

IBM DB2 12 for z/OS Technical Overview

Meg Bernal

Tammie Dang

Acacio Ricardo Gomes Pessoa



 Analytics

Information Management



International Technical Support Organization

IBM DB2 12 for z/OS Technical Overview

December 2016

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

First Edition (December 2016)

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

© Copyright International Business Machines Corporation 2016. 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	ix
Trademarks	x
Preface	xi
Authors	xi
Now you can become a published author, too!	xii
Comments welcome	xiii
Stay connected to IBM Redbooks	xiii
Part 1. Overview	1
Chapter 1. DB2 12 for z/OS at a glance	3
1.1 Subsystem	4
1.2 Application functions	4
1.3 Operations and performance	5
Chapter 2. Continuous delivery	7
2.1 Function level	8
2.1.1 Star function level	11
2.2 Catalog level	11
2.3 Code level	12
2.4 Activating a function level	12
2.5 DISPLAY GROUP command	14
2.6 Catalog changes	19
2.6.1 The SYSIBM.SYSLEVELUPDATES table	19
2.7 Application compatibility (APPLCOMPAT)	20
2.7.1 Managing applications through function level changes	22
2.7.2 Data Definition Language (DDL) statements sensitive to APPLCOMPAT	24
2.8 SQL processing option SQLLEVEL	25
2.9 New built-in global variables	26
2.9.1 PRODUCTID_EXT	26
2.9.2 CATALOG_LEVEL	26
2.9.3 DEFAULT_SQLLEVEL	26
2.10 DB2 supplied stored procedures	27
2.10.1 ADMIN_COMMAND_DB2	27
2.10.2 GET_CONFIG	27
2.11 Resource access control facility exit	28
2.12 Instrumentation	29
Part 2. Subsystem	31
Chapter 3. Scalability	33
3.1 Range-partitioned table spaces changes	34
3.1.1 PBR RPN table space characteristics	34
3.1.2 PBR RPN partitioned index characteristics	34
3.1.3 PBR RPN non-partitioned index characteristics	34
3.1.4 PBR RPN considerations	35
3.2 DB2 internal latch contention relief	36
3.3 Buffer pool simulation	37

3.4 Support for sizes greater than 4 GB of active log data sets	40
Chapter 4. Availability	43
4.1 Improved availability for pending definition changes	44
4.1.1 Altering index compression attribute	44
4.1.2 Altering column	44
4.2 Catalog availability improvements	45
4.2.1 Handling dynamic SQL statement	45
4.2.2 Single phase catalog migration	45
4.3 Removal of point-in-time recovery restrictions for PBG table spaces	46
4.4 PBR RPN DSSIZE increase	46
4.5 Insert partition	46
4.6 REORG enhancements for PBGs, FlashCopy and LOBs	46
4.6.1 Partition-by-growth (PBG)	47
4.6.2 FlashCopy	47
4.6.3 Large object (LOB)	47
4.7 LOAD RESUME YES BACKOUT YES option	47
4.8 Faster point-in-time recovery	48
4.8.1 Single object by defaulting to the PARALLEL(1) option	48
4.8.2 SCOPE UPDATED keyword	48
4.9 TRANSFER OWNERSHIP SQL statement	48
4.10 Auto-retry of GRECP and LPL recovery	48
Chapter 5. Data sharing	49
5.1 DISPLAY GROUP command	50
5.2 XA support for global transactions	50
5.3 Peer recovery	52
5.4 Automatic retry of GRECP and LPL recovery	53
5.5 Improved lock avoidance checking	54
5.6 Asynchronous lock duplexing	54
Part 3. Application functions	59
Chapter 6. SQL	61
6.1 Introduction	62
6.2 Additional support for triggers	64
6.2.1 Basic triggers	65
6.2.2 Advanced triggers	65
6.2.3 Differences between basic triggers and advanced triggers	66
6.2.4 Maintaining trigger activation order	70
6.3 Pagination support	70
6.3.1 Returning a subset of rows	71
6.3.2 Data-dependent pagination support	71
6.3.3 Numeric-based pagination	74
6.4 Additional support for arrays	76
6.4.1 Arrays as global variables	77
6.4.2 Associative array support on ARRAY_AGG aggregate function	78
6.4.3 Optional ORDER BY clause on ARRAY_AGG aggregate function	80
6.5 MERGE statement enhancements	81
6.5.1 Additional source value support	84
6.5.2 Additional data modification support	84
6.5.3 Additional matching condition option	84
6.5.4 Additional predicates on matching conditions support	84
6.5.5 Atomicity	84

6.5.6	Enhanced MERGE statement example	85
6.6	New built-in functions	87
6.6.1	Aggregate functions for statistics	87
6.6.2	Scalar functions for hashing	90
6.6.3	GENERATE_UNIQUE_BINARY scalar function	91
6.6.4	VARCHAR_BIT_FORMAT scalar function enhancement	93
6.7	Enhanced built-in function support	94
6.7.1	TIMESTAMP scalar function enhancement	94
6.7.2	XMLMODIFY scalar function enhancement	94
Chapter 7	Application enablement	99
7.1	Ensuring application compatibility	100
7.2	Temporal table enhancements	101
7.2.1	Enhanced application periods	101
7.2.2	Referential constraints for temporal tables	104
7.2.3	Temporal logical transactions	107
7.2.4	Auditing capabilities using temporal tables	114
Chapter 8	Connectivity and administration routines	119
8.1	Maintaining session data on the target server	120
8.2	Preserving prepared dynamic statements after a ROLLBACK	122
8.3	DRDA fast load	123
8.4	Profile monitoring for remote threads and connections	123
8.4.1	Automatic start of profiles during subsystem start	123
8.4.2	Support for global variables	124
8.4.3	Support for wildcarding	124
8.4.4	Idle thread enhancement	125
8.5	Stored procedures supplied by DB2	126
Part 4	Operations and performance	129
Chapter 9	Administrator function	131
9.1	Dynamic plan stability	132
9.1.1	Stabilization into and loading from catalog tables	132
9.1.2	Stabilization method	132
9.1.3	Catalog tables	135
9.1.4	Calculating the EDM statement cache hit ratio	137
9.1.5	Invalidation of stabilized dynamic statements	138
9.1.6	EXPLAIN changes	139
9.1.7	The FREE STABILIZED DYNAMIC QUERY subcommand	139
9.1.8	Monitor for stabilization	141
9.1.9	DSNZPARM and installation panel	144
9.2	Resource limit facility for static SQL	146
9.2.1	Reactive governing static SQL	146
9.2.2	Use cases	148
9.2.3	RLF DSNZPARMs and installation panels	151
9.3	Column level deferred alter (pending alter column)	154
9.3.1	Utility	158
9.3.2	ALTER INDEX	159
9.4	Insert partition	159
9.4.1	ALTER ADD PARTITION	160
9.4.2	Utilities affected	162
9.4.3	Catalog changes	163

Chapter 10. Security	167
10.1 Installation or migration without requiring SYSADM	168
10.2 UNLOAD privilege	169
10.2.1 Enforcing new privilege	170
10.2.2 Using DB2 security facility	170
10.2.3 Using Resource Access Control Facility (RACF)	171
10.3 Object ownership transfer	172
10.3.1 Supported objects	174
10.3.2 New owner	175
10.3.3 Revoking privileges of current owner	175
Chapter 11. Utilities	177
11.1 Backup and recovery enhancements	178
11.1.1 Sequential image copy enhancements	178
11.1.2 Copy support for FASTREPLICATION	180
11.1.3 Alternate copy pools for system-level backups	181
11.1.4 FLASHCOPY_PPRCP keyword option	182
11.1.5 Point-in-time recovery enhancements	183
11.1.6 MODIFY RECOVERY enhancements	185
11.2 RUNSTATS enhancements	186
11.2.1 Specifying FREQVAL without the COUNT n keywords	186
11.2.2 USE PROFILE support for inline statistics	187
11.2.3 INVALIDATECACHE option	187
11.2.4 RUNSTATS TABLESPACE LIST INDEX improvements	189
11.2.5 New keyword REGISTER for RUNSTATS utility	190
11.3 REORG enhancements	191
11.3.1 Improved FlashCopy management	191
11.3.2 Preventing COPY-pending on a LOB table space during REORG of PBG ...	192
11.3.3 Improved partition-level PBG REORGs	193
11.3.4 REORG option for empty PBG partitions deletion	194
11.3.5 Support for the COMPRESSRATIO catalog column	194
11.3.6 Display claimers information on each REORG drain failure	195
11.3.7 Additional REORG enhancements	196
11.4 LOAD and UNLOAD enhancements	196
11.4.1 LOAD enhancements	196
11.4.2 UNLOAD enhancements	197
Chapter 12. Installation and migration	199
12.1 Prerequisites for DB2 12	200
12.1.1 Data sharing	200
12.1.2 Processor requirements	200
12.1.3 Software requirements	200
12.1.4 DB2 Connect prerequisites	200
12.1.5 Programming language requirements, minimum levels	201
12.1.6 Minimum configuration (IEASYSxx)	201
12.2 Single-phase migration and function level	201
12.2.1 Fallback SPE	202
12.2.2 EARLY code	203
12.2.3 Pre-migration checkout	204
12.2.4 Creating DSNZPARM and DECP modules	205
12.2.5 Creating and verifying routines supplied by DB2	205
12.2.6 REBIND at each new release	206
12.2.7 Activating new function level	206
12.2.8 Deprecated in earlier releases and removed in DB2 12	206

12.3	Installing a new DB2 12 system	208
12.3.1	Defines DB2 to z/OS	208
12.4	Subsystem parameters	209
12.4.1	New subsystem parameters	209
12.4.2	Removed subsystem parameters	213
12.4.3	Install Parameters Default Changes	214
12.4.4	Deprecated system parameters	214
12.5	Installation or migration without requiring SYSADM	214
12.6	Installation with z/OS Management Facility (z/OSMF)	214
12.6.1	How to use DB2 installation CLIST and panels to generate z/OSMF artifacts	216
12.6.2	Feeding the generated artifacts to z/OSMF	222
12.7	Temporal catalog	227
12.7.1	System-period data versioning for two RTS catalog tables	228
12.7.2	Real-time statistics externalization during migration	229
Chapter 13.	Performance	231
13.1	Performance expectations	232
13.2	In-memory buffer pool	232
13.3	In-memory index optimization	232
13.4	Improved insert performance for non-clustered data	234
13.4.1	DDL clause on CREATE TABLESPACE and ALTER TABLESPACE	235
13.4.2	SYSIBM.SYSTABLESPACE new column: INSERTALG	235
13.4.3	ZPARM: DEFAULT_INSERT_ALGORITHM	235
13.5	Query performance enhancements	236
13.5.1	UNION ALL and Outer Join enhancements	236
13.5.2	Sort improvements	243
13.5.3	Predicate optimization	246
13.5.4	Execution time adaptive index	248
Part 5.	Appendixes	251
Appendix A.	Information about IFCID changes	253
	IFCID header changes	254
	New IFCIDs	254
	IFCID 382: Begin parallel task synchronization suspend	255
	IFCID 383: End parallel task synchronization suspend	255
	IFCID 389: Fast index traversal	255
	IFCID 404: Serviceability trace record for new subsystem parameter AUTH_COMPATIBILITY	256
	IFCID 413: Begin of pipe wait for a fast insert	257
	IFCID 414: End of pipe wait for a fast insert	257
	IFCID 477: Fast index traversal	257
	Application compatibility IFCID changes	258
	IFCID 366: Deprecated application compatibility trace	258
	IFCID 376: Application compatibility trace	258
	Dynamic SQL plan stability IFCID changes	259
	IFCID 002: RDS statistics block	259
	IFCID 002: EDM pool statistics block	260
	IFCID 021: Lock types	260
	IFCID 029: EDM request begin identifier and new block	260
	IFCID 030: EDM request end identifier and new block	261
	IFCID 106: New subsystem parameter	261
	IFCID 316: Stabilization and hash ID information	262

Fast INSERT IFCID changes	262
IFCID 002: Package level pipe wait information	262
IFCID 002: Data Manager Statistics block	263
IFCID 003: Accounting control block	263
IFCID 018: End of inserts and scans	263
IFCID 058: Accumulated pipe wait time	263
IFCID 106: New subsystem parameter	264
IFCID 316: Statement level pipe wait information	264
IFCID 401: Accumulated pipe wait time	264
IFCID 413: Begin pipe wait	264
IFCID 414: End pipe wait	265
Lift partition limits IFCID changes limits	265
IFCID 006: Pre-read page number flag and partition number	266
IFCID 007: Post-read page number flag and partition number	266
IFCID 021: Resource name	266
IFCID 106: New subsystem parameter	267
IFCID 124: Page number within pageset	267
IFCID 127: Agent suspend	267
IFCID 128: Agent resume	267
IFCID 150: Resource name	268
IFCID 172: Resource name	268
IFCID 196: Resource name	269
IFCID 198: Page numbering flag	269
IFCID 223: Identifier, new constant, and partition number	269
IFCID 226: Page numbering flag	270
IFCID 227: Page numbering flag	270
IFCID 255: Partition number and relative page number	270
IFCID 259: Partition number and relative page number	271
IFCID 305: Table space partition number and type	271
Large object (LOB) compression IFCID changes	271
IFCID 003: Accounting control block	271
IFCID 106: New subsystem parameter	272
Transfer ownership IFCID changes	272
IFCID 002: RDS statistics block	272
IFCID 062: Statement type	272
IFCID 140: Source object owner and name	273
IFCID 361: Source object owner and name	273
UNLOAD privilege for UNLOAD utility IFCID changes	274
IFCID 106: New subsystem parameter	274
Additional changed IFCIDs	274
IFCID 106: Modifications and enhancements	274
IFCID 125: Adaptive index processing	276
New QWAC_WORKFILE_MAX and QWAC_WORKFILE_CURR fields	277
Appendix B. Additional material	279
Locating the web material	279
Downloading and extracting the web material	279

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](http://www.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.

CICS®	IBM z Systems®	Redbooks®
DB2®	IMS™	Redbooks (logo)  ®
DB2 Connect™	Language Environment®	z Systems®
FlashCopy®	Parallel Sysplex®	z/OS®
IBM®	pureXML®	zEnterprise®
IBM z™	RACF®	

The following terms are trademarks of other companies:

Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

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

Preface

IBM® DB2® 12 for z/OS® delivers key innovations that increase availability, reliability, scalability, and security for your business-critical information. In addition, DB2 12 for z/OS offers performance and functional improvements for both transactional and analytical workloads and makes installation and migration simpler and faster. DB2 12 for z/OS also allows you to develop applications for the cloud and mobile devices by providing self-provisioning, multitenancy, and self-managing capabilities in an agile development environment. DB2 12 for z/OS is also the first version of DB2 built for continuous delivery.

This IBM Redbooks® publication introduces the enhancements made available with DB2 12 for z/OS. The contents help database administrators to understand the new functions and performance enhancements, to plan for ways to use the key new capabilities, and to justify the investment in installing or migrating to DB2 12.

Authors

This book was produced by a team of specialists from around the world working at the International Technical Support Organization, Raleigh Center.



Meg Bernal is a Senior Software Engineer with the DB2 for z/OS development team based in the IBM Silicon Valley Laboratory. Meg is celebrating her twentieth year as a Developer in DB2 and was involved in several SQL and SQL PL enhancements in DB2 12. Meg often presents at conferences and DB2 user groups, nationally and internationally.



Tammie Dang is a Senior Software Engineer in the US. She has 27 years of experience in the Relational Database field. She holds a degree in Computer Science from the University of Houston. Some areas of her expertise include intersystem locking, data sharing on the IBM z/OS Parallel Sysplex® platform, query parallelism, application development using SQL, ODBC, JCC API, and stabilized dynamic SQL. She has written extensively about DB2 for z/OS enhancements and features.



Acacio Ricardo Gomes Pessoa is an IBM z/OS IMS™ and DB2 Database Administrator in Brazil for IBM Global Services. He started working at IBM in 2008 as a Mainframe Operator and in Production Support. In 2010 he moved to the DBA Team. He supports telecommunication applications using IMS Full Function Databases and DB2 tables. He holds a degree in Information Systems and his areas of expertise include IMS and DB2 databases maintenance.

This project was led by:

- ▶ Martin Keen

Thanks to the following people for their contributions to this project:

- ▶ Maria Sueli Almeida
- ▶ Tom Beavin
- ▶ Gayathiri Chandran
- ▶ Steve Chen
- ▶ Chris Crone
- ▶ Jason Cu
- ▶ Xiaohong Fu
- ▶ Akiko Hoshikawa
- ▶ Jeff Josten
- ▶ Ravi Kumar
- ▶ Jae Lee
- ▶ Dorothy Lin
- ▶ Fen-Ling Lin
- ▶ Irene Liu
- ▶ Regina Liu
- ▶ Jane Man
- ▶ Claire McFeely
- ▶ Manvendra Mishra
- ▶ Emily Prakash
- ▶ Terence Purcell
- ▶ Haakon Roberts
- ▶ Hugh Smith
- ▶ Xiao Bo Wang
- ▶ Maryela Weihrauch

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 from two to six 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 at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

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

- ▶ Use the online **Contact us** review Redbooks form found at:

ibm.com/redbooks

- ▶ Send your comments in an email to:

redbooks@us.ibm.com

- ▶ Mail your comments to:

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>



Part 1

Overview

This part contains the following chapters:

- ▶ Chapter 1, “DB2 12 for z/OS at a glance” on page 3
- ▶ Chapter 2, “Continuous delivery” on page 7



DB2 12 for z/OS at a glance

IBM DB2 12 for z/OS delivers key innovations that increase availability, reliability, scalability, and security for your business-critical information. In addition, DB2 12 for z/OS offers performance and functional improvements for both transactional and analytical workloads and makes installation and migration simpler and faster. DB2 12 for z/OS also allows you to develop applications for the cloud and mobile devices by providing self-provisioning, multitenancy, and self-managing capabilities in an agile development environment.

DB2 12 for z/OS is the first version of DB2 built for continuous delivery. As a growing number of applications need to be created or modified in a matter of days, not months or years, access to the latest functions and features in DB2 becomes more vital. With the introduction of continuous delivery, DB2 will ship new functions and features when those items are ready. Whatever your role is, you will now be able to select when you want a new item available in your databases and applications. A full chapter is dedicated to continuous delivery to help you be successful in leveraging this new capability.

The following sections have a brief overview of some of the important functions that are provided by IBM DB2 Version 12.1 for z/OS (which throughout this book is also referred to as DB2 12 for z/OS or just DB2 12). For the purposes of this discussion, these functions are divided into the following categories, which correspond to several parts of this book:

- ▶ Subsystem
- ▶ Application functions
- ▶ Operations and performance

1.1 Subsystem

The chapters in the Subsystem part focus on the scalability, availability, and data sharing enhancements.

One of the biggest enhancements on the scalability front in DB2 12 is the ability to store 256 trillion rows in a single table. DB2 12 is able to store all of those rows in a single table through a new partition-by-range (PBR) table space structure. The new structure will use relative page numbers (RPN) and remove the dependency between the number of partitions and the partition size. By eliminating this dependency, you no longer need to research the rules regarding their relationship or concern yourself with running out of space in a partition that would cause you to change the partition limit keys and possibly result in an application outage. As part of the new RPN structure, DB2 12 moves to seven-byte record identifiers (RIDs), which consist of a two-byte part number and five-byte page number that allows the table to increase its size.

DB2 12 is improving the availability of universal table space indexes by allowing alteration of an index compression to be a pending change that places the index in advisory-REORG pending (AREOR) status. By placing the index in AREOR status, applications may continue to access those indexes during alteration. Prior to DB2 12, those indexes would have been placed in a restrictive REBUILD-pending (RBDP) status. Several new features in DB2 12 also prevent the database objects from being dropped and re-created, which affects critical data availability.

One data sharing feature you might consider using in DB2 12 is the ability for one member of a DB2 data sharing group to automatically initiate the recovery process of another failed member in that same data sharing group. This type of processing is called *peer recovery* whereby the assisting member initiates a restart in light mode operation on the failed member. You may also see the simplicity in enabling asynchronous duplexing for the IRLM lock structure, with performance similar to the simplex mode, while still taking advantage of dual structures for availability.

1.2 Application functions

The application functions chapters focus on SQL, application enablement, and also connectivity and administration routines.

In DB2 12, you can write triggers by using SQL Programming Language (SQL PL). SQL PL allows developers to code logic in the form of an IF statement, WHILE statement, GOTO statement, and so on, around their traditional SQL. SQL PL allows handlers to be coded so that error conditions and warning conditions can be handled gracefully. With SQL PL support for triggers, you can also debug, deploy, and maintain multiple versions of triggers. In fact, just about everything you can do today with SQL PL routines, you can do with enhanced trigger support in DB2 12.

DB2 12 has extended its support for the array data type. The array data type can be specified for a global variable. With arrays as global variables, you can now reference arrays outside of SQL PL routines because the array data type was an exclusive data type for SQL PL routines prior to DB2 12. The array data type support provides the capability to treat data in your tables as arrays and to treat your arrays as tables. With the enhancements to the array data type, the restriction of referencing arrays across different types of objects and applications has been lifted.

DB2 12 continues to deliver enhancements to temporal tables as more customers identify additional use cases for applications based on time. Starting in DB2 12, referential constraints can be enforced on application-period temporal tables that contain a BUSINESS_TIME period. Another enhancement to application-period temporal tables is the ability to use the inclusive/inclusive model such that both the start time and the end time are included in the time interval. With support for both the inclusive/exclusive model and now the inclusive/inclusive model, applications developers have more flexibility in coding applications to satisfy their business needs.

DB2 12 also provides additional support for system-period temporal tables with logical temporal transactions that provide greater flexibility in data modifications and auditing capabilities that answer the classic questions of who made changes to the database, when did the changes occur, and what changes were made. The auditing capabilities became so popular that the feature was retrofitted back to DB2 11 for z/OS.

1.3 Operations and performance

The operations and performance chapters focus on administrator functions, security, utilities, installation and migration, and performance.

One of the biggest achievements with DB2 12 for z/OS in the performance area is to potentially observe over 11 million inserts per second for unclustered data. DB2 12 introduces a fast insert algorithm that not only can increase throughput but also has the potential to reduce logging and reduce certain elapsed and CPU times. The fast insert algorithm is enabled for only universal table spaces using MEMBER CLUSTER. The algorithm can be enabled system-wide or for individual table spaces.

DB2 12 also introduces a number of advanced in-memory database techniques that have the potential of providing CPU reductions. One in-memory technique is the introduction of a fast traversal block (FTB) for indexes. FTBs are in-memory optimized index structures that allow DB2 to traverse the FTB much faster than doing traditional page-oriented page traversal for indexes that are cached in buffer pools.

Continuing the commitment in providing analytical insights, DB2 12 has the potential of providing faster execution times for query workloads and even faster times for targeted queries. One performance area of focus in DB2 12 is for the SQL UNION ALL operator. DB2 uses the UNION ALL operator for bi-temporal tables and transparent archiving, while application developers often use UNION ALL as a good alternative to very large tables where several smaller tables are joined by the UNION ALL to simulate the larger table. Outer joins share similar performance challenges to UNION ALL in that both can result in materializing some portion of the data into a workfile; as such, the materialization into a workfile can result in significant performance degradation, consume workfile resources, or both. In DB2 12, materialization to a workfile is minimized, trimming unnecessary columns from the materialization, pushing predicates down to lower query blocks, pushing down the ordering of data and fetching first counters into lower query blocks, and reordering outer join tables to avoid materialization. All contribute to the enhanced performance for the UNION ALL operator and outer joins.

DB2 12 introduces an extensive amount of improvements to its utilities such as REORG, LOAD, and RUNSTATS, and has made several enhancements to assist you with backup and recovery. One attractive change to the REORG TABLESPACE utility is that the RELOAD phase can now be offloaded to zIIP¹. Prior to DB2 12, a statistics profile was allowed only on the RUNSTATS utility; however, starting in DB2 12, both the LOAD and REORG TABLESPACE online utilities allow you to use a statistics profile to collect inline statistics.

As for the RUNSTATS utility itself and other utilities that collect inline statistics, an option exists to *not* invalidate dynamic statements in the dynamic statement cache. Prior to DB2 12, the collection of statistics for an object always invalidated cached dynamic statements that referred to the objects. The advantage of not invalidating the statement in the cache is that DB2 can reuse the previously determined access path for the statement. Remember, the default option of whether or not to invalidate the statements in the cache for RUNSTATS is to not invalidate them, which differs from prior releases of DB2.

DB2 12 also provides new functions that are useful for the database administrators such as administering the complete life cycle of stabilized dynamic SQL. The administration capabilities allow you to capture the runtime structures in the persistent DB2 catalog and reload them on subsequent executions, much like static SQL.

With DB2 12, you can migrate or install DB2 without requiring SYSADM authority and ease your migration process from anywhere and from any computer using a web-based application, z/OS Management Facility.

¹ IBM z™ Systems Integrated Information Processor (zIIP)



Continuous delivery

Each DB2 version delivers many enhancements in different areas that are welcomed by customers worldwide. However, sometimes certain functionalities are required sooner than the scheduled date of the next version. New release migration is also costly and time-consuming. Waiting for years to receive the much needed features in the next version might be impossible for certain business needs, thus the enhanced features must be retrofitted to the version in the current service stream. Continuous delivery is a solution to make enhancements available in the service stream sooner than the next version.

Continuous delivery is being approached in various ways in the industry, and also within IBM. The various approaches run from changes that occur constantly (literally hundreds of times a day,) to fix packs that contain enhancements, to point releases and modification levels. DB2 for z/OS customers want a solution that provides value, but does not create a large legacy of point releases needing maintenance. DB2 12 is adopting a step approach to new functions that involves shipping changes and service (both preventative service and defect fixes) in the same service stream. By enabling customers to adopt new function and service in a controlled manner, DB2 is giving customers the confidence to continue to move forward with maintenance in their normal scheduling scheme, giving them the flexibility they need to run their businesses that require stable systems, and enabling them to activate new functions when they want.

This chapter covers the following topics:

- ▶ Function level
- ▶ Catalog level
- ▶ Code level
- ▶ Activating a function level
- ▶ DISPLAY GROUP command
- ▶ Catalog changes
- ▶ Application compatibility (APPLCOMPAT)
- ▶ SQL processing option SQLLEVEL
- ▶ New built-in global variables
- ▶ DB2 supplied stored procedures
- ▶ Resource access control facility exit
- ▶ Instrumentation

2.1 Function level

DB2 11 has various migration modes such as conversion mode (CM), star mode (CM*), enabled new function mode (ENFM), and new function mode (NFM). With DB2 12, DB2 changes to a new concept of function level that helps to introduce the continuous delivery model and to manage groups of new enhancements. This concept of function level starts when you begin your migrations to DB2 12.

A *function level* enables a particular set of new DB2 capabilities and enhancements that were previously delivered and applied in the single continuous stream of DB2 code. The single continuous stream of DB2 code will include support for new capabilities, defect fixes, and preventative service items. The new capabilities and enhancements are dormant when first applied. Before the new capabilities of a function level can be used, that function level must be activated using the **ACTIVATE** command. In addition, before an application can take advantage of new capabilities, the application must also specify the corresponding application compatibility value.

Function levels are specified by the 9-byte strings that correspond to the DB2 version, release, and modification value, in the following format:

VvvRrrMmmm

In this format, *vv* is the version, *rr* is the release and *mmm* is the modification indicator.

As soon as DB2 12 general availability code is applied and DB2 is restarted, that code level is capable of supporting functions compatible to DB2 11, and the new functions of DB2 12. Therefore, the DB2 12 general availability code supports the following function levels:

► V12R1M100

Identifies the function level before activating a new function, after migration to DB2 12 (function level 100). You can think of function level 100 as being similar to conversion mode (CM) in prior releases of DB2. DB2 11 members of the data sharing group may be started. DB2 subsystems and data sharing members can fall back to DB2 11.

► V12R1M500

Identifies the lowest function level that enables a new function in DB2 12 (function level 500). This function level can be thought of as the new function mode (NFM) in DB2 11. DB2 11 members of the data sharing group may not be started. DB2 subsystems and data sharing members cannot fall back to DB2 11. The DB2 subsystem or all active members of a data sharing group must have the code level of V12R1M500 (by migrating to DB2 12) and the catalog has been converted to the V12R1M500 level using the CATMAINT utility.

► V12R1M100*

Identifies the function level where new functions are no longer available. DB2 11 members of the data sharing group may not be started. DB2 subsystems and data sharing members cannot fall back to DB2 11.

Figure 2-1 shows a time line (from left to right) for how DB2 11 data sharing group can be migrated to function level V12R1M100, in coexistence mode, the CATMAINT utility is used to convert to DB2 12 catalog, and a few months later function level V12R1M500 is activated for new feature exploitation. Note that the application packages are bound with the appropriate APPLCOMPAT options before and after a new function is available.

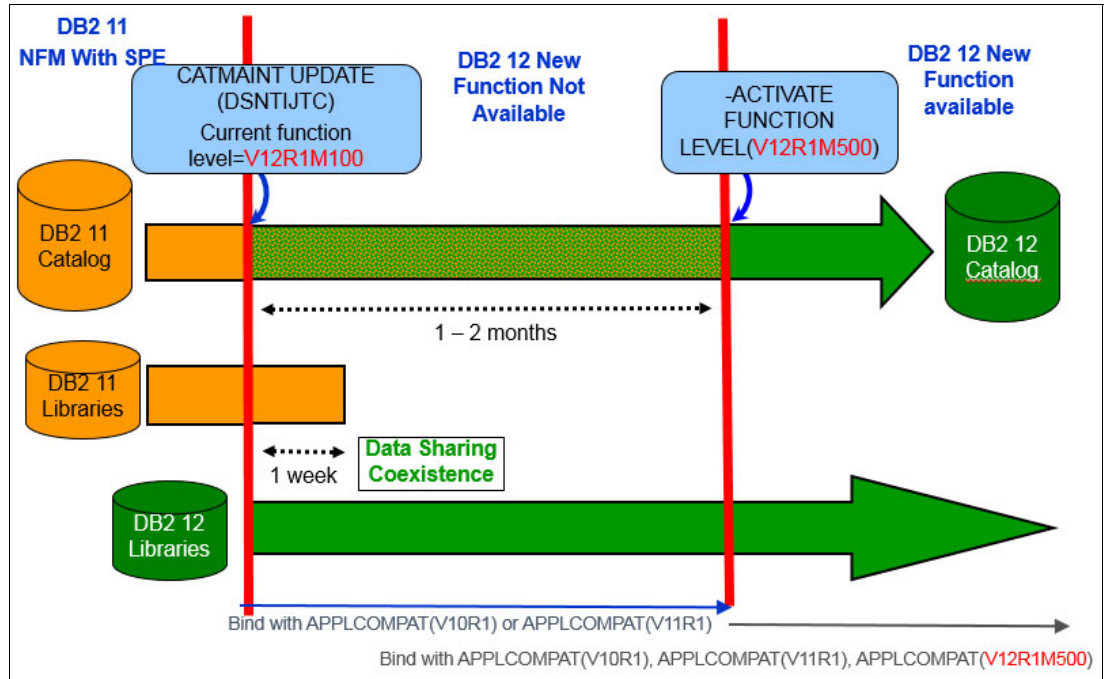


Figure 2-1 Migration from DB2 11 to DB2 12 then activating the first new function level 500

As DB2 delivers more function levels in the continuous stream of DB2 code, the modification level, mmm, in the function level string with value of 500 or greater is reserved for the various function levels where new enhancements are supported.

Assume that function level V12R1M5xx (without a catalog change) and V12R1M5yy (with a required catalog change) are delivered after V12R1M500 to support new enhancements. The following figures show the activities moving to these function levels.

In Figure 2-2, after the maintenance application of the V12R1M5xx code level, an **ACTIVATE** command can be used to enable that function level.

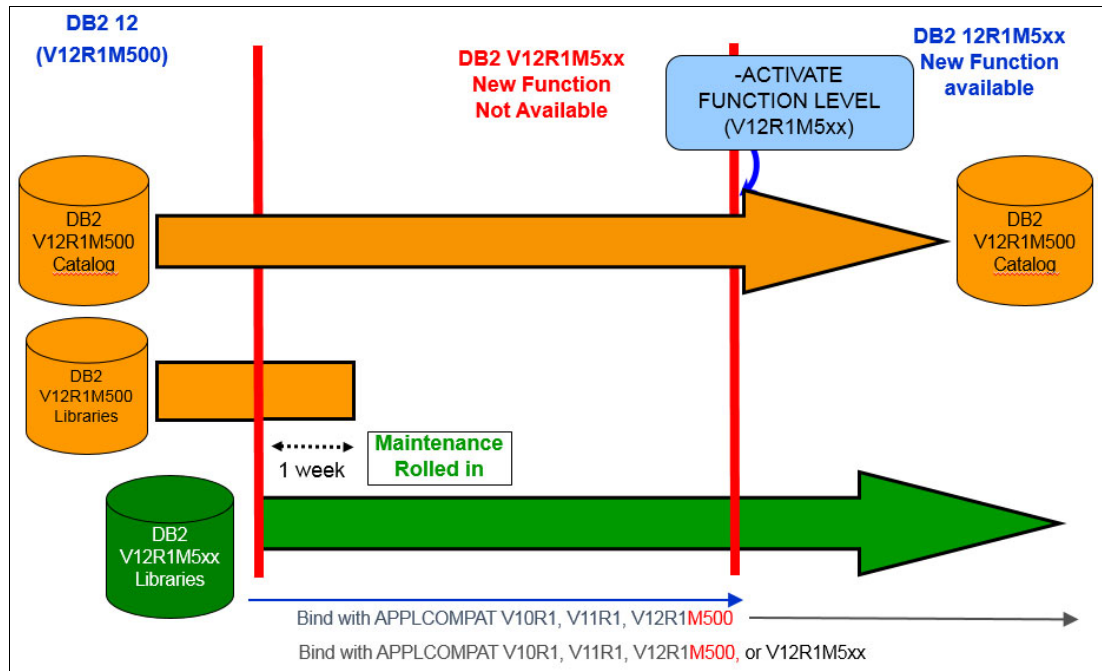


Figure 2-2 Function level V12R1M5xx (without a catalog change)

In Figure 2-3, after the maintenance application of the V12R1M5yy code level and the catalog level upgrade through the CATMAINT utility, an **ACTIVATE** command can be used to enable that function level.

