was last used by DML and SELECT. Default is now NULL	If NULL, consider dropping this IX.

DRIVETYPE	HDD and SSD (for Solid State Drive)	
GETPAGES	# of GETPAGE requests since last REORG or creation	
SYS_START	Used for system-period versioning	
SYS_END	Used for system-period versioning	
TRANS_START	Used for system period versioning	

5.3.1 SYSINDEXSPACESTATS columns affected by the COPY utility

SYSINDEXSPACESTATS columns affected by the COPY utility are shown in Table 5-9.

Table 5-9 SYSINDEXSPACESTATS columns affected by the COPY utility

Column name	Short Description
COPYLASTTIME	Timestamp of full image copy
COPYUPDATEDPAGES	# of pages updates since full copy
COPYCHANGES	# of updates since last full copy
COPYDATELRSN	RBA or LRSN since last full copy
COPYUPDATETIME	Timestamp of first update since last full copy

5.3.2 SYSINDEXSPACESTATS columns affected by the REORG INDEX utility

The SYSINDEXSPACESTATS columns affected by the REORG INDEX utility are shown in Table 5-10.

Table 5-10 SYSINDEXSPACESTATS columns affected by the REORG INDEX utility

Column name	Short description
REORGLASTTIME	Timestamp since last REORG INDEX
REORGINSERTS	# of inserts since last REORG, REBUILD, LOAD REPLACE or creation
REORGDELETES	# of deletes since last REORG, REBUILD, LOAD REPLACE or creation
REORGAPPENDINSERT	# if entries that have a key > than max key value since last utility of creation
REORGPSEUDODELETES	# of pseudo deleted entries since REORG, LOAD REPLACE of creation
REORGMASSDELETE	# of mass deletes from segmented or LOB space
REORGLEAFNEAR	# of leaf pages near previous leaf pages
REORGLEAFFAR	# of leaf pages away from previous leaf pages
REORGNUMLEVELS	# of levels in the index tree since last REOG, REBUILD INDEX or creation

REORGINDEXACCESS	# of times this index was accessed with SELECT, FETCH, UPDATE or DELETE since last REORG, REBUILD or creation – this includes RI enforcement

5.3.3 SYSINDEXSPACESTATS columns affected by the LOAD utility

The SYSINDEXSPACESTATS columns affected by the LOAD utility are shown in Table 5-11.

Table 5-11 SYSINDEXSPACESTATS columns affected by the LOAD utility

Column name	Short description
LOADLASTTIME	Timestamp of the last LOAD REPLACE

5.3.4 SYSINDEXSPACESTATS columns affected by the REBUILD INDEX utility

The SYSINDEXSPACESTATS columns affected by the REBUILD INDEX utility are shown in Table 5-12.

Table 5-12 SYSINDEXSPACESTATS columns affected by the REBUILD INDEX utility

Column name	Short description
REBUILDLASTTIME	Timestamp of the last REBUILD IX

5.3.5 SYSINDEXSPACESTATS columns affected by RUNSTATS

The SYSINDEXSPACESTATS columns affected by RUNSTATS are shown in Table 5-13.

Table 5-13 SYSINDEXSPACESTATS columns affected by RUNSTATS

Column name	Short description
STATSLASTTIME	Timestamp of the last RUNSTATS
STATSINSERTS	# of IX entries inserted since last RUNSTATS
STATSDELETES	# of IX entries deleted since last RUNSTATS
STATSMASSDELETE	# of IX entries mass deleted since last RUNSTATS

6

Repairability and REPAIR CATALOG

As of V11, Db2 does a “sanity check” with the first physical open of a page set to ensure that the schema definitions for tables within the page set match their descriptions in the catalog. If there is a mismatch, there will be an abend with reason code `X'00C900E3'`. Utilities that scan table spaces also do this check, and on a mismatch, the utility fails with the following message:

```
DSNU590I,  RESOURCE NOT AVAILABLE, REASON=X'00C900E3'
```

This reliability checking failure is typically caused when moving objects using **`DSN1COPY`** to clone table spaces across systems. In this environment with changing schema definitions, the catalog definition for the table can get out of sync with the underlying format of the data rows in the page set.

Customers using cloning tools are also at risk because these products can do similar copying of underlying data sets. It’s important to catch schema mismatches early, because they can result in internal overlays of memory, and these types of problems have been known to occasionally crash Db2 systems.

In the past, it was difficult to detect schema mismatches, but in V11, a design point of “self-describing objects” when creating an object with **`CREATE TABLESPACE`** and **`CREATE TABLE`** enable this checking to take place.

© Copyright IBM Corp. 2018. All rights reserved.

For all newly created object-based table spaces, the initial metadata description is embedded in the table space within the `HEADER` page and `SYSTEM` pages, as shown in Figure 6-1.