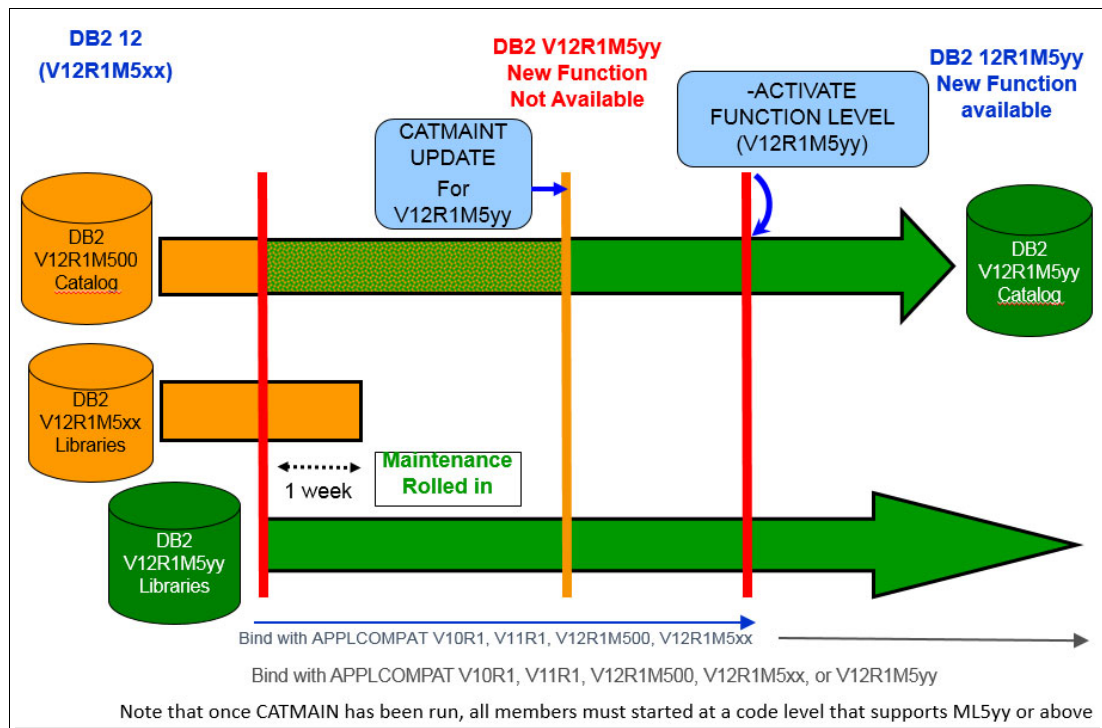


Figure 2-3 Function level V12R1M5yy (with a catalog change)

2.1.1 Star function level

Without the code levels changing, a lower function level can be activated. If a user activates a function level lower than the current function level, then the user activates the corresponding star (*) function level. When the current function level is a star function level, DB2 continues to tolerate objects, packages, and structures that were created or bound at the higher function level that use capabilities supported in the higher level. Such objects can still be accessed in the lower star function level, but they cannot be dropped and re-created while the star function level remains activated. Such packages can still be executed, rebound, and automatically bound. Note that rebinding of such packages work when the APPLCOMPAT bind option value is the same or lower than the current star function level. Newly bound packages can be bound at only the APPLCOMPAT levels up to the current function level.

If an application exploits new SQL features provided by a function level, binding the application package with the corresponding APPLCOMPAT option is recommended even for dynamic SQL statements. In this way, those existing packages behave the same after the * function level is activated. For more details and examples, see 2.7, “Application compatibility (APPLCOMPAT)” on page 20.

After activating function level V12R1M500 with new DB2 12 functionality, the function level can be changed back to function level V12R1M100, which means the V12R1M100* function level (that is, V12R1M100 star function level).

2.2 Catalog level

A *catalog level* indicates that a particular CATMAINT job has been run on the DB2 catalog and converted it to the appropriate level. Each function level requires a specific catalog level. However, not every function level introduces a new catalog level. If the catalog is not at the minimum required catalog level when you attempt to activate a function level, the **ACTIVATE** command fails, and the message output indicates the current and required catalog levels.

The CATMAINT utility is enhanced with the LEVEL keyword so the user can migrate the current catalog to the specified catalog level. The syntax diagram (Figure 2-4) shows the CATMAINT control statement.

```
>>-CATMAINT--UPDATE--+-----+----->
                               '-LEVEL-- (--catalog-level--)-'
```

Figure 2-4 CATMAIN control statement

The format for the catalog level is changed from DB2 11 to align with function level format. The new format is a 9-byte string:

VvVrRrMmmm

In this format, vv is the version, rr is the release, and mmm is the modification level. If the catalog level is omitted, V12R1M500 is used.

For the initial migration to DB2 12 general availability code, the CATMAINT UPDATE LEVEL V12R1M500 control statement must be specified. This is because the DB2 12 migration process converts the catalog to be ready for new functions with function level V12R1M500. There is no other separate step to convert the catalog for new functions. This single-phase catalog migration reduces the change windows involved in migrating completely to the DB2 12 release and improve availability of the system for application workload. Additionally,

the V12R1M500 catalog level is compatible to DB2 11 to support both coexistence and fallback cases.

Each catalog level is identified by the same identifier as the lowest function level that requires it. For example, the function level immediately after V12R1M500 would require that the DB2 data sharing group or subsystem is at catalog level V12R1M500. The DB2 catalog level is shown on the DISPLAY GROUP command output.

2.3 Code level

Each function level requires the DB2 data sharing group or subsystem to be at a specific code level. That means the required code shipments that support new functions must be applied on the DB2 data sharing group or subsystem before the corresponding function level can be activated. Unlike function level, which is a data sharing group's concept, *code level* is pertinent to each member because maintenance can be applied differently on each member. Each DB2 member in the data sharing group may have its own code level that differs from another member's code level.

Each code level is identified by the same identifier as the function level that requires it. For example, function level V12R1M501 requires that each member of the data sharing group, or a single DB2 subsystem is at code level V12R1M501. Code level is shown on the DISPLAY GROUP command output abbreviated by a six-character string such as '121500' (for V12R1M500), '121501' (for V12R1M501), '111500' (for DB2 11 new function mode).

Notes:

- ▶ The DB2 12 general availability code is at code level '121500' (or V12R1M500). This is because this code level is capable of new functions delivered at the DB2 12 general availability announcement. Enhancements may be applied at various times followed by a program temporary fix (PTF), which sets the code level for the appropriate enhancements. When that PTF is applied, the DB's code level is changed.
- ▶ Changing the code level does not automatically change the function level.

2.4 Activating a function level

When first starting DB2 12, the current active function level is V12R1M100. This is also true for the data sharing group even if there is another DB2 11 coexist member. The **ACTIVATE** command is used to activate a function level which could be either higher or lower than the current function level. The **ACTIVATE** command checks for all required actions before the command is successful.

Currently, to activate function level V12R1M500 for the new functions shipped at DB2 12 general availability, all members of the data sharing group must be at a DB2 12 code level and the catalog must already be converted to V12R1M500 level. After the function level V12R1M500 is activated, another DB2 11 member cannot be started. Future function levels might have these requirements:

- ▶ A DB2 subsystem or all members of a data sharing group must be at a certain code level.
- ▶ The catalog must be at a certain level.
- ▶ Any other documented requirement.

Figure 2-5 shows the **ACTIVATE** command syntax:

```
>>--ACTIVATE FUNCTION LEVEL--(--function-level--)---+-----+-----><
                                     '-TEST-'
```

Figure 2-5 The **ACTIVATE** command

The scope of the **ACTIVATE** command is **GROUP**. If the specified function level is successfully activated, a successful message **DSNU757I** is displayed. A row will also be inserted into the **SYSIBM.SYSLEVELUPDATES** catalog table to indicate that the activation completed successfully.

If unsuccessful, then the same **DSNU757I** message is issued, indicating the reason the group or **DB2** subsystem could not be taken to the specified function level.

The **TEST** option is useful to determine whether the **DB2** subsystem or data sharing group meets all requirements for the specified function level to be activate. The function level is not changed with the **TEST** option.

The next three figures show examples of various **ACTIVATE** command outputs.

Figure 2-6 shows the user-specified **V12R1M500** function level along with **TEST**. The user wants to know if the subsystem is ready for this function level. The **DSNU757I** message indicates that the subsystem is eligible for the specified level. In addition, because **TEST** was not specified, the user received detailed information about each active group member. All of the members have a current code level of **V12R1M500** and the catalog has been migrated to the **V12R1M500** level. This is why the subsystem can be taken to the specified **V12R1M500** level.

```
-ACTIVATE FUNCTION LEVEL (V12R1M500) TEST
DSNU757I  -DB2A DSNUGCCA
*** BEGIN ACTIVATE FUNCTION LEVEL (V12R1M500)
          GROUP ELIGIBLE FOR FUNCTION LEVEL (V12R1M500)
          CATALOG LEVEL (V12R1M500)
          CURRENT FUNCTION LEVEL (V12R1M500)
          HIGHEST ACTIVATED FUNCTION LEVEL (V12R1M500)
          HIGHEST POSSIBLE FUNCTION LEVEL (V12R1M501)
-----
DB2      CURRENT      CAPABLE FUNCTION LEVELS
MEMBER  ID  CODE-LEVEL  LOWEST      HIGHEST      STATUS
-----
DB2A    1  V12R1M500  V12R1M100  V12R1M500  ELIGIBLE
DB2B    2  V12R1M500  V12R1M100  V12R1M500  ELIGIBLE
-----
DSN9022I  -DB2A DSNZACMD '-ACTIVATE FUNC' NORMAL COMPLETION
```

Figure 2-6 The **ACTIVATE** command with the **TEST** option

Figure 2-7 on page 14 shows the user-specified **V12R1M500** function level with no **TEST** option specified. This means the user wants to activate the **V12R1M500** function level. The **DSNU757I** message is displayed, indicating that the group successfully activated the specified function level **V12R1M500**.

```

-ACTIVATE FUNCTION LEVEL (V12R1M500)
DSNU757I  -DB2A DSNUGCCA
*** BEGIN ACTIVATE FUNCTION LEVEL (V12R1M500)
          FUNCTION LEVEL (V12R1M500) SUCCESSFULLY ACTIVATED
          CATALOG LEVEL (V12R1M500)
          CURRENT FUNCTION LEVEL (V12R1M500)
          HIGHEST ACTIVATED FUNCTION LEVEL (V12R1M500)
          HIGHEST POSSIBLE FUNCTION LEVEL (V12R1M500)
DSN9022I  -DB2A DSNZACMD '-ACTIVATE FUNC' NORMAL COMPLETION

```

Figure 2-7 The successful ACTIVATE command

Figure 2-8 shows the user-specified V12R1M500 function level for activation. The DB2 data sharing group is in the migration process from DB2 11. The DSNU757I message is displayed indicating that the data sharing group is not yet ready for the specified level because of missing catalog update to the V12R1M500 level. In other words, the mandatory CATMAINT update job for DB2 12 has not yet been run. Because the subsystem or group has not yet migrated to DB2 12, the catalog level reflects the V11R1M500 level which is DB2 11 NFM level.

```

-ACTIVATE FUNCTION LEVEL (V12R1M500)
DSNU757I  -DB2A DSNUGCCA
*** BEGIN ACTIVATE FUNCTION LEVEL (V12R1M500)
          SUBSYSTEM NOT ELIGIBLE FOR FUNCTION LEVEL (V12R1M500)
          CATALOG NOT AT REQUIRED LEVEL (V12R1M500)
          CATALOG LEVEL (V11R1M500)
          CURRENT FUNCTION LEVEL (V12R1M100)
          HIGHEST ACTIVATED FUNCTION LEVEL (V12R1M100)
          HIGHEST POSSIBLE FUNCTION LEVEL (V12R1M100)
DSN9022I  -DB2A DSNZACMD '-ACTIVATE FUNC' NORMAL COMPLETION

```

Figure 2-8 The failed ACTIVATE command

2.5 DISPLAY GROUP command

There are a few changes in the output of the **DISPLAY GROUP** command on DB2 12. DB2 11 shows the catalog level with 3-byte string. In DB2 12, the catalog level is shown in the new format of 9-byte string, VvvRrrMmmm. The migration mode is no longer displayed. The output now always shows the following three function levels:

- ▶ The current function level is a 9-byte string that indicates the *currently* activated function level.
- ▶ The highest activated function level is a 9-byte string that indicates *highest* function level that was *ever activated*.
- ▶ The highest possible function level is a 9-byte string that indicates the *highest* function level that *can be activated*.

That information can be used to determine the status of the subsystem or data sharing group prior to activating the next function level.

Another change on the **DISPLAY GROUP** command is the code level for each DB2 member. DB2 11 showed the individual DB2 code level as a 3-byte string. In DB2 12, the **DISPLAY**

GROUP command shows each DB2's code level in the format of 6-byte string, which indicates version, release, and modification level (vvrrmmm).

The examples in the following figures show various **DISPLAY GROUP** commands used throughout the migration process, starting from migrating members in the data sharing to DB2 12 code one at a time, to activating a new function.

Figure 2-9 shows a data sharing group with active, coexist members in DB2 12 and DB2 11 code levels. Some members have migrated to DB2 12 thus the catalog level is V12R1M500. The DB2 LVL column shows members at code levels of version 12, release 1, modification 500 and version 11, release 1, modification 500.

```

-DIS GROUP
DSN7100I -DB1A DSN7GCMD
*** BEGIN DISPLAY OF GROUP(DSNCAT) CATALOG LEVEL(V12R1M500)
      CURRENT FUNCTION LEVEL(V12R1M100)
      HIGHEST ACTIVATED FUNCTION LEVEL(V12R1M100)
      HIGHEST POSSIBLE FUNCTION LEVEL(V12R1M100)
      PROTOCOL LEVEL(2)
      GROUP ATTACH NAME(DSNG)
-----
DB2      SUB      DB2      SYSTEM      IRLM
MEMBER  ID  SYS  CMDPREF  STATUS  LVL  NAME  SUBSYS  IRLMPROC
-----
DB1A    1  DB1A  -DB1A    ACTIVE  121500  MVSA  DJ1A  DB1AIRLM
DB1B    2  DB1B  -DB1B    ACTIVE  121500  MVSB  DJ1B  DB1BIRLM
DB1C    3  DB1C  -DB1C    ACTIVE  121500  MVSC  DJ1C  DB1CIRLM
DB1D    4  DB1D  -DB1D    FAILED  111500  MVSD  DJ1D  DB1DIRLM
DB1E    5  DB1E  -DB1E    QUIESCED 111500  MVSE  DJ1E  DB1EIRLM
DB1F    6  DB1F  -DB1F    ACTIVE  111500  MVSF  DJ1F  DB1FIRLM
DB1G    7  DB1G  -DB1G    ACTIVE  121500  MVSG  DJ1G  DB1GIRLM
-----
SCA  STRUCTURE SIZE:      1024 KB, STATUS= AC,   SCA IN USE:      11 %
LOCK1 STRUCTURE SIZE:      1536 KB
NUMBER LOCK ENTRIES:      262144
NUMBER LIST ENTRIES:      7353, LIST ENTRIES IN USE:      0
*** END DISPLAY OF GROUP(DSNDB10 )
DSN9022I -DB1A DSN7GCMD 'DISPLAY GROUP ' NORMAL COMPLETION

```

Figure 2-9 The **DISPLAY GROUP** command on a data sharing group with coexistence

Figure 2-10 shows a data sharing group where all active members have been migrated to DB12 code level before activating any new function, thus the current function level is V12R1M100 and the highest possible function level is V12R1M500.

```

-DIS GROUP
DSN7100I -DB1A DSN7GCMD
*** BEGIN DISPLAY OF GROUP(DSN7CAT) CATALOG LEVEL(V12R1M500)
      CURRENT FUNCTION LEVEL(V12R1M100)
      HIGHEST ACTIVATED FUNCTION LEVEL(V12R1M100)
      HIGHEST POSSIBLE FUNCTION LEVEL(V12R1M500)
      PROTOCOL LEVEL(2)
      GROUP ATTACH NAME(DSNG)
-----
DB2      SUB      DB2      SYSTEM      IRLM
MEMBER  ID  SYS  CMDPREF  STATUS  LVL  NAME  SUBSYS  IRLMPROC
-----
DB1A    1  DB1A  -DB1A    ACTIVE  121500  MVSA  DJ1A  DB1AIRLM
DB1B    2  DB1B  -DB1B    ACTIVE  121500  MVSB  DJ1B  DB1BIRLM
DB1C    3  DB1C  -DB1C    ACTIVE  121500  MVSC  DJ1C  DB1CIRLM
DB1D    4  DB1D  -DB1D    FAILED  111500  MVSD  DJ1D  DB1DIRLM
DB1E    5  DB1E  -DB1E    QUIESCED 111500  MVSE  DJ1E  DB1EIRLM
DB1F    6  DB1F  -DB1F    ACTIVE  121500  MVSF  DJ1F  DB1FIRLM
DB1G    7  DB1G  -DB1G    ACTIVE  121500  MVSG  DJ1G  DB1GIRLM
-----
SCA  STRUCTURE SIZE:      1024 KB, STATUS= AC,   SCA IN USE:      11 %
LOCK1 STRUCTURE SIZE:      1536 KB
NUMBER LOCK ENTRIES:      262144
NUMBER LIST ENTRIES:      7353, LIST ENTRIES IN USE:      0
*** END DISPLAY OF GROUP(DSNDB10 )
DSN9022I -DB1A DSN7GCMD 'DISPLAY GROUP ' NORMAL COMPLETION

```

Figure 2-10 The DISPLAY GROUP command on a data sharing group before new function activated

Figure 2-11 shows a data sharing group where all active members have been migrated to V12 code level and new function has been activated. Note that member DB1C at DB2 11 code level will not be able to restart after new function activation because DB2 has a hard requirement that all members of a data sharing group must be at the code level equal or greater than the activated function level.

```

-DIS GROUP
DSN7100I -DB1A DSN7GCMD
*** BEGIN DISPLAY OF GROUP (DSNCAT) CATALOG LEVEL (V12R1M500)
          CURRENT FUNCTION LEVEL (V12R1M500)
          HIGHEST ACTIVATED FUNCTION LEVEL (V12R1M500)
          HIGHEST POSSIBLE FUNCTION LEVEL (V12R1M500)
          PROTOCOL LEVEL (2)
          GROUP ATTACH NAME (DSNG)
-----
DB2      SUB      DB2      SYSTEM      IRLM
MEMBER  ID  SYS  CMDPREF  STATUS  LVL  NAME  SUBSYS  IRLMPROC
-----
DB1A    1  DB1A  -DB1A    ACTIVE  121500  MVSA  DJ1A  DB1AIRLM
DB1B    2  DB1B  -DB1B    ACTIVE  121500  MVSB  DJ1B  DB1BIRLM
DB1C    3  DB1C  -DB1C    QUIESCED  111500  MVSC  DJ1C  DB1CIRLM
DB1D    6  DB1D  -DB1D    ACTIVE  121500  MVSD  DJ1D  DB1DIRLM
-----
SCA  STRUCTURE SIZE:      1024 KB, STATUS= AC,   SCA IN USE:      11 %
LOCK1 STRUCTURE SIZE:      1536 KB
NUMBER LOCK ENTRIES:      262144
NUMBER LIST ENTRIES:      7353, LIST ENTRIES IN USE:      0
*** END DISPLAY OF GROUP (DSNDB10 )
DSN9022I -DB1A DSN7GCMD 'DISPLAY GROUP ' NORMAL COMPLETION

```

Figure 2-11 The DISPLAY GROUP command on a data sharing group after new function activated

Figure 2-12 shows a data sharing group where all active members have been migrated to DB2 12 code level, a new function has been activated at level V12R1M500 and then activated down to level V12R1M100 (no more DB2 12 new function enabled). Note that members DB1D and DB1E with DB2 11 code level will not be able to restart because new function was previously activated, thus the highest activated function level is V12R1M500.

```

-DIS GROUP
DSN7100I -DB1A DSN7GCMD
*** BEGIN DISPLAY OF GROUP(DSN7CAT) CATALOG LEVEL(V12R1M500)
      CURRENT FUNCTION LEVEL(V12R1M100*)
      HIGHEST ACTIVATED FUNCTION LEVEL(V12R1M500)
      HIGHEST POSSIBLE FUNCTION LEVEL(V12R1M500)
      PROTOCOL LEVEL(2)
      GROUP ATTACH NAME(DSNG)
-----
DB2      SUB      DB2      SYSTEM      IRLM
MEMBER  ID  SYS  CMDPREF  STATUS  LVL  NAME  SUBSYS  IRLMPROC
-----
DB1A    1  DB1A  -DB1A    ACTIVE  121500  MVSA  DJ1A  DB1AIRLM
DB1B    2  DB1B  -DB1B    ACTIVE  121500  MVSB  DJ1B  DB1BIRLM
DB1C    3  DB1C  -DB1C    ACTIVE  121500  MVSC  DJ1C  DB1CIRLM
DB1D    4  DB1D  -DB1D    FAILED  111100  MVSD  DJ1D  DB1DIRLM
DB1E    5  DB1E  -DB1E    QUIESCED 111100  MVSE  DJ1E  DB1EIRLM
DB1F    6  DB1F  -DB1F    ACTIVE  121500  MVSF  DJ1F  DB1FIRLM
DB1G    7  DB1G  -DB1G    ACTIVE  121500  MVSG  DJ1G  DB1GIRLM
-----
SCA  STRUCTURE SIZE:      1024 KB, STATUS= AC,   SCA IN USE:      11 %
LOCK1 STRUCTURE SIZE:      1536 KB
NUMBER LOCK ENTRIES:      262144

```

Figure 2-12 The DISPLAY GROUP command with star function level

Figure 2-13 shows a non-data sharing DB2 subsystem migrated from DB2 11 and new function has not yet been activated.

```

-DIS GROUP
DSN7100I -DB1A DSN7GCMD
*** BEGIN DISPLAY OF GROUP(.....) CATALOG LEVEL(V12R1M500)
      CURRENT FUNCTION LEVEL(V12R1M100)
      HIGHEST ACTIVATED FUNCTION LEVEL(V12R1M100)
      HIGHEST POSSIBLE FUNCTION LEVEL(V12R1M500)
      PROTOCOL LEVEL(2)
      GROUP ATTACH NAME(....)
-----
DB2      SUB      DB2      SYSTEM      IRLM
MEMBER  ID  SYS  CMDPREF  STATUS  LVL  NAME  SUBSYS  IRLMPROC
-----
.....    0  DB1A  -DB1A    ACTIVE  121500  MVSA  DJ1A  DB1AIRLM
-----
*** END DISPLAY OF GROUP(DSNDB10 )
DSN9022I -DB1A DSN7GCMD 'DISPLAY GROUP ' NORMAL COMPLETION

```

Figure 2-13 The DISPLAY GROUP command on non-data sharing member

2.6 Catalog changes

When a package or package copy is bound or rebound, the current function level is saved in the following new columns:

- ▶ SYSIBM.SYSPACKAGE.FUNCTION_LEVEL
- ▶ SYSIBM.SYSPACKCOPY.FUNCTION_LEVEL

When a row is inserted in SYSQUERY on the BIND QUERY subcommand, the current function level is saved in the following new column:

- ▶ SYSIBM.SYSQUERY.FUNCTION_LVL

2.6.1 The SYSIBM.SYSLEVELUPDATES table

DB2 12 introduces the SYSIBM.SYSLEVELUPDATES catalog table to help track the history of various changes to code level, catalog level, and function level over time. Each successful execution of the **ACTIVATE** command is recorded in the SYSIBM.SYSLEVELUPDATES table so history of these commands can be reviewed. When a **CATMAINT UPDATE LEVEL** utility is run to change the catalog level, a row is also inserted in this table. In addition, the table records the information when the code level is updated for the DB2 subsystem or members in the data sharing group.

Table 2-1 shows the columns defined in the SYSIBM.SYSLEVELUPDATES catalog table.

Table 2-1 The SYSIBM.SYSLEVELUPDATES catalog table

Column name	Data type	Description
FUNCTION_LVL	VARCHAR(10)	Function level when this record was inserted
PREV_FUNCTION_LVL	VARCHAR(10)	Previous function level
HIGH_FUNCTION_LVL	VARCHAR(10)	Highest activated function level
CATALOG_LVL	VARCHAR(10)	Catalog level when this record was inserted
OPERATION_TYPE	CHAR(1)	Type of operation: <ul style="list-style-type: none">▶ "C" for catalog change▶ "F" for function level▶ "M" for maintenance update
EFFECTIVE_TIME	TIMESTAMP(12)	Time when the operation completed
EFFECTIVE_LRSN	CHAR(10)	RBA (or LRSN for data sharing) depicting the time that an operation completed
OPERATION_TEXT	VARCHAR(256)	The text of the operation
GROUP_MEMBER	VARCHAR(24)	Name of the group member on which the operation was executed

For the example assume that function levels V12R1M503, V12R1M505, and V12R1M506 are available after function level V12R1M500. The content of the SYSLEVELUPDATES table in this example indicates that three **ACTIVATE** commands were used at different times. The code levels for both DB2A and DB2B members in the data sharing group were updated prior to activating the function level V12R1M505. The user can also see the previous function level immediately before the **ACTIVATE** command and the highest ever activated function level. Note the action column values might change from this example when DB2 delivers the next code level.

Table 2-2 The SYSLEVELUPDATES table's rows

Function_lvl	Prev_Function_lvl	Catalog_lvl	Operation_type	Effective_time	Operation_text	Group_Member
V12R1M100	V12R1M100	V12R1M500	C	2017-03-01	CATMAINT PROCESSING	DB2A
V12R1M500	V12R1M100	V12R1M500	F	2017-03-01	ACTIVATE FUNCTION LEVEL(V12R1500)	DB2A
V12R1M500	V12R1M100	V12R1M500	M	2017-06-01	TBD	DB2A
V12R1M500	V12R1M100	V12R1M500	M	2017-06-01	TBD	DB2B
V12R1M505	V12R1M500	V12R1M500	F	2017-07-01	ACTIVATE FUNCTION LEVEL(V12R1505)	DB2B
V12R1M503	V12R1M505	V12R1M500	F	2017-08-01	ACTIVATE FUNCTION LEVEL(V12R1503)	DB2A
V12R1M503	V12R1M505	V12R1M506	C	2017-12-01	CATMAINT (V12R1506)	DB2B

2.7 Application compatibility (APPLCOMPAT)

DB2 11 introduced the concept of application compatibility as a package's APPLCOMPAT bind option on the **BIND PACKAGE**, **REBIND PACKAGE**, and **REBIND TRIGGER PACKAGE** subcommands or the CURRENT APPLICATION COMPATIBILITY special register. The APPLCOMPAT bind option's default value is the APPLCOMPAT subsystem parameter. Their format consists of version and release levels, and the two values supported in DB2 11 are V10R1 and V11R1. They were added for the following purposes:

- ▶ To handle the application's incompatible behaviors introduced by a new release
- ▶ To enable new SQL functionalities such as new SQL syntax and semantics

APPLCOMPAT will be extended to support specification of function levels. In DB2 12, APPLCOMPAT is extended to a more granular level with the modification level to follow the same format as function level. Initially, support for the following specifications of APPLCOMPAT are available:

- ▶ V10R1: SQL behaviors are compatible to DB2 10.
- ▶ V11R1: SQL behaviors are compatible to DB2 11.
- ▶ V12R1M100: Is equivalent to V11R1.
- ▶ V12R1M500: SQL behaviors are compatible to DB2 12 and can be specified only if the subsystem or data sharing group is at function level V12R1M500 or greater.

As new function levels are introduced, the corresponding new APPLCOMPAT values will also be introduced for application usage. Example 2-1 shows the BIND PACKAGE syntax.

Example 2-1 The BIND PACKAGE syntax with the APPLCOMPAT option

```
BIND PACKAGE
>-----+-----+----->
-----|-----APPLCOMPAT-----(--V10R1--)-----|
                                   (--V11R1--)
                                   (--function-level--)
```

Before the user can take advantage of new application capabilities delivered and enabled by a function level, the user must also specify the corresponding application compatibility value through the **BIND** or **REBIND** subcommand. For instance, to use the SQL syntax specified in Chapter 6, “SQL” on page 61 the application must be bound with the APPLCOMPAT (V12R1M500) or greater value.

There are no changes necessary to retrieve the APPLCOMPAT level of a package—the information is available in the same place as it was in DB2 11. The existing catalog SYSPACKAGE.APPLCOMPAT column’s data type is VARCHAR(10) and is long enough to carry the extended format added in DB2 12. Also note that the APPLCOMPAT option on some Data Definition Language (DDL) statements for SQL routines and advanced triggers also follows the same format because they are to become the APPLCOMPAT bind option for the routine or trigger’s corresponding package.

Figure 2-14 shows the updated DSNEBP10 panel for application compatibility.

```

DSNEBP10                                DEFAULTS FOR BIND PACKAGE                                SSID: DSN
COMMAND ==>

Change default options as necessary:
----- Use the UP/DOWN keys to access all options -----

 1 ISOLATION LEVEL ..... ==>          (CS, RR, RS, UR, or NC)
 2 VALIDATION TIME ..... ==>          (RUN or BIND)
 3 RESOURCE RELEASE TIME ... ==>      (COMMIT, DEALLOCATE, or
                                       INHERITFROMPLAN)
 4 EXPLAIN PATH SELECTION .. ==>      (NO, YES, or ONLY)
 5 DATA CURRENCY ..... ==>          (NO or YES)
 6 PARALLEL DEGREE ..... ==>          (1 or ANY)
 7 SQLERROR PROCESSING ..... ==>    (NOPACKAGE or CONTINUE)
 8 REOPTIMIZE FOR INPUT VARS ==>      (ALWAYS, NONE, ONCE, or AUTO)
 9 DEFER PREPARE ..... ==>           (NO, YES, INHERITFROMPLAN)
10 KEEP DYN SQL PAST COMMIT. ==>      (NO or YES)
11 APPLICATION ENCODING..... ==>      (Blank, ASCII, EBCDIC, UNICODE,
                                       or ccsid)
12 OPTIMIZATION HINT ..... ==>        > (Blank or 'hint-id')
13 IMMEDIATE WRITE..... ==>          (NO, YES, INHERITFROMPLAN)
14 DYNAMIC RULES ..... ==>          (RUN, BIND, DEFINE, or INVOKE)
15 DBPROTocol ..... ==>            (blank, DRDA, or DRDACBF)
16 ACCESS PATH REUSE ..... ==> NONE  (ERROR, NONE, or WARN)
17 ACCESS PATH COMPARISON .. ==> NONE (ERROR, NONE, or WARN)
18 SYSTEM_TIME SENSITIVE ... ==>     (blank, NO, or YES)
19 BUSINESS_TIME SENSITIVE . ==>     (blank, NO, or YES)
20 ARCHIVE SENSITIVE ..... ==>      (blank, NO, or YES)
21 APPLICATION COMPATIBILITY ==>      (blank, DB2 function level,
                                       V10R1, or V11R1)

-----
PRESS:  ENTER to continue  UP/DOWN to scroll  RETURN to EXIT

```

Figure 2-14 The DSNEBP10 migration panel with the APPLICATION COMPATIBILITY zparm

2.7.1 Managing applications through function level changes

On a BIND or REBIND package's subcommand, DB2 12 checks that the specified application compatibility value is equal or less than the current function level. This check is skipped on autobind and REBIND without a new APPLCOMPAT option specified. This check is to ensure that the new SQL syntax is allowed for the new and changed applications at the appropriate level and disallowed when the function level has been changed to a lower function level (star function level). The assumption is that when the function level is activated to a lower level, the user does not want to expose more new applications to new enhancements in the previous function level. However, when the existing applications use those enhancements, they should still be able to execute.

The CURRENT APPLICATION COMPATIBILITY special register has the default value from the package's bind option and is applicable to dynamic data manipulation statements. In DB2 11, the SET CURRENT APPLICATION COMPATIBILITY SQL statement can be used to change the special register to a value other than the APPLCOMPAT bind option. The CURRENT APPLICATION COMPATIBILITY special register is still supported in DB2 12. However, the specified value is V12R1M500 or greater; that value must be the same or less than the APPLCOMPAT bind option in the package where the SQL statement is executed.

DB2 mandates this restriction to avoid failure for existing applications with dynamic SQL statements when executed in lower star function level. The package's APPLCOMPAT bind option does not change when function level changes.

The preference is not to rely on the SET CURRENT APPLICATION COMPATIBILITY SQL statement in applications if the function level can become a star function level later. Instead, bind the application's package with the APPLCOMPAT option at the required function level and let it be the special register's default value.

Consider this next example scenario. Assume a DB2 subsystem was activated with function level V12R1M500 and then later activated back to function level V12R1M100, which is a star function level. Package A is bound with the APPLCOMPAT(V12R1M500) option and has a static SELECT ARRAY_AGG statement (which is a DB2 11 syntax) and a dynamic SELECT OFFSET statement (which is a DB2 12 syntax). Package B is bound with the APPLCOMPAT(V12R1M100) option and has the same two SQL statements and a SET CURRENT APPLICATION COMPATIBILITY = V12R1M500 statement before the dynamic SELECT statement.

Figure 2-15 shows the execution status for each SQL statement in the function levels described for this scenario. All of the statements are successful except for package B's SET CURRENT APPLICATION COMPATIBILITY statement and SELECT statement with the new OFFSET clause.

Package	SQL statement	Function level 500	Function level 100*
A bound with APPLCOMPAT(V12R1M500)	SELECT ARRAY_AGG(PHONENO ORDER BY SALARY) FROM DSN81110.EMP WHERE SALARY > LOWSAL	ok	ok
A	PREPARE S1 FROM SELECT * FROM T1 OFFSET 100	ok (CURRENT APPLICATION COMPATIBILITY V12R1M500)	ok
B bound with APPLCOMPAT(V12R1M100)	SELECT ARRAY_AGG(PHONENO ORDER BY SALARY) FROM DSN81110.EMP WHERE SALARY > LOWSAL	ok	ok
B	SET CURRENT APPLICATION COMPATIBILITY = 'V12R1M500'	ok	fail
B	PREPARE S1 FROM SELECT * FROM T1 OFFSET 100	ok (CURRENT APPLICATION COMPATIBILITY V12R1M100)	fail

Figure 2-15 Two packages with statements executing in function levels V12R1M500 and V12R1M100*

2.7.2 Data Definition Language (DDL) statements sensitive to APPLCOMPAT

Similar to when applications that issue DDL statements need to be protected when the system is activated to a star function level, DDL changes introduced in DB2 12 now require the APPLCOMPAT setting also. For example, the PAGENUM option on the CREATE TABLESPACE statement to create a partition-by-growth with relative page number, or the new TRANSFER OWNERSHIP statement are allowed only when the APPLCOMPAT option of the containing package is V12R1M500 or greater.

Assuming the time is in increasing order, Figure 2-16 shows the behaviors in various function levels, including star function level, for new versus existing applications that exploit new functions.

Time	Action	Result
1	-ACTIVATE FUNCTION LEVEL(V12R1M500)	Successful
2	BIND PACKAGE A with APPLCOMPAT(V12R1M500) that has a TRANSFER OWNERSHIP statement	Successful because current function level is V12R1M500
3	Execute package A	Successful because current function level is V12R1M500
4	-ACTIVATE FUNCTION LEVEL(V12R1M100)	Successful
5	BIND PACKAGE B with APPLCOMPAT(V12R1M500) that has a CREATE TABLESPACE.. PAGENUM() statement	Failed because new package and current function level is V12R1M100
6	REBIND PACKAGE A	Successful because package already existed
7	Execute package A	Successful because package already existed

Figure 2-16 Star function level and new versus existing applications that exploit new functions

DB2 12 saves the APPLCOMPAT option that is in effect when executing the DDL statement in the new SYSENVIRONMENT.APPLCOMPAT column. This value can be used at various activities such as pending definition changes, explicit and implicit regeneration of objects, REPAIR DBD utility, and others. The need might exist to change the value that is saved in the SYSENVIRONMENT.APPLCOMPAT column for an object. Therefore, a new USING APPLICATION COMPATIBILITY option is added to the ALTER REGENERATE statement in DB2 12 so the user can update the APPLCOMPAT value for the altered object.

2.8 SQL processing option SQLLEVEL

A new SQL processing option (precompile and coprocess option), SQLLEVEL, is added in DB2 12 and has the same format as function level. The SQLLEVEL option plays a similar role as the package's APPLCOMPAT bind option. New SQL syntax is allowed by the DB2 12 precompiler or coprocessor based on the setting of the SQLLEVEL option. Similar to the NEWFUN option, which is only a CHAR(3) data type, SQLLEVEL is used to precompile or compile without DB2 online. Hence, when the DBRM is bound, the actual bind result depends on whether or not DB2 is at the appropriate function level.

The NEWFUN processing option is deprecated, although still allowed up to the V12 value. If the NEWFUN and SQLLEVEL options are specified together, the NEWFUN value is ignored and the message DSNH4789I is issued. If neither NEWFUN nor SQLLEVEL is specified on a precompilation or compilation, the DSNHDECP SQLLEVEL parameter is used as the default value.