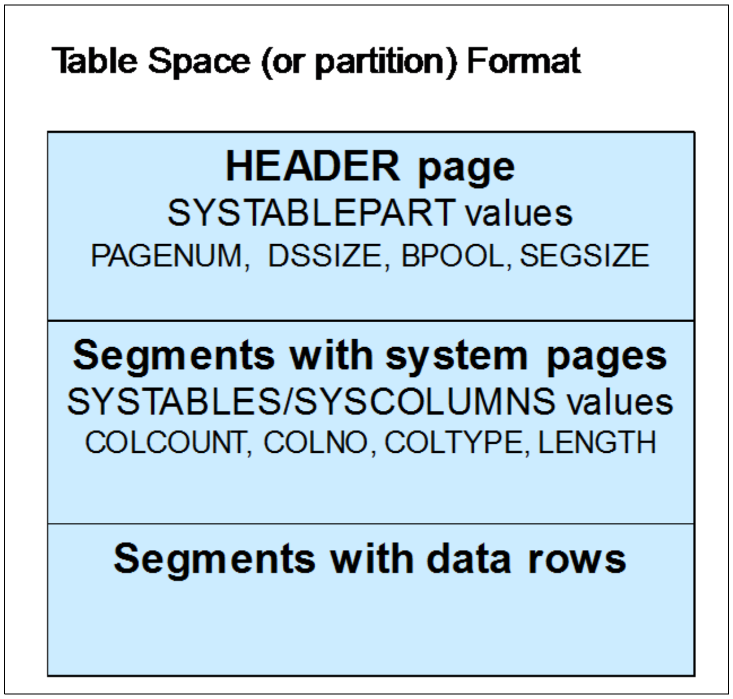


Table Space (or partition) Format
HEADER page
SYSTABLEPART values
PAGENUM, DSSIZE, BPOOL, SEGSIZE
Segments with system pages
SYSTABLES/SYSCOLUMNS values
COLCOUNT, COLNO, COLTYPE, LENGTH
Segments with data rows

Figure 6-1 Table space (or partition) format

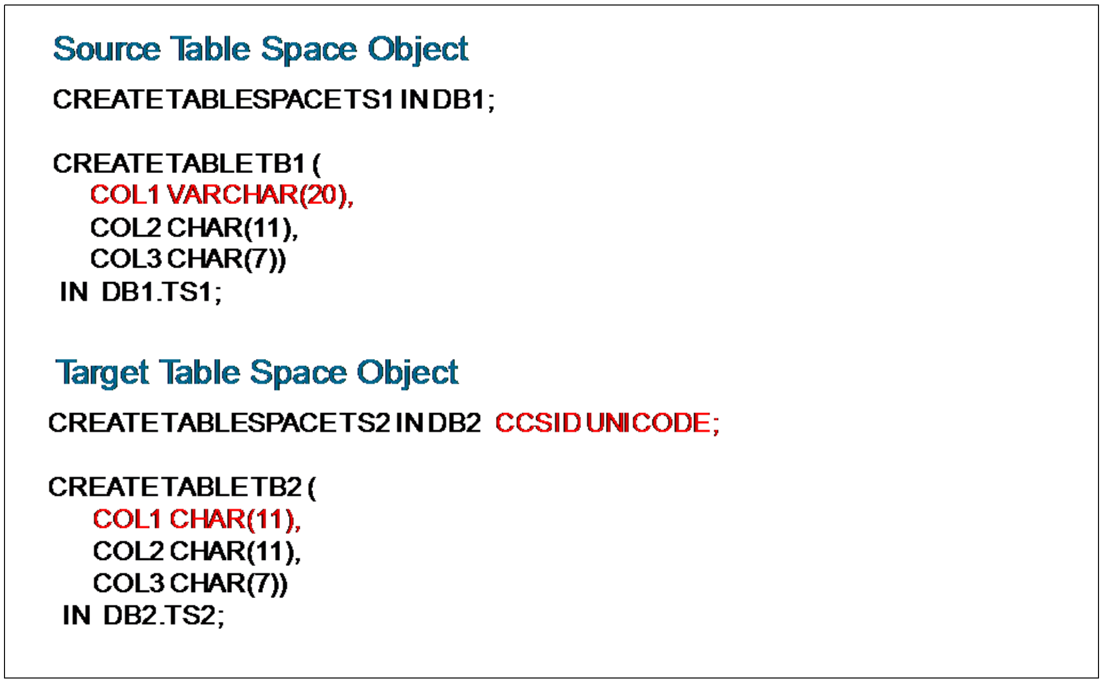
Critical table space attributes are kept in the header page for each partition, while the table and column descriptions for each table are kept within system pages. For tables, **`CREATE TABLE`** embeds a system page with a version 0 definition of the table, and if **`ALTER TABLE`** statements are executed to change data types of columns, change the length of columns, or add or drop columns, the new system page with version `n+1` is embedded with the current description when the first data row is inserted or updated to use that new description.

It's one thing to catch schema mismatches, but it's another to diagnose the exact problem so that the problem can be remedied. That's where the **`REPAIR CATALOG`** and **`REPAIR CATALOG TEST`** utilities help.

6.1 REPAIR CATALOG TEST

REPAIR with the `CATALOG TEST` keywords should be run whenever a potential schema mismatch is suspected (after an abend with reason code `00C900E3`), or it can be run periodically as a general health check.

As an example of how it can help, consider an example with a source table space `TS1` containing table `TB1`, and a target table space `TS2` containing table `TB2`. Suppose the tables should be maintained in parallel, but for some reason, while changes were being made, they weren't applied together, so the definition got out of sync, as shown in Figure 6-2.