The DSNHDECP SQLLEVEL parameter can be set on the DSNTIP41 installation panel (Figure 2-17). In installation mode, the default setting of this field is the maximum supported DB2 function level (currently V12R1M500); in migration mode, it is V12R1M100.

```
DSNTIP41          INSTALL DB2 - APPLICATION PROGRAMMING DEFAULTS PANEL 3
===>

Enter the application compatibility level:
 1 APPL COMPAT LEVEL    ==> V12R1M500 (DB2 function level or V11R1 or V10R1)

Enter the default SQL level for the precompiler:
 2 PRECOMPILER SQL LEVEL ==> V12R1M500 (DB2 function level or V11R1 or V10R1)

PRESS:  ENTER to continue  RETURN to exit  HELP for more information
```

Figure 2-17 The DSNTIP41 panel for setting the Pre-compiler SQLLEVEL option

2.9 New built-in global variables

To enable an application programming interface (API) for the various levels introduced for continuous delivery in DB2 12, the following built-in global variables are created. Because of single-phase migration, an application that retrieves these new variables can execute on a coexist or fallback DB2 11 member and the values to be returned are indicated.

2.9.1 PRODUCTID_EXT

The PRODUCTID_EXT global variable contains the extended product identifier of the database manager that invoked the function. This global variable has the following characteristics:

- ▶ It is read only, with values maintained by the system.
- ▶ The type is VARCHAR(30).
- ▶ The schema is SYSIBM.
- ▶ The scope of this global variable is session.

The format of the extended product identifier values is pppvvrmmm, where ppp is a three-letter product code (such as DSN for DB2), vv is the version, rr is the release, and mmm is the modification level (such as 100, 500, 501). For example, DSN1201500 identifies DB2 12 after the activation of DB2 12 new function (function level 500 or greater). If retrieved on a coexist DB2 11 member of a data sharing group, the value DSN1101500 is returned.

2.9.2 CATALOG_LEVEL

The CATALOG_LEVEL global variable contains the level of the current catalog. This global variable has the following characteristics:

- ▶ It is read only, with values maintained by the system.
- ▶ The type is VARCHAR(30).
- ▶ The schema is SYSIBM.
- ▶ The scope of this global variable is session.

The format of the catalog level values is VvvRrMmmm, where vv is the version, r is the release, and mmm is the modification level (such as 100, 500, 501). For example, V12R1M500 identifies DB2 12 after the activation of DB2 12 new function (function level 500 or greater). If retrieved on a coexist DB2 11 member of a data sharing group, the value V12R1M500 is returned because the catalog has been converted to that level.

2.9.3 DEFAULT_SQLLEVEL

The DEFAULT_SQLLEVEL variable contains the default value of the SQLLEVEL SQL processing option (DECPSQLL). This global variable has the following characteristics:

- ▶ It is read only, with values maintained by the system.
- ▶ The type is VARCHAR(30).
- ▶ The schema is SYSIBM.
- ▶ The scope of this global variable is session.

The format of the catalog level values is V10R1, V11R1, or VvvRrMmmm, where vv is the version, r is the release, and mmm is the modification level (such as 100, 500, 501). For example, V12R1M500 identifies DB2 12 after the activation of DB2 12 new function (function level 500 or higher). If retrieved on a coexist DB2 11 member of a data sharing group, the value V12R1M100 is returned because the group has not yet been activated to new function.

2.10 DB2 supplied stored procedures

To accommodate the output change on the **DISPLAY GROUP** command, the following stored procedures supplied by DB2 are affected:

- ▶ ADMIN_COMMAND_DB2
- ▶ GET_CONFIG

2.10.1 ADMIN_COMMAND_DB2

The ADMIN_COMMAND_DB2 stored procedure's result set data type for the DB2_LVL parameter is changed as documented in Figure 2-18.

Column name	Data type	Contents
ROWNUM	INTEGER	Sequence number of the table row, from 1 to n
DB2_MEMBER	CHAR(8)	Name of the DB2 group member
ID	INTEGER	ID of the DB2 group member
SUBSYS	CHAR(4)	Subsystem name of the DB2 group member
CMDPREF	CHAR(8)	Command prefix for the DB2 group member
STATUS	CHAR(8)	Status of the DB2 group member
DB2_LVL	CHAR(6)	DB2 version, release and modification level
SYSTEM_NAME	CHAR(8)	Name of the z/OS® system where the member is running, or was last running in cases when the member status is QUIESCED or FAILED
IRLM_SUBSYS	CHAR(4)	Name of the IRLM subsystem to which the DB2 member is connected
IRLMPROC	CHAR(8)	Procedure name of the connected IRLM

Figure 2-18 Result set row for second ADMIN_COMMAND_DB2 result set (processing-type = "GRP")

Running the DSNTIJRT job with MODE(INSTALL) will define the result table SYSIBM.DATA_SHARING_GROUP accordingly and bind the DBRMs for the stored procedure.

Existing DB2 11 applications that call the SYSPROC.ADMIN_COMMAND_DB2 stored procedure and retrieve the SYSIBM.DATA_SHARING_GROUP result set to access the DB2_LVL column must be updated to use CHAR(6) instead of CHAR(3).

2.10.2 GET_CONFIG

DB2 12 modifies the XML output of the SYSPROC.GET_CONFIG stored procedure as follows:

- ▶ Returns the following additional information from the output of the **-DISPLAY GROUP** command under Common Data Sharing Group Information:
 - Data Sharing Group Current Function Level
 - Data Sharing Group Highest Activated Function Level
 - Data Sharing Group Highest Possible Function Level
- ▶ Changes the Data Sharing Group Level key to Data Sharing Group Catalog Level key.
- ▶ Removed the Data Sharing Group Mode key.

Figure 2-19 shows the XML output.

```
<key>Data Sharing Group Catalog Level</key>
<dict>
  <key>Display Name</key>
  <string>Data Sharing Group Catalog Level</string>
  <key>Value</key><string>V12R1M500</string>
  <key>Hint</key><string/>
</dict>
<key>Data Sharing Group Current Function Level</key>
<dict>
  <key>Display Name</key>
  <string>Data Sharing Group Current Function Level</string>
  <key>Value</key><string>V12R1M100</string>
  <key>Hint</key><string/>
</dict>
<key>Data Sharing Group Highest Activated Function Level</key>
<dict>
  <key>Display Name</key>
  <string>Data Sharing Group Highest Activated Function Level</string>
  <key>Value</key><string>V12R1M100</string>
  <key>Hint</key><string/>
</dict>
<key>Data Sharing Group Highest Possible Function Level</key>
<dict>
  <key>Display Name</key>
  <string>Data Sharing Group Highest Possible Function Level</string>
  <key>Value</key><string>V12R1M100</string>
  <key>Hint</key><string/>
</dict>
```

Figure 2-19 The XML output of the SYSPROC.GET_CONFIG stored procedure

Running the DSNTIJRT job with MODE(INSTALL) defines the result table SYSIBM.DATA_SHARING_GROUP accordingly and binds the DBRMs for the stored procedure. Existing applications that call the SYSPROC.GET_CONFIG stored procedure must be updated to handle the new information.

2.11 Resource access control facility exit

The Resource access control module's parameter list is changed with a new CHAR(10) field to reflect the new code level's format with version, release, and modification levels. The new field is XAPLLVLX and replacing the deprecated XAPLLVL field.

2.12 Instrumentation

The header for all DB2 12 trace records has several new fields to return information for code level and incompatible changes to a trace record in the continuous delivery code stream:

- ▶ **QWHS_MOD_LVL**
Denotes the code level where the trace record was written.
- ▶ **QWHS_REC_COMPAT** and **QWHS_REC_INCOMPAT**
Can be used to check whether a trace record's mapping has a compatible or incompatible change.
- ▶ **QWHS_MOD_LVL DS CL10**
Code level for continuous delivery.
- ▶ **QWHS_REC_INCOMPAT DS XL2**
Incompatible change value. Incremented each time an incompatible change occurs, such as changing the size of existing fields in a record or removing fields no longer being set, which causes the offset to other fields to change.
- ▶ **QWHS_REC_COMPAT DS XL2**
Compatible change value. Incremented each time a compatible change occurs, such as adding a new field in a reserved area, no longer setting an existing field, or increasing the size of a record to add a new field) is made.

Part 2



Subsystem

This part contains the following chapters:

- ▶ Chapter 3, “Scalability” on page 33
- ▶ Chapter 4, “Availability” on page 43
- ▶ Chapter 5, “Data sharing” on page 49



Scalability

The scalability enhancements for DB2 12 are described in the following topics:

- ▶ Range-partitioned table spaces changes
- ▶ DB2 internal latch contention relief
- ▶ Buffer pool simulation
- ▶ Support for sizes greater than 4 GB of active log data sets

3.1 Range-partitioned table spaces changes

Before DB2 12, the maximum number of partitions for a partitioned table was limited to 4,096 and the maximum partition size was also limited to 256 GB, so the partitioned table could only have up to 64 partitions if the page size is 4K. The maximum number of partitions for a partitioned table was dependent on the page size and the partition size. These limitations existed because data pages were sequentially formatted based on the table space, not based on each data partition.

DB2 12 introduces a solution for that scenario. A new type of partition-by-range (PBR) structure called *partition-by-range relative page numbering (PBR RPN)*.

3.1.1 PBR RPN table space characteristics

PBR RPN table space provides many structure differences compared to the previous format, starting with relative page numbers (RPNs) that represent page numbers without embedded partition numbers, instead of absolute page numbers.

In PBR structure, the partition size had a limit of 256 GB and PBR RPN enables the DSSIZE to grow up to 1 TB for a partition, had an increase of the maximum table size from 16 TB (4K page) to 4 PB, and is designed to go even larger.

Several DB2 internal data structures are expanded to 7-byte RIDs (2-byte part number, 5-byte page number); achieving up to 256 trillion rows in a single table is also possible.

3.1.2 PBR RPN partitioned index characteristics

The maximum index partition size can be up to 1 TB and the size is independent from the data partition size.

The record ID (RID) for each index entry will also be 7 bytes. The RID contains a 2-byte partition number, a 4-byte page number, and a 1-byte record-id within the data page.

The index header page will record the index partition number and an indicator to indicate it is a PBR RPN partitioned index.

The index page numbers within each index partition do not include the index partition number.

3.1.3 PBR RPN non-partitioned index characteristics

Non-partitioned index (NPI), on PBR RPN table spaces, continues to have the same characteristics as for NPI in the previous DB2 versions, with some exceptions:

- ▶ For 4K, 8K, 16K, or 32K index page size, the maximum index space size will continue to be limited to 16 TB, 32 TB, 64 TB, or 128 TB, respectively.
- ▶ The maximum PIECESIZE size is limited to the DSSIZE of the table space, with a maximum value of 256 GB; the default is 4 GB. Altering the PIECESIZE will result in the index being placed into page set rebuild pending (PSRBP).
- ▶ The RID length in NPI is 7 bytes.
- ▶ The PAGENUM setting in SYSINDEXES and SYSINDEXPART will be 'A' to reflect absolute page numbering.
- ▶ The header page will indicate that relative page numbering is used.

3.1.4 PBR RPN considerations

The creation of PBR RPN table space is performed by CREATE table space or CREATE table (implicit space) that has page format that differs from PBR table spaces. DEFINE NO is used to avoid formatting of page sets. The table may have partitioned and non-partitioned indexes.

A ZPARM parameter (PAGESET_PAGENUM) is used to control whether creation of range-partitioned uses relative page numbering. This ZPARM parameter applies when the PAGENUM keyword is not specified on a CREATE TABLESPACE or CREATE TABLE with implicit table space creation.

Figure 3-1 shows the syntax diagram for CREATE TABLESPACE with PAGENUM RELATIVE option.

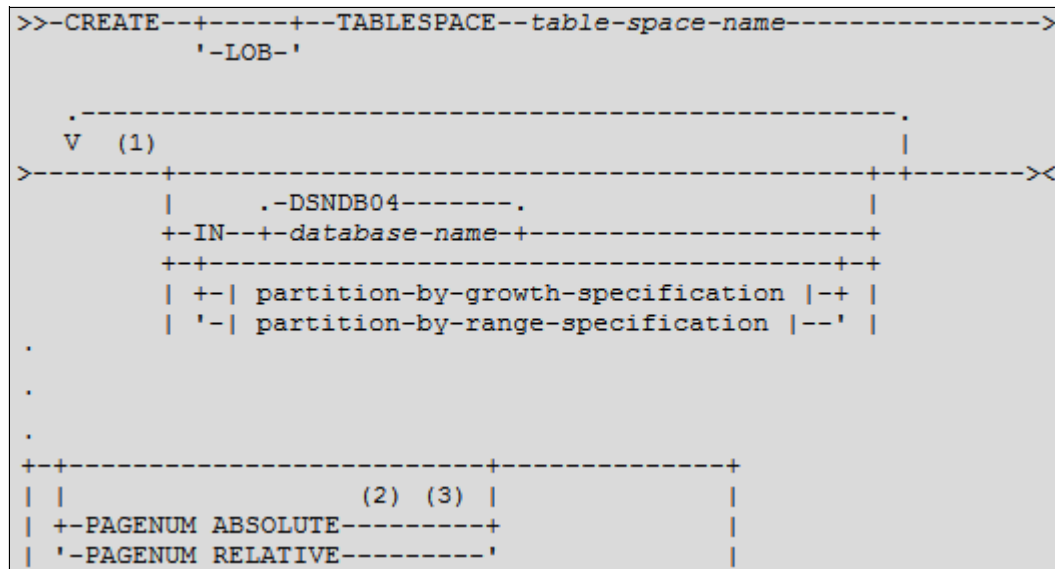


Figure 3-1 Syntax diagram for CREATE TABLESPACE

The existing PBR table spaces can be converted to PBR RPN with the ALTER TABLESPACE statement by specifying PAGENUM RELATIVE. The ALTER TABLESPACE statement syntax representation is shown in Figure 3-2 on page 36.

Figure 3-4 compares DB2 12 and DB2 11 for which log latch reduction results were obtained in a special insert test case.

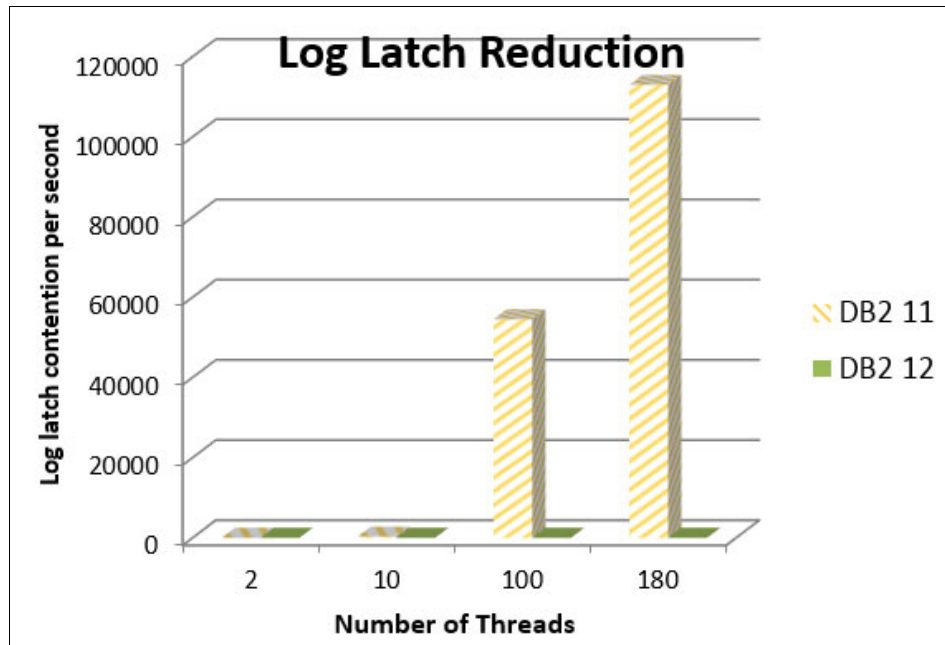


Figure 3-4 Log latch reduction comparison

3.3 Buffer pool simulation

The benefit from increasing the size of buffer pool varies depending on the environment workload. Buffer pool simulation provides accurate simulation results from increasing the buffer pool size as it was in a real workload for a production environment.

To explain how buffer pool simulation works, the following scenario is used in an example:

- ▶ IBM Fictional Brokerage online transaction workload uses two-way data sharing, with approximately 3000 transactions per second (TPS).
- ▶ Originally uses a total of 14 GB (7G * 2) local buffer pools.
- ▶ Simulate buffer pool one (BP1) to expand from 8 MB to 2 GB, total 18GB (9G * 2) buffer pools.
- ▶ Collect statistics data or **DISPLAY BUFFER POOL** command outputs.

Figure 3-5 shows the buffer pool simulation steps.

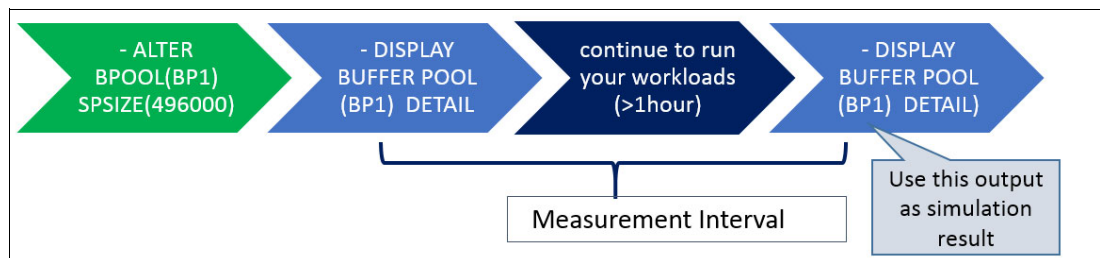


Figure 3-5 Buffer pool simulation steps

Note: The preference is to reset SPSIZE(0) when simulating more than one SPSIZE and also to take enough samples (2 - 3 hours) per SPSIZE.

Figure 3-6 shows the representation for display output from `-DIS BPOOL (BP1) DETAIL` for the measurement interval.

```
DSNB432I  -CEA1 SIMULATED BUFFER POOL ACTIVITY -
          AVOIDABLE READ I/O -
          SYNC READ I/O (R) =25463982
          SYNC READ I/O (S) =81181
          ASYNC READ I/O      =15470503
          SYNC GBP READS (R) =11172099
          SYNC GBP READS (S) =4601
          ASYNC GBP READS     =1181076
          PAGES MOVED INTO SIMULATED BUFFER POOL =53668641
          TOTAL AVOIDABLE SYNC I/O DELAY =35321543 MILLISECONDS
```

Figure 3-6 Display output from `-DIS BPOOL (BP1) DETAIL`

To calculate *numbers of avoidable sync I/O per second*, the formula in Example 3-1 is used.

Example 3-1 Numbers of avoidable sync I/O per second calculation

$$\text{Sync READ I/O (R) + SYNC READ I/O (S) / interval =}$$
$$\text{(avoidable sync I/O per second)}$$

By adding the values related to the example, the calculation is as follows:

$$(25463982 + 81181) / 360 = \mathbf{70958} \text{ I/O per sec}$$

The simulation and validation results are shown in Figure 3-7 on page 39 and Figure 3-8 on page 39.

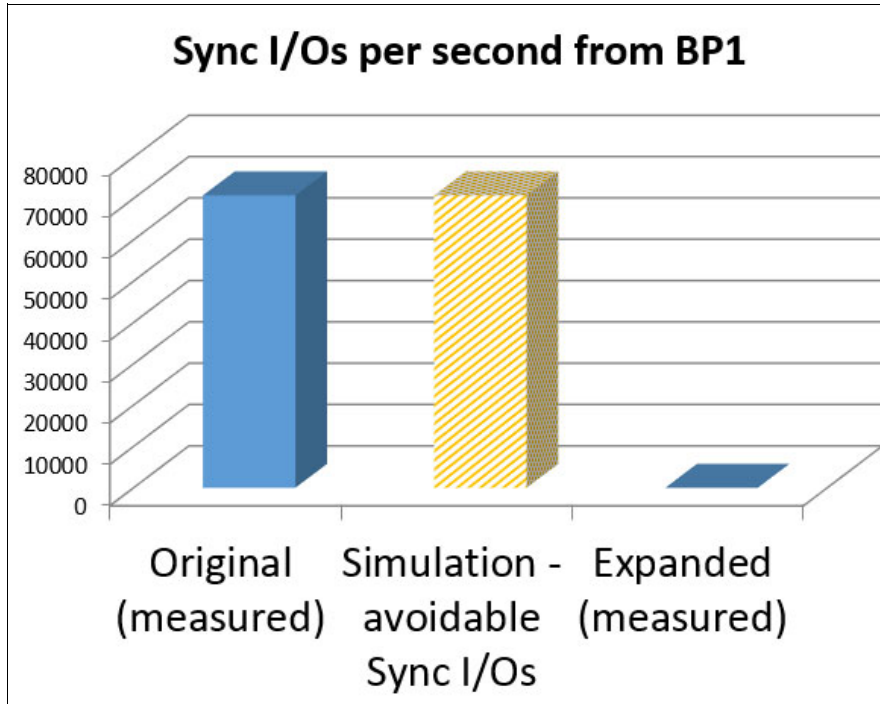


Figure 3-7 Simulation and validation results: Sync I/Os per second from BP1

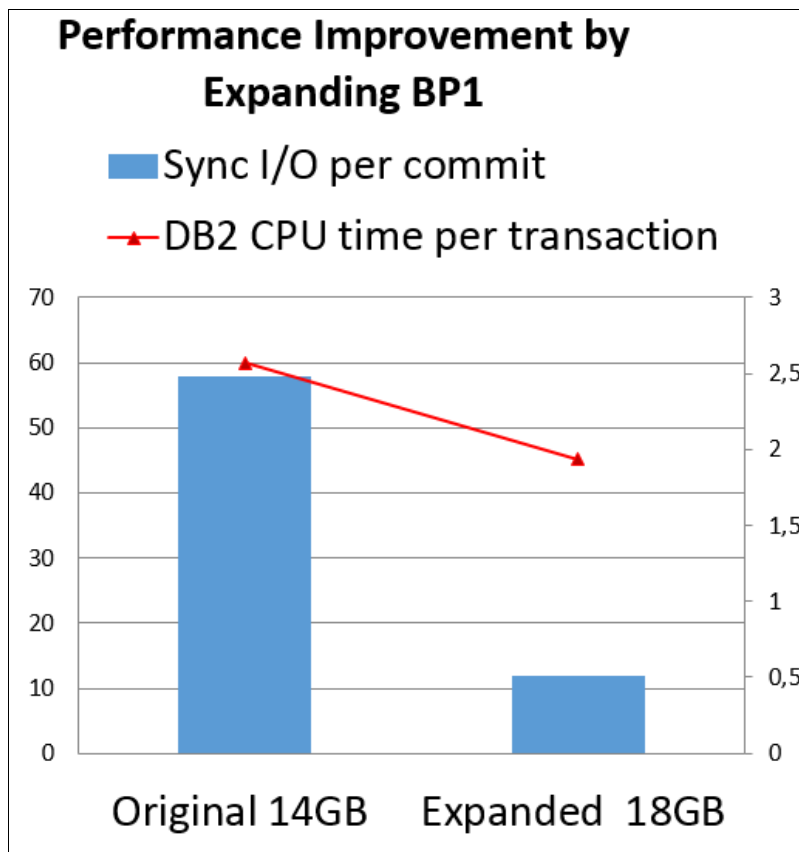


Figure 3-8 Simulation and validation results: Performance improvement by expanding BP1

3.4 Support for sizes greater than 4 GB of active log data sets

Versions before DB2 12 support up to 4 GB size of active log data sets; DB2 12 greatly increased the size of active log data sets to 768 GB.

Note: When a VSAM data set greater than 4 GB is defined, an association with System Management Services (SMS) data class is required where the extended addressability (EA) attribute is set to YES. The *support >4GB log* will be allowed only after DB2 12 new-function mode has been activated.

Two methods are available to add new active log data sets that are greater than 4 GB:

- ▶ DB2 stand-alone utility DSNJU003
After DB2 new function is activated, stop DB2, run DSNJU003 NEWLOG to add the new log, then restart DB2.
- ▶ DB2 **-SET LOG NEWLOG** command
After DB2 new-function mode is up, issue the command to add the new log.

When detecting >4GB logs in non-new function mode, log manager will issue a new error message DSNJ158I. Furthermore, if detected during DB2 start, DB2 will be terminated with abend code 00E80084. If it is from a **-SET LOG** command, the command will fail.

Alternatively, new message DSNJ159I is issued in new function when >768GB active log data set size is detected.

Changes to messages DSNJ158I and DSNJ159I are described in the following list:

- ▶ New message DSNJ158I
DSNJ158I csect-name ACTIVE LOG DATA SET DSN=dsname IS GREATER THAN 4 GB IN SIZE
 - Explanation
DB2 detected an active log data set greater than 4 GB in size. At that time DB2 V12 new function was not activated yet.
 - dsname
The data set name for the active log encountering the error.
 - System action
If it occurred from processing the **-SET LOG NEWLOG** command, the command failed. If it occurred during DB2 start, DB2 abnormally terminated with 00E80084.
 - System programmer response
If from the **-SET LOG NEWLOG** command, reallocate the data set to be less than 4 GB in size. Then, retry the command. Greater than 4 GB is valid only after DB2 V12 new function has been activated. Use the **-DISPLAY GROUP** command to show NEW FUNCTION(Y) or (N).
If from DB2 start, reallocate the data set to be less than 4 GB in size or use DSNJU003 utility with **DELETE dsname** to remove the data set. Then restart DB2. Greater than 4 GB is valid for DB2 restart only after DB2 V12 new function has been activated before.

► New message DSNJ159I

DSNJ159I csect-name ACTIVE LOG DATA SET DSN=dsname IS GREATER THAN 768 GB IN SIZE

– Explanation

DB2 detected an active log data set greater than 768 GB in size, which is not supported. The maximum size is 768 GB in DB2 V12 new function.

– dsname

The data set name for the active log encountering the error.

– System Action

If it occurred from processing the **-SET LOG NEWLOG** command, the command failed. If it occurred during DB2 start, DB2 abnormally terminated with 00E80084.

– System programmer response

If from **-SET LOG NEWLOG**, reallocate the data set to be less than 768 GB in size. Then, retry the command.

If from DB2 start, reallocate the data set to be less than 768 GB in size or use DSNJU003 DELETE DSNAME to remove the data set. Then restart DB2.

Although the new maximum size for active log data sets is 768 GB, you might opt for size less than 768 GB. The active log data set is allocated with a primary allocation quantity and zero secondary quantity. The maximum volume size is 54 GB for a 3390-9, and 223 GB for a 3390-A EAV (extended address volume) unless copy services IBM FlashCopy®, mirror, or PPRC) are not used, and then it is 1 TB. Offload to archive logs will be another factor in choosing the optimal size. The archive log data set (vs tape), which is a sequential data set, might require extended format (EF) to accommodate the new large size.



Availability

This chapter introduces availability improvements by the following enhancements:

- ▶ Improved availability for pending definition changes
- ▶ Catalog availability improvements
- ▶ Removal of point-in-time recovery restrictions for PBG table spaces
- ▶ PBR RPN DSSIZE increase
- ▶ Insert partition
- ▶ REORG enhancements for PBGs, FlashCopy and LOBs
- ▶ LOAD RESUME YES BACKOUT YES option
- ▶ Faster point-in-time recovery
- ▶ TRANSFER OWNERSHIP SQL statement
- ▶ Auto-retry of GRECP and LPL recovery

4.1 Improved availability for pending definition changes

DB2 12 introduces availability improvements for pending definition changes, allowing applications to access objects that in the previous DB2 releases had a restriction status that prevented the access. This section describes two enhancement improvements:

- ▶ Altering index compression attribute
- ▶ Altering column

4.1.1 Altering index compression attribute

Indexes with a page size greater than 4 KB can be compressed to 4K page on DB2, and as a result, DASD space is saved and achieves improved I/O efficiency.

However, when the compress attribute of an index in universal table space was altered, the index was marked as a rebuild pending (RBDP) status, preventing applications from using the index until the REORG TABLESPACE utility or the REBUILD INDEX utility completed.

DB2 12 introduces an improvement to the availability of indexes in universal table spaces, now alterations to index compression are a pending change, placing the index in advisory REORG-pending (AREOR) status, therefore applications can continue to access the indexes.

The updated value for the COMPRESS attribute in the ALTER INDEX statement is materialized by a subsequent online REORG INDEX or online REORG TABLESPACE at the table space level. With this improvement, database administrators can correct or remove a pending change to index compression without affecting the target index. Also, this improvement reduces the planning and costs that are associated with an application outage caused by the previous behavior in DB2 11.

A new record is inserted in the SYSIBM.SYSPENDINGDDL catalog table with the following information, when an **ALTER INDEX COMPRESS** is issued:

- ▶ OBJTYPE column = 'I'
- ▶ OPTION_KEYWORD = 'COMPRESS'
- ▶ OPTION_VALUE = 'xxx' ('YES' or 'NO')

Notes:

- ▶ If the index is defined with the DEFINE NO attribute and data sets are not created yet, the alteration is still immediate.
- ▶ Also, for an index that is not in a universal table space, an alteration to index compression can be a pending change if other pending changes exist at the index, table, or table space level when the ALTER INDEX COMPRESS statement runs.

4.1.2 Altering column

In the previous DB2 releases, DB2 provided the capability to alter column attributes such as data type, length, precision, or scale of columns for a table through ALTER TABLE statement with the ALTER COLUMN clause, but these alterations impact the availability or extra performance overhead is required.

For example, in DB2 11, some column definition changes are immediate changes. That means the affected indexes on the table are put in a restricted status. These indexes are not

available to be used in an access path optimization process. If a unique index is placed in restrictive status, it results in an outage to the table.

DB2 12 enables ALTER COLUMN statements that change the data type, length, precision, or scale of columns in pending alterations.

For more information about columns be pending alteration, see 9.3, “Column level deferred alter (pending alter column)” on page 154.

4.2 Catalog availability improvements

This section describes the following catalog availability improvements in DB2 12:

- ▶ Handling dynamic SQL statement
- ▶ Single phase catalog migration

4.2.1 Handling dynamic SQL statement

When a transaction issues dynamic SQL statements, DB2 dynamically prepares the SQL statements for execution. During preparation of dynamic SQL, DB2 acquires read claims on several catalog table spaces and related indexes, and acquired a DBD lock on the catalog. The DBD lock is needed to serialize catalog operations with CATMAINT and other DDL that can execute against the catalog.

In previous releases of DB2, the transaction released the DBD lock and the read claims at commit points. If transactions with dynamic SQL statements did not issue commit operations for a long period of time, CATMAINT and online REORG on the catalog were blocked during that long period.

DB2 12 manages releases DBD locks on the catalog and read claims against catalog objects, as soon as PREPARE statement execution is complete, so this change improves availability for CATMAINT utility and online REORG on catalog objects.

4.2.2 Single phase catalog migration

In DB2 11, the catalog is converted to DB2 11 format during migration to conversion mode. Then, it is converted again during the enable new function mode. Each time the catalog is updated, that can impact applications that access the catalog and other internal DB2 activities such as real-time statistics, which also updates certain catalog tables. These contentions can result in an unavailable resource condition and failed application or catalog update processes.

DB2 12 improves availability on the catalog resource by having only a single-phase migration. There is only one CATMAINT job to convert the catalog to DB2 12 level. The V12R1M500 catalog level can be used for new function activation (to get to new-function mode) and handled by the coexistence of fallback DB2 11 also. For more information, see Chapter 12, “Installation and migration” on page 199.

4.3 Removal of point-in-time recovery restrictions for PBG table spaces

In DB2 12, the following restrictions were removed for point-in-time (PIT) recovery for partition-by-growth (PBG) table spaces:

- ▶ Alteration of SEGSIZE
- ▶ Alteration of DSSIZE
- ▶ Alteration of Buffer Pool
- ▶ Alteration of MEMBER CLUSTER

With these restrictions removed, DB2 enables the data to be available for recovery even after any of those alterations were performed. In this way, running an additional REORG to materialize the data and then recover to the required recovery point is unnecessary.

More information is described in 11.1.5, “Point-in-time recovery enhancements” on page 183.

4.4 PBR RPN DSSIZE increase

DB2 12 introduces a solution for the scenario described above described in 4.3, “Removal of point-in-time recovery restrictions for PBG table spaces” on page 46. A new type of partition-by-range (PBR) structure called partition-by-range relative page numbering (PBR RPN) allows DSSIZE to grow up to 1 TB for a partition. Also, the maximum table size has increased from 16 TB (4K page) to 4 PB, and designed to grow even larger.

PBR RPN improves application availability when is necessary to change the DSSIZE because up to DB2 11 a REORG execution was required and now an immediate ALTER is allowed, this way, not preventing the access to the related objects.

For more information about PBR RPN structure characteristics and more considerations, see Chapter 3, “Scalability” on page 33.

4.5 Insert partition

DB2 12 allows a partition be added in the middle of the table dynamically through the ALTER statement. With this enhancement, the availability of the objects is much better because the objects do not have to be dropped, re-created, and populated with data.

For more information, see Chapter 9, “Administrator function” on page 131.

4.6 REORG enhancements for PBGs, FlashCopy and LOBs

This section describes the following REORG enhancements:

- ▶ Partition-by-growth (PBG)
- ▶ FlashCopy
- ▶ Large object (LOB)

4.8 Faster point-in-time recovery

Point-in-time recovery can run faster with the following enhancements that are described in this section, improving the availability of the related objects in the recovery:

- ▶ Single object by defaulting to the PARALLEL(1) option
- ▶ SCOPE UPDATED keyword

4.8.1 Single object by defaulting to the PARALLEL(1) option

This option indicates that the RECOVER utility should perform parallel restoring of image copies when processing multiple objects. However, the PARALLEL(1) option can also improve performance for recovery of a single object. In DB2 12, the RECOVER utility defaults to use the PARALLEL(1) option even for recovery of a single object, thereby improving the performance of data recovery.

4.8.2 SCOPE UPDATED keyword

DB2 12 introduces the SCOPE UPDATED keyword, applied when the RECOVER utility uses the TORBA option or the TOLOGPOINT option. As a result, the RECOVER utility runs faster because the objects that are specified in LISTDEFlist that have not changed since the recovery point are not recovered. DB2 12 does not waste time recovering unnecessary data sets.

For more information about SCOPE UPDATE keyword, see Chapter 11, “Utilities” on page 177.

4.9 TRANSFER OWNERSHIP SQL statement

DB2 12 introduces the TRANSFER OWNERSHIP SQL statement, providing support for changing ownership of an object while keeping the object available. Up through DB2 11, the object must be dropped and re-created, which affects availability.

More detailed information related to TRANSFER OWNERSHIP SQL statement is in Chapter 10, “Security” on page 167.

4.10 Auto-retry of GRECP and LPL recovery

DB2 12 implements retry logic for the logical page list (LPL) and group buffer pool recovery pending (GRECP) recovery works. This logic is applied when automatic recovery of GRECP and LPL fail, so the object becomes available faster.

For more information about how auto-retry of GREPC and LPL recovery works, see Chapter 5, “Data sharing” on page 49.



Data sharing

DB2 data sharing can provide the following advantages over other database architectures:

- ▶ Separate, independent DB2 systems
- ▶ Improved DB2 availability during both planned and unplanned outages
- ▶ Increased scalability because you are not bound by the limits of a single DB2 system
- ▶ Greater flexibility when configuring systems

These advantages and an overview of the operational aspects of data sharing are described in detail in *DB2 12 for z/OS Data Sharing: Planning and Administration*, SC27-8849.

DB2 12 for z/OS offers a number of data sharing enhancements to provide improved availability, scalability, and performance:

- ▶ DDF shared session data across the data sharing group.
- ▶ In-memory indexes can reduce the number of Get Page requests and group buffer pool requests.
- ▶ Improved insert space search can avoid P-lock contention and streamline inserts.
- ▶ UNLOAD ISOLATION(UR) utility avoids coupling facility page registration.