Source Table Space Object

```
CREATE TABLESPACE TS1 IN DB1;

CREATE TABLE TB1 (
    COL1 VARCHAR(20),
    COL2 CHAR(11),
    COL3 CHAR(7))
 IN DB1.TS1;
```

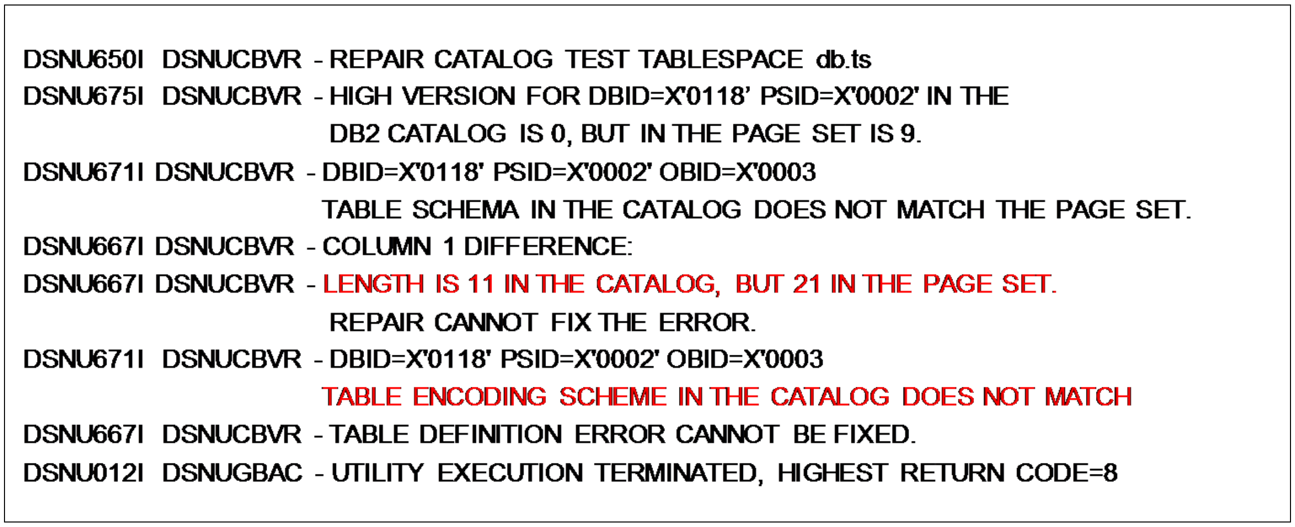
Target Table Space Object

```
CREATE TABLESPACE TS2 IN DB2  CCSID UNICODE;

CREATE TABLE TB2 (
    COL1 CHAR(11),
    COL2 CHAR(11),
    COL3 CHAR(7))
 IN DB2.TS2;
```

Figure 6-2 Source and target space objects

Notice the highlighted differences in red. If Source data sets for table space `TS1` are moved using **`DSN1COPY`** to copy over the data sets for `TS2`, then 20 characters for `COL1` cannot fit within the `COL1` definition of 11 characters for `COL1` in `TB2`. In addition, EBCDIC encoding can be recognized if the table space is defined as UNICODE.

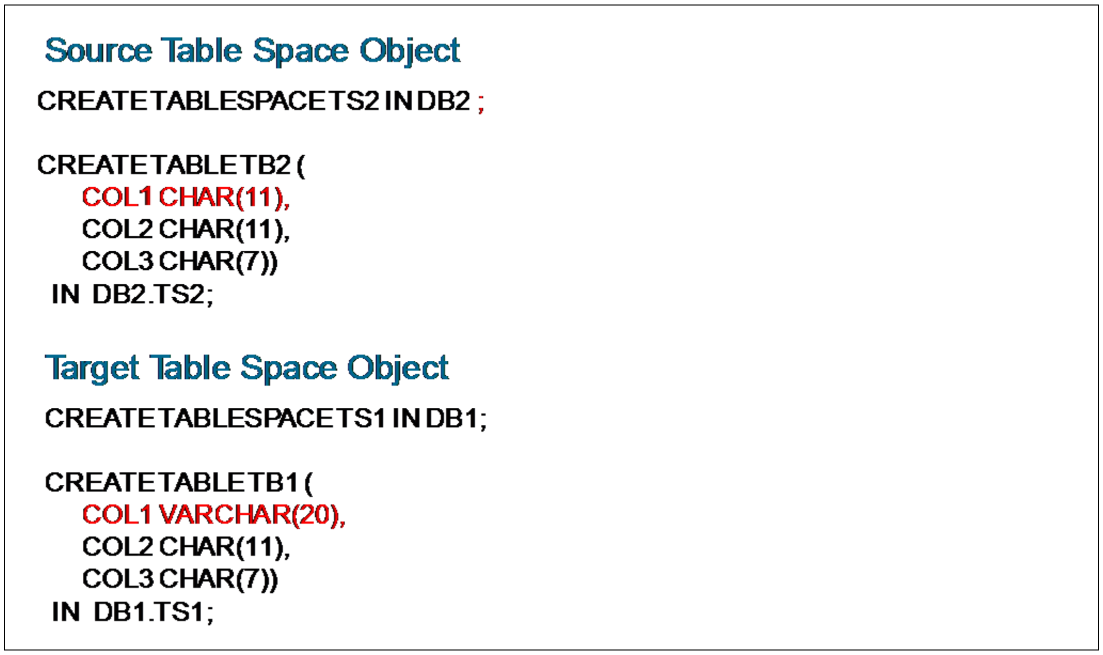
When **REPAIR CATALOG TEST DB2.TS2** is run, the system pages found within the underlying page set are compared against the catalog definition. In this case, the following messages are printed out, clearly showing the problem (Figure 6-3).



```
DSNU650I  DSNUCBVR - REPAIR CATALOG TEST TABLESPACE db.ts
DSNU675I  DSNUCBVR - HIGH VERSION FOR DBID=X'0118' PSID=X'0002' IN THE
                     DB2 CATALOG IS 0, BUT IN THE PAGE SET IS 9.
DSNU671I DSNUCBVR - DBID=X'0118' PSID=X'0002' OBID=X'0003
                    TABLE SCHEMA IN THE CATALOG DOES NOT MATCH THE PAGE SET.
DSNU667I DSNUCBVR - COLUMN 1 DIFFERENCE:
DSNU667I DSNUCBVR - LENGTH IS 11 IN THE CATALOG, BUT 21 IN THE PAGE SET.
                     REPAIR CANNOT FIX THE ERROR.
DSNU671I  DSNUCBVR - DBID=X'0118' PSID=X'0002' OBID=X'0003
                    TABLE ENCODING SCHEME IN THE CATALOG DOES NOT MATCH
DSNU667I  DSNUCBVR - TABLE DEFINITION ERROR CANNOT BE FIXED.
DSNU012I  DSNUGBAC - UTILITY EXECUTION TERMINATED, HIGHEST RETURN CODE=8
```

Figure 6-3 Output messages

Now change the scenario a bit by reversing the **DSN1COPY** and removing the UNICODE encoding as shown in Figure 6-4.



Source Table Space Object

```
CREATE TABLESPACE TS2 IN DB2 ;

CREATE TABLE TB2 (
   COL1 CHAR(11),
   COL2 CHAR(11),
   COL3 CHAR(7))
 IN DB2.TS2;
```

Target Table Space Object

```
CREATE TABLESPACE TS1 IN DB1;

CREATE TABLE TB1 (
   COL1 VARCHAR(20),
   COL2 CHAR(11),
   COL3 CHAR(7))
 IN DB1.TS1;
```

Figure 6-4 Source and target table space objects

In this case, the difference is noted with **REPAIR CATALOG TEST**. However, the discrepancy can be fixed by running **REPAIR CATALOG**. Why? Because, if the source table existed, and an **ALTER TABLE ALTER COLUMN COL1 VARCHAR(20)** was done, it would look like the target table with version `N+1`. **REPAIR CATALOG** changes the version in the catalog to `N+1`, so that when the underlying data is updated, it's changed to the latest version with format `VARCHAR(20)`.

6.2 Multi-table tablespaces and version wrapping

When making schema changes on a table or tables in a table space, there is a limit of 255 versions that can be active at one time for a table space. The most frequent type of schema change that creates a new version is the **ALTER** of a column in a table to a compatible data type, typically to hold larger values. For example, a **SMALLINT** column COL2 must be made larger, so the followed DDL is run:

```
ALTER TABLE TB1 ALTER COLUMN COL2 SET DATA TYPE INTEGER;
```

In addition to changing column types, other changes like an **ALTER** to add a new column to a table also creates a new version. If a table is never **ALTER**ed, then it remains at version zero.

The limit on the number of versions at 255 is usually plenty with a single table in the table space, but if there are a few hundred tables in the table space, then a version changing alter or two for each table can quickly reach the limit.

Wrapping is the term given when a version number is reused after reaching the limit of 255, and new versions are allocated again starting at 1. This can only occur if there are no current tables using version 0 or 1, and there are no image copies containing tables with rows for version 0 or 1.

If the current table space version is 255 (SYSTABLESPACE column CURRENT_VERSION) there is a blocking version 0 or version 1 indicated by (SYSTABLESPACE column OLDEST_VERSION, then the **ALTER** will return the following SQL error:

```
SQLCODE = -4732, ERROR:  THE MAXIMUM NUMBER OF ALTERS ALLOWED
HAS BEEN EXCEEDED FOR TABLE
54055 SQLSTATE RETURN CODE
```

The method to remove the blocking versions 0 and 1 in the table space are the following procedures.

6.2.1 Single table tablespaces

The following steps can be taken

1. Run the **REORG** utility. This will ensure that all data rows are materialized to the current version of 255.
2. Run the **MODIFY** utility to get rid of all image copies that have an oldest version of 0 or 1. The date range needed for the **MODIFY** can be determined by running **REPORT RECOVERY** to see the oldest versions as captured in the SYSCOPY records. After **MODIFY** is run, it will update the SYSTABLESPACE column OLDEST_VERSION to represent the oldest version still left in the SYSCOPY records.
3. Rerun the **ALTER**, and version 1 will be allocated as the current version after the previous version of 255.

6.2.2 Multi-table tablespaces

If all of the tables have a non-zero version, the procedure is the same as for table spaces with a single table. Otherwise, the procedure must first get rid of these version zero tables as follows:

1. Run **`REPAIR TABLESPACE`** with the keyword option `SETCURRENTVERSION`. This updates each table version to be the current table space version.
2. Run **`REORG`** to bring all table data rows to their current version.
3. Run **`MODIFY`** to get rid of all of the previous image copies that have an `OLDEST_VERSION` of zero.
4. Rerun the **`ALTER`** and the version can wrap to one.

For example, consider the following example of a table space with 4 tables. The lowest table versions are at zero, but the highest table version after the later **`REORG`** brought those tables to version 5.