This chapter describes the following enhancements to data sharing:

- ▶ DISPLAY GROUP command
- ▶ XA support for global transactions
- ▶ Peer recovery
- ▶ Automatic retry of GRECP and LPL recovery
- ▶ Improved lock avoidance checking
- ▶ Asynchronous lock duplexing

5.1 DISPLAY GROUP command

DB2 12 for z/OS introduces the concept of *continuous delivery* where enhancements are delivered continuously in the maintenance stream as they are available. Starting at release migration or new installation, a function level is used to identify the set of enhancements and capabilities available on the system. Similar to DB2 11 for z/OS where compatibility mode and new function mode are available modes for a data sharing group, a group with DB2 12 members can have the corresponding function level V12R1M100 or V12R1M500 respectively (or 100 and 500 for short). In DB2 12 for z/OS, the **-DIS GROUP** command no longer shows the mode. Instead, its output is changed to show the following function levels, which belong to the entire data sharing group (not individual member's property):

- ▶ Current function level

This is the *currently* active function level for the data sharing group. The **-ACTIVATE** command is used to change the current function level.

- ▶ Previously activated function level

This is the function level that was *last activated* before the current function level. The previous activate function level might be higher than or level with the current function level. If a previously activated function level is higher than the current function level, then the current function level can be displayed with an asterisk (*) at the end, which is also called star function level (similar to the star modes CM*, ENFM* in DB2 11 for z/OS).

- ▶ Highest activated function level

This is the *highest* function level that is activated so far. The highest activated function level might be higher than or level with the current function level. If the highest activated function level is higher than the current function level, then the current function level can be displayed with an asterisk (*) at the end (as mentioned, this is similar to the star modes CM*, ENFM* in DB2 11 for z/OS).

Other useful information about the **-DIS GROUP** command is each DB2 member's code level. The code level is shown with each member's subsystem name, member ID, command prefix, and so on. The code level is the applied PTF that provides the capability on each member. All members must have at least the equivalent code level to the function level that is specified on the **-ACTIVATE** command. The individual member's code level is shown in the format VVRMMM where VV is version, R is release, and MMM is the modification level. This differs from DB2 11 for z/OS where the DB2 code level is three characters in the format VVR where VV is version, and R is release.

For more information on the **-DISPLAY GROUP** command changes, see Chapter 2, "Continuous delivery" on page 7.

5.2 XA support for global transactions

To use XA support with client applications communicating to a DB2 data sharing group, the user must set up dynamic virtual IP addresses. A dynamic IP address must be configured for the DB2 group, and one IP address must be set up for each DB2 member in the group. Prior to DB2 12, DB2 was restricted in terms of its transaction architecture. For example, multiple XA transactions using the same transaction ID (XID) or multiple branches of a global transaction may not share resources (locks) in a data sharing environment. When different XA resources on the same global XID connect to different members of the data sharing group (either intentionally or by Workload Manger), the queries running on the global transaction may experience contention against each other and time out. The same condition can occur

when multiple XA transactions with different branches of the same global transaction connect to different DB2 members with Sysplex Workload Management enabled. This is because the DB2 server threads on different members are not able to share locks.

DB2 12 improves that situation where multiple connections on the same global transaction (global transaction ID and format ID but branch qualifier IDs are different) are serviced by different members of the data sharing group. A member is considered the owner of the global transaction when that member is the first member connected to by the first XA resource using an XA connection. That member writes an entry in the SCA structure with the global transaction ID, format ID, the owning member's IP address and port. Then when a subsequent XA resource using an XA connection to connect to another member with the same global transaction, that other member queries the SCA structure to determine the owning member. A DRDA connect request is built and routed to the owning member using its IP address and port. In this way, the subsequent queries actually run on the same member and avoid a lock issue when branches of the global transaction connect to different members.

Figure 5-1 shows the interaction between two XA connections in the same global transaction and one is being rerouted.

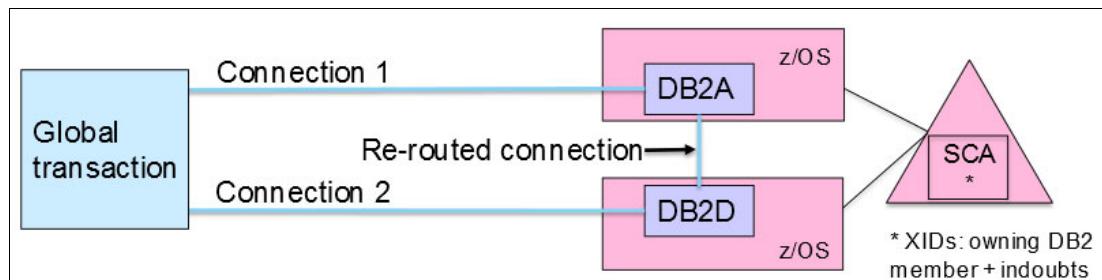


Figure 5-1 XA global transaction

Example 5-1 shows some Java code snippet that builds different branches of the same global transaction connecting to different DB2 data sharing members and issuing UPDATE statements on the same table.

Example 5-1 Java code with multiple branches of the same global transaction executing UPDATE on different members

```

/* Initialize data source members */
com.ibm.db2.jcc.DB2XADataSource dataSource1 = new com.ibm.db2.jcc.DB2XADataSource();
com.ibm.db2.jcc.DB2XADataSource dataSource2 = new com.ibm.db2.jcc.DB2XADataSource();
dataSource1.setDatabaseName("STLEC1");
    dataSource1.setServerName("9.30.85.33");           // 1st member's IP address
    dataSource1.setPortNumber(446);
dataSource1.setClientApplcompat("V12R1");           // set DB2 12 DDF new function
    dataSource2.setDatabaseName("STLEC1");
    dataSource2.setServerName("9.30.85.36");           // 2nd member's IP address
    dataSource2.setPortNumber(446);
dataSource2.setClientApplcompat("V12R1");           // set DB2 12 DDF new function
/* XA Transaction */
javax.sql.XAConnection xaCon1 = null;
javax.sql.XAConnection xaCon2 = null;
    java.sql.Connection con1 = null;
    java.sql.Connection con2 = null;

/* Get XA Connections from data source members */
    xaCon1 = dataSource1.getXAConnection();
xaCon2 = dataSource2.getXAConnection();

```

```

/* Get XA Resources from XA Connections */
javax.transaction.xa.XAResource xaR1 = xaCon1.getXAResource();
    javax.transaction.xa.XAResource xaR2 = xaCon2.getXAResource();

/* Get Connections from XA Connections */
    con1 = xaCon1.getConnection();
con2 = xaCon2.getConnection();

/* Get Statements from Connections */
    java.sql.Statement s1 = con1.createStatement();
    java.sql.Statement s2 = con2.createStatement();

/* Assuming bid1 and bid2 are different branch values, the following code */
/* generates 2 XID's: xid1 and xid2 using the same format ID and global ID */
com.ibm.db2.jcc.DB2Xid xid1 = new com.ibm.db2.jcc.DB2Xid(fid, gid, bid1);
com.ibm.db2.jcc.DB2Xid xid2 = new com.ibm.db2.jcc.DB2Xid(fid, gid, bid2);

/* Begin transactions by invoking the start method with the TMLCS parm */
/* requesting that loosely coupled transactions are able to share locks. */
    xaR1.start(xid1, com.ibm.db2.jcc.DB2XAResource.TMLCS);
    xaR2.start(xid2, com.ibm.db2.jcc.DB2XAResource.TMLCS);
    /* Both UPDATE statements below run on the same member with IP */
        /* address of 9.30.85.33. Though the 2nd UPDATE is first routed to */
        /* member with IP address of 9.30.85.36, it is routed to owning */
        /* whose IP address is 9.30.85.33. */
s1.execute("UPDATE TEMP SET ID = 123456789 WHERE ID = 159357");
s2.execute("UPDATE TEMP SET ID = 789456123 WHERE ID = 753951");

xaR1.end(xid1, javax.transaction.xa.XAResource.TMSUCCESS);
xaR2.end(xid2, javax.transaction.xa.XAResource.TMSUCCESS);
/* All connections with different branch qualifiers need to separately prepare */
xaR1.prepare(xid1);
xaR2.prepare(xid2);
/* All connections with different branch qualifier need to separately commit */
xaR1.commit(xid1, false);
xaR2.commit(xid2, false);

```

5.3 Peer recovery

Since the data sharing introduction in DB2 Version 4, customers have been using Automatic Restart Manager (ARM) or other external mechanisms to restart a member automatically when DB2 or the LPAR that DB2 is on fails. Many data sharing customers want DB2 to be able to restart another peer DB2 member in the same data sharing group in the event that a member fails. With this feature, customers no longer need to implement an external mechanism to perform the automatically recovery process for a failed DB2 and would address the DB2's goal for reliability.

DB2 12 introduces an enhancement to restart a peer member automatically for retained lock recovery in case of LPAR failures without requiring ARM or external automation. A new DSNZPARM is introduced, PEER_RECOVERY, to specify whether this data sharing member is to participate in data sharing peer recovery.

These are the acceptable values:

- ▶ NONE
This member is not involved in the peer recovery process. NONE is the default value.
- ▶ RECOVER
This member should be recovered by a peer member in case this member fails.
- ▶ ASSIST
This member is to assist to recovery another peer.
- ▶ BOTH
This is the combination of RECOVER and ASSIST options.

PEER_RECOVERY DSNZPARM can be specified on the installation panel DSNTIPK and is online-changeable.

Upon receiving the notification from z/OS that a peer member with a PEER_RECOVERY setting of RECOVER or BOTH has failed; all surviving members that have the PEER_RECOVERY setting of ASSIST or BOTH serializes the recovery process by obtaining a global lock. The first assisting member having the lock attempts to restart the failed peer DB2 by issuing the **START DB2** command with the **LIGHT(YES)** option. The last DSNZPARM load module used by the failed peer is also used to automatically restart it. Light restart is requested because the goal is to handle retained locks only.

If this light restart results in a failure, another assisting member can retry the process to recover the same instance of the failed member. When all members have tried and the failed member still cannot be recovered, a manual intervention is needed to restart that DB2.

5.4 Automatic retry of GRECP and LPL recovery

When a group buffer pool structure fails before data in the structure is written into DASD, the group buffer pool is put in the group buffer pool recovery pending (GRECP) state. The **START DATABASE** command must be issued to recover the data. DB2 also attempts to recover the GRECP state as soon as it occurs.

The logical page list (LPL) contains a list of pages (or a page range) that cannot be read or written for reasons such as the transient disk read and write problems, which can be fixed without redefining new disk tracks or volumes.

Specific to data sharing, the LPL also contains pages that cannot be read or written for “must-complete” operations, such as a commit or a restart, because of a problem with the coupling facility. For example, pages can be added if a channel failure occurs to the coupling facility or disk, or if locks are held by a failed subsystem, disallowing access to the needed page.

As soon as the pages are added to the logical page list, an automatic LPL recovery is attempted by DB2. Automatic LPL recovery is also performed when the user issues the **START DATABASE** command with **ACCESS(RW)** or **ACCESS(RO)** option for the table space, index, or partition.

Automatic recovery of GRECP and LPL was introduced in DB2 9. However, if these automatic recovery processes fail for some reason, then the object in the GRECP or LPL status remains as is, and the user must reinitiate the recovery process by issuing the **START DATABASE** command.

To improve availability, DB2 12 implements the retry logic for the GRECP and LPL recovery processes. This retry is initiated at three-minute intervals.

5.5 Improved lock avoidance checking

DB2 keeps track of the oldest write claim on a page set/partition and stores it as the system-level commit log record sequence number (LRSN). In a data sharing environment, for insert applications, the system-level commit LRSN is the key to better lock avoidance. To take the best advantage of avoiding locks or reusing deleted space, all applications that access data concurrently should issue COMMIT statements frequently. However, some applications cannot issue COMMIT often because application logic causes the system-level commit and read LRSN to remain old even though the majority of objects are already committed. This can cause other threads running other applications to experience an increase of lock requests because DB2 cannot exploit the lock avoidance logic.

In some cases, DB2 also keeps track of the read interest on a page set/partition and stores it as the system-level read LRSN. Read LRSN is used by DB2 space management scheme to determine if a deleted LOB space can be reused. In certain cases, with DB2 10, a deleted LOB space is not reused effectively and causes the LOB space to grow. You can define LOBs as inline LOBs to avoid the LOB space reuse problem, but this workaround leads to another problem which is the base table space reuse problem. You must then run **REORG** to reclaim the unused space.

DB2 12 improves both performance and space issues by providing greater granularity for commit LRSN and read LRSN. These values are kept at the object level (page set or partition) in each DB2 member's memory. Each member can track a maximum of 500 object-level commit and read LRSN values. These are the oldest values for each object and are advanced at commit. When a page is accessed, the page's LRSN will be compared to the object-level LRSN to determine whether a lock is needed. When the object is not in the track list, the highest LRSN value from the list will be used for comparison. This technique is also used in non-data sharing. In data sharing, each member also needs a global view of which other members are also using this object. Therefore, the object-level commit and read LRSN values are also stored in the SCA structure for each member.

5.6 Asynchronous lock duplexing

Up through DB2 11, the DB2 lock structure managed by IMS Resource Lock Manager (IRLM) can be set up with system-managed duplexing for availability in case a coupling facility (CF) failure occurs. This feature is available with CF Level 11. System-managed duplexing for any coupling facility structure requires every structure update to occur in both primary and secondary structures at the same time. Two identical commands are sent to both structures with sequence numbers to be executed in parallel, and both must complete successfully before the update request is returned to IRLM as the exploiter. The drawback with this synchronous approach is performance overhead when the two structures are located further away from each other (such as distance greater than 10 kilometers). This overhead can be visible to a DB2 transaction making updates to the database and waiting for lock requests (both primary and secondary structure updates) to be completed.

With DB2 12, the new feature, asynchronous duplexing for lock structures, is introduced to address the performance issue while maintaining the availability advantage. Figure 5-2 shows the 2 DB2 data centers with larger distance and the system-managed duplexing group buffer pools, SCA and lock structures.

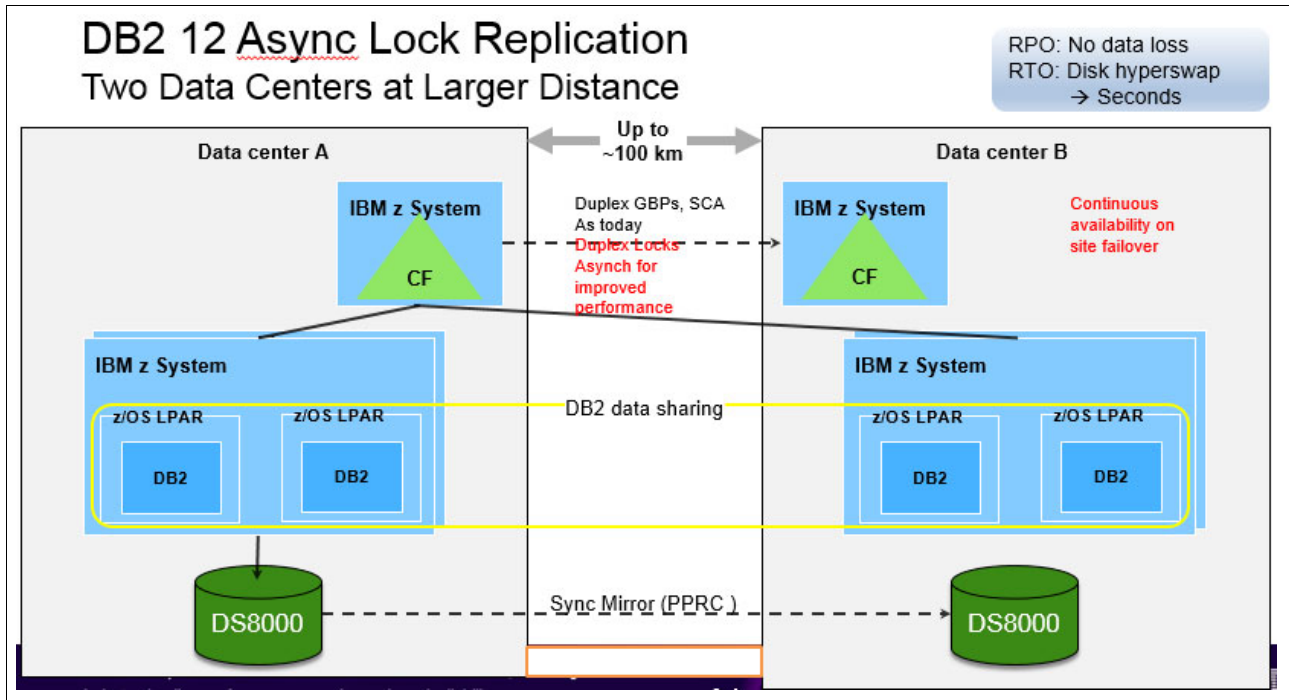


Figure 5-2 Data centers for system-managed duplexing structures

This feature, which uses the z/OS asynchronous duplexing support for lock structures in the coupling facility, has these requirements:

- ▶ CF Level 21 with service level 02.16
- ▶ z/OS V2.2 SPE with PTFs for APAR OA47796
- ▶ DB2 12 with PTFs for APAR PI66689
- ▶ IRLM 2.3 with PTFs for APAR PI68378

With asynchronous duplexing, the lock requests from a DB2 transaction will be sent to IRLM and the z/OS cross-system extended services (XES) to the primary lock structure only. XES will return a sequence number to IRLM and DB2 as soon as the primary structure's update is done. Meanwhile a request to update the secondary structure is sent by XES in the background. Each thread keeps track of the secondary structure's update sequence number and the oldest is saved at the DB2 member level. When a forced log write is needed for the transaction (such as at COMMIT), DB2 and IRLM will check with XES that the oldest sequence number's request has been successfully written to the secondary structure. Most of the time, the secondary structure that was updated should have been completed because continuous requests have been submitted to it in the background. If not, DB2 log write process is suspended until updates to the secondary structure are done, and this suspend time is accounted in the DB2 log write suspend time.

This approach usually has performance result similar to the simplex lock structure because there is no waiting for dual structure updates in the mainline request to the primary structure. The continuous, asynchronous background updates to secondary structure ensures completion in a timely manner and achieves the duplexing functionality for availability in the case when the primary structure fails.

To exploit asynchronous duplexing, the CFRM couple data set (CDS) must be formatted with new **ASYNCDUPLEX** keyword, and the CFRM policy for DB2 lock structure must be updated with new **ASYNCONLY** keyword. After this CDS is in effect, a simple z/OS **SET XCF,REALLOCATE** command can start rebuilding the lock structure with asynchronous duplexing protocol, assuming all DB2 and IRLM members in the data sharing group have joined with the appropriate code level PTFs.

The z/OS **DISPLAY XCF,STR** command support can be used to check structure status with regard to the CFRM policy for duplexing (structures that are duplexed, or are in the process of establishing duplexing, or are in the process of falling out of duplexing).

Figure 5-3 shows the z/OS **DISPLAY XCF** command output, which displays the asynchronous duplexing status for the lock structure.

```

D XCF,STRUCTURE,STRNAME=DSNCAT_LOCK1
IXC360I 14.16.48 DISPLAY XCF 644
STRNAME: DSNCAT_LOCK1
STATUS: REASON SPECIFIED WITH REBUILD START:
POLICY-INITIATED
DUPLEXING REBUILD
METHOD: SYSTEM-MANAGED
AUTO VERSION: D15B6C28 08153502
PHASE: ASYNC DUPLEX ESTABLISHED
EVENT MANAGEMENT: MESSAGE-BASED
TYPE: LOCK
POLICY INFORMATION:
POLICY SIZE : 18 M
POLICY INITSIZE: 16 M
POLICY MINSIZE : 12 M
FULLTHRESHOLD : 90
ALLOWAUTOALT : YES
REBUILD PERCENT: 1
DUPLEX : ENABLED ASYNCONLY
ALLOWREALLOCATE: YES
PREFERENCE LIST: LF01 CACHE01
ENFORCEORDER : YES
EXCLUSION LIST IS EMPTY

DUPLEXING REBUILD NEW STRUCTURE
  
```

Figure 5-3 The z/OS **DISPLAY XCF** command output

Upon restarting DB2 12, DB2 identifies to IRLM and if IRLM has the PTF for APAR PI68378, IRLM will request asynchronous duplexing protocol to XES. When the lock structure is reallocated with the CFRM couple data set and CFRM policy indicates asynchronous duplexing for the IRLM lock structure, the rebuild process begins as shown in the figure. From this point on, the lock structure is allocated with this protocol and a subsequent DB2 IRLM member restarting with the PTF for APAR PI68378 is needed. Otherwise (such as restarting of a coexisting DB2 11 member), XES will automatically rebuild the IRLM lock structure with simplex mode.

The z/OS messages listed in Example 5-2 will be issued to show that duplexing is no longer active.

Example 5-2 Lock structure is rebuilt with simplex mode

```
IXC522I SYSTEM-MANAGED DUPLEXING REBUILD FOR STRUCTURE 423
  DSNCAT_LOCK1 IS BEING STOPPED
  TO FALL BACK TO THE OLD STRUCTURE DUE TO
  DUPLEXING PREVENTING A CHANGE IN THE SET OF CONNECTORS
  SYSTEM CODE: 00801000
IXC571I SYSTEM-MANAGED DUPLEXING REBUILD FOR STRUCTURE 424
  DSNCAT_LOCK1 HAS COMPLETED THE ASYNC DUPLEX ESTABLISHED PHASE
  AND IS ENTERING THE QUIESCE FOR STOP PHASE.
  TIME: 10/04/2016 14:53:05.725002
  AUTO VERSION: D17214AE 13530010
IXC571I SYSTEM-MANAGED DUPLEXING REBUILD FOR STRUCTURE 425
  DSNCAT_LOCK1 HAS COMPLETED THE QUIESCE FOR STOP PHASE
  AND IS ENTERING THE STOP PHASE.
  TIME: 10/04/2016 14:53:05.865551
  AUTO VERSION: D17214AE 13530010
IXC577I SYSTEM-MANAGED DUPLEXING REBUILD HAS 426
  BEEN STOPPED FOR STRUCTURE DSNCAT_LOCK1
  STRUCTURE NOW IN COUPLING FACILITY LF01
  PHYSICAL STRUCTURE VERSION: D17214AE 05DDB390
  LOGICAL STRUCTURE VERSION: D17214AE 05DDB390
  AUTO VERSION: D17214AE 13530010
```

When the coexisting DB2 11 is stopped, XES will rebuild the lock structure back to asynchronous duplexing mode because all members are in DB2 12 function level V12R1M100. Example 5-3 lists messages that are issued to show reduplexing.

Example 5-3 Lock structure is rebuilt to asynchronous duplexing mode

```
IXC536I DUPLEXING REBUILD OF STRUCTURE DSNCAT_LOCK1 506
  INITIATED.
  REASON: CONNECTOR DISCONNECTED FROM STRUCTURE
  IXC570I SYSTEM-MANAGED DUPLEXING REBUILD STARTED FOR STRUCTURE 507
  DSNCAT_LOCK1 IN COUPLING FACILITY LF01
  IXC577I SYSTEM-MANAGED DUPLEXING REBUILD HAS 523
  ESTABLISHED ASYNC DUPLEXING FOR STRUCTURE DSNCAT_LOCK1
  STRUCTURE IS DUPLEXED
```



Part 3

Application functions

This part contains the following chapters:

- ▶ Chapter 6, “SQL” on page 61
- ▶ Chapter 7, “Application enablement ” on page 99
- ▶ Chapter 8, “Connectivity and administration routines ” on page 119



SQL

In this chapter, SQL-related functionality introduced in DB2 12 for z/OS is described. Many examples are provided to help demonstrate how the new features work. The examples can help you in adapting and using the new features in your applications.

This chapter covers the following topics:

- ▶ Introduction
- ▶ Additional support for triggers
- ▶ Pagination support
- ▶ Additional support for arrays
- ▶ MERGE statement enhancements
- ▶ New built-in functions
- ▶ Enhanced built-in function support

Many SQL changes are available only when new function is activated and with an application compatibility value of 'V12R1M500' or greater. For a description of new function activation, see Chapter 2, “Continuous delivery” on page 7. For a summary of which SQL changes are available with new function activation, see the “What’s new in the initial DB2 12 release” section of *DB2 12 for z/OS What’s New?*, GC27-8861.

Note: Several SQL examples described here are in Appendix B, “Additional material” on page 279. You can use the SQL examples in Appendix as templates to create your own applications, and as a learning tool to understand DB2 functionality.

6.1 Introduction

This section describes several SQL changes introduced or modified in DB2 12 for z/OS. For a complete description of each item, see *DB2 12 for z/OS SQL Reference*, SC27-8859.

Table 6-1 summarizes the set of SQL statement changes, new and modified, introduced in DB2 12 for z/OS.

Table 6-1 Summary of SQL statement enhancements in DB2 12 for z/OS

SQL statement	Description
ALTER FUNCTION (compiled SQL scalar)	New clause: ▶ CONCENTRATE STATEMENTS
ALTER INDEX	New clause: ▶ DSSIZE Changed clause: ▶ COMPRESS
ALTER PROCEDURE (SQL native)	New clause: ▶ CONCENTRATE STATEMENTS
ALTER TABLE	New clauses: ▶ CCSID ▶ EXCLUSIVE and INCLUSIVE for BUSINESS_TIME period-definition ▶ PERIOD BUSINESS_TIME clause for referential-constraints Changed clauses: ▶ ADD PERIOD FOR ▶ ADD PARTITION
ALTER TABLESPACE	New clauses: ▶ PAGENUM ▶ INSERT ALGORITHM Changed clauses: ▶ COMPRESS ▶ DSSIZE
ALTER TRIGGER (advanced)	New statement that allows for the altering of triggers that contain SQL PL (SQL Procedural Language).
ALTER TRIGGER (basic)	Was ALTER TRIGGER in prior releases
COMMENT	Changed clause: ▶ TRIGGER trigger-name VERSION trigger-version-id
CREATE FUNCTION (compiled SQL scalar)	New clauses: ▶ CONCENTRATE STATEMENTS ▶ WRAPPED
CREATE FUNCTION (inline SQL scalar)	New clause: ▶ WRAPPED
CREATE FUNCTION (SQL table)	New clause: ▶ WRAPPED
CREATE INDEX	New clause: ▶ DSSIZE

SQL statement	Description
CREATE PROCEDURE (SQL native)	New clauses: <ul style="list-style-type: none"> ▶ CONCENTRATE STATEMENTS ▶ WRAPPED
CREATE TABLE	New clauses: <ul style="list-style-type: none"> ▶ CCSID on a CHAR, GRAPHIC, CLOB, or DBCLOB column ▶ EXCLUSIVE and INCLUSIVE clauses for PERIOD BUSINESS_TIME clause ▶ PAGENUM ▶ PERIOD BUSINESS_TIME clause for referential-constraints Changed clauses: <ul style="list-style-type: none"> ▶ DSSIZE ▶ NUMPARTS
CREATE TRIGGER (advanced)	New statement that allows for the creation of triggers that contain SQL PL (SQL Procedural Language).
CREATE TRIGGER (basic)	New clause: <ul style="list-style-type: none"> ▶ WRAPPED
CREATE VARIABLE	Changed clause: <ul style="list-style-type: none"> ▶ data-type
DELETE	New clauses: <ul style="list-style-type: none"> ▶ BETWEEN value-1 AND value-2 clause of the period-clause ▶ fetch-clause
EXECUTE	Changed clause: <ul style="list-style-type: none"> ▶ USING
EXECUTE IMMEDIATE	Changed clause: <ul style="list-style-type: none"> ▶ variable
EXPLAIN	New clause: <ul style="list-style-type: none"> ▶ STABILIZED DYNAMIC QUERY STMTID
fullselect	New clause: <ul style="list-style-type: none"> ▶ offset-clause Changed clauses: <ul style="list-style-type: none"> ▶ fetch-clause ▶ order-by-clause
GRANT (table or view privileges)	New clause: <ul style="list-style-type: none"> ▶ UNLOAD
MERGE	New clauses: <ul style="list-style-type: none"> ▶ signal-statement ▶ ELSE IGNORE ▶ delete-operation on modification-operation Changed clauses: <ul style="list-style-type: none"> ▶ AS correlation-name ▶ assignment-clause ▶ WHEN matching-condition ▶ THEN modification-operation
OPEN	Changed clause: <ul style="list-style-type: none"> ▶ USING

SQL statement	Description
PREPARE	New clause: ▶ offset-clause Changed clause: ▶ fetch-clause
REVOKE (table or view privileges)	New clause: ▶ UNLOAD
SELECT INTO	New clause: ▶ offset-clause Changed clauses: ▶ INTO ▶ fetch-clause
SET assignment-statement	Changed clause: DEFAULT
subselect	New clause: ▶ offset-clause Changed clauses: ▶ correlation-clause of collection-derived-table in a table-reference of a FROM clause ▶ fetch-clause
TRANSFER OWNERSHIP	New statement that allows for the transferal of ownership, of certain objects, from one user to another user.
UPDATE	New clause: ▶ BETWEEN value-1 AND value-2 clause of the period-clause
VALUES INTO	Changed clauses: ▶ assignment clause source ▶ INTO
compound-statement for SQL routines	New clause: ▶ ATOMIC Changed clause: ▶ DEFAULT or CONSTANT

6.2 Additional support for triggers

Triggers are a set of SQL statements that represents the business logic that should occur when certain data modifications occur. The data modification that activates the trigger can be an insert, update, or delete operation. The target of the data modification may be a table or a view. When the trigger is activated by the data modification, the set of SQL statements specified inside the trigger body will be executed. Starting in DB2 12 for z/OS, triggers can be written in SQL Procedural Language (SQL PL). If a trigger is written with SQL PL, it is known as an advanced trigger.

SQL Procedural Language (SQL PL)

SQL PL is a subset of SQL that can be used to implement control logic around traditional SQL statements. DB2 for z/OS supports the following control statements:

- ▶ assignment-statement
- ▶ CALL statement
- ▶ CASE statement
- ▶ compound-statement
- ▶ FOR statement
- ▶ GET DIAGNOSTICS statement
- ▶ GOTO statement
- ▶ IF statement
- ▶ ITERATE statement
- ▶ LEAVE statement
- ▶ LOOP statement
- ▶ REPEAT statement
- ▶ RESIGNAL statement
- ▶ RETURN statement
- ▶ SIGNAL statement
- ▶ WHILE statement

For detailed information about the control statements, see the “SQL control statements for SQL routines and triggers” section in the *DB2 12 for z/OS SQL Reference*, SC27-8859.

6.2.1 Basic triggers

Prior to DB2 12, the CREATE TRIGGER statement was used to define a basic trigger. Starting in DB2 12, the CREATE TRIGGER statement is renamed to the CREATE TRIGGER (basic) statement, and as such, defines a basic trigger. The MODE DB2SQL clause continues to be required in order to create a basic trigger. Basic triggers created prior to DB2 12 can still be used on DB2 12 without modification.

6.2.2 Advanced triggers

Starting in DB2 12, the CREATE TRIGGER (advanced) statement defines an advanced trigger. The body of an advanced trigger can contain traditional SQL as well as SQL PL. The set of SQL statements supported within an advanced trigger is more extensive than for basic triggers. For example, dynamic SQL statements can be used within an advanced trigger. Additionally, with advanced triggers, multiple versions of an advanced trigger can be defined, similar to how multiple versions of an SQL PL routine can be defined. Advanced triggers also have debugging support, similar to SQL PL routines.

Because advanced triggers use SQL PL, they have several characteristics that are similar to SQL PL routines. For instance, with an advanced trigger, you can specify various options through the option-list on the CREATE TRIGGER (advanced) statement and ALTER TRIGGER (advanced) statement. All variables, including transition variables, are nullable. A larger set of SQL statements are supported in an advanced trigger compared to basic triggers.

Advanced triggers can be created only when new function is activated and with an application compatibility setting of 'V12R1M500' or greater. The MODE DB2SQL clause may not be specified for an advanced trigger.

6.2.3 Differences between basic triggers and advanced triggers

Catalog differences and behavioral differences exist between basic and advanced triggers.

Catalog information

The SYSIBM.SYSPACKAGE and SYSIBM.SYSTRIGGERS catalog tables provide information about whether or not a trigger is a basic trigger or an advanced trigger. The SYSIBM.SYSPACKAGE catalog table TYPE and VERSION columns and the SYSIBM.SYSTRIGGERS catalog table SQLPL, DEBUG_MODE, and VERSION columns contain the relevant information.

For example, suppose a table MYEMP exists that is like the DSN8C10.EMP sample table and a basic trigger was created to calculate an employee's bonus:

```
CREATE TRIGGER BAS_TRG_BONUS
AFTER INSERT ON MYEMP
FOR EACH ROW MODE DB2SQL -- MODE DB2SQL = basic trigger
BEGIN ATOMIC
  UPDATE MYEMP SET BONUS = BONUS + 5000;
END!
```

An advanced trigger was created to calculate an employee's raise:

```
CREATE TRIGGER ADV_TRG_SAL
AFTER INSERT ON MYEMP
FOR EACH ROW          -- no MODE DB2SQL = advanced trigger
BEGIN ATOMIC
  UPDATE MYEMP SET SALARY = SALARY + 1000;
END!
```

For basic triggers, in the SYSIBM.SYSPACKAGE catalog table, the TYPE will have the value 'T' and the VERSION will be an empty string because the VERSION option-list is not supported with basic triggers; the SYSIBM.SYSTRIGGERS catalog table will have blanks or the empty string for the SQLPL, DEBUG_MODE, and VERSION columns, respectively. For advanced triggers, the SYSIBM.SYSPACKAGE catalog table TYPE will have the value '1' and the VERSION will contain 'V1', because no version was specified and 'V1' is the default value on the CREATE TRIGGER (advanced) statement; the SYSIBM.SYSTRIGGERS catalog table will have the SQLPL column set to 'Y', the DEBUG_MODE set to '0', because DEBUG_MODE was not specified, and the VERSION column set to 'V1', because no version was specified.

The following select statement can be issued against the SYSIBM.SYSPACKAGE catalog table to view the trigger specific column values for both basic and advanced triggers:

```
SELECT TYPE, VERSION, LENGTH(VERSION) AS LEN_VER, NAME
FROM SYSIBM.SYSPACKAGE
WHERE NAME = 'BAS_TRG_BONUS' OR NAME = 'ADV_TRG_SAL'!
```

That returns the following information:

TYPE	VERSION	LEN_VER	NAME
1	V1	2	ADV_TRG_SAL
T		0	BAS_TRG_BONUS

Issuing a select statement against the SYSIBM.SYSTRIGGERS catalog table shows the trigger specific column values for both basic and advanced triggers:

```
SELECT SQLPL, HEX(SQLPL) AS HEX_SQLPL,
       DEBUG_MODE AS DBG, HEX(DEBUG_MODE) AS HEX_DBG,
       VERSION, LENGTH(VERSION) AS LEN_VER, NAME
FROM SYSIBM. SYSTRIGGERS
WHERE NAME = 'BAS_TRG_BONUS' OR NAME = 'ADV_TRG_SAL'!
```

That returns the following information:

SQLPL	HEX_SQLPL	DBG	HEX_DBG	VERSION	LEN_VER	NAME
Y	59	0	30	V1	2	ADV_TRG_SAL
	20		20		0	BAS_TRG_BONUS

Note: These SQL examples are in Appendix B, “Additional material” on page 279 in the sqlpl_trigger_catalog.sql file.

Issuing CREATE TRIGGER or ALTER TRIGGER statements

The CREATE TRIGGER (basic) statement and ALTER TRIGGER (basic) statement may be embedded in an application program or issued interactively. They are executable statements that can be dynamically prepared but only if the DYNAMICRULES(RUN) behavior is specified, either implicitly or explicitly. By comparison, the CREATE TRIGGER (advanced) statement and ALTER TRIGGER (advanced) statement can only be dynamically prepared and the DYNAMICRULES(RUN) behavior must be in effect.