Table 2 and Table 3 are at version zero. However, when **`REPAIR TABLESPACE`** with `SETCURRENVERSION` is run, the `SYSTABLES VERSION` column in the catalog is updated to 5. The next **`REORG TABLESPACE`** can then materialize all of the data rows to the current version of 5. See Figure 6-5.

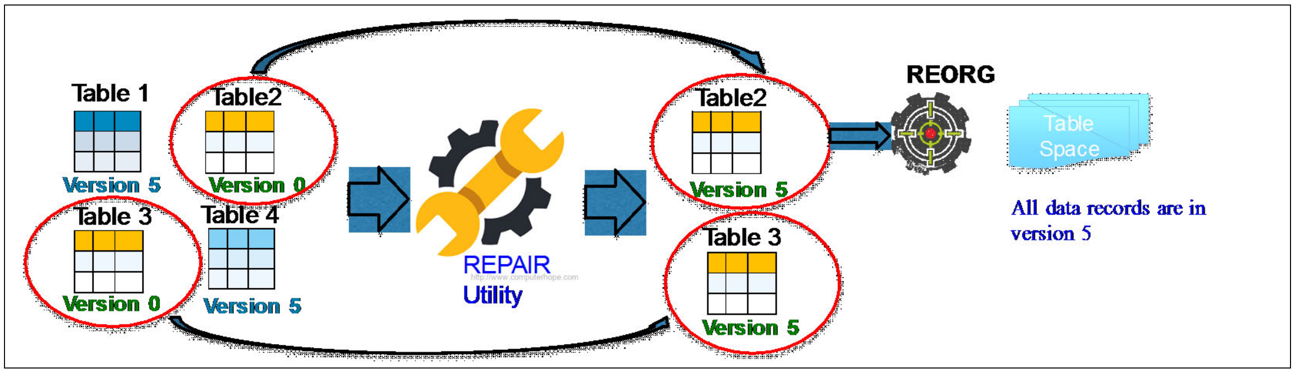


Figure 6-5 Multi-table tablespaces

7

Db2 DSNZPARMs for utilities

There are many Db2 subsystem parameters, DSNZPARMs, also known as ZPARMs, that you can tailor to manage Db2. In this chapter, we cover only those subsystem parameters that affect your Db2 utilities. These Db2 utility DSNZPARMs can be altered and changed by you as and when required, and almost all of them can be changed without recycling Db2, i.e. dynamically. The effect of some ZPARMs can be overwritten with utility syntax at utility execution time.

There are macros that get assembled to produce the DSNZPARM object modules into your `SDSNEXIT` load library: `DSN6ENV`, `DSN6ARVP`, `DSN6LOGP`, `DSN6FAC`, `DSN6GRP`, `DSN6SYSP`, and `DSN6SPRM`. The information in this chapter does not cover the installation and customization process of ZPARMs.

This chapter describes the ZPARMs (because they affect the utilities) by sections based upon general usage, sort, dynamic allocation of work data sets, and so on.

© Copyright IBM Corp. 2018. All rights reserved.

7.1 ZPARMs affecting dynamic allocation of data sets

The Db2 utilities may require many data sets for sorting purposes. The following table denotes the two ZPARMs that affect the work data sets, which affect the dynamic allocation for sorting when invoking **DFSORT** or the Db2 Sort product.

When **UTSORTAL** is set to YES, DB2 uses RTS for the estimates.

When not using the **SORTNUM** elimination feature (subsystem parameter UTSORTAL=NO) and no **SORTNUM** is specified, DB2 queries the **DFSORT** installation option value, DYNALLOC, to be used for **SORTNUM**.

In addition, when not using **SORTNUM** elimination (UTSORTAL=NO), Db2 uses the **RUNSTATS SPACE** statistics from the Db2 catalog to estimate the FILSZ value. If **RUNSTATS** has not been run, when the table space is compressed or the table space contains rows with VARCHAR columns, Db2 may not be able to accurately estimate the number of rows. If the estimated number of rows is too high and the sort work space is not available, or if the estimated number of rows is too low, **DFSORT** might fail and cause an abend (Table 7-1).

Table 7-1 ZPARMS affecting sort dynamic allocation

Parameter	Values	What it does
UTSORTAL	<u>YES</u> | NO	Enable dynamic allocation of SORTWK data sets. Uses DB2 RTS to estimate the number of rows to sort.
IGNSORTN	YES | NO	Ignore SORTNUM statements when coded.

Tip: It is strongly advised to set these parameters to YES and let the utility (**LOAD**, **REORG**, **REBIULD IX**, or **CHECK INDEX**) size and allocate the data sets for sorting purposes.

7.2 ZPARMs affecting utility timeout operations

The **UTIMOUT** subsystem parameter specifies how long, in number of resource values, that a utility or utility command is to wait for a resource. The utility or utility command waits until a lock or all claims on a resource of a claim class is released (Table 7-2).

Table 7-2 ZPARMS affecting utility timeouts

Parameter	Values	What it does
UTIMOUT	1-254	How long a resource is to wait for resources. This value multiplied by IRLMRWT.

Tip: For example, if you use the default value of 6, a utility can wait six times longer than an SQL application for a resource. This option allows utilities to wait longer than SQL applications to access a resource. The value of **UTILITY TIMEOUT** is used as the default value for the **RETRY** parameter of DB2 utilities, such as **CHECK INDEX** and online **REBUILD INDEX**.

7.3 ZPARMs affecting SORT operations

The following three system parameters affect the **SORT** operations for utilities, such as **REBUILD INDEX**, **LOAD**, **REORG**, and **CHECK**. When the utilities allocate the relevant temporary and work data sets, it uses these values in conjunction with your DFSMS storage classes. In addition, should you have the IBM DB2 SORT separately licensed product, this is where you enable the usage without making any other changes to your JOBS or even JCL (Table 7-3).

Table 7-3 ZPARMS affecting sort operations

Parameter	Values	What it does
VOLTDEFT	SYSDA	Device type or unit name that is to be used by DB2 utilities for dynamically allocating temporary data sets. Used for COPY CONCURRENT (DFSMSdss) and CHECK data sets.
UTIL_TEMP_STORCLAS		DFSMS storage class that the CHECK INDEX, CHECK DATA, and CHECK LOB utilities are to use when allocating temporary shadow data sets. (These utilities allocate shadow data sets when the SHRLEVEL CHANGE option is used.)
DB2SORT	ENABLE/ DISABLE	To enable the use of DB2 SORT TOOL instead of DFSORT for utility sort processing. DB2 SORT should be licensed and installed.

Tip: With the Db2 Catalog and Directory objects being SMS Managed, it is also strongly advised to have all of your application table spaces SMS managed. The utilities can use the DFSMS Storage Class and Management Class constructs, making data set management from a Db2 point of view much more efficient and optimized.

UTIL_TEMP_STORCLAS: The default value of blank indicates that the shadow data sets are to be defined in the same storage class as the production page set.

7.4 ZPARMS affecting RUNSTATS operations

From a utilities point of view, **REAL TIME STATS** (also known as RTS) is used for dynamic allocation, sizing of the work and temporary data sets, and many more functions. When running **RUNSTATS**, it is good to know how, when and how much **RUNSTATS** adds to the Db2 Catalog tables (Table 7-4 on page 66).

If you must record history, note the acceptable values of SPACE, NONE, ALL, and ACCESSPATH (with a default of NONE). Here a description:

- SPACE: All inserts and updates that DB2 makes to space-related catalog statistics are recorded.
- NONE: None of the changes that DB2 makes in the catalog are not recorded.
- ALL: All inserts and updates that DB2 makes in the catalog are recorded.
- ACCESSPATH: All inserts and updates that DB2 makes to ACCESSPATH-related catalog statistics are recorded.

Table 7-4 ZPARMS affecting RUNSTATS

Parameter	Values	What it does
STATROLL	YES/NO	Specifies whether the RUNSTATS utility is to aggregate the partition-level statistics, even though some parts might not contain data
STATHIST	SPACE/NONE/ALL/ACCESSPATH	Should inserts and updates be recorded in catalog history tables

Tip: Consider setting `STATROLL` to `YES`, if you have large partitioned table spaces and indexes: it may help optimizer to choose a better access path.

7.5 ZPARMS affecting IBM FlashCopy operations

Previously, in Db2 V8, with the **BACKUP** and **RESTORE SYSTEM** utilities, the utilities started to use and exploit DASD fast replication, also known as IBM FlashCopy.

In Db2 V9, more enhancements were made to allow individual table spaces or index spaces to be recovered from a System Level Backup (SLB). Also, Db2 9 provided the backup to be offloaded to tape. The I/O for physical copying to disk in the background can be reduced by the use of incremental FlashCopy. Even if incremental FlashCopy is used, the dump to tape is always a full dump.

Increased availability with new online functions **CHECK DATA**, **CHECK INDEX**, and **CHECK LOB** is enhanced to run in a **SHRLEVEL CHANGE** mode. **SHRLEVEL CHANGE CHECK DATA** (**CHECK INDEX** and **CHECK LOB**) work on a copy of the related table spaces and indexes. The copies (shadows) are taken by Db2 using the DFSMSdss (**ADRDSSU**) utility. Integrity checking is done on the shadow data sets.

REBUILD INDEX can run now in **SHRLEVEL CHANGE** mode. The **REBUILD** utility now has a `LOGPHASE` when **SHRLEVEL CHANGE** is specified. There are **DRAIN WAIT** options similar to **REORG** to control the final drain of writers before the index can be made available.

The **REPAIR** utility has been enhanced so that **LOCATE** can be run against indexes, index spaces, and table spaces with **SHRLEVEL CHANGE**. This does not apply to `LOB` table spaces. Starting in Db2 10 and later, Db2 provides FlashCopy Image Copy (FCIC) for copying table space. The following system parameters affect the use of IBM FlashCopy. Their operations can be overwritten on utility syntax for the most part (Table 7-5 on page 67).