Authorization requirements

For a *basic* trigger, the privilege set must include SYSADM authority, or if the REFERENCING clause is specified, the SELECT privilege on the table or view on which the trigger is defined.

For an *advanced* trigger, the privilege set must include the SYSADM authority, or the SELECT privilege on the table or view on which the trigger is defined, regardless of whether the REFERENCING clause is specified.

Default encoding scheme

The default encoding scheme for a basic trigger is Unicode. The default encoding scheme for an advanced trigger is determined from the DEFAULT APPLICATION ENCODING SCHEME field on the DSNTIPF installation panel. On an advanced trigger, the APPLICATION ENCODING SCHEME trigger-option on the CREATE TRIGGER (advanced) statement and ALTER TRIGGER (advanced) statement can be used to set the application encoding scheme value to EBCDIC, ASCII, or UNICODE. For advanced triggers to use the same UNICODE encoding scheme as basic triggers, specify the APPLICATION ENCODING SCHEME UNICODE trigger-option on the CREATE TRIGGER (advanced) statement or ALTER TRIGGER (advanced) statement.

Unhandled warnings at the completion of a trigger

For basic triggers, if a warning occurs during execution of the last SQL statement in the basic trigger's body, the warning is not returned to the SQL statement that activated the trigger. By contrast, with an advanced trigger, unhandled warnings are returned to the statement that activated the trigger.

Look at the following example to better understand how unhandled warnings are handled. Once again, suppose a table MYEMP exists that is like the DSN8C10.EMP sample table and a basic trigger was created to modify an employee's job title. Remember that the JOB column in the sample table is defined as a CHAR(8) field:

```
CREATE TRIGGER BAS_TRG_WARN
AFTER INSERT ON MYEMP
FOR EACH ROW MODE DB2SQL -- MODE DB2SQL = basic trigger
BEGIN ATOMIC
    UPDATE MYEMP SET JOB = CAST(JOB AS CHAR(1));
END! -- exclamation point as SQL terminator
```

An advanced trigger was created to modify an employee's work department. Recall, the WORKDEPT column in the sample table is defined as a CHAR(3) field:

```
CREATE TRIGGER ADV_TRG_WARN
AFTER INSERT ON MYEMP
FOR EACH ROW -- no MODE DB2SQL = advanced trigger
BEGIN ATOMIC
    UPDATE MYEMP SET WORKDEPT = CAST(WORKDEPT AS CHAR(1));
END! -- exclamation point as SQL terminator
```

The following INSERT statement activates both triggers. Notice that the JOB value contains a 4-byte character string 'PRES' while the WORKDEPT value contains a 3-byte character string 'A00':

```
INSERT INTO MYEMP
VALUES ('000011', 'CHRISTINE', 'I', 'HAAS', 'A00', 'A1A1',
       '1965-01-01', 'PRES', 18, 'F', '1933-08-14',
       52750, 1000, 4220);
```

Only the advanced trigger returned the warning as noted by the message token 'A00' which corresponds to the WORKDEPT value used on the INSERT statement:

```
SQLCODE = 445, WARNING: VALUE A00 HAS BEEN TRUNCATED
```

Note: The SQL examples are in Appendix B, “Additional material” on page 279 in the sqlpl_trigger_warning.sql file.

Transition variables as procedure OUT and INOUT parameters

For basic triggers, if a stored procedure is invoked from inside the trigger and a transition variable was specified for an OUT or INOUT parameter, the updated transition variable value is not visible to the trigger upon return from the stored procedure. For an AFTER basic trigger, changes to transition variables are allowed. With an advanced trigger, changes to the transition variable, as a result of the transition variable being set from a stored procedure, are visible to the trigger upon return from the stored procedure. For an AFTER advanced trigger, changes to transition variables are not allowed.

Look at an SQL PL stored procedure that sets an output parameter to a specific value, namely 999:

```
CREATE PROCEDURE SP1 (OUT out_value INTEGER)
LANGUAGE SQL
BEGIN
    SET out_value = 999;
END!
```

A basic trigger is created whereby a stored procedure is invoked and the employee's bonus is passed as the OUT parameter:

```
CREATE TRIGGER BAS_TRG_SP
NO CASCADE BEFORE INSERT ON MYEMP
REFERENCING NEW AS new_bonus
FOR EACH ROW MODE DB2SQL -- MODE DB2SQL = basic trigger
BEGIN ATOMIC
  CALL SP1(new_bonus.BONUS);
END!
```

Similarly, an advanced trigger is created to also invoke the stored procedure, passing in the employee's salary as the argument for the OUT parameter:

```
CREATE TRIGGER ADV_TRG_SP
NO CASCADE BEFORE INSERT ON MYEMP
REFERENCING NEW AS new_salary
FOR EACH ROW -- no MODE DB2SQL = advanced trigger
BEGIN ATOMIC
  CALL SP1(new_bonus.SALARY);
END!
```

The following INSERT statement activates both triggers. Notice the BONUS value contains the value 1000 while the SALARY value contains the value 52750.

```
INSERT INTO MYEMP
VALUES ('000011', 'CHRISTINE', 'I', 'HAAS', 'A00', 'A1A1',
       '1965-01-01', 'PRES', 18, 'F', '1933-08-14',
       52750, 1000, 4220);
```

Only the advanced trigger set the OUT parameter as indicated by only the SALARY column having the value of 999.00, the value set by the stored procedure. The BONUS column contains the original source value of 1000.00:

```
SELECT EMPNO, BONUS, SALARY FROM MYEMP;
```

That returns the following information:

```
EMPNO  BONUS    SALARY
-----  -
000011 1000.00  999.00
```

Note: The SQL examples are in Appendix B, “Additional material” on page 279 in the sqlpl_trigger_outparm.sql file.

Stand-alone fullselect and VALUES statement

Basic triggers allow both a fullselect and a VALUES statement to be specified. Advanced triggers do not support either. As an alternative to the VALUES statement in an advanced trigger, you can specify either the SELECT INTO statement or the VALUES INTO statement.

6.2.4 Maintaining trigger activation order

Basic triggers and advanced triggers are alike in that multiple triggers may be created for the same table, view, event, or activation time. The order in which those triggers are activated is the order in which the triggers were created. DB2 records the timestamp when each CREATE TRIGGER statement executes. When an event occurs in a table or view that activates more than one trigger, DB2 uses the stored timestamps to determine which trigger to activate first. DB2 always activates all the BEFORE triggers that are defined on a table or view before the AFTER triggers that are defined on that table or view. Within the set of BEFORE triggers, the activation order is by stored timestamp and within the set of AFTER triggers, the activation order is by stored timestamp.

For example, the following three BEFORE advanced triggers are created against table MYEMP:

```
CREATE TRIGGER MYTRIG1 BEFORE INSERT ON MYEMP VERSION V1 . . . ;
CREATE TRIGGER MYTRIG2 BEFORE INSERT ON MYEMP VERSION V1 . . . ;
CREATE TRIGGER MYTRIG3 BEFORE INSERT ON MYEMP VERSION V1 . . . ;
```

When an INSERT statement is issued against the MYEMP table, MYTRIG1 is executed, then MYTRIG2 is executed, then MYTRIG3 is executed.

Now, suppose you want to modify MYTRIG2 and keep the activation order of all triggers. There are three ways to modify the MYTRIG2 trigger:

- ▶ Issue a **CREATE OR REPLACE** with the same version:

```
CREATE OR REPLACE TRIGGER MYTRIG2 BEFORE INSERT ON MYEMP VERSION V1 . . . ;
```
- ▶ Issue an **ALTER TRIGGER REPLACE VERSION** with the same version:

```
ALTER TRIGGER MYTRIG2 REPLACE VERSION V1 . . . ;
```
- ▶ Issue an **ALTER TRIGGER ADD VERSION** with a different version followed by an **ALTER TRIGGER ACTIVATE VERSION**:

```
ALTER TRIGGER MYTRIG2 ADD VERSION V2 . . . ;
ALTER TRIGGER MYTRIG2 ACTIVATE VERSION V2 . . . ;
```

6.3 Pagination support

With the increase in mobile and web applications, often application developers want, or need, to display only a subset of rows back to the user. Displaying a subset of rows at a time allows the user to “page” through the data. Think of a typical search engine result page that displays N number of rows and then allows the user to “page” to the next set of results. DB2 12 for z/OS introduces two types of pagination support:

- ▶ Data-dependent
- ▶ Numeric-based

In addition, application developers have more flexibility in telling DB2 how many rows total should be returned through the enhanced fetch-first-clause clause in the subselect clause.

6.3.1 Returning a subset of rows

Application developers might want DB2 to return only a certain number of rows. Additionally, application developers might want to have flexibility in telling DB2 how many rows to return. Prior to DB2 12, the fetch-first-clause clause could be used to tell DB2 to return only N number of rows. Starting in DB2 12, the fetch-first-clause clause is renamed to the fetch-clause clause, which allows variables and parameter markers to be specified. Also starting in DB2 12, the value specified for N can be a big integer. The value for N can also be set to 0, which indicates to DB2 that no rows should be returned. Lastly, for readability, application developers can specify NEXT instead of FIRST on the fetch-clause clause.

Figure 6-1 shows the syntax diagram for the fetch-clause clause. For complete information about the clause, see *DB2 12 for z/OS SQL Reference*, SC27-8859.

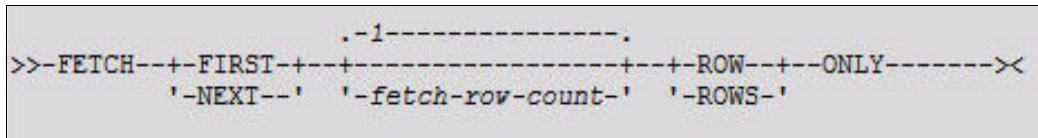


Figure 6-1 Syntax diagram for fetch-clause

6.3.2 Data-dependent pagination support

Application developers code their applications to return relevant data not only to leverage performance benefits from the database but also to minimize the amount of data processed by their applications. One such method that application developers have in their toolbox is to code their SQL with predicates. DB2 12 simplifies applications by allowing the row-value-expression to be specified in a basic predicate along with the less than, less than or equal to, greater than, or greater than or equal to comparison operators (<, <=, >, >=). Prior to DB2 12, only the equal and not equal comparison operators (=, <>) were allowed with the row-value-expression. A row-value-expression returns a single row of data where each column in the row is an expression, expressions can be constants, variables, expressions themselves, and others.

Figure 6-2 shows the basic predicate syntax diagram.

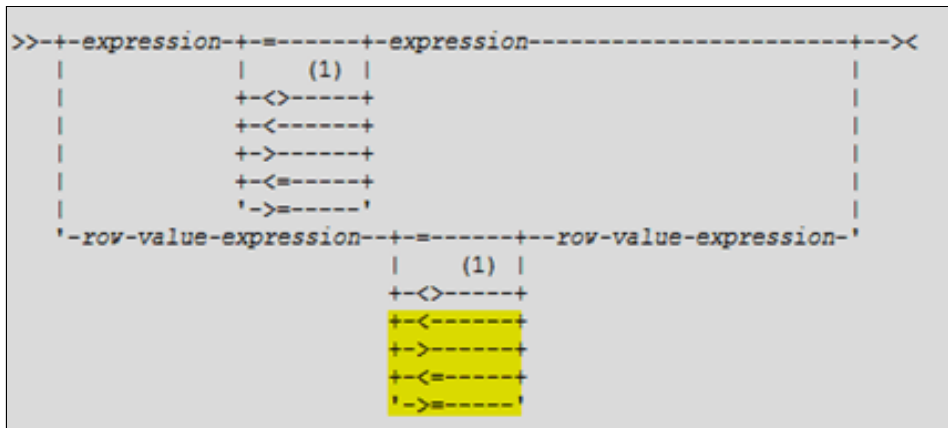


Figure 6-2 Syntax diagram for a basic predicate

Data-dependent pagination features

Data-dependent pagination is a method whereby an application developer can display pages of data at a time. The data returned in the last row of the page is the data used to fetch the next page of data. Application developers code this style of pagination in their applications, returning pages of data until no more rows qualify.

Several key features ensure data-dependent pagination returns the correct and proper data:

- ▶ All of the columns specified in the ORDER BY clause must be in either ascending or descending order.
- ▶ All of the columns specified in the ORDER BY clause must generate a unique value.

You can have multiple rows with the same values but at least one column in that row must distinguish that row from all other rows. For instance, in the phone book, there multiple rows of people have the same last name and first name but the phone number is unique among those rows.

Example 6-1 shows a high-level overview of how a data-dependent application might be coded prior to DB2 12.

Example 6-1 Data-dependent application prior to DB2 12

```
-- determine the number of rows in the table
SELECT COUNT(*) INTO :NUMROWS FROM T1;

-- initialize input variables so all rows qualify
HV1 = 0;
HV2 = 0;
HV3 = '';
-- initialize row counter
ROWCTR = 0;

-- declare a cursor
DECLARE CSR CURSOR FOR
SELECT . . .
FROM T1
WHERE ((COL1 = :HV1 AND COL2 = :HV2 AND COL3 > :HV3) OR
       (COL1 = :HV1 AND COL2 > :HV2) OR
       (COL1 = :HV1)
ORDER BY COL1 ASC, COL2 ASC, COL3 ASC
FETCH FIRST 3 ROWS ONLY;

-- define a looping mechanism, in this example we use a GOTO
REFETCH_LABEL:

-- open the cursor
OPEN CSR;

-- fetch from the cursor until EOF (+100)
DO WHILE (SQLCODE != 0)
    FETCH FROM CSR INTO :HV1, :HV2, :HV3;

-- increment row counter
    ROWCTR = ROWCTR + 1;
END WHILE;
```



```

-- decrement number of rows still needed to fetch
  NUMROWS = NUMROWS - ROWCTR;

-- close the cursor
  CLOSE CSR;

-- if more rows still exist, open the cursor again using
-- with HV1, HV2, HV3 values from last fetch request
  IF (NUMROWS > 0) THEN
    GOTO REFETCH_LABEL;

```

Data-dependent pagination example

Consider a mobile phone application that uses data-dependent pagination to display automobile information. In the mobile app, the automobiles with the lowest prices and mileage are displayed first. Suppose the application developer wants to display only four rows at a time on the mobile phone screen because the mobile phone screen is small in comparison to a laptop or desktop computer.

A table, myAUTOS, has information about cars by a manufacturer:

```

SELECT VIN, MODEL, PRICE, MILEAGE FROM myAUTOS
  ORDER BY PRICE ASC, MILEAGE ASC, VIN ASC!

```

The results are as follows:

VIN	MODEL	PRICE	MILEAGE
JA4KA12340C000316	TL	995.00	140000
JF3KA96631C000012	TL	1995.00	110000
JH4KA96637C003617	RDX	5995.00	80000
JB1KA23667D000001	MDX	6995.00	100000
JH4KA96637C007613	MDX	6995.00	110000
JH5LA61637D761300	RDX	7995.00	65000
JH1KA63637C009625	RDX	7995.00	69000
JA4KA96638C111111	MDX	7995.00	100000
JD1CA26638C007613	MDX	7995.00	110000
JD2KA96638C001122	RDX	9995.00	80000

Notice that several cars have the same price, but the VIN column differs across all rows.

Prior to DB2 12, the application developer might have used the following SQL to display the lowest priced autos with the lowest mileage:

```

SELECT VIN, MODEL, PRICE, MILEAGE
FROM myAUTOS
WHERE ((PRICE = :PRICE_VAR AND MILEAGE = :MILEAGE_VAR AND
      VIN > :VIN_VAR) OR
      (PRICE = :PRICE_VAR AND MILEAGE > :MILEAGE_VAR) OR
      (PRICE > :PRICE_VAR))
ORDER BY PRICE ASC, MILEAGE ASC, VIN ASC
FETCH FIRST 4 ROWS ONLY;

```

In that SQL, the columns are in ascending order. The first predicate searches for all three attributes, while the second predicate searches for the two other columns that you are interested in, while the last predicate searches for the most important column, the PRICE column, to display the lowest priced auto first.

Starting in DB2 12, the basic predicate can now compare one row-value-expression with another row-value-expression with the greater than (>) operator:

```
SELECT VIN, MODEL, PRICE, MILEAGE
FROM myAUTOS
WHERE (PRICE, MILEAGE, VIN) > (:PRICE_VAR, :MILEAGE_VAR, :VIN_VAR)
ORDER BY PRICE ASC, MILEAGE ASC, VIN ASC
FETCH FIRST 4 ROWS ONLY;
```

With the additional comparison operators supported with row-value-expression comparisons, application developers can choose to simplify their SQL and potentially make their applications more readable. For detailed information about row-value-expression, basic predicates, and expression, see the “Language elements” section in the *DB2 12 for z/OS SQL Reference*, SC27-8859.

Note: SQL examples are in Appendix B, “Additional material” on page 279 in the following files:

- ▶ pagination_myautos_ddl.sql
- ▶ pagination_myautos_preV12appl.txt
- ▶ pagination_myautos_V12appl.txt
- ▶ pagination_myautos_drptbl.sql

6.3.3 Numeric-based pagination

Another form of pagination application that developers might employ in their applications is to use numeric-based pagination whereby data is returned to the user from a particular starting row. DB2 12 provides another method for application developers to display a set of rows from a particular starting row and for a particular number of rows with the new offset-clause clause. The offset-clause clause tells DB2 to start returning rows to the application after a skipping a certain number of rows. Prior to DB2 12, application developers might have used other methods such as a scrollable cursor, a rowset cursor, an OLAP function, a stored procedure, or application logic that consumed the rows prior to displaying the actual rows to the user.

For example, an application developer might have received a request from the business to not display the three lowest items. This example uses a product table, MYPRODUCT, like the DSN8C10.PRODUCT sample table, but with more rows in it:

```
SELECT NAME, PRICE FROM MYPRODUCT ORDER BY PRICE ASC;
```

The results are as follows:

NAME	PRICE
-----	-----
Tricycle	99.00
Unicycle	199.00
Fixie	500.00
Single-speed	699.00
Road bike	1000.00
Tandem	1699.00
Tri bike	2500.00
Mountain bike	5000.00
Cargo	5999.00
Electric	6999.00

The application developer might declare a cursor with the offset-clause, as in the following example, to prevent the three lowest items from being displayed:

```
DECLARE CS1 CURSOR FOR
SELECT NAME, PRICE
FROM MYPRODUCT
ORDER BY PRICE ASC
OFFSET 3 ROWS;
```

The data displayed to the user might be as follows, so that the 'Tricycle', 'Unicycle', and 'Fixie' bicycles are not displayed:

NAME	PRICE
-----	-----
Single-speed	699.00
Road bike	1000.00
Tandem	1699.00
Tri bike	2500.00
Mountain bike	5000.00
Cargo	5999.00
Electric	6999.00

Similarly, if the business wants to return only items 5 - 9 from the original table, the application developer might code the cursor to use the offset-clause in conjunction with the fetch-clause as in this example:

```
DECLARE CS1 CURSOR FOR
SELECT NAME, PRICE
FROM MYPRODUCT
ORDER BY PRICE ASC
OFFSET 4 ROWS
FETCH FIRST 5 ROWS ONLY;
```

The data displayed to the user might be as follows, so that the 'Tricycle', 'Unicycle', 'Fixie', 'Single-speed', and 'Electric' bicycles are not displayed:

NAME	PRICE
-----	-----
Road bike	1000.00
Tandem	1699.00
Tri bike	2500.00
Mountain bike	5000.00
Cargo	5999.00

Consider another example where the application developer wants to display only twenty rows at a time. The application developer might have fetched data from multiple cursors, skipping rows in the application, then display the data back to the user:

```
DECLARE CS1 CURSOR FOR SELECT ..... FETCH FIRST 20 ROWS ONLY;
OPEN CS1;
loop 20 times:
    FETCH FROM CS1 INTO ...;
    output result to user
end loop;
CLOSE CS1;
DECLARE CS2 CURSOR FOR SELECT ..... FETCH FIRST 40 ROWS ONLY;
OPEN CS2;
loop 20 times:
    FETCH FROM CS2;
```

```

end loop;
loop 20 times:
  FETCH FROM CS2 INTO ...;
  output result to user
end loop;
CLOSE CS2;

```

Starting in DB2 12, an application developer can add the following code to the application to display twenty rows at a time while using the new offset-clause in combination with the enhancement to the fetch-clause:

```

:offset_hv = 0;
:fetch_hv = 20;
DECLARE CS1 CURSOR FOR SELECT ..... OFFSET ? ROWS FETCH FIRST ? ROWS ONLY;
OPEN CS1 USING :offset_hv, :fetch_hv;
loop 2 times:
  loop 20 times:
    FETCH FROM CS1 INTO ...;
    output result to user
  end loop;
CLOSE CS1;
:offset_hv = 20;
OPEN CS1 USING :offset_hv, :fetch_hv;
end loop;

```

Note: SQL examples are in Appendix B, “Additional material” on page 279 in the following files:

- ▶ pagination_myproduct_ddl.sql
- ▶ pagination_myproduct_appl.txt
- ▶ pagination_myproduct_drptbl.sql

6.4 Additional support for arrays

DB2 11 provided support for the array data type. Arrays could be defined as ordinary arrays or associative arrays. Ordinary arrays have a user-defined number of elements that are referenced by their ordinal position in the array. Associative arrays have no user-defined number of elements that are referenced by the array index value. An associative array’s index values do not have to be contiguous but they are unique. SQL PL variables and parameters for SQL PL routines could be defined as arrays.

DB2 11 also provided support for global variables. Global variables allow application developers to create a variable once, and then use that variable across multiple SQL statements within the same transaction. However, DB2 11 did not allow a global variable to be defined as an array.

Starting in DB2 12, application developers can perform the following functions:

- ▶ Create global variables with an array data type.
- ▶ Specify an associative array as an argument to the ARRAY_AGG aggregate function.
- ▶ Optionally specify the order-by-clause on the ARRAY_AGG aggregate function.

Those functions allow applications to exploit variables with the array data type outside of SQL PL objects (routines and triggers).

6.4.1 Arrays as global variables

Consider this small example of an SQL PL variable having an ordinary array type containing INTEGER values. The ordinary array can hold a maximum of five elements. The steps in the following example demonstrate how to create an array type, declare an SQL PL variable of that array type, set an element in the array variable, and populate a column of a table.

```
CREATE TYPE myOrdIntArray AS INTEGER ARRAY[5]!
CREATE TABLE myResultsTB (countCOL INTEGER)!
COMMIT!
CREATE PROCEDURE SP1
LANGUAGE SQL
BEGIN
    DECLARE myOrdIntArrayVar myOrdIntArray;
    SET myOrdIntArrayVar[1] = 999;
    INSERT INTO myResultsTB VALUES(myOrdIntArrayVar[1]);
END!
CALL SP1!
COMMIT!
SELECT countCOL FROM myResultsTB!
```

The result of the SELECT statement displays the following information:

```
countCOL
-----
          999
```

Note: This SQL example is in Appendix B, “Additional material” on page 279 in the `sqlplvar.sql` file.

The next example is of a *global variable* that has a data type of INTEGER. The global variable is then shared across multiple SQL statements.

```
CREATE VARIABLE myIntGV INTEGER!
CREATE TABLE myResultsTB (countCOL INTEGER)!
COMMIT!
SET myIntGV = 99!
COMMIT!
CREATE PROCEDURE SP1
LANGUAGE SQL
BEGIN
    INSERT INTO myResultsTB VALUES(myIntGV);
END!
COMMIT!
CALL SP1!
COMMIT!
SELECT countCOL FROM myResultsTB!
```

The result of the SELECT statement displays the following information:

```
countCOL
-----
          99
```

Note: This SQL example is in Appendix B, “Additional material” on page 279 in the `globalvar.sql` file.

Starting in DB2 12, a *global variable* can now be created as an *array type*. Using the two previous examples, an *array global variable* is now used to populate the result table:

```
CREATE TYPE myOrdIntArray AS INTEGER ARRAY[5]!  
CREATE TABLE myResultsTB (countCOL INTEGER)!  
COMMIT!  
CREATE VARIABLE myIntAryGV myOrdIntArray!  
COMMIT!  
SET myIntAryGV [1] = 9!  
CREATE PROCEDURE SP1  
LANGUAGE SQL  
BEGIN  
    INSERT INTO myResultsTB VALUES(myIntAryGV[1]);  
END!  
COMMIT!  
SELECT countCol FROM myResultsTB!
```

The result of the SELECT statement displays the following information:

```
countCOL  
-----  
          9
```

Note: This SQL example is in Appendix B, “Additional material” on page 279 in the `aryglobalvar.sql` file.

6.4.2 Associative array support on ARRAY_AGG aggregate function

DB2 11 provided support for the ARRAY_AGG aggregate function, which allowed you to create an array based on data in your tables. For all rows referenced in the query, the result column value is assigned to an array element. The ARRAY_AGG aggregate produce only an ordinary array in DB2 11.

For example, using a product table, MYPRODUCT, like the DSN8C10.PRODUCT sample table, but with more rows in it:

```
SELECT NAME, PRICE FROM MYPRODUCT ORDER BY PRICE ASC;
```

The results are as follows:

NAME	PRICE
-----	-----
Tricycle	99.00
Unicycle	199.00
Fixie	500.00
Single-speed	699.00
Road bike	1000.00
Tandem	1699.00
Tri bike	2500.00
Mountain bike	5000.00
Cargo	5999.00
Electric	6999.00

Create an ordinary array global variable containing all product prices:

```
CREATE TYPE PRICES AS DECIMAL(30,2) ARRAY[25]!  
COMMIT!  
CREATE VARIABLE myPRICES PRICES!  
COMMIT!  
SET myPRICES = (SELECT ARRAY_AGG(PRICE ORDER BY PRICE ASC)  
                FROM myPRODUCT)!  
SELECT T.PRICE FROM UNNEST(myPRICES) AS T(PRICE)!
```

The results are as follows:

```
PRICE  
-----  
    99.00  
   199.00  
   500.00  
   699.00  
  1000.00  
  1699.00  
  2500.00  
  5000.00  
  5999.00  
  6999.00
```

Starting in DB2 12, you can use the ARRAY_AGG aggregate function to create an associative array. The next example creates an associative array global variable for a price sheet where the array's index is the NAME and the elements of the array contain the PRICE:

```
CREATE TYPE PRICESHEET  
  AS DECIMAL(30,2) ARRAY[VARCHAR(128) CCSID UNICODE]!  
COMMIT!  
CREATE VARIABLE myPRICESHEET PRICESHEET!  
COMMIT!  
SET myPRICESHEET = (SELECT ARRAY_AGG(NAME, PRICE)  
                   FROM MYPRODUCT)!  
SELECT T.NAME, T.PRICE  
FROM UNNEST(myPRICESHEET) AS T(NAME,PRICE)!
```

Results are as follows; notice that because this associative array has an index of character data (NAME), the results are naturally ordered by NAME:

NAME	PRICE
-----	-----
Cargo	5999.00
Electric	6999.00
Fixie	500.00
Mountain bike	5000.00
Road bike	1000.00
Single-speed	699.00
Tandem	1699.00
Tri bike	2500.00
Tricycle	99.00
Unicycle	199.00

Note: This SQL example is in Appendix B, "Additional material" on page 279 in the arrayagg.sql file.

6.4.3 Optional ORDER BY clause on ARRAY_AGG aggregate function

As discussed previously, DB2 11 provided support for the ARRAY_AGG aggregate function. The ARRAY_AGG aggregate function could produce only an ordinary array in DB2 11. If multiple ordinary arrays were created, and the order-by-clause was specified for one of the arrays, the order-by-clause had to be specified for all of the arrays.

For example, using the product table MYPRODUCT like the DSN8C10 . PRODUCT sample table, but with more rows in it.

```
SELECT NAME, PRICE FROM MYPRODUCT ORDER BY PRICE ASC;
```

Results are as follows:

NAME	PRICE
Tricycle	99.00
Unicycle	199.00
Fixie	500.00
Single-speed	699.00
Road bike	1000.00
Tandem	1699.00
Tri bike	2500.00
Mountain bike	5000.00
Cargo	5999.00
Electric	6999.00

Create two ordinary array global variables containing all product names and prices:

```
CREATE TYPE NAMES AS VARCHAR(128) CCSID UNICODE ARRAY[25]!  
CREATE TYPE PRICES AS DECIMAL(30,2) ARRAY[25]!  
COMMIT!  
CREATE VARIABLE myNAMES NAMES!  
CREATE VARIABLE myPRICES PRICES!  
COMMIT!  
SET (myNAMES, myPRICES) =  
    (SELECT ARRAY_AGG(NAME ORDER BY PRICE ASC),  
     ARRAY_AGG(PRICE ORDER BY PRICE ASC)  
     FROM myPRODUCT)!  
COMMIT!  
SELECT T.NAME, T.PRICE  
FROM UNNEST(myNAMES,myPRICES) AS T(NAME,PRICE)!
```

Results are as the follows:

NAME	PRICE
Tricycle	99.00
Unicycle	199.00
Fixie	500.00
Single-speed	699.00
Road bike	1000.00
Tandem	1699.00
Tri bike	2500.00
Mountain bike	5000.00
Cargo	5999.00
Electric	6999.00

Starting in DB2 12, you can specify only one order-by-clause. In the next example, the order-by-clause was removed for the PRICE column, which is the source of the myPRICES ordinary array global variable:

```
SET (myNAMES, myPRICES) =
    (SELECT ARRAY_AGG(NAME ORDER BY PRICE ASC),
        ARRAY_AGG(PRICE)
     FROM myPRODUCT)!
SELECT T.NAME, T.PRICE
FROM UNNEST(myNAMES,myPRICES) AS T(NAME,PRICE)!
```

Results are as follows:

NAME	PRICE
-----	-----
Tricycle	99.00
Unicycle	199.00
Fixie	500.00
Single-speed	699.00
Road bike	1000.00
Tandem	1699.00
Tri bike	2500.00
Mountain bike	5000.00
Cargo	5999.00
Electric	6999.00

Note: This SQL example is in Appendix B, “Additional material” on page 279 in the arrayagg.sql file.

6.5 MERGE statement enhancements

DB2 9 provided the initial support for the MERGE statement. The MERGE statement allowed application developers to either update existing rows in a table or insert new rows into a table, depending on the source values specified. If the source values match the target values then the rows are updated, otherwise the new values are inserted into the target table. The source values could be an expression, host-variable-array, or NULL.

Starting in DB2 12, the MERGE statement is enhanced to allow the following support:

- ▶ Additional source value support
- ▶ Additional data modification support
- ▶ Additional matching condition options
- ▶ Additional predicates on the matching conditions support
- ▶ Atomicity

Refresher on MERGE statement support prior to DB2 12

Consider this DB2 11 MERGE statement example of a MYPRODUCT product table that is similar to the DSN8C10.PRODUCT sample table:

```
SELECT * FROM MYPRODUCT ORDER BY PRICE ASC;
```

Results are as follows:

PID	NAME	PRICE
-----	-----	-----
00800	Tricycle	99.00
00900	Unicycle	199.00
00400	Fixie	500.00
00500	Single-speed	699.00
00200	Road bike	1000.00
00600	Tandem	1699.00
00300	Tri bike	2500.00
00100	Mountain bike	5000.00
00700	Cargo	5999.00
01000	Electric	6999.00

Using the MERGE statement, the price of the mountain bike will be updated to the value 4599.00 because its PID matches the of '00100' source value:

```
SET myPRICE = 4599.00;
SET myPID = '00100';
MERGE INTO MYPRODUCT Tgt
  USING (VALUES(myPID,myPRICE,myNAME))
  AS Src(myPID,myPRC,myNME)
  ON (Tgt.PID = Src.myPID)
  WHEN MATCHED THEN UPDATE
    SET PRICE = Src.myPRC
  WHEN NOT MATCHED THEN
    INSERT (PID,PRICE,NAME)
    VALUES (Src.myPID,Src.myPRC,Src.myNME)
  NOT ATOMIC CONTINUE ON SQLEXCEPTION;
SELECT * FROM MYPRODUCT ORDER BY PRICE ASC;
```

Results are shown in the following update:

PID	NAME	PRICE
-----	-----	-----
00800	Tricycle	99.00
00900	Unicycle	199.00
00400	Fixie	500.00
00500	Single-speed	699.00
00200	Road bike	1000.00
00600	Tandem	1699.00
00300	Tri bike	2500.00
00100	Mountain bike	4599.00
00700	Cargo	5999.00
01000	Electric	6999.00

This next example adds a new type of bicycle, called a 'Cruiser', to the product table:

```
SET myPRICE = 899.00;
SET myNAME = 'Cruiser';
SET myPID = '00101';
MERGE INTO MYPRODUCT Tgt
  USING (VALUES(myPID,myPRICE,myNAME))
  AS Src(myPID,myPRC,myNME)
  ON (Tgt.PID = Src.myPID)
  WHEN MATCHED THEN UPDATE
    SET PRICE = Src.myPRC
```

```

WHEN NOT MATCHED THEN
  INSERT (PID,PRICE,NAME)
    VALUES (Src.myPID,Src.myPRC,Src.myNME)
NOT ATOMIC CONTINUE ON SQLEXCEPTION;
SELECT * FROM MYPRODUCT ORDER BY PRICE ASC;

```

Results in the following row have a PID of '101' inserted into the table:

PID	NAME	PRICE
00800	Tricycle	99.00
00900	Unicycle	199.00
00400	Fixie	500.00
00500	Single-speed	699.00
00101	Cruiser	899.00
00200	Road bike	1000.00
00600	Tandem	1699.00
00300	Tri bike	2500.00
00100	Mountain bike	4599.00
00700	Cargo	5999.00
01000	Electric	6999.00

Note: These SQL examples are in Appendix B, “Additional material” on page 279 in the merge_preV12.sql file.

DB2 12 MERGE statement syntax diagram

Figure 6-3 shows the MERGE statement syntax diagram for DB2 12. Notice the addition of the table-reference for source values, the signal-statement as an action for the matching-condition, and the ELSE IGNORE clause.

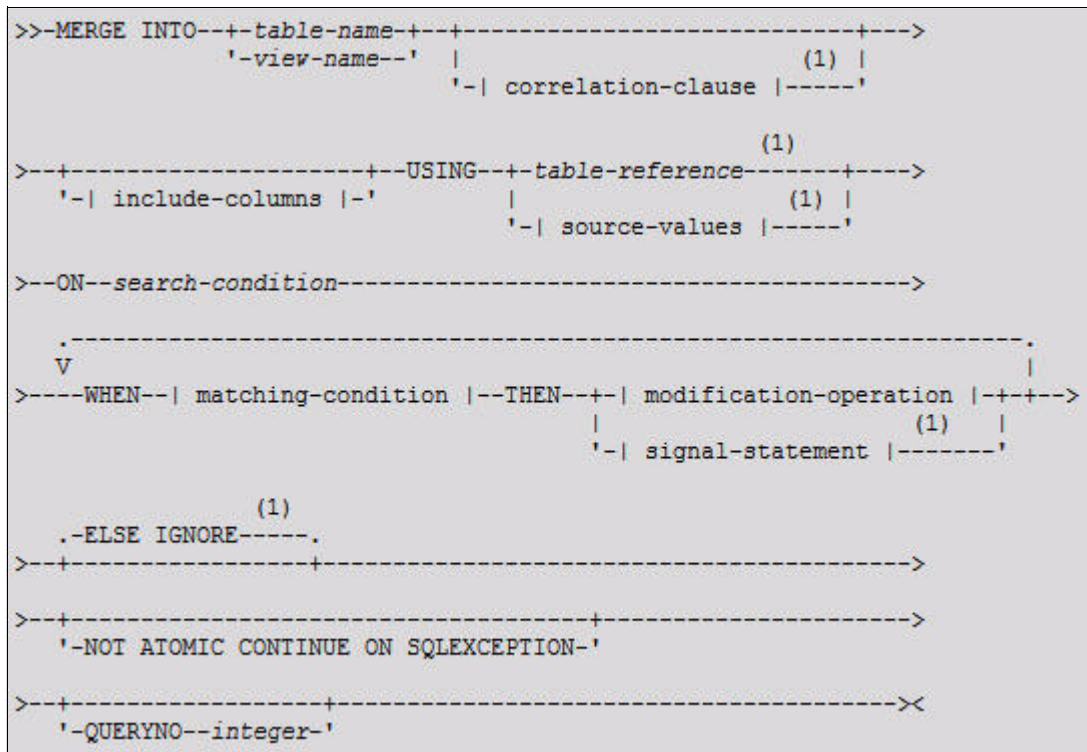


Figure 6-3 MERGE statement syntax diagram (partial)

6.5.1 Additional source value support

As described previously, prior releases of DB2 allowed the source values to be expression, host-variable-array, or NULL. Starting in DB2 12, the source values can be any of the following values:

- ▶ Data from tables
- ▶ Data from views
- ▶ Result from a fullselect
- ▶ An expression, host-variable-array, NULL

6.5.2 Additional data modification support

DB2 11 allowed application developers to update the target table or insert into the target table. Either the single update occurred or the single insert occurred. Starting in DB2 12, developers can issue a delete modification in addition to an update or insert modification. Additionally, more than one delete, update, or insert may be specified. Lastly, users may include the ELSE IGNORE clause to specify that no action should be taken for the rows where no matching-condition evaluates to True.