Table 7-5 ZPARMS affecting FlashCopy operations

Parameter	Values	What it does
COPY_FASTREPLICATION	PREFERRED/ REQUIRED/ NONE	PREFERRED - The COPY utility directs DSS COPY to use fast replication if possible. If FlashCopy cannot be used, then DSS uses traditional data movement methods. REQUIRED - The COPY utility directs DSS COPY to only use fast replication, ensuring that object copies occur as quickly as possible. This option causes the COPY utility to fail if FlashCopy cannot be used. This option might reduce the opportunity for resource contention and unavailability for SHRLEVEL REFERENCE copies. NONE - The COPY utility directs DSS COPY not to use fast replication. Traditional data movement methods are be used.
FLASHCOPY_COPY	YES/NO	Specifies that the COPY utility uses FC technology when the FLASHCOPY option / keyword is not specified in the utility control statement
FLASHCOPY_LOAD	YES/NO	LOAD utility uses FC technology when the FLASHCOPY option is not specified in the control statement – inline copy
FLASHCOPY_REORG_TS	YES/NO	Inline copy at load phase for REORG – when not coded on REORG control statement
FLASHCOPY_REORG_IX	YES/NO	As above
FLASHCOPY_REBUILD_IX	YES/NO	Can Rebuild IX use FlashCopy by default when not specified on the REBUILD INDEX utility control statement
CHECK_FASTREPLICATION	PREFERRED/ REQUIRED	Specifies the type of replication that DFSMSdss COPY uses to copy objects to shadow data sets
REC_FASTREPLICATION	NONE/ PREFERRED/ REQUIRED	Should RECOVER utility use FC to recover from a FlashCopy image copy
FLASHCOPY_PPRC	NONE/ PREFERRED/ REQUIRED/ blank	Whether DFSMSdss preserves mirroring while processing a DB2 utilities request Whether the target device pair is allowed to go to duplex pending state
FCCOPYDDN	HLQ.&DB..&SN.. N&DSNUM..&UQ.	Template for the output VSAM data set name for any FlashCopy image copy.

Tip: Consider (depending on if you are considering data set level `FCIC` or `SYSTEM LEVEL BACKUP`) enabling FlashCopy through utility syntax. When, and if, you are using FlashCopy backups everywhere, update the ZPARMS to match your criteria.

7.6 ZPARMS affecting BACKUP and RESTORE SYSTEM

Starting in Db2 12, **`COPY_FASTREPLICATION`** is a new subsystem parameter to specify whether fast replication is required, preferred, or not needed during the creation of a FlashCopy image copy by the **`COPY`** utility. This new parameter was necessary because the creation of a FlashCopy image copy by the **`COPY`** utility used a default of `FASTREP (PREF)` (fast replication preferred) and no options to override existed.

In addition, the system-level backup supports multiple copy pools in which you can keep, extra system level backups on disk during upgrades. Also, an alternate copy pool includes the same defined set of storage groups as the standard copy pool. However, different backup storage groups are specified.

To use an alternate copy pool, specify the `ALTERNATE_CP` option and the related backup storage group options (`DBBSG` and `LGBSG`) on the **`BACKUP SYSTEM`** utility control statement:

- `DBBSG` refers to the backup storage group name for the database copy pool. It can be up to eight characters and must be defined to DFSMS with the `COPY POOL BACKUP` attribute.
- `LGBSG` refers to the backup storage group name for the log copy pool. It can be up to eight characters and must be defined to DFSMS with the `COPY POOL BACKUP` attribute

And now, also starting in Db2 12, the `FLASHCOPY_PPRCP` keyword is added to **`RESTORE SYSTEM`** and the **`RECOVER`** utilities, enabling you to control the *preserve mirror* option for the Db2 production volumes during FlashCopy operations when the recovery base is a system-level backup. `FLASHCOPY_PPRCP` also applies to the **`RECOVER`** utility that uses a FlashCopy image copy as a recovery base (Table 7-6).

Table 7-6 ZPARMS affecting BACKUP and RESTORE SYSTEM operations

Parameter	Values	What it does
SYSTEM_LEVEL_BACKUPS	YES/NO	Enables the RECOVER utility to use SLB as input for object-level recoveries.
RESTORE_RECOVER_FROMDUMP	YES/NO	Specifies if RESTORE SYSTEM and RECOVER can use SLB dump on tape as input for recovery. Yes: use tape. No: use only disk.
UTILS_DUMP_CLASS_NAME		Name of the DFSMShsm DUMP CLASS for BACKUP SYSTEM dump to tape.
RESTORE_TAPEUNITS	NOLIMIT/ 1-255	The number of tape drives RESTORE SYSTEM allocates for restore from dump.
ALTERNATE_CP		If left blank, BACKUP SYSTEM uses the standard copy pool only. If not blank, BACKUP SYSTEM uses the specified copy pool as the alternate copy pool for system-level backups. BACKUP SYSTEM alternates between using the standard copy pool and the alternate copy pool for system-level backups.
UTIL_DBBSG		The name of a backup DFSMS storage group to be used by the BACKUP SYSTEM utility for the database copy pool.
UTIL_LGBSG		The name of a backup SMS storage group to be used by the BACKUP SYSTEM utility.

UTILS_HSM_ MSGDS_HLQ		The high-level qualifier for data sets to be allocated by the Db2 BACKUP SYSTEM and RESTORE SYSTEM utilities in order to receive messages from IBM Hierarchical Storage Management (HSM) and IBM Data Facility Data Set Services (DFDSS). These messages will be included for diagnostic purposes in Db2 utility SYSPRINT DD output.

Note: For **`ALTERNATE_CP`**, **`BACKUP SYSTEM`** checks the BSDS to determine whether the standard copy pool or alternate copy pool was used for the most recent system-level backups. The **`ALTERNATE_CP`** value applies to both the database copy pool and the log copy pool, if one is specified.

For **`UTILS_HSM_MSGS_HLQ`**: This high-level qualifier must also be registered in HSM through a **`SETSYS`** command. Data sets that use this high-level qualifier are defined and populated by HSM and DFDSS during **`BACKUP SYSTEM`** and **`RECOVER SYSTEM`** processing, then allocated by Db2 and the content written to the utility's `SYSPRINT DD`. Db2 does not delete the data set.