6.5.3 Additional matching condition option

Besides allowing a delete, update, or insert operation, developers may also issue a SIGNAL statement. The SIGNAL statement can be used to return an error when the matching-condition evaluates to True.

6.5.4 Additional predicates on matching conditions support

DB2 11 allows you to specify either a MATCHED or a NOT MATCHED condition. If MATCHED, then DB2 would perform the update modification; if NOT MATCHED, then DB2 would perform the insert modification. With DB2 12, a search-condition can be used with either the MATCHED or NOT MATCHED condition to further qualify source values that matched the ON search-condition.

6.5.5 Atomicity

The enhanced MERGE statement is atomic and therefore the NOT ATOMIC CONTINUE ON SQLEXCEPTION clause is not allowed. In DB2 12, if you use the enhanced features introduced on the MERGE statement, the MERGE statement is atomic in that the source rows are processed as though a set of rows is processed by each WHEN clause. In comparison, for a non-atomic MERGE statement, each source row is processed independently as though a separate MERGE statement were executed for each source row. For a non-atomic MERGE statement, if the data modification was an update modification, the update is cumulative in that the rows that are updated from a source row are subject to more updates by subsequent source rows in the same statement.

Also, for an atomic MERGE statement, if any source row encounters an error, processing stops and no rows in the target table are modified. Conversely, for a non-atomic MERGE statement, if any row encounters an error, processing continues for the remaining rows and the row that encountered the error is not inserted or updated into the target table.

6.5.6 Enhanced MERGE statement example

Suppose each group in a company has an activities table that contains all activities that this group organizes; the ACTIVITIES_MASTER table contains all upcoming activities organized by all groups in that company.

For example, the ACTIVITIES_GROUPA table contains all activities group A organizes. The ACTIVITIES_GROUPB table contains all activities group B organizes, and so forth. All activities tables have ACTIVITY as the primary key.

The ACTIVITIES_MASTER table has (GROUP, ACTIVITY) as the primary key, and ADATE is not nullable. The ACTIVITIES_MASTER table has a LAST_MODIFIED column defined as a row change timestamp column.

The following example is the corresponding SQL for the tables and global variable:

```
CREATE VARIABLE myGROUP CHAR(1)!
CREATE TABLE ACTIVITIES_MASTER
  (ACTIVITY CHAR(4) NOT NULL
  ,DESCRIPTION VARCHAR(128)
  ,ADATE DATE NOT NULL
  ,LAST_MODIFIED TIMESTAMP WITH DEFAULT
  ,GROUP CHAR(1) NOT NULL
  ,PRIMARY KEY (GROUP,ACTIVITY)
  )!
CREATE TABLE ACTIVITIES_GROUPA
  (ACTIVITY CHAR(4) NOT NULL PRIMARY KEY
  ,DESCRIPTION VARCHAR(128)
  ,ADATE DATE NOT NULL
  ,LAST_MODIFIED TIMESTAMP
  )!
```

Let's assume the ACTIVITIES_MASTER table already has a few rows:

```
SELECT *
FROM ACTIVITIES_MASTER
ORDER BY GROUP ASC, LAST_MODIFIED ASC!
```

Table 6-2 shows the contents of the ACTIVITIES_MASTER table.

Table 6-2 Current content of ACTIVITIES_MASTER table

ACTIVITY	DESCRIPTION	ADATE	LAST_MODIFIED	GROUP
0001	Luncheon at the golf course	2016-09-15	2016-09-28-09.12.36.731879	A
0003	Movie at the mall	2016-09-15	2016-09-28-09.12.36.811142	A
0002	Game of bowling	2016-09-16	2016-09-28-09.12.36.807500	B

Here is what group 'A' is organizing:

```
SELECT * FROM ACTIVITIES_GROUPA ORDER BY LAST_MODIFIED ASC!
```

Table 6-3 shows that group 'A' has two events being organized for 21 October 2016.

Table 6-3 Current content of ACTIVITIES_GROUPA table

ACTIVITY	DESCRIPTION	ADATE	LAST_MODIFIED
0004	Round of golf	2016-10-21	2016-09-13-09.12.36. 815246
0001	Luncheon at the golf course	2016-10-21	2016-09-28-09.12.36. 886441

Assuming the date today is 4 October 2016, you want to merge all activities into the ACTIVITIES_MASTER table. Based on conditions, the MERGE statement will perform one of these actions:

- ▶ Update the list of activities organized by group 'A'.
- ▶ Delete all outdated activities.
- ▶ Update the activities information (DESCRIPTION and ADATE), if they have changed.
- ▶ For new upcoming activities:
 - Insert a new row representing the new upcoming activity.
 - Signal an error if the date of the activity is not known.

The corresponding SQL is as follows:

```
SET myGROUP = 'A';
MERGE INTO ACTIVITIES_MASTER Tgt
USING (SELECT ACTIVITY, DESCRIPTION, ADATE, LAST_MODIFIED
      FROM ACTIVITIES_GROUPA) Src
ON (Tgt.ACTIVITY = Src.ACTIVITY) AND Tgt.GROUP = myGROUP
WHEN MATCHED AND Src.ADATE IS NULL THEN
  SIGNAL SQLSTATE '70001'
  SET MESSAGE_TEXT = Src.ACTIVITY ||
  ' cannot be modified. Reason: Date is unknown'
WHEN MATCHED AND Src.ADATE < CURRENT DATE THEN
  DELETE
WHEN MATCHED AND Tgt.LAST_MODIFIED < Src.LAST_MODIFIED THEN
  UPDATE SET (DESCRIPTION, ADATE, LAST_MODIFIED) =
  (Src.DESCRPTION, Src.ADATE, DEFAULT)
WHEN NOT MATCHED AND Src.ADATE IS NULL THEN
  SIGNAL SQLSTATE '70002'
  SET MESSAGE_TEXT = Src.ACTIVITY ||
  ' cannot be inserted. Reason: Date is unknown'
WHEN NOT MATCHED AND Src.ADATE >= CURRENT DATE THEN
  INSERT (GROUP, ACTIVITY, DESCRIPTION, ADATE)
  VALUES ('A',Src.ACTIVITY, Src.DESCRPTION, Src.ADATE)
ELSE IGNORE;
```

That SQL uses a global variable to indicate that the activities from group 'A' were being merged into the table. To merge all of the activities from group 'B', you only need to modify the global variable from 'A' to 'B' and use the same MERGE statement.

The MERGE statement used here successfully merged two rows from the ACTIVITIES_GROUPA table into the ACTIVITIES_MASTER table. A new row having an activity of '0004' was inserted into the table while the activity having '0001' was updated.

Table 6-4 shows the contents of the updated ACTIVITIES_MASTER table.

Table 6-4 Updated ACTIVITIES_MASTER table

ACTIVITY	DESCRIPTION	ADATE	LAST_MODIFIED	GROUP
0003	Movie at the mall	2016-09-15	2016-09-28-09.12.36. 811142	A
0004	Round of golf	2016-10-21	2016-10-04-15.54.38. 256115	A
0001	Luncheon at the golf course	2016-10-21	2016-10-04-15.54.38. 256115	A
0002	Game of bowling	2016-09-16	2016-09-28-09.12.36. 807500	B

Note: These SQL examples are in Appendix B, “Additional material” on page 279 in the merge_V12.sql file.

6.6 New built-in functions

DB2 12 introduces a set of aggregate functions and scalar functions. The aggregate functions can help with statistical analysis. The scalar functions provide enhancements for generating BINARY data and generating hashing values.

DB2 12 introduces these new aggregate functions:

- ▶ MEDIAN
- ▶ PERCENTILE_CONT
- ▶ PERCENTILE_DISC

The following new scalar functions are for hashing and for generating unique binary values:

- ▶ HASH_CRC32
- ▶ HASH_MD5
- ▶ HASH_SHA1
- ▶ HASH_SHA256
- ▶ GENERATE_UNIQUE_BINARY

One other new built-in function you might not know about is the VARCHAR_BIT_FORMAT scalar function. Although VARCHAR_BIT_FORMAT was available in DB2 11, it was made generally available after DB2 11.

6.6.1 Aggregate functions for statistics

Analytics are becoming more important as businesses want to gain insight into their customers, suppliers, products, and more. Performing analytics closer to where the data resides might provide better performance. Applications built for analytics can be complex. With additional SQL support, analytics can be performed by the database and applications can be simplified.

MEDIAN aggregate function

The MEDIAN function returns the median value of a set of numbers. You might often think of the median to be the “middle” value. For instance, if a table has 3 rows and you sort the data, the data in row 2 contains the median value. A table with 4 rows, results in a median of adding the value from the sorted rows of row 2 with row 3 then dividing that value in half. The result of the MEDIAN aggregate function produces a double precision floating-point number.

The following example uses product table MYPRODUCT, like the DSN8C10.PRODUCT sample table, but with more rows in it:

```
SELECT NAME, PRICE FROM MYPRODUCT ORDER BY PRICE ASC;
```

Results are in the following 10 rows:

NAME	PRICE
Tricycle	99.00
Unicycle	199.00
Fixie	500.00
Single-speed	699.00
Road bike	1000.00
Tandem	1699.00
Tri bike	2500.00
Mountain bike	5000.00
Cargo	5999.00
Electric	6999.00

You can calculate the median price of the bikes directly in DB2 by using the new MEDIAN aggregate function:

```
SELECT MEDIAN(PRICE) AS MEDIAN_PRICE,  
       CAST(MEDIAN(PRICE) AS DECIMAL(30,2)) AS MEDIAN_PRICE_DEC  
FROM MYPRODUCT;
```

Results are as follows:

MEDIAN_PRICE	MEDIAN_PRICE_DEC
1.3495000000000000E+03	1349.50

PERCENTILE_CONT aggregate function

The PERCENTILE_CONT aggregate function returns a percentile of a set of values. The value returned corresponds to the specified percentile given a sort specification by using a continuous distribution model. The PERCENTILE_CONT is calculated over a set of rows identified in a group. Similar to the MEDIAN aggregate function, the result of the PERCENTILE_CONT aggregate function produces a double precision floating-point number.

PERCENTILE_CONT syntax diagram

Figure 6-4 shows the syntax diagram for the PERCENTILE_CONT aggregate function.

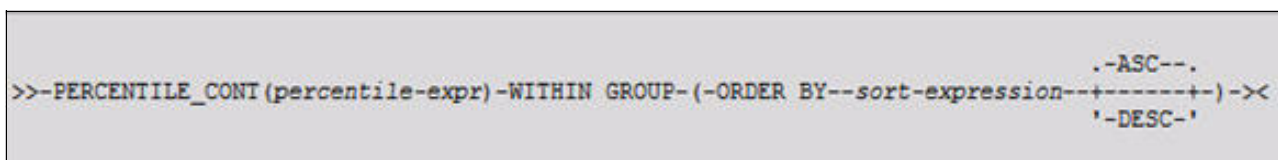


Figure 6-4 Syntax diagram for PERCENTILE_CONT aggregate function

PERCENTILE_CONT examples

The PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY sort-expression) can be used to calculate the MEDIAN aggregate function:

```
SELECT PERCENTILE_CONT(0.5)
       WITHIN GROUP (ORDER BY PRICE)
       AS MEDIAN_PRICE,
       CAST(PERCENTILE_CONT(0.5)
           WITHIN GROUP (ORDER BY PRICE) AS DECIMAL(30,2))
       AS MEDIAN_PRICE_DEC
FROM MYPRODUCT;
```

Results are as follows:

MEDIAN_PRICE	MEDIAN_PRICE_DEC
1.3495000000000000E+03	1349.50

In the next example, the line of business wants to know the 95th percentile of the total compensation per department from a table MYEMP that is like the DSN8C10.EMP sample table:

```
SELECT WORKDEPT, PERCENTILE_CONT(0.95)
       WITHIN GROUP (ORDER BY COMM+BONUS+SALARY)
       AS SAL_95TH,
       CAST(PERCENTILE_CONT(0.95)
           WITHIN GROUP (ORDER BY COMM+BONUS+SALARY) AS DEC(30,2))
       AS SAL_95TH_DEC
FROM MYEMP
GROUP BY WORKDEPT;
```

Results are as follows:

WORKDEPT	SAL_95TH	SAL_95TH_DEC
A00	5.6720000000000000E+04	56720.00
B01	4.5350000000000000E+04	45350.00
C01	4.0487600000000000E+04	40487.60
D11	3.4128500000000000E+04	34128.50
D21	3.7332400000000000E+04	37332.40
E01	4.4189000000000000E+04	44189.00
E11	3.1566000000000000E+04	31566.00
E21	2.8531500000000000E+04	28531.50

PERCENTILE_DISC aggregate function

Similar to the PERCENTILE_CONT aggregate function, the PERCENTILE_DISC aggregate function returns a percentile of a set of values. For the PERCENTILE_DISC aggregate function, each value in the input set is treated as a discrete value. The value returned corresponds to the specified percentile given a sort specification by using a discrete distribution model. The value returned is always a value that appeared in the input set. The PERCENTILE_DISC is calculated over a set of rows identified in a group. Unlike the MEDIAN and PERCENTILE_CONT aggregate functions, the result of the PERCENTILE_DISC aggregate function produces a result data type the same as the sort-expression data type.

The next example looks for the 95th percentile, by using the discrete distribution model, of the total compensation per department from a table MYEMP that is like the DSN8C10.EMP sample table:

```
SELECT WORKDEPT, PERCENTILE_DISC(0.95)
           WITHIN GROUP (ORDER BY COMM+BONUS+SALARY)
           AS SAL_95TH
FROM MYEMP
GROUP BY WORKDEPT!
```

Results are as follows:

WORKDEPT	SAL_95TH
A00	57970.00
B01	45350.00
C01	42110.00
D11	35430.00
D21	39763.00
E01	44189.00
E11	32730.00
E21	28742.00

Note: These SQL examples are in Appendix B, “Additional material” on page 279 in the `bifs_stats.sql` file.

6.6.2 Scalar functions for hashing

DB2 12 introduces hashing functions for the first time. Hashing functions in a database are typically used to convert a character string into a fixed length value (or key). The new hash value is a representation of the original character string. Using hashed values can help with query performance if the hashed value is shorter in length than the original character string. In addition, hashed values are often stored in the database and used as an index for data retrieval.

HASH_CRC32, HASH_MD5, HASH_SHA1, and HASH_SHA256 scalar functions

DB2 12 introduces a set of hashing functions with varying degrees of cryptographic strength. That set of hashing functions is shown in the following list, from weakest cryptography to strongest cryptography:

- ▶ HASH_CRC32
- ▶ HASH_MD5
- ▶ HASH_SHA1
- ▶ HASH_SHA256

Typically the stronger cryptographic hashes carry a longer execution time, therefore DB2 provides several hashing functions because user requirements differs, and DB2 provides a variety of functions.

6.6.3 GENERATE_UNIQUE_BINARY scalar function

DB2 provides a GENERATE_UNIQUE scalar function that generates a unique value that includes the internal form of Coordinated Universal Time (UTC), CPUID, and, if in a sysplex environment, the sysplex member where the function was processed. The GENERATE_UNIQUE function returns a CHAR(13) FOR BIT DATA value. With modern processors reaching ever increasing speeds, the timestamp part of the GENERATE_UNIQUE scalar function result is not precise enough to guarantee uniqueness. Therefore DB2 12 introduces the GENERATE_UNIQUE_BINARY function. The GENERATE_UNIQUE_BINARY scalar function provides a more precise result as a BINARY(16) value.

Consider that the GENERATE_UNIQUE_BINARY and GENERATE_UNIQUE scalar functions are non-deterministic. Additionally, the special register CURRENT_TIMESTAMP returns a timestamp value, and the GENERATE_UNIQUE_BINARY and GENERATE_UNIQUE functions return a modified timestamp value. Further, when you reference the CURRENT_TIMESTAMP special register in an SQL statement, the value is determined for the entire statement whereas the GENERATE_UNIQUE_BINARY and GENERATE_UNIQUE scalar functions generate a unique value for each affected row.

The following types of SQL statements will generate unique values for each row when the GENERATE_UNIQUE_BINARY function or GENERATE_UNIQUE function is referenced:

- ▶ A multiple row insert statement
- ▶ An insert statement with a fullselect
- ▶ An insert operation in a MERGE statement

The following example demonstrates the use of these scalar functions in an insert statement with a fullselect to generate the values to be inserted. The CURRENT_TIMESTAMP special register is also specified to provide input values.

```
CREATE TABLE myTABLE
  (ID CHAR(6),
   CURTMS TIMESTAMP,
   GENUNQ_TMS TIMESTAMP,
   GENUNQB_TMS TIMESTAMP)!
COMMIT!
INSERT INTO myTABLE
  SELECT EMPNO,
         CURRENT_TIMESTAMP,
         TIMESTAMP(GENERATE_UNIQUE()),
         TIMESTAMP(GENERATE_UNIQUE_BINARY())
  FROM DSN8C10.EMP
  ORDER BY EMPNO
  FETCH FIRST 3 ROWS ONLY!
COMMIT!
```

Table 6-5 shows the rows inserted into the table.

Table 6-5 *myTABLE* data populated from an insert with fullselect

ID	CURTMS	GENUNQ_TMS	GENUNQB_TMS
000010	2016-10-31-16.23.47. 737945	2016-10-31-23.23.47. 738026	2016-10-31-23.23.47. 738035
000020	2016-10-31-16.23.47. 737945	2016-10-31-23.23.47. 826548	2016-10-31-23.23.47. 826560
000030	2016-10-31-16.23.47. 737945	2016-10-31-23.23.47. 826594	2016-10-31-23.23.47. 826595

Notice, the column values generated by the scalar functions are unique for each row, but the special register was evaluated once and provided the same value for all rows.

Note: The SQL example is in Appendix B, “Additional material” on page 279 in the `bifs_genuniq_inssel.sql` file.

Method for storing both `GENERATE_UNIQUE_BINARY` and `GENERAGE_UNIQUE` values in the same column

If you have been storing values generated by the `GENERATE_UNIQUE` scalar function in a column and now want to store values with greater precision generated by the `GENERATE_UNIQUE_BINARY` scalar function, consider using the enhancement to the `TIMESTAMP` scalar function that DB2 12 introduces. By using the `TIMESTAMP` scalar function enhancement, the values produced by both the `GENERATE_UNIQUE_BINARY` and `GENERATE_UNIQUE` scalar functions may coexist in the same column. DB2 suggests you alter the existing column, presumably, of type `CHAR(13) FOR BIT DATA` to `VARBINARY(16)`. The `TIMESTAMP` scalar function is enhanced to allow a binary string with an actual length as input of 13 bytes, of which is assumed to be a result from the `GENERATE_UNIQUE` function.

In this example, your existing table has a `CHAR(13) FOR BIT DATA` column that contains the values generated from the `GENERATE_UNIQUE` scalar function:

```
CREATE TABLE myTABLE
  (ID INTEGER,
   GENUNQ CHAR(13) FOR BIT DATA);
COMMIT;
INSERT INTO myTABLE VALUES(GENERATE_UNIQUE(), 1);
COMMIT;
SELECT ID, LENGTH(GENUNQ) AS ULEN, HEX(GENUNQ) AS UHEX, TIMESTAMP(GENUNQ) AS UTMS
FROM myTABLE;
```

Results are as follows:

```
ID  ULEN  UHEX                                     UTMS
--  ----  -
1   13   00D161645F2B9B322600010001  2016-09-21-15.13.35.865267
```

The `GENUNQ` column can be altered to store values from `GENERATED_UNIQUE_BINARY` values. Be sure to alter the column to a varying length string to ensure no padding takes place.

```
ALTER TABLE myTABLE ALTER GENUNQ SET DATA TYPE VARBINARY(16);
COMMIT;
INSERT INTO myTABLE VALUES(2, GENERATE_UNIQUE_BINARY());
```

```
COMMIT;
SELECT ID, LENGTH(GENUNQ) AS ULEN, HEX(GENUNQ) AS UHEX, TIMESTAMP(GENUNQ) AS UTMS
FROM myTABLE;
```

Results are as follows:

```
ID  ULEN  UHEX
--  ----  -----
 1   13   00D161645F2B9B322600010001
 2   16   00D161645F39797DAA00000001010001
UTMS
-----
2016-09-21-15.13.35.865267
2016-09-21-15.13.35.922071
```

Note: These examples are in Appendix B, “Additional material” on page 279 in the `bifs_genuniq.sql` file.

6.6.4 VARCHAR_BIT_FORMAT scalar function enhancement

The `VARCHAR_BIT_FORMAT` scalar function became available in DB2 11 after DB2 11 was made generally available. The scalar function is available after new function is activated with the application compatibility of 'V11R1' or greater is specified.

The `VARCHAR_BIT_FORMAT` scalar function returns a bit data string representation of a character string that has been formatted using a format-string. The format-string contains a template of how the input source string should be interpreted. The following format strings are valid:

- ▶ 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'
- ▶ 'XXXXXXXX-XXXX-XXXX-XXXXXXXXXXXXXXXX'

Each x or X corresponds to one hexadecimal digit in the result. If you use an uppercase “X” then all input values must be in uppercase. If you use a lowercase “x” then all input values must be lowercase.

The `VARCHAR_BIT_FORMAT` scalar function is often used for Universally Unique Identifiers (UUIDs). UUIDs are 128-bit values that uniquely identify objects in software.

Here is an example with a UUID as input:

```
SELECT
  HEX(VARCHAR_BIT_FORMAT('db200c12-defa-11ad-b22f-b648bbb0916a',
                        'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'))
  AS HEX_UUID
FROM SYSIBM.SYSDUMMY1;
```

Results are as follows:

```
HEX_UUID
-----
DB200C12DEFA11ADB22FB648BBB0916A
```

Note: This example is in Appendix B, “Additional material” on page 279 in the `bifs_vchbitfmt.sql` file.

6.7 Enhanced built-in function support

DB2 12 provides enhancements to the following scalar functions:

- ▶ `TIMESTAMP`
- ▶ `XMLMODIFY`

6.7.1 `TIMESTAMP` scalar function enhancement

The `TIMESTAMP` scalar function is enhanced to allow a binary string with an actual length of 13 bytes, of which is assumed to be a result from the `GENERATE_UNIQUE` function.

An example of the `TIMESTAMP` enhancement is in “Method for storing both `GENERATE_UNIQUE_BINARY` and `GENERAGE_UNIQUE` values in the same column” on page 92.

6.7.2 `XMLMODIFY` scalar function enhancement

The Extensible Markup Language (XML) defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. The first working draft of an XML specification was published in 1996. XML 1.0 became a Worldwide Web Consortium (W3C) recommendation in February 1998.

DB2 9 for z/OS introduced support for the XML data type through the use of its IBM pureXML® capabilities and a hybrid database engine. With DB2 9, XML data previously stored in the traditional relational format can be stored natively as XML.

Many enhancements to XML processing were provided through maintenance in DB2 9 and in DB2 10. Those functions and enhancements are documented in the following publications:

- ▶ *DB2 10 for z/OS Technical Overview*, SG24-7892
- ▶ *Extremely pureXML in DB2 10 for z/OS*, SG24-7915

DB2 11 provided many enhancements to XML functionality. Some of those enhancements were retrofitted to DB2 10.

This section describes the XML enhancements in DB2 12, in particular, the following expressions can now be used with the `XMLMODIFY` scalar function:

- ▶ `FLWOR` expressions
- ▶ Conditional expressions
- ▶ Sequence expressions

With the addition of these expressions, you can perform multiple update actions in a single `XMLMODIFY` scalar function invocation. For the examples in the next sections, assume you have a table `purchaseOrders` containing a column named `PO` that contains an XML document as shown in Example 6-2.

Example 6-2 XML document sample original

```
<ipo:purchaseOrder
  xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
  xmlns:ipo=http://www.example.com/IPO
  xmlns:pyd=http://www.examplepayment.com
  orderDate="1999-12-01" pyd:paidDate="2000-01-07">

  <shipTo exportCode="1" xsi:type="ipo:UKAddress">
```

```

    <name>Helen Zoe</name>
    <street>47 Eden Street</street>
    <city>Cambridge</city>
    <postcode>CB1 1JR</postcode>
  </shipTo>
  <items>
    <item partNum="872-AA">
      <productName>Lawnmower</productName>
      <quantity>1</quantity>
      <USPrice>149.99</USPrice>
      <shipDate>2011-05-20</shipDate>
    </item>
    <item partNum="945-ZG">
      <productName>Sapphire Bracelet</productName>
      <quantity>2</quantity>
      <USPrice>178.99</USPrice>
      <comment>Not shipped</comment>
    </item>
  </items>
</ipo:purchaseOrder>

```

Support for FLWOR expression

The following example demonstrates how you can use the FLWOR expression to increase the US Price of each item by 10%. Notice the use of the for-let-where-return (FLWOR) expression.

```

UPDATE purchaseOrders
SET PO = XMLMODIFY(
    'declare namespace ipo="http://www.example.com/IPO";
    for$i in /ipo:purchaseOrder/items/item
    let $p := $i/USPrice
    where xs:decimal($p)>0
    return
    replace value of node $p with $p *1.1
    ');

```

Results are shown in Example 6-3.

Example 6-3 XML document sample after FLWOR update

```

<ipo:purchaseOrder
  xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
  xmlns:ipo=http://www.example.com/IPO
  xmlns:pyd=http://www.examplepayment.com
  orderDate="1999-12-01" pyd:paidDate="2000-01-07">
  <shipTo exportCode="1" xsi:type="ipo:UKAddress">
    <name>Helen Zoe</name>
    <street>47 Eden Street</street>
    <city>Cambridge</city>
    <postcode>CB1 1JR</postcode>
  </shipTo>
  <items>
    <item partNum="872-AA">
      <productName>Lawnmower</productName>
      <quantity>1</quantity>

```

```

    <USPrice>164.98</USPrice>
    <shipDate>2011-05-20</shipDate>
  </item>
  <item partNum="945-ZG">
    <productName>Sapphire Bracelet</productName>
    <quantity>2</quantity>
    <USPrice>196.89</USPrice>
    <comment>Not shipped</comment>
  </item>
</items>
</ipo:purchaseOrder>

```

Support for conditional expression

Given the original XML document (Example 6-2 on page 94), this next example demonstrates how you can update all items with a quantity greater than one with a discount of 10% and if the quantity is one or less then increase the price by 5%. Notice the use of the if-else condition.

```

UPDATE purchaseOrders
SET PO = XMLMODIFY(
  'declare namespace ipo="http://www.example.com/IPO";
  for $i in /ipo:purchaseOrder/items/item
  let $p := $i/USPrice
  let $q := $i/quantity
  where xs:decimal($p)>0 and xs:integer($q)>0
  return
  (
    if(xs:integer($q) > 1) then
      replace value of node $p with $p *0.9
    else
      replace value of node $p with $p *1.05 )
  ');

```

Results are shown in Example 6-4.

Example 6-4 XML document sample after conditional expression update

```

<ipo:purchaseOrder
  xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
  xmlns:ipo=http://www.example.com/IPO
  xmlns:pyd=http://www.examplepayment.com
  orderDate="1999-12-01" pyd:paidDate="2000-01-07">
  <shipTo exportCode="1" xsi:type="ipo:UKAddress">
    <name>Helen Zoe</name>
    <street>47 Eden Street</street>
    <city>Cambridge</city>
    <postcode>CB1 1JR</postcode>
  </shipTo>
  <items>
    <item partNum="872-AA">
      <productName>Lawnmower</productName>
      <quantity>1</quantity>
      <USPrice>149.99</USPrice>
      <shipDate>2011-05-20</shipDate>
    </item>
  </items>
</ipo:purchaseOrder>

```



```

    <item partNum="945-ZG">
      <productName>Sapphire Bracelet</productName>
      <quantity>2</quantity>
      <USPrice>178.99</USPrice>
      <comment>Not shipped</comment>
    </item>
  </items>
</ipo:purchaseOrder>

```

Support for sequence expression

Given the original XML document (Example 6-2 on page 94), the next example demonstrates how you can perform the following two actions in a single invocation of the XMLMODIFY scalar function:

- ▶ Replace the quantity of the item with partNum of 872-AA with the value 2.
- ▶ Delete the item with partNum of 945-ZG'.

Notice how the comma operator is used to construct a sequence of an update expression and delete expression:

```

UPDATE purchaseOrders
SET PO = XMLMODIFY(
  'declare namespace ipo="http://www.example.com/IPO";
  replace value of node
    /ipo:purchaseOrder/items/item[@partNum="872-AA"]/quantity
  with xs:integer(2),
  delete node /ipo:purchaseOrder/items/item[@partNum="945-ZG"]
  ');

```

Results are shown in Example 6-5.

Example 6-5 XML document sample after the use of a sequence expression

```

<ipo:purchaseOrder
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:ipo="http://www.example.com/IPO"
  xmlns:pyd="http://www.examplepayment.com"
  orderDate="1999-12-01" pyd:paidDate="2000-01-07">

  <shipTo exportCode="1" xsi:type="ipo:UKAddress">
    <name>Helen Zoe</name>
    <street>47 Eden Street</street>
    <city>Cambridge</city>
    <postcode>CB1 1JR</postcode>
  </shipTo>
  <items>
    <item partNum="872-AA">
      <productName>Lawnmower</productName>
      <quantity>2</quantity>
      <USPrice>149.99</USPrice>
      <shipDate>2011-05-20</shipDate>
    </item>
  </items>
</ipo:purchaseOrder>

```



Application enablement

This chapter describes DB2 12 for z/OS functions, not strictly confined to SQL, that provide infrastructure support for new applications or that simplify portability of existing applications to DB2 for z/OS from other database systems.

This chapter covers the following topics:

- ▶ Ensuring application compatibility
- ▶ Temporal table enhancements

7.1 Ensuring application compatibility

DB2 11 for z/OS introduced mechanisms to limit potential SQL and XML incompatibilities on application DML statements. In addition, the mechanisms allow you to control whether or not new SQL syntax and semantics are supported for an application.

For instance, in DB2 11 new function mode with the application compatibility set to 'V11R1', the maximum lengths for values returned for the SPACE and VARCHAR built-in functions decreased from 32767 to 32764. If the length of the output string for either of those functions was greater than 32764 bytes, applications received SQLCODE -171.

Additionally, all new SQL syntax changes introduced in DB2 11 were fenced by the application compatibility value of 'V11R1'. For instance, DB2 11 introduced support for arrays as SQL variables inside of SQL PL procedures and functions. On the CREATE PROCEDURE or CREATE FUNCTION statement, you needed to specify the application compatibility option to 'V11R1'.

The same infrastructure is in place for DB2 12 to ensure application compatibility:

- ▶ Identify applications that are affected by incompatible SQL and XML changes through the migration job reports and trace records.

At the time of this writing, DB2 12 has no incompatibilities. If DB2 does introduce an incompatible change in the future, use IFCID 376 to discover which applications will be affected.

IFCID 376 reports incompatibilities for both static and dynamic SQL and is written once per unique instance of a static statement, and also a dynamic statement in the dynamic statement cache.

- ▶ Control the compatibility level to DB2 10 or DB2 11 at the application level. The following methods apply to transitioning to new behavior:
 - APPLCOMPAT(VnnR1) option on BIND/REBIND PACKAGE, REBIND TRIGGER PACKAGE, CREATE/ALTER PROCEDURE (SQL native), and CREATE/ALTER FUNCTION (compiled SQL scalar) for static SQL.
 - APPLCOMPAT DSNZPARM for static SQL indicates the default value for the APPLCOMPAT bind option.
 - DSN_PROFILE_ATTRIBUTES for IBM DRDA applications.
 - CURRENT APPLICATION COMPATIBILITY special register for dynamic SQL.

Starting in DB2 12, DB2 recommends explicitly specifying the APPLCOMPAT option as follows:

- Packages for SQL routines, specify the option on the CREATE or ALTER SQL statement.
- Packages for advanced triggers, specify on the CREATE or ALTER SQL statement.
- Packages associated with basic trigger, specify the option on the **REBIND TRIGGER PACKAGE** command.
- Otherwise, specify the option on the **BIND** or **REBIND PACKAGE** command.

For details about application compatibility, see Chapter 2, “Continuous delivery” on page 7.

7.2 Temporal table enhancements

Before introducing the temporal table enhancements in DB2 12, here is a review the temporal table feature.

DB2 10 for z/OS added the initial support for temporal tables and data versioning. The term *temporal* pertains to the concept of time, in particular, data that is associated with some point in time. Many business applications associate data and time, as in these examples:

- ▶ Insurance policies

An insurer needs to keep track of when a client started insurance coverage, made changes to their insurance coverage, or stopped and restarted their insurance coverage.

- ▶ Regulatory and compliance laws

A business needs to track of who made changes to the data and when the changes were made. These changes might need to be preserved for an extended period of time.

Another key term used often with temporal tables is the notion of a *period*, which is a time interval represented by a start time and an end time. DB2 supports two types of time periods:

- ▶ System period:

- A table with a system period is known as a system temporal table (STT).
- A pair of columns with system-maintained values that indicate the period of time when a row is valid, which can also be referred to as a `SYSTEM_TIME` period.
- The system period is intended for supporting a concept of data versioning and is used to provide the system maintained history of the data.

- ▶ Application period:

- A table with an application period is known as an application temporal table (ATT).
- A pair of columns with application-maintained values that indicate the period of time when a row is valid, which can also be referred to as a `BUSINESS_TIME` period.
- The application period is intended for supporting a user-controlled way of establishing the notion of validity of the data. It allows user applications to specify and control the periods when certain data is considered valid to the user.

This section describes the DB2 12 enhancements related to temporal tables:

- ▶ Enhanced application periods
- ▶ Referential constraints for temporal tables
- ▶ Temporal logical transactions
- ▶ Auditing capabilities using temporal tables

Note: Several of the SQL examples are in Appendix B, “Additional material” on page 279. You can use the SQL examples in the appendix as templates to create your own applications, and as a learning tool to understand DB2 functionality.

7.2.1 Enhanced application periods

As 7.2.3, “Temporal logical transactions” on page 107 discussed, a *period* is a time interval represented by a start time and an end time. If a start time of a period is considered to be included in that period and the end time of the period is considered to be excluded from that period, it is referred to as an *inclusive-exclusive* (or closed/opened) time interval. For example, if the begin column has a value of '01/01/2015' and the end column has a value of '03/21/2015', the row is valid from January 1, 2015 to March 20, 2015, including both of those

dates. The row is not valid on 03/21/2015 (March 21, 2015)), the last day of the end column. The inclusive-exclusive time interval has been the only time interval semantic available in DB2 for an application-period temporal table with a BUSINESS_TIME period (ATT).

DB2 12 is enhanced to allow an application-period temporal table with a BUSINESS_TIME period that use the inclusive-inclusive semantic. An inclusive-inclusive (or closed/closed) time interval is a time interval where both the start time and end time are included in the interval. Revisiting the previous example, if the begin column has a value of '01/01/2015' and the end column has a value of '03/21/2015', the row is valid from January 1, 2015 to March 21, 2015, including both of those dates.

To specify an inclusive-inclusive period, use the new INCLUSIVE keyword in the PERIOD BUSINESS_TIME clause of the CREATE TABLE or ALTER TABLE statement. Notice, the EXCLUSIVE keyword is the default (Figure 7-1).

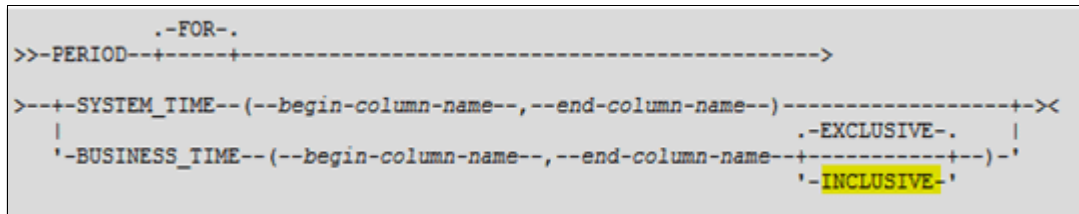


Figure 7-1 Updated period-definition syntax diagram

With support for the inclusive-inclusive time interval, DB2 will generate the corresponding temporal predicates for a time travel query. In Table 7-1, assume BUS_START and BUS_END represent the BUSINESS_TIME period columns. Notice, the inclusive-inclusive predicates now contain a greater than or equal to (>=) predicate; the previously supported inclusive-exclusive model supported only a greater than (>) predicate.

Table 7-1 Temporal predicates for inclusive-inclusive time interval

SQL	inclusive-inclusive ATT (DB2 12)	inclusive-exclusive ATT (pre-DB2 12)
AS OF expression	BUS_START <= expression AND BUS_END >= expression	BUS_START <= expression AND BUS_END > expression
FROM expression-1 TO expression-2	BUS_START < expression-2 AND BUS_END >= expression-1 AND expression-1 < expression-2	BUS_START < expression-2 AND BUS_END > expression-1 AND expression-1 < expression-2
BETWEEN expression-1 AND expression-2	BUS_START <= expression-2 AND BUS_END >= expression-1 AND expression-1 <= expression-2	BUS_START <= expression-2 AND BUS_END > expression-1 AND expression-1 <= expression-2

Temporal inclusive-inclusive BETWEEN-AND example

Suppose you have a myPOLICIES table with BUSINESS_TIME period defined on the BUS_START and BUS_END columns as INCLUSIVE:

PID	COVERAGE	BUS_START	BUS_END
A121	12000	2008-01-01	2008-07-01
A122	13000	2008-07-01	2008-07-01
A123	14000	2008-07-01	2009-01-01
A124	15000	2009-01-01	2010-01-01

If you want to set the coverage to be \$55,555 for all policies like 'A12%' with a portion of BUSINESS_TIME between '2008-07-01' and '2008-08-01' then issue the following UPDATE statement:

```
UPDATE myPOLICIES
  FOR PORTION OF BUSINESS_TIME
  BETWEEN '2008-07-01' AND '2008-08-01'
SET COVERAGE = 55555
WHERE PID LIKE 'A12%';
```

This UPDATE statement has the following results:

- ▶ Three rows ('A121', 'A122', 'A123') have their coverages updated to \$55,555.
- ▶ Policy 'A124' is unchanged.
- ▶ Rows for policies 'A121' and 'A123' have additional rows.

The content of each row after the update is as follows:

PID	COVERAGE	BUS_START	BUS_END
A121	55555	2008-07-01	2008-07-01
A122	55555	2008-07-01	2008-07-01
A123	55555	2008-07-01	2008-08-01
A124	15000	2009-01-01	2010-01-01
A121	12000	2008-01-01	2008-06-30
A123	14000	2008-08-02	2009-01-01

Temporal inclusive-inclusive AS OF example

Again, suppose you have a myPOLICIES table with BUSINESS_TIME period defined on the BUS_START and BUS_END columns as INCLUSIVE:

PID	COVERAGE	BUS_START	BUS_END
A123	55555	2008-07-02	2008-08-01
A124	15000	2009-01-01	2010-01-01
A121	12000	2008-01-01	2008-06-30
A123	14000	2008-08-02	2009-01-01

If you want to see all policies with a BUSINESS_TIME as of '2009-01-01', issue the following SELECT statement:

```
SELECT * FROM myPOLICIES
  FOR BUSINESS_TIME AS OF '2009-01-01';
```

Notice in the following output that policy 'A124' is returned because its BUS_END is *greater* than '2009-01-01' and the policy for 'A123' with coverage of \$14,000 is returned because its BUS_END is *equal* to '2009-01-01':

PID	COVERAGE	BUS_START	BUS_END
A124	15000	2009-01-01	2010-01-01
A123	14000	2008-08-02	2009-01-01

Note: The SQL examples are in Appendix B, “Additional material” on page 279 in the temporal_app_periods.sql file.

7.2.2 Referential constraints for temporal tables

In DB2 12, a temporal referential constraint can now be defined for an application-period temporal table (ATT) that contains a BUSINESS_TIME period. With a temporal referential constraint in place, DB2 will ensure the period of each child row is contained within a period of a parent row (or a set of contiguous rows without gap). You can specify the PERIOD BUSINESS_TIME clause in the definition of a referential constraint that enforces a temporal referential constraint for an application-period temporal table. On the CREATE TABLE or ALTER TABLE statement add PERIOD BUSINESS_TIME to the referential-constraint clause or to the references-clause clause.

For DB2 to enforce the referential constraint, a unique index must be defined on the parent table, with the BUSINESS_TIME WITHOUT OVERLAPS clause. The child table must also have an index that corresponds to the foreign key defined with the BUSINESS_TIME WITHOUT OVERLAPS clause. The BUSINESS_TIME period can be either inclusive-exclusive or the newly introduced inclusive-inclusive period and both the parent and child ATT must have the same semantics.

Updated syntax diagram for the referential constraint clause

An updated referential-constraint syntax diagram is shown in Figure 7-2.

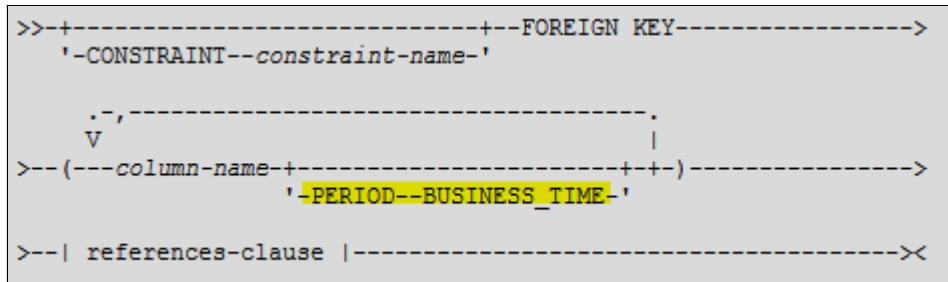


Figure 7-2 Updated referential-constraint syntax diagram

Updated syntax diagram for the references clause

An updated references-clause syntax diagram is shown in Figure 7-3.

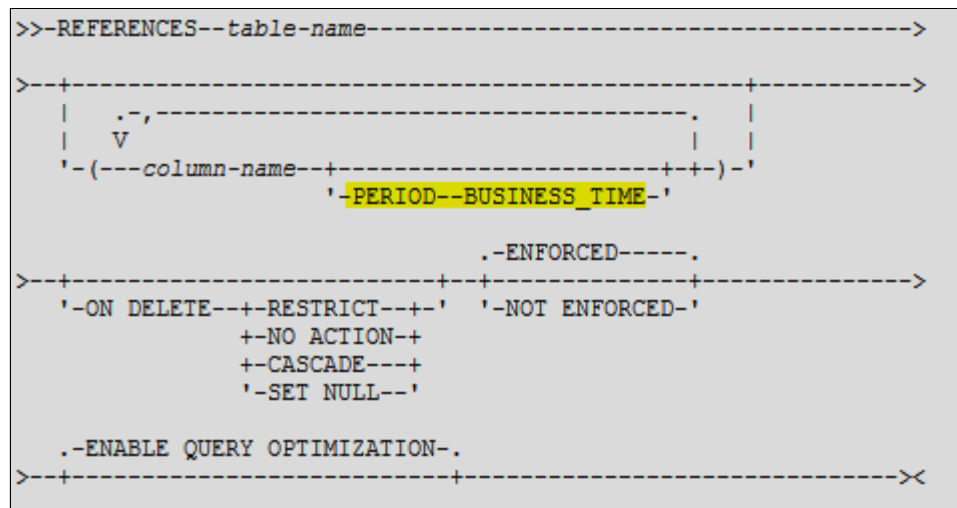


Figure 7-3 Updated references-clause syntax diagram

Referential constraint example

Suppose two tables exist:

- ▶ myDEPT: The parent application-period temporal table (ATT)
- ▶ myEMP: The child ATT

Notice that the myDEPT table has an application period defined on the DSTART and DEND columns. The myEMP table has an application period defined on the ESTART and EEND columns and the referential-constraint for the foreign key has the references-clause specified with it:

```
CREATE TABLE myDEPT
(DNO INTEGER NOT NULL,
 DNAME VARCHAR(30),
 DSTART DATE NOT NULL,
 DEND DATE NOT NULL,
 PERIOD BUSINESS_TIME (DSTART, DEND),
 PRIMARY KEY (DNO, BUSINESS_TIME WITHOUT OVERLAPS)
);
CREATE TABLE myEMP
(ENO INTEGER NOT NULL,
 ESTART DATE NOT NULL,
 EEND DATE NOT NULL,
 EDEPTNO INTEGER,
 PERIOD BUSINESS_TIME (ESTART, EEND),
 PRIMARY KEY (ENO, BUSINESS_TIME WITHOUT OVERLAPS),
 FOREIGN KEY (EDEPTNO, PERIOD BUSINESS_TIME)
 REFERENCES DEPT (DNO, PERIOD BUSINESS_TIME)
);
```

You create a unique index on the parent table, a unique index on the child table, plus an index on the child's foreign key:

```
CREATE UNIQUE INDEX DPRIMARY_KEY ON myDEPT
  (DNO, BUSINESS_TIME WITHOUT OVERLAPS);
CREATE UNIQUE INDEX EPRIMARY_KEY ON myEMP
  (ENO, BUSINESS_TIME WITHOUT OVERLAPS);
CREATE INDEX EFOREIGN_KEY ON myEMP
  (EDEPTNO, BUSINESS_TIME WITH OVERLAPS);
```

INSERT statement examples

Assume the following three rows exist in the myDEPT parent table:

DNO	DNAME	DSTART	DEND
3	SVT	2009-01-01	2010-01-01
3	FVT	2010-01-01	2011-12-31
4	DEV	2011-06-01	2011-12-31

The myEMP child table has one row:

ENO	ESTART	EEND	EDEPTNO
22218	2009-05-01	2011-02-03	3

You try to insert a row into the myEMP child table with an EDEPTNO of 3 having an end date of August 1, 2012:

```
INSERT INTO myEMP VALUES(22300,'2009-05-01','2012-08-01',3);
```

The results show the following referential constraint error:

```
SQLCODE = -530, ERROR: THE INSERT OR UPDATE VALUE OF FOREIGN KEY EDEPTNO IS
INVALID
```

The following insert with EDEPTNO of 4 having valid start and end dates succeeds:

```
INSERT INTO myEMP VALUES(22218,'2011-06-03','2011-11-12',4);
```

UPDATE statement examples

Continuing with the example above, the myEMP table now has two rows:

ENO	ESTART	EEND	EDEPTNO
22218	2009-05-01	2011-02-03	3
22218	2011-06-03	2011-11-12	4

The myDEPT table has the original three rows:

DNO	DNAME	DSTART	DEND
3	SVT	2009-01-01	2010-01-01
3	FVT	2010-01-01	2011-12-31
4	DEV	2011-06-01	2011-12-31

Issue an UPDATE statement against the myDEPT table having a department number of 4 and set the start date to August 3, 2011:

```
UPDATE myDEPT SET DSTART = '2011-08-03' WHERE DNO = 4!
```

This results in the following parent key error:

```
SQLCODE = -531, ERROR: PARENT KEY IN A PARENT ROW CANNOT BE UPDATED BECAUSE IT HAS ONE OR MORE DEPENDENT RELATIONSHIP EDEPTNO
```

Similarly, you issue an UPDATE statement against the myEMP table having a department number of 3 and set the end date to August 1, 2012:

```
UPDATE myEMP SET EEND = '2012-08-01' WHERE EDEPTNO = 3;
```

This results in the following referential constraint error:

```
SQLCODE = -530, ERROR: THE INSERT OR UPDATE VALUE OF FOREIGN KEY EDEPTNO IS INVALID
```

UPDATE statement with split rows example

Assume the myEMP table contains the following data:

ENO	ESTART	EEND	EDEPTNO
-----	-----	-----	-----
22218	2009-05-01	2011-02-03	3
22218	2011-06-03	2011-11-12	4

If the rows are split as a result of an update request, the referential constraint will be enforced during row splitting on the child table ATT. The following update is successful:

```
UPDATE myEMP
  FOR PORTION OF BUSINESS_TIME FROM '2011-07-01' TO '2011-08-01'
SET EDEPTNO = 3
WHERE EDEPTNO = 4;
```

The results are as follows. Notice, row 1 is unchanged, row 2 contains the updated period, rows 3 and 4 represent the split rows.

ENO	ESTART	EEND	EDEPTNO
-----	-----	-----	-----
22218	2009-05-01	2011-02-03	3
22218	2011-07-01	2011-08-01	3
22218	2011-06-03	2011-07-01	4
22218	2011-08-01	2011-11-12	4

Note: The SQL examples are in Appendix B, “Additional material” on page 279 in the temporal_ref_const.sql file.

7.2.3 Temporal logical transactions

As explained in 7.2, “Temporal table enhancements” on page 101, a system-period temporal table (STT) is a table that contains a pair of columns with system-maintained values that indicate the period of time when a row is valid. The period is referred to as a SYSTEM_TIME period.

DB2 defines a *logical unit of work* as a recoverable sequence of operations within an application process. An application process can consist of a single unit of work, that contains several logical units of work as a result of issuing COMMIT or ROLLBACK statements. This type of unit of work is often referred to as a *physical unit of work*.

Starting in DB2 12, application developers can initiate temporal logical units of work against an STT that are not determined by COMMIT or ROLLBACK. The values for the SYSTEM_TIME period, start and end columns, are determined based on the value of a

new built-in global variable that you set. The new built-in global variable is called SYSIBM.TEMPORAL_LOGICAL_TRANSACTION_TIME and has a TIMESTAMP(12) data type. When that built-in global variable is set to NULL, which is the default, no temporal logical transactions are allowed.

The following data change operations are affected by temporal logical transactions:

▶ INSERT or UPDATE of STT

The *begin* column value of the *current* table is generated by DB2 based on the value of the global variable.

▶ UPDATE or DELETE of STT

The *end* column value of the *history* table is generated by DB2 based on the value of the global variable.

▶ DELETE from STT with extra row generation through the ON DELETE ADD EXTRA ROW clause

The *begin* column value and *end* column value of the extra row in the *history* table are generated by DB2 based on the value of the global variable. For additional information regarding the new clause, see *DB2 12 for z/OS SQL Reference*, SC27-8859.

In addition to the SYSIBM.TEMPORAL_LOGICAL_TRANSACTION_TIME global variable, DB2 introduces another built-in global variable, SYSIBM.TEMPORAL_LOGICAL_TRANSACTIONS, to help application developers control physical units of temporal work versus logical units of temporal work. This SYSIBM.TEMPORAL_LOGICAL_TRANSACTION_TIME built-in global variable has a SMALLINT data type:

▶ A value of 0 (zero), the default, *disallows* multiple temporal logical transactions in a single commit scope.

▶ A value of 1 (one) *allows* multiple temporal logical transactions in a single commit scope.

The next examples demonstrate the following scenarios:

- ▶ Single application, single logical temporal transaction, multiple physical transactions
- ▶ Single application, multiple logical temporal transactions, single physical transaction
- ▶ Single application, multiple logical temporal transactions, single physical transaction, timestamp violation
- ▶ Two concurrent applications, competing multiple logical temporal transactions

Temporal logical transaction example

Suppose you have a myPOLICIES table with SYSTEM_TIME period defined on the SYS_START and SYS_END columns and a history table myPOLICIES_HIST:

```
CREATE TABLE myPOLICIES
(PID CHAR(4) NOT NULL,
 PTYPE VARCHAR(4) NOT NULL,
 COPAY SMALLINT NOT NULL,
 SYS_START TIMESTAMP(12) NOT NULL IMPLICITLY HIDDEN
   GENERATED ALWAYS AS ROW BEGIN,
 SYS_END TIMESTAMP(12) NOT NULL IMPLICITLY HIDDEN
   GENERATED ALWAYS AS ROW END,
 TRANS_ID TIMESTAMP(12) NOT NULL IMPLICITLY HIDDEN
   GENERATED ALWAYS AS TRANSACTION START ID,
 PRIMARY KEY (PID),
 PERIOD SYSTEM_TIME(SYS_START, SYS_END))!
COMMIT!
CREATE TABLE myPOLICIES_HIST LIKE myPOLICIES!
```

```

COMMIT!
ALTER TABLE myPOLICIES
  ADD VERSIONING USE HISTORY TABLE myPOLICIES_HIST!

```

Assume the myPOLICIES STT has a single row (Table 7-2).

Table 7-2 STT myPOLICIES initial data

PID	PTYPE	COPAY	SYS_START	SYS_END	TRANS_ID
C882	PPO	10	2016-10-05-13.04.14. 460764263000	9999-12-30-00.00.00. 000000000000	2016-10-05-13.04.14. 460764263000

Single application, single logical temporal transaction, multiple physical transactions example

This example demonstrates multiple physical transactions by having a **COMMIT** between each data modification to the myPOLICIES table with rows added to the myPOLICIES_HIST table, as appropriate, in a single logical temporal transaction as the SYSIBM.TEMPORAL_LOGICAL_TRANSACTION_TIME global variable is modified only once.

The following steps demonstrate this scenario:

1. Set the SYSIBM.TEMPORAL_LOGICAL_TRANSACTION_TIME global variable to CURRENT TIMESTAMP.
2. Issue a **COMMIT**.
3. Modify the policy type from 'PPO' to 'HMO'.
4. Issue a **COMMIT**.
5. Modify the copayment from 10 to 5.
6. Issue a **COMMIT**.

This is the corresponding SQL:

```

SET SYSIBM.TEMPORAL_LOGICAL_TRANSACTION_TIME = CURRENT TIMESTAMP!
COMMIT!
UPDATE myPOLICIES SET PTYPE = 'HMO' WHERE PID = 'C882'!
COMMIT!
UPDATE myPOLICIES SET COPAY = 5 WHERE PID = 'C882'!
COMMIT!

```

Notice that a **COMMIT** statement is issued after each UPDATE statement to produce multiple physical transactions.

After the *first* update is committed, the myPOLICIES table has the updated policy type with updated start and transaction times (Table 7-3) and the myPOLICIES_HIST table was populated with a single row having an end time the same as the myPOLICIES start time (Table 7-4).

Table 7-3 STT, myPOLICIES after policy type update

PID	PTYPE	COPAY	SYS_START	SYS_END	TRANS_ID
C882	HMO	10	2016-10-05-13.04.14. 570404000000	9999-12-30-00.00.00. 000000000000	2016-10-05-13.04.14. 657414608000

Table 7-4 History table, myPOLICIES_HIST, after policy type update

PID	PTYPE	COPAY	SYS_START	SYS_END	TRANS_ID
C882	PPO	10	2016-10-05-13.04.14. 460764263000	2016-10-05-13.04.14. 570404000000	2016-10-05-13.04.14. 460764263000

After the *second* update is committed, the myPOLICIES table has the updated copayment with a new transaction time while its start time stayed the same (Table 7-5). The myPOLICIES_HIST table is unchanged (Table 7-6).

Table 7-5 STT myPOLICIES after co-payment update

PID	PTYPE	COPAY	SYS_START	SYS_END	TRANS_ID
C882	HMO		2016-10-05-13.04.14. 570404000000	9999-12-30-00.00.00. 000000000000	2016-10-05-13.04.14. 756788072000

Table 7-6 History table, myPOLICIES_HIST, still contains unmodified single row

PID	PTYPE	COPAY	SYS_START	SYS_END	TRANS_ID
C882	PPO	10	2016-10-05-13.04.14. 460764263000	2016-10-05-13.04.14. 570404000000	2016-10-05-13.04.14. 460764263000

Single application, multiple logical temporal transactions within a single physical transaction

This example demonstrates multiple data modifications to the myPOLICIES table within a single physical transaction. The **COMMIT** is issued only once at the end with rows added to the myPOLICIES_HIST table, as appropriate, in multiple logical temporal transactions as the SYSIBM.TEMPORAL_LOGICAL_TRANSACTION_TIME global variable is modified multiple times.

The following steps demonstrate the scenario:

1. Set the SYSIBM.TEMPORAL_LOGICAL_TRANSACTION_TIME global variable to 1 to allow multiple logical temporal transactions to occur in a single commit scope.
2. Set the SYSIBM.TEMPORAL_LOGICAL_TRANSACTION_TIME global variable to the CURRENT TIMESTAMP.
3. Modify the policy type from 'PPO' to 'HMO'.
4. Set the SYSIBM.TEMPORAL_LOGICAL_TRANSACTION_TIME global variable again to the CURRENT TIMESTAMP.
5. Modify the copayment from 10 to 5.
6. Issue a **COMMIT**.

Assume you finished creating the myPOLICIES table (Table 7-7) and myPOLICIES_HIST table.

Table 7-7 STT myPOLICIES initial data

PID	PTYPE	COPAY	SYS_START	SYS_END	TRANS_ID
C882	PPO	10	2016-10-05-13.23.54. 354977492000	9999-12-30-00.00.00. 000000000000	2016-10-05-13.23.54. 354977492000

This is the corresponding SQL:

```
SET SYSIBM.TEMPORAL_LOGICAL_TRANSACTIONS = 1!
SET SYSIBM.TEMPORAL_LOGICAL_TRANSACTION_TIME = CURRENT TIMESTAMP!
UPDATE myPOLICIES SET PTYPE = 'HMO' WHERE PID = 'C882'!
SET SYSIBM.TEMPORAL_LOGICAL_TRANSACTION_TIME = CURRENT TIMESTAMP!
UPDATE myPOLICIES SET COPAY = 5 WHERE PID = 'C882'!
COMMIT!
```

Notice that only a single **COMMIT** is issued after the last data modification to produce a single physical transaction.

Looking at the data in the tables after the *first* update, the myPOLICIES table has the updated policy type with updated start and transaction times (Table 7-8) and the myPOLICIES_HIST table was populated with a single row containing an end time that is the same as the start time in the myPOLICIES table (Table 7-9).

Table 7-8 STT myPOLICIES after policy type update

PID	PTYPE	COPAY	SYS_START	SYS_END	TRANS_ID
C882	HMO	10	2016-10-05-13.23.54. 460015000000	9999-12-30-00.00.00. 000000000000	2016-10-05-13.23.54. 461328022000

Table 7-9 History table, myPOLICIES_HIST, after policy type update

PID	PTYPE	COPAY	SYS_START	SYS_END	TRANS_ID
C882	PPO	10	2016-10-05-13.23.54. 354977492000	2016-10-05-13.23.54. 460015000000	2016-10-05-13.23.54. 354977492000

However, after the *second* update is committed, the myPOLICIES table has the updated copayment with a new start time and an un-modified transaction time (Table 7-10) while the myPOLICIES_HIST table has an additional row to record the copayment change (Table 7-11).

Table 7-10 STT myPOLICIES after copayment update

PID	PTYPE	COPAY	SYS_START	SYS_END	TRANS_ID
C882	HMO		2016-10-05-13.23.54. 540575000000	9999-12-30-00.00.00. 000000000000	2016-10-05-13.23.54. 461328022000

Table 7-11 History table, myPOLICIES_HIST, after copayment update

PID	PTYPE	COPAY	SYS_START	SYS_END	TRANS_ID
C882	PPO	10	2016-10-05-13.23.54. 354977492000	2016-10-05-13.23.54. 460015000000	2016-10-05-13.23.54. 354977492000
C882	HMO	10	2016-10-05-13.23.54. 460015000000	2016-10-05-13.23.54. 540575000000	2016-10-05-13.23.54. 461328022000

Single application, multiple logical temporal transactions within a single physical transaction, timestamp violation

This example demonstrates multiple data modifications to the myPOLICIES table and rows added to the myPOLICIES_HIST table, as appropriate, with multiple logical temporal transactions in a single physical transaction. However, this example uses the global variable SYSIBM.TEMPORAL_LOGICAL_TRANSACTION_TIME to set the transaction time to a point in time prior to the initial time that the myPOLICIES table is populated.

The following steps demonstrate the scenario:

1. Set the SYSIBM.TEMPORAL_LOGICAL_TRANSACTIONS global variable to 1 to allow multiple logical temporal transactions to occur in a single commit scope.
2. Set the SYSIBM.TEMPORAL_LOGICAL_TRANSACTION_TIME global variable to the CURRENT TIMESTAMP.
3. Modify the policy type from 'PPO' to 'HMO'.
4. Set the SYSIBM.TEMPORAL_LOGICAL_TRANSACTION_TIME global variable to a time that is *prior to the initial SYS_START*.
5. Modify the copayment from 10 to 5.
6. Issue a **COMMIT**.

Assume you finished creating the myPOLICIES table (Table 7-12) and myPOLICIES_HIST tables.

Table 7-12 STT myPOLICIES initial data

PID	PTYPE	COPAY	SYS_START	SYS_END	TRANS_ID
C882	PPO	10	2016-10-05-13.49.04. 166123255000	9999-12-30-00.00.00. 000000000000	2016-10-05-13.49.04. 166123255000

This is the corresponding SQL:

```
SET SYSIBM.TEMPORAL_LOGICAL_TRANSACTIONS = 1!
SET SYSIBM.TEMPORAL_LOGICAL_TRANSACTION_TIME = CURRENT TIMESTAMP!
UPDATE myPOLICIES SET PTYPE = 'HMO' WHERE PID = 'C882'!
SET SYSIBM.TEMPORAL_LOGICAL_TRANSACTION_TIME =
  '2016-09-27-05.54.14.000000000000'!
UPDATE myPOLICIES SET COPAY = 5 WHERE PID = 'C882'!
COMMIT!
```

Notice that only a single **COMMIT** is issued after the last data modification to produce a single physical transaction.

Looking at the data in the tables after the *first* update, the myPOLICIES table has the updated policy type with an updated start time (Table 7-13) and the myPOLICIES_HIST table was populated with a single row (Table 7-14).

Table 7-13 STT myPOLICIES after policy type update

PID	PTYPE	COPAY	SYS_START	SYS_END	TRANS_ID
C882	HMO	10	2016-10-05-13.49.04. 277947000000	9999-12-30-00.00.00. 000000000000	2016-10-05-13.49.04. 279435625000

Table 7-14 History table, myPOLICIES_HIST, after policy type update

PID	PTYPE	COPAY	SYS_START	SYS_END	TRANS_ID
C882	PPO	10	2016-10-05-13.49.04. 166123255000	2016-10-05-13.49.04. 277947000000	2016-10-05-13.49.04. 166123255000

However, after the *second* update is committed, the following error occurs because the SYS_END was less than the SYS_START:

SQLCODE = -20528, ERROR: THE TARGET OF THE DATA CHANGE OPERATION IS A TABLE MYPOLICIES, WHICH INCLUDES A PERIOD SYSTEM_TIME. A ROW THAT THIS DATA CHANGE OPERATION ATTEMPTED TO MODIFY WAS ALSO MODIFIED BY ANOTHER TRANSACTION.

Two concurrent applications, competing multiple logical temporal transactions

This example demonstrates two concurrent applications with competing logical temporal transactions. Both applications set the SYSIBM.TEMPORAL_LOGICAL_TRANSACTION_TIME global variable but application-2 set the SYSIBM.TEMPORAL_LOGICAL_TRANSACTION_TIME global variable to a time *prior to the initial* SYS_START time.

Table 7-15 describes the scenario.

Table 7-15 Two applications with competing logical temporal transactions

Application-1	Application-2
SET TEMPORAL_LOGICAL_TRANSACTION_TIME = CURRENT_TIMESTAMP	SET TEMPORAL_LOGICAL_TRANSACTION_TIME = '2016-09-27-05.54.14.000000000000'
UPDATE myPOLICIES SET PTYPE = 'HMO' WHERE PID = 'C882'	UPDATE myPOLICIES SET PTYPE = 'POS' WHERE PID = 'C882'

Similar to the previous example of a single application having multiple logical temporal transactions in a single physical transaction with a timestamp violation, Application-2 will also experience SQLCODE -20528 because the SYS_END value was less than SYS_START value.

Note: The SQL examples are in Appendix B, “Additional material” on page 279 in the following files:

- ▶ temporal_log_trans.sql
- ▶ temporal_log_trans2.sql
- ▶ temporal_log_trans3.sql

7.2.4 Auditing capabilities using temporal tables

As discussed in 7.2, “Temporal table enhancements ” on page 101, temporal tables are often used to satisfy regulatory and compliance laws where a business needs to keep track of who made what changes to the data and when the data changes were made. Using a `SYSTEM_TIME` period with non-deterministic generated-expression columns, DB2 can now keep track of the “who, what, and when” of data modifications.

A non-deterministic generated expression column is a generated column that is defined using a non-deterministic expression. As such, the `as-generated-expression-clause` can now be specified as part of the column-definition on the `CREATE TABLE` and `ALTER TABLE` statements where the `as-generated-expression-clause` allows the following items to be specified: `DATA CHANGE OPERATION`, special register, or session variable. For columns defined with the new `as-generated-expression-clause`, DB2 automatically generates values based on the source expression during data change operations (insert/update/delete) and also with the `LOAD` utility when the `OVERRIDE` option is not specified.

For a delete operation, a new `ON DELETE ADD EXTRA ROW` clause is added to the `ALTER TABLE` statement. This optional clause tells DB2 to insert an additional row into the associated history table when a row is deleted from a system-period temporal table (STT). The additional row is used to track information about the delete operation.

This auditing support with temporal tables also became available in DB2 11 for z/OS after DB2 11 was made generally available. The functionality is available after migration to DB2 11 new function mode.

Updated syntax diagrams

The following syntax diagrams are updated or newly introduced:

- ▶ Updated column-definition with renaming of the `generated-column-definition` to the `generated-clause`
- ▶ Inclusion of the `as-generated-expression-clause` to the `generated-clause`
- ▶ Definition of the new `non-deterministic-expression` to the `as-generated-expression-clause`

Definition of non-deterministic-expression

The syntax diagram for the non-deterministic-expression is shown in Figure 7-7.

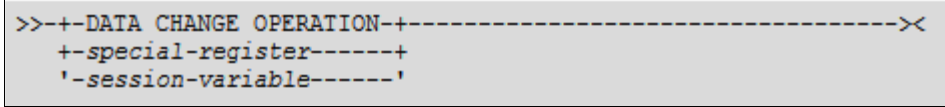


Figure 7-7 Syntax diagram for the non-deterministic-expression

Non-deterministic generated expression column specification

If you specify DATA CHANGE OPERATION for the non-deterministic-expression clause, DB2 will generate a CHAR(1) value representing the data modification:

- 'D' Delete operation
- 'I' Insert operation
- 'U' Update operation

Currently, DB2 supports special registers (Table 7-16) and session variables (Table 7-17) to be specified on the non-deterministic-expression clause. In addition, the data types of the columns must comply with the data types listed in Table 7-16 and Table 7-17. These tables indicate the expected data type for a non-deterministic generated expression column if the special register or session variable in the first column is specified as the expression.

Table 7-16 Special registers supported with the non-deterministic-expression

Special Register	Column data type
CURRENT CLIENT_ACCTG	VARCHAR(255)
CURRENT CLIENT_APPLNAME	VARCHAR(255)
CURRENT CLIENT_CORR_TOKEN	VARCHAR(255)
CURRENT CLIENT_USERID	VARCHAR(255)
CURRENT CLIENT_WRKSTNNAME	VARCHAR(255)
CURRENT SERVER	CHAR(16)
CURRENT SQLID	VARCHAR(n) where n>=8
SESSION_USER or USER	VARCHAR(128)

Table 7-17 Session variables supported with the non-deterministic-expression

Session variable	Column data type
SYSIBM.PACKAGE_NAME	VARCHAR(128)
SYSIBM.PACKAGE_SCHEMA	VARCHAR(128)
SYSIBM.PACKAGE_VERSION	VARCHAR(122)

Temporal table auditing example

Suppose you slightly modified myPOLICIES table from the previous examples. The myPOLICIES table also has a SYSTEM_TIME period defined on the SYS_START and SYS_END columns and a history table, myPOLICIES_HIST. In addition, you included non-deterministic generated expression columns in your table definition:

- ▶ A column that is based on the SESSION_USER special register
- ▶ A column that is based on the DATA CHANGE OPERATION clause

The DDL looks like the following example:

```
CREATE TABLE myPOLICIES
(PID CHAR(4) NOT NULL,
 PTYPE VARCHAR(4) NOT NULL,
 COPAY SMALLINT NOT NULL,
 USER_ID VARCHAR(128) GENERATED ALWAYS AS (SESSION_USER),
 OPCODE CHAR(1) GENERATED ALWAYS AS (DATA CHANGE OPERATION),
 SYS_START TIMESTAMP(12) NOT NULL
   GENERATED ALWAYS AS ROW BEGIN,
 SYS_END TIMESTAMP(12) NOT NULL
   GENERATED ALWAYS AS ROW END,
 TRANS_ID TIMESTAMP(12) NOT NULL
   GENERATED ALWAYS AS TRANSACTION START ID,
 PRIMARY KEY (PID),
 PERIOD SYSTEM_TIME(SYS_START, SYS_END))!
CREATE TABLE myPOLICIES_HIST
(PID CHAR(4) NOT NULL,
 PTYPE VARCHAR(4) NOT NULL,
 COPAY SMALLINT NOT NULL,
 USER_ID VARCHAR(128),
 OPCODE CHAR(1),
 SYS_START TIMESTAMP(12) NOT NULL,
 SYS_END TIMESTAMP(12) NOT NULL,
 TRANS_ID TIMESTAMP(12) NOT NULL)!
COMMIT!
ALTER TABLE myPOLICIES
  ADD VERSIONING USE HISTORY TABLE myPOLICIES_HIST
  ON DELETE ADD EXTRA ROW!
```

Assume the myPOLICIES STT has a single row and the myPOLICIES_HIST has no rows (Table 7-18).

Table 7-18 STT myPOLICIES initial data

PID	COPAY	USER_ID	OPCODE	SYS_START	SYS_END
C882	10	SALLY	'I'	2016-10-05-14.00.35. 565119448000	9999-12-30-00.00.00. 000000000000

About two hours later, Jen updates the row, increasing the copayment by \$10:

```
UPDATE myPOLICIES SET COPAY = COPAY + 10!
```

The myPOLICIES table has the updated row (Table 7-19) and a row is added to the myPOLICIES_HIST table (Table 7-20).