7.7 ZPARMS affecting the RECOVER utility

ZPARMS affecting the RECOVER are shown in Table 7-7.

Table 7-7 ZPARMS affecting the RECOVER utility

Parameter	**Values**	**What it does**
REC_FASTREPLICATION	REQUIRED/ PREFERRED/ NONE	REQUIRED - The RECOVER utility forces use of FlashCopy when performing recovery from a FlashCopy image copy, to ensure that recovery occurs as quickly as possible. However, this option will cause RECOVERY to fail if FlashCopy cannot be used. PREFERRED - The RECOVER utility uses FlashCopy only if FlashCopy support is available. NONE - The RECOVER utility will use standard input/output to restore the FlashCopy image copy. This setting is not permitted when the FLASHCOPY_PPRC parameter is set to PREFERRED or REQUIRED.

7.8 ZPARMS affecting the CHECK utility

ZPARMS affecting the CHECK utility are shown in Table 7-8.

Table 7-8 ZPARMS affecting the CHECK utility

Parameter	**Values**	**What it does**
CHECK_SETCHKP	YES/NO	Specifies whether the CHECK DATA and CHECK LOB utilities are to place inconsistent objects in CHECK PENDING
CHECK_ FASTREPLICATION	PREFERRED /REQUIRED	Specifies which DSS COPY option is used to shadow the data sets during CHECK processing Preferred – If FC is available, then DSS COPY will use it Required – Forces DSS COPY to use FC – will fail if FC is not avail

Note: There is no impact if the **CHECK** utilities are running in mode SHRLEVEL CHANGE because the utilities are running on a shadow copy and a restrictive state is not set in this mode.

7.9 ZPARMS affecting the REORG utility

Starting in version 12, Db2 now avoids leaving the page set in **COPY**-pending when the **REORG** utility is run to create an inline FlashCopy with no sequential inline image copy and FlashCopy fails. If FlashCopy runs unsuccessfully, the **REORG** completes with a return code of 8 (Table 7-9).

Table 7-9 ZPARMS affecting the REORG utility

Parameter	Values	What it does
REORG_LIST_ PROCESSING	PARALLEL /SERIAL	This parameter controls the PARALLEL option for REORG. Used during LISTDEF processing PARALLEL = PARALLEL YES (when not specified on REORG TS) SERIAL = PARALLEL NO (when not specified on REORG TS)
REORG_IGNORE_ FREESPACE	YES/NO	Controls whether DB2 ignores the PCTFREE and FREEPAGE values that are defined for PBG table spaces in these situations YES = DB2 will use 0
REORG_PART_SORT _NPSI	AUTO/ YES/NO	AUTO - Specifies that if sorting all keys of the non-partitioned secondary indexes improves the elapsed time and CPU performance, all keys are sorted. YES - Specifies that if sorting all keys of the non-partitioned secondary indexes improves the elapsed time, all keys are sorted. NO - Specifies that only keys of the non-partitioned secondary indexes that are in the scope of the REORG are sorted.
REORG_MAPPING_ DATABASE		Default DB for mapping table for SHRLEVEL CHANGE REORG Blank – denotes an implicitly defined database will be used Or 8 byte DB name
REORG_DROP_PBG _PARTS	DISABLE/ ENABLE	Specifies whether the REORG utility removes trailing empty partitions when operating on an entire PBG table space- Only when running a REORG on the entire PGB table
TEMPLATE_TIME	UTC/ LOCAL	Specifies setting for the TIME option of the TEMPLATE statement
PARAMDEG_UTIL	0 to 32767	Specified the maximum number of parallel subtasks: REORG TS, REBUILD INDEX, CHECK INDEX, LOAD and UNLOAD 0 – no constraints for parallelism

7.10 ZPARM affecting object conversion

The default behavior typically applies when the utility control statement does not specify the `RBALRSN_CONVERSION` option: Db2 uses the value specified in ZPARM. `UTILITY_OBJECT_CONVERSION` and `RBALRSN_CONVERSION` options to convert existing table spaces and indexes to 6-byte page format (Table 7-10).

Table 7-10 ZPARM affecting object conversion

Parameter	Values	What it does
UTILITY_ OBJECT_ CONVERSION	BASIC/ EXTENDED/ NOBASIC/ NONE	Specifies whether DB2 utilities that accept the RBALRSN_CONVERSION option will convert existing table spaces and indexes to: ▶ A 6-byte page format ▶ A 10-byte page format ▶ Perform no conversion Utility parameter is RBALRSN_CONVERSION for LOAD REPLACE and REBUILD and REORG

Note: Table spaces and indexes that already use extended 10-byte page format cannot be returned to the 6-byte page format. When this setting is in effect, utilities that specify the `RBALRSN_CONVERSION` keyword with `BASIC` fail. In addition, utilities that specify the `RBALRSN_CONVERSION` keyword with `NONE` when the object is in 6-byte page format fail. `NOBASIC` is allowed in this field only if the `OBJECT CREATE FORMAT` field is set to `EXTENDED`. If a value is not specified for `RBALRSN_CONVERSION`, the `RBALRSN_CONVERSION` value defaults to `EXTENDED`.

REDP-5503-00

ISBN DocISBN

Printed in U.S.A.

Get connected

Redbooks ®

ibm.com/redbooks