Table 7-19 STT, myPOLICIES, after copayment was increased

PID	COPAY	USER_ID	OPCODE	SYS_START	SYS_END
C882	20	JEN	'U'	2016-10-05-16.00.35. 565119448000	9999-12-30-00.00.00. 000000000000

Table 7-20 History table, myPOLICIES_HIST after co-payment was increased

PID	COPAY	USER_ID	OPCODE	SYS_START	SYS_END
C882	10	SALLY	'I'	2016-10-05-14.00.35. 565119448000	2016-10-05-16.00.35. 565119448000

Approximately, thirty minutes later, Chris deletes the policy:

```
DELETE FROM myPOLICIES!
```

The myPOLICIES table no longer has any rows but two rows are added to the myPOLICIES_HIST table, one for the update and one for the delete (Table 7-21).

Table 7-21 History table, myPOLICIES_HIST after policy was deleted

PID	COPAY	USER_ID	OPCODE	SYS_START	SYS_END
C882	10	SALLY	'I'	2016-10-05-14.00.35. 565119448000	2016-10-05-16.00.35. 565119448000
C882	20	JEN	'U'	2016-10-05-16.00.35. 565119448000	2016-10-05-16.30.35. 000000000000
C882	20	CHRIS	'D'	2016-10-05-14.30.35. 000000000000	2016-10-05-16.30.35. 000000000000

Note: The SQL example are in Appendix B, “Additional material” on page 279 in the following files:

- ▶ temporal_audit1.sql
- ▶ temporal_audit2.sql
- ▶ temporal_audit3.sql



Connectivity and administration routines

This chapter describes enhancements in the connectivity and administration routines of DB2 12:

- ▶ Maintaining session data on the target server
- ▶ Preserving prepared dynamic statements after a ROLLBACK
- ▶ DRDA fast load
- ▶ Profile monitoring for remote threads and connections
- ▶ Stored procedures supplied by DB2

8.1 Maintaining session data on the target server

Maintaining session data on the target DB2 12 for z/OS server is also called session token support. When this feature is enabled, a single small session token (a few bytes) generated by DB2 represents a client session. A transport is then associated with the session by flowing the appropriate session token on it.

Tip: You need to configure `clientApplcompat` at the Connection/DataSource level with a value `'V12R1'` to enable session token support to control behavior of applications migrating to DB2 12 for z/OS. Also, a new client driver level is required for your CLI and Java driver.

If this feature is not enabled, as in DB2 11 for z/OS, the client drivers using sysplex workload balancing (WLB) must remember and track session information (such as global variables, client information, and special registers) to support transaction pooling or connection reroute as shown in Figure 8-1.

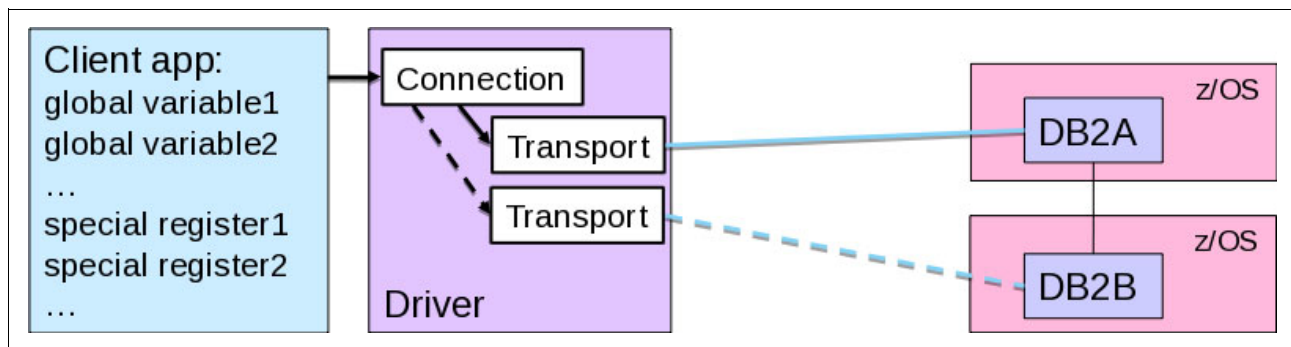


Figure 8-1 Remembering and tracking session information

Thus the WLB or client reroute initiated transport switch requires the client drivers to replay the session data on the new transport. This results in growing size of replay data, security exposure, and performance degradation.

In DB2 12 for z/OS, with the session token support, the session data will not be returned to client from DB2 for z/OS server. Here the client using sysplex WLB will pass a session token to DB2 server and DB2 will maintain session data for the data sharing group. DB2 will use the following catalog tables to manage session data:

- ▶ SYSIBM.SYSSESSION
- ▶ SYSIBM.SYSSESSION_EX
- ▶ SYSIBM.SYSSESSION_STATUS

In the sample scenario illustrated in Figure 8-2, if client connection is rerouted to DB2B, the client will pass a session token on establishing transport (global variables, special registers will be kept intact).

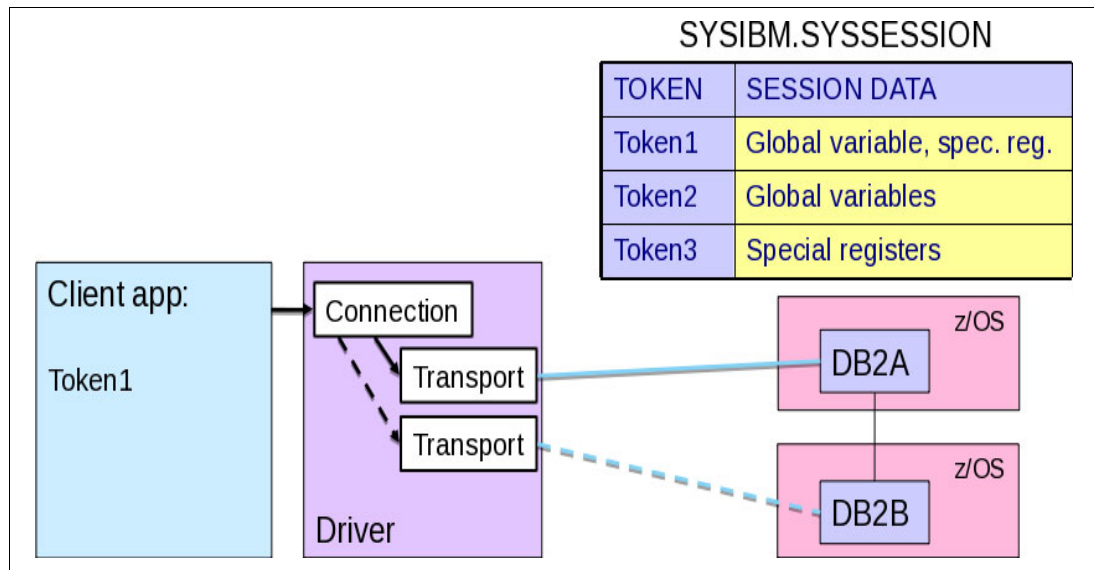


Figure 8-2 Sample scenario

Thus, with the session token support, the performance is better because you are now dealing with much smaller replay data (only the session token), far fewer session commands, and also it provides increased security for sensitive information in variables and registers.

When the application closes a logical connection, the driver flows a terminate connection to DB2, resulting in the following sequence of events:

1. The distributed thread (DBAT) is pooled.
2. Connection is made inactive.
3. The session information is released.

Tip: Applications should explicitly close result sets, statements, and connections thereby releasing database resources in a timely manner.

You may use the **MODIFY DDF** command to set the session data timeout value. The session data timeout value determines how long the session data will stay active before it can be removed from the SYSIBM.SYSSESSION table. For example, **-MODIFY DDF SESSIDLE(100)** sets the SESSIDLE value to 100 minutes.

Note: If session data was purged, you receive a DRDA reply with the error code -30062 and the invalid session token message. The application that receives -30062 should explicitly close the logical connection.

8.2 Preserving prepared dynamic statements after a ROLLBACK

Up to DB2 11, when the application package is bound with `KEEPDYNAMIC(YES)`, this option is applicable to the `COMMIT` request only. Normally statements are cleaned up on `COMMIT`. With the `KEEPDYNAMIC(YES)` option, DB2 keeps the runtime structures of cached dynamic statements on `COMMIT`. Example 8-1 shows `COMMIT` and dynamic SQL in DB2 11, assuming the package is bound with the `KEEPDYNAMIC(NO)` option.

Example 8-1 COMMIT and dynamic SQL with KEEP DYNAMIC(NO)

```
PREPARE STMTID FOR INSERT INTO T1 SELECT * FROM T2 WHERE T2.C1 = ?
EXECUTE STMTID USING :H1
EXECUTE STMTID USING :H2
COMMIT                                     → the dynamic INSERT is unprepared
EXECUTE STMTID USING :H3                 → fails because statement is not prepared
```

Example 8-2 shows the result when the package is bound with `KEEPDYNAMIC(YES)` option.

Example 8-2 Package bound with KEEP DYNAMIC(YES)

```
PREPARE STMTID FOR INSERT INTO T1 SELECT * FROM T2 WHERE T2.C1 = ?
EXECUTE STMTID USING :H1
EXECUTE STMTID USING :H2
COMMIT                                     → the dynamic INSERT is not unprepared
EXECUTE STMTID USING :H3                 → successful because statement is still prepared
```

The `KEEPDYNAMIC(YES)` functionality is to avoid a trip from the application to re-prepare dynamic SQL statements after `COMMIT`, which improves performance. However, this functionality does not apply to `ROLLBACK` on DB2 11. Example 8-3 shows `ROLLBACK` and dynamic SQL in DB2 11, assuming the package is bound with the `KEEPDYNAMIC(YES)` option.

Example 8-3 ROLLBACK and dynamic SQL

```
PREPARE STMTID FOR INSERT INTO T1 SELECT * FROM T2 WHERE T2.C1 = ?
EXECUTE STMTID USING :H1
EXECUTE STMTID USING :H2
ROLLBACK                                   → the dynamic INSERT is unprepared
EXECUTE STMTID USING :H3                 → fails because statement is not prepared
```

DB2 12 extends the `KEEPDYNAMIC(YES)` functionality to `ROLLBACK` also, where prepared dynamic statements are retained at the end of a unit of work. With DB2 12, and the package bound with `APPLCOMPAT(V12R1M500)` and `KEEPDYNAMIC(YES)`; Example 8-4 shows the result.

Example 8-4 Package bound with APPLCOMPAT(V12R1M500) and KEEP DYNAMIC(YES)

```
PREPARE STMTID FOR INSERT INTO T1 SELECT * FROM T2 WHERE T2.C1 = ?
EXECUTE STMTID USING :H1
EXECUTE STMTID USING :H2
ROLLBACK                                   → the dynamic INSERT is not unprepared
EXECUTE STMTID USING :H3                 → successful because statement is still prepared
```

The `DBAT` is also kept active so the next re-execution of the statement can resume immediately without the need for re-prepare.

Note: This function is available for local application as well.

8.3 DRDA fast load

DB2 supplies the stored procedure DSNUTILU, which can be invoked by a DB2 application program to run DB2 online utilities. The LOAD online utility is one such utility that DSNUTILU supports. As such, DSNUTILU can be used to load data from distributed clients to DB2 for z/OS servers by invoking the LOAD online utility.

Starting in DB2 12, the client drivers are enhanced to support remote loads to DB2 for z/OS servers. The DB2 Call Level Interface (CLI) APIs and the Command Line Processor (CLP) are modified to stream data to the load process in continuous blocks. By using the enhanced client drivers, the loading of data from distributed clients can significantly reduce elapsed time because the task that extracts data blocks and passes them to the LOAD utility is 100% zIIP offloadable using the goal of the distributed address space. The function is available before activating new function. For information about new function activation, see Chapter 2, “Continuous delivery” on page 7.

8.4 Profile monitoring for remote threads and connections

DB2 provides monitoring capabilities by the use of profiles tables. Profile tables enable you to monitor the use of system resources by remote applications, including remote threads and connections, and to control performance-related subsystem parameters in particular contexts on your DB2 subsystem. Each monitored situation is defined by a set of criteria called a profile.

A *profile* is a set of criteria that identifies a particular situation on a DB2 subsystem. A profile is defined by a record in the SYSIBM.DSN_PROFILE_TABLE table. The profile tables and related indexes are created by the **DSNTIJSJG** job during DB2 installation or migration. After profiling is correctly defined, use the **START PROFILE** command to start profile monitoring. Issue a **STOP PROFILE** command to disable all profile functions.

For further information about profiles, see the topic about using profiles to monitor and optimize performance in *DB2 12 for z/OS Managing Performance*, SC27-8857.

DB2 12 offers several enhancements to system profiles:

- ▶ Automatic start of profiles during subsystem start
- ▶ Support for global variables
- ▶ Support for wildcarding
- ▶ Idle thread enhancement

8.4.1 Automatic start of profiles during subsystem start

A new subsystem parameter, PROFILE_AUTOSTART, is introduced to allow the **START PROFILE** command processing to be automatically initiated as part of DB2 startup processing. If you set this subsystem parameter to YES, DB2 will automatically start the **START PROFILE** command processing.

Consider the following values:

- ▶ If you specify YES and your DB2 system was started with the ACCESS(MAINT) option or LIGHT(YES) option, DB2 will ignore the setting of the PROFILE_AUTOSTART subsystem parameter and not initiate **START PROFILE**.
- ▶ A value of NO, which is the default value, tells DB2 to not initiate **START PROFILE** during startup processing.

8.4.2 Support for global variables

You can create profiles to define a set of criteria to monitor processes within DB2. For remote applications, client information like application name, user ID name, and workstation name are particularly helpful.

Consider these two tables:

- ▶ `SYSIBM.DSN_PROFILE_TABLE`. Contains a row for each profile.
- ▶ `SYSIBM.DSN_PROFILE_ATTRIBUTES`. Allows you to specify actions for DB2 to take when a process, such as a SQL statement, thread, or connection meets the criteria of the profile. The `KEYWORDS` column in the `SYSIBM.DSN_PROFILE_ATTRIBUTES` table indicates which action DB2 should perform. Each action can have up to three attributes that control how the specified action is applied by DB2. The attributes are defined by the `ATTRIBUTE1`, `ATTRIBUTE2`, and `ATTRIBUTE3` columns.

Starting in DB2 12, you can specify `GLOBAL_VARIABLE` as a `KEYWORDS` column value in the `SYSIBM.DSN_PROFILE_ATTRIBUTES` table. The following built-in global variables are supported:

- ▶ `GET_ARCHIVE`
- ▶ `MOVE_TO_ARCHIVE`
- ▶ `TEMPORAL_LOGICAL_TRANSACTION_TIME`
- ▶ `TEMPORAL_LOGICAL_TRANSACTIONS`

The `ATTRIBUTE1` column must specify a valid `SET assignment-statement` statement for the global variable. For example, if you want DB2 to allow temporal logical transactions to occur, set the `ATTRIBUTE1` column value as follows:

```
SET SYSIBM.TEMPORAL_LOGICAL_TRANSACTIONS = 1
```

If a profile filter matches a connection, DB2 will automatically apply the built-in global variable value to the DB2 process of that connection when the connection is initially established, and when a connection is reused.

Setting a built-in global variable in the profile table applies only to remote applications. This function is available after new function is activated. For detailed information about new function activation, see Chapter 2, “Continuous delivery” on page 7. For additional information about temporal logical transactions, see Chapter 7, “Application enablement” on page 99.

8.4.3 Support for wildcarding

As indicated previously, `SYSIBM.DSN_PROFILE_TABLE` contains a row for each profile. Each column in the table tells DB2 which connection to monitor. Prior to DB2 12, to handle various connections, multiple rows had to be defined in the table. Starting in DB2 12, you may choose to have a single row that represents more than one connection.

The following columns are enhanced in DB2 12 for remote connections:

- ▶ `AUTHID`
- ▶ `LOCATION`
- ▶ `PRDID`

The support described next is available before new function activation. For information about new function activation, see Chapter 2, “Continuous delivery” on page 7.

AUTHID column enhancement

The AUTHID column contains the authorization ID of a monitored user. The AUTHID column can contain a VARCHAR(128) value. Starting in DB2 12, you can monitor all authorization IDs that start with the same prefix by specifying an asterisk (*) at the end of the prefix. For example, if you want to set your profile criteria to all authorization IDs that start with 'USER', specify 'USER*' for the AUTHID column.

LOCATION column enhancement

The LOCATION column contains the IP address of a monitored connection. The LOCATION column can contain a VARCHAR(254) value. Starting in DB2 12, you can specify a LOCATION with the following values:

- ▶ IPv4 subnet address subnet prefix in the form of IPv4address/mm where mm is 8, 16, or 24 and represents the number of initial bits of an IPv4 address that belongs in the subnet.
For example, if you specify a value of 9.30.222.0/24, any IP address in the range 9.30.222.1 to 9.30.222.254 will be monitored.
- ▶ IPv6 subnet address subnet prefix in the form of IPv6address/mmm where mmm is 16, 32, 48, 64, 80, 96, or 112 and represents the number of initial bits of an IPv6 address that belongs in the subnet.
- ▶ The form of either 0.0.0.0 or ::0, which represents any IP address.

PRDID column enhancement

The PRDID column contains the product-specific identifier of a monitored remote requester. The PRDID column is a CHAR(8) value. Similar to the AUTHID column enhancement, starting in DB2 12, you can monitor all product-specific identifiers that start with the same prefix by specifying an asterisk (*) at the end of the prefix. For example, if you want to monitor all versions of DB2 for z/OS, use 'DSN*' for the PRDID column.

8.4.4 Idle thread enhancement

As discussed previously, the SYSIBM.DSN_PROFILE_TABLE contains a row for each profile, the SYSIBM.DSN_PROFILE_ATTRIBUTES table allows you to specify actions for DB2 to take when a process meets the criteria of the profile. The MONITOR IDLE THREADS column in the SYSIBM.DSN_PROFILE_ATTRIBUTES table tells DB2 to monitor, for an approximate amount of time, an active server thread's idle time. The ATTRIBUTE1 column is used to specify the type of messages and level of detail of messages issued for monitored threads. The ATTRIBUTE2 column specifies the threshold the thread is allowed to be idle.

The ATTRIBUTE1 column is enhanced to allow the following values:

- ▶ EXCEPTION_ROLLBACK
- ▶ EXCEPTION_ROLLBACK_DIAGLEVEL1
- ▶ EXCEPTION_ROLLBACK_DIAGLEVEL2

If the threshold is exceeded according to the `ATTRIBUTE2` value, DB2 issues the appropriate type and level of messages, and if database changes occurred, but were not committed, the following actions are performed:

- ▶ The thread is aborted.
- ▶ The database access thread (DBAT) is pooled.
- ▶ The database changes are rolled back.
- ▶ The connection becomes inactive as it is placed in a must-abort states.

Note: DB2 hides the `EXCEPTION_ROLLBACK` event from the remote application environment in either of the following scenarios:

- ▶ The aborted thread performed only read-only operations.
- ▶ The thread committed or aborted but the associated DBAT remained active before it became idle.

It might appear to the remote application that database resources were lost, such as held cursors, held LOB locators, declared global temporary tables, and `KEEPDYNAMIC` sections.

This enhancement to idle threads for `EXCEPTION_ROLLBACK` became available in DB2 11 for z/OS after DB2 11 was made generally available. On a DB2 11 system, the functionality is available after migration to DB2 11 new function mode. On a DB2 12 system, the function becomes available after new function activation. For information about new function activation, see Chapter 2, “Continuous delivery” on page 7.

8.5 Stored procedures supplied by DB2

The following administration routines are changed in DB2 12:

- ▶ `ADMIN_COMMAND_DB2`

This routine supports the `DISPLAY GROUP` command changes for continuous delivery. For more information, see Chapter 2, “Continuous delivery” on page 7.

- ▶ `ADMIN_COMMAND_DSN`

This routine supports the `FREE STABILITY DYNAMIC QUERY` subcommand. For more information, see Chapter 9, “Administrator function” on page 131.

- ▶ `ADMIN_EXPLAIN_MAINT`

This routine upgrades the `EXPLAIN` table definitions for the new release. The suggested approach is to invoke this routine in the migration process with the `STANDARDIZE_AND_CREATE` action to alter the existing `EXPLAIN` tables to conform to DB2 12 format.

You can set the stored procedure input parameters to upgrade each existing `EXPLAIN` table to the current format for DB2 12 and create any new tables needed for `EXPLAIN`:

- `MODE` to 'RUN'
- `ACTION` to 'STANDARDIZE_AND_CREATE'
- `SCHEMA-NAME` to the creator identified by the query

Note: You can set `MODE` to 'PREVIEW' to obtain a report of any changes without processing them.

▶ ADMIN_INFO_SQL

This routine provides a serviceability mechanism to collect information needed for service issues. This stored procedure is enhanced in DB2 12 as follows:

- Collect the EXPLAIN table, DSN_STATEMENT_TABLE, in the EXPL file. This is needed for service cases where the function level information is useful.
- Issue the **ALTER BUFFERPOOL** commands earlier in the process.

▶ GET_CONFIG

This routine supports the **DISPLAY GROUP** command changes for continuous delivery. For more information, see Chapter 2, “Continuous delivery” on page 7.



Part 4

Operations and performance

This part contains the following chapters:

- ▶ Chapter 9, “Administrator function” on page 131
- ▶ Chapter 10, “Security” on page 167
- ▶ Chapter 11, “Utilities” on page 177
- ▶ Chapter 12, “Installation and migration” on page 199
- ▶ Chapter 13, “Performance” on page 231



Administrator function

DB2 12 introduces several features useful for the database administrators. This chapter describes the following functions:

- ▶ Dynamic plan stability
- ▶ Resource limit facility for static SQL
- ▶ Column level deferred alter (pending alter column)
- ▶ Insert partition

9.1 Dynamic plan stability

Dynamic SQL statements allow applications to build SQL statements dynamically at run time. Unlike static SQL statements, the full text of the dynamic SQL statement is not known at the application's bind time. In an ad hoc query environment, performance of dynamic SQL statements is an important priority because enterprise applications use repeating dynamic SQL and they often suffer from instability compared to static SQL. Although the risk of any individual query regressing on any one day is small, exposing thousands of queries to access path changes every day as statistics, the environment, maintenance level, and even release of DB2 changes result in exposure to query performance regression.

To optimize a dynamic SQL statement's performance, the prepared or bound form of the SQL statement is often saved in the in-memory global dynamic statement cache (DSC). Then, for the next request to prepare the dynamic SQL statement for execution, a copy of the bound form of the SQL statement from the cache may be used. However, the storage of the bound form of the dynamic SQL statement in the in-memory cache is not persistent and only provides stabilization for dynamic queries over an instance of a database system. Further, stabilization is not realized across different version of the database system. After the bound form of the SQL statement is deleted from the cache, the dynamic SQL statement must be prepared again. Even when the access plan for the dynamic SQL statement is persistently stored and reused on the next PREPARE SQL, there is no guarantee that the same bound form will result when the PREPARE process is repeated using the stored access plan.

DB2 12 addresses that concern by supporting the ability to stabilize and reuse the runtime structures (the prepared form) for dynamic SQL, extending the stability that was available to only statically bound SQL in DB2 11 to stabilized dynamic SQL.

9.1.1 Stabilization into and loading from catalog tables

Similar to static SQL statements that are bound in packages and saved in the persistent DB2 catalog and directory, DB2 12 saves the prepared runtime structures of dynamic SQL statements and their corresponding EXPLAIN and dependency (object and authorization) information, in the persistent new catalog tables, SYSDYNQRY and SYSDYNQRYDEP. The source statements in the DSC can be saved in those catalog tables through the new **START DYNQRYCAPTURE** command.

During subsequent prepared request of the same dynamic statement, DB2 looks in the DSC. If the statement is not found in the DSC (cache miss), DB2 will look up the SYSDYNQRY for a stabilized statement using the same criteria for DSC look up. If a match is found on the catalog table, a running copy of the statement is made along with its dependency information to insert into the DSC and to be executed by the running thread. This process is called loading a stabilized dynamic statement into the DSC, which bypasses the full PREPARE process altogether.

9.1.2 Stabilization method

When the function level is activated to 500 or above, **START DYNQUERYCAPTURE** (STA DYNQRY for short) can be used to capture qualified statements in the DSC into the catalog tables. The DSNT758 message is issued when attempting to issue the commands before new function activation. This command can be executed with different scopes:

- ▶ **SCOPE(LOCAL)**: Qualified statements in the DSC of the member where the command is executed are handled.
- ▶ **SCOPE(GROUP)**: Qualified statements in the DSC of all active members are handled.

To execute this command, a privilege set of the process must include one of the following authorities:

- ▶ SQLADM authority
- ▶ System DBADM authority
- ▶ SYSOPR authority
- ▶ SYSCTRL authority
- ▶ SYSADM authority

Figure 9-1 shows the **STA DYNQRY** command syntax.

```

>>-START DYNQUERYCAPTURE STBLGRP(stabilization-group) ----->
  .-THRESHOLD(integer-constant)-|-cache-snap-specification-----|
>+-----+-----+-----+-----+-----+-----+-----+-----+<<
  |-SIMTID--(integer-constant)-----+
  '-SIMTKN-(string-constant)-----'

Cache snap specification
>>+-----+-----+-----+-----+-----+-----+-----+-----+<<
  |          .-*-----, | |          .-NO--, | |          .-LOCAL-, |
  '-CURSQLID(+-----+)-' '-MONITOR(-----)-' '-SCOPE(-+-----+)-'
          +-SQLID-+          +-YES-+          +-GROUP-+
  
```

Figure 9-1 The **START DYNQUERYCAPTURE** command

You provide a stabilization group name to logically associate a set of queries that are captured or stabilized. The stabilization group name can be used as input to the **FREE** command to free all the queries for a stabilization group.

Various criteria can be specified to find the statements that are stable and performed well to be captured:

- ▶ Queries that were cached in the DSC under a specific CURRENT SQLID or all and have been executed a number of times (threshold). Note that DB2 increments each statement's execution count only if IFCID318 is enabled. Therefore, to turn on this trace record or TYPE(MONITOR) CLASS(29), then use EXPLAIN STATEMENT CACHE or other methods to determine which statements are to be stabilized. The default threshold value is 2.
- ▶ A query with a specific statement ID or statement token as shown on the EXPLAIN STATEMENT CACHE output or IFCID 316 trace record. These are IDs and tokens in the DSC; hence, only the member's DSC where the command is executed is examined.

The DSNX221I message is issued if the command has no syntax or no authorization error thus is started successfully. An asynchronous service task is scheduled to begin the stabilization process. The DSNX222I message is issued when the stabilization process finishes with summarized information on the numbers of statements that DB2 attempted to capture and actually captured. A command number is displayed on both messages to help coordinate them together. The DSNX223I message is written if another dynamic query capture process is already active for the specified SQLID.

More than one **STA DYNQRY** command can be executed at the same time, and DB2 manages the process so that a query can be stabilized with a stabilization group only. For a data sharing group, the preference is to use the **SCOPE(GROUP)** option to reduce the contention among members. DB2 captures queries in the member where the command starts, then notifies other members to capture queries in their DSC concurrently. Thus, if a query exists on multiple members, the number of scheduled queries for stabilization might be less than the number of queries actually stabilized.

Example 9-1 specifies to DB2 to stabilize queries in the dynamic cache with **CURRENT SQLID** of **ADMF001** and that have been executed at least 50 times.

Example 9-1 The STA DYNQRY command and output

```
-STA DYNQUERYCAPTURE STBLGRP(ABC) THRESHOLD(50) CURSQLID(ADMF001)
DSNX221I -DB2A DSNXESTC DYNAMIC QUERY CAPTURE FOR
COMMAND NUMBER 3 STARTED SUCCESSFULLY.
DSNX222I -DB2A DSNXESC1 DYNAMIC QUERY CAPTURE
COMPLETED FOR COMMAND NUMBER 3 WITH 20 STATEMENTS SCHEDULED,
20 STATEMENTS STABILIZED, AND 0 STATEMENTS ALREADY STABILIZED.
```

Example 9-2 stabilizes all queries in the dynamic cache of each member in the data sharing group that have been executed at least 200 times.

Example 9-2 The STA DYNQRY command and output

```
-STA DYNQUERYCAPTURE STBLGRP(DEF) THRESHOLD(200) SCOPE(GLOBAL)
DSNX221I -DB2A DSNXESTC DYNAMIC QUERY CAPTURE FOR
COMMAND NUMBER 2 STARTED SUCCESSFULLY.
DSNX222I -DB2A DSNXESC1 DYNAMIC QUERY CAPTURE
COMPLETED FOR COMMAND NUMBER 2 WITH 50 STATEMENTS SCHEDULED,
47 STATEMENTS STABILIZED, AND 3 STATEMENTS ALREADY STABILIZED.
```

Limitation

Qualified queries to be captured are those inserted in the DB2 dynamic cache excluding these queries:

- ▶ Queries prepared with **REOPT(AUTO)**
- ▶ Queries prepared with **CONCENTRATE STATEMENT WITH LITERALS**
- ▶ Queries transformed due to referencing System Temporal, Application Temporal, or Archived Transparency table and non-default setting of the **CURRENT SYSTEM TEMPORAL TIME** special register, **CURRENT BUSINESS TEMPORAL TIME** special register, or **GET_ARCHIVE** global variable.

9.1.3 Catalog tables

Dynamic statements are stabilized in the SYSIBM.SYSDYNQRY table with an unique stabilized statement ID and copy ID. The table has the following columns (Table 9-1).

Table 9-1 Catalog tables

Column name	Data type	Description
SDQ_STMT_ID	BIGINT NOT NULL	Stabilized dynamic query statement ID.
STBLGRP	VARCHAR(128) NOT NULL	Stabilization group name
COPYID	SMALLINT NOT NULL	The copy of the stabilized runtime structures for the query in this row: <ul style="list-style-type: none"> ▶ 0 – CURRENT ▶ 4 – INVALID
CURSQLID	VARCHAR (128) NOT NULL	Current SQLID for stabilized dynamic query.
CURSCHEMA	VARCHAR (128) NOT NULL	Current schema for stabilized dynamic query.
CURAPPLCOMPAT	VARCHAR (10) NOT NULL	Current application compatibility for stabilized dynamic query.
QUERY_HASH	CHAR(16) NOT NULL FOR BIT DATA	The hash key generated by statement text.
QUERY_HASH_VERSION	INTEGER NOT NULL	The version of the query hash.
VALID	CHAR(1) NOT NULL	Whether the stabilized dynamic query is valid: <ul style="list-style-type: none"> ▶ N for No ▶ Y for Yes
LASTUSED	DATE NOT NULL	Date query using these runtime structures was last run.
RELBOUND	CHAR(1) NOT NULL	The release when the query was stabilized. See Release dependency indicators.
GROUP_MEMBER	VARCHAR(24) NOT NULL	The data sharing member name that updates the row
STBLTIME	TIMESTAMP NOT NULL	Timestamp when statement was stabilized.
ROWID	ROWID NOT NULL GENERATED ALWAYS	Internal use only.
STMTTEXT	CLOB (2M) NOT NULL	The text of the SQL statement.
DATA1	BLOB(2G) INLINE LENGTH (32329) NOT NULL	Internal use only.
DATA2	BLOB(2G) NOT NULL	Internal use only.
DATA3	BLOB(2G) NOT NULL	Internal use only.
DATA4	BLOB(2G) NOT NULL	Internal use only.
DATA5	VARCHAR(128) NOT NULL	Internal use only.
DATA6	CHAR(8) NOT NULL FOR BIT DATA	Internal use only.

The object and authorization dependency information are stored in the SYSDYNQRYDEP table with the primary key as SDQ_STMT_ID and COPY ID column values to correlate back to rows in the SYSDYNQRY table.

Table 9-2 shows the columns that exist in the SYSDYNQRYDEP table.

Table 9-2 Column names in the SYSDYNQRYDEP table

Column name	Data type	Description
SDQ_STMT_ID	BIGINT NOT NULL	Stabilized dynamic query statement ID.
COPYID	SMALLINT NOT NULL	The copy of the dynamic query in this row. Current version of the dynamic query.
BQUALIFIER	VARCHAR(128) NOT NULL	The value of the column depends on the type of object: <ul style="list-style-type: none"> ▶ If BNAME identifies a table space (BTYPE is R), the value is the name of its database. ▶ If BNAME identifies a table on which a period is defined (BTYPE is B or C), the value is the qualifier of that table. ▶ If BNAME identifies user-defined function, a cast function, a stored procedure, or a sequence (BTYPE is F, O, or Q), the value is the schema name. ▶ If BNAME identifies a role, the value is blank. ▶ Otherwise, the value is the schema of BNAME.
BNAME	VARCHAR(128) NOT NULL	The name of the object that the query depends on.
BTYPE	CHAR(1) NOT NULL	Type of object identified by BNAME and BQUALIFIER: <ul style="list-style-type: none"> ▶ 'A' = Alias ▶ 'E' = INSTEAD OF trigger ▶ 'F' = User-defined function or cast function ▶ 'G' = Created global temporary table ▶ 'H' = Global Variable ▶ 'I' = Index ▶ 'M' = Materialized query table ▶ 'O' = Stored procedure ▶ 'P' = Partitioned table space if it is defined as LARGE or with the DSSIZE parm ▶ 'Q' = Sequence object ▶ 'R' = Table space ▶ 'S' = Synonym ▶ 'T' = Table ▶ 'V' = View ▶ 'W' = SYSTEM_TIME period ▶ 'Z' = BUSINESS_TIME period ▶ '0' (zero) = Sequence alias
CLASS	CHAR(1) NOT NULL	<ul style="list-style-type: none"> ▶ 'A' = Authorization dependency ▶ 'D' = Data Definition Language dependency
BAUTH	SMALLINT NOT NULL	The privilege that is held on the object on which the query is dependent. The privilege only applies when CLASS is 'A': <ul style="list-style-type: none"> ▶ 50 = SELECTAUTH ▶ 51 = INSERTAUTH ▶ 52 = DELETEAUTH ▶ 53 = UPDATEAUTH ▶ 64 = EXECUTEAUTH ▶ 263 = USAGEAUTH ▶ 291 = READAUTH ▶ 292 = WRITEAUTH ▶ 0 = Column is not used

Column name	Data type	Description
AUTHID_TYPE	CHAR(1) NOT NULL	The type of authorization ID indicated by AUTHID. The authorization type only applies when CLASS is 'A': ' ' = Authorization ID 'L' = Role
AUTHID	VARCHAR(128) NOT NULL	The authorization ID or role of the user who holds the privilege on the object on which the query is dependent. The authorization ID only applies when CLASS is 'A'.
DBNAME	VARCHAR(24) NOT NULL	If the value of BADMINAUTH is 'D' (DBADMAUTH), DBNAME contains the name of the database on which the user or role indicated by AUTHID holds DBADM authority. Otherwise the value is blank.
BADMINAUTH	CHAR(1) NOT NULL	The authority that allowed access to the object on which the query is dependent. The admin authority only applies when CLASS is 'A': <ul style="list-style-type: none"> ▶ 'B' = SDBADMAUTH ▶ 'D' = DBADMAUTH ▶ 'G' = ACCESSCTRLAUTH ▶ 'K' = SQLADMAUTH ▶ 'L' = SYSCTRLAUTH ▶ 'S' = SYSADMAUTH ▶ 'T' = DATAACCESSAUTH ▶ ' ' = Authority not held
PUBLICAUTH	CHAR(1) NOT NULL	Whether the privilege or authority is held by PUBLIC. The PUBLIC privilege only applies when CLASS is 'A': <ul style="list-style-type: none"> ▶ 'Y' = Privilege is held ▶ ' ' = Privilege is not held
ALLOBJAUT	CHAR(1) NOT NULL	Whether the privilege is held on all objects within the schema. The all objects privilege only applies when CLASS is 'A': <ul style="list-style-type: none"> ▶ 'Y' = Privilege is held ▶ ' ' = Privilege is not held
QUERY_HASH	CHAR(16) NOT NULL FOR BIT DATA	The hash key of the statement text if the value of CLASS is 'D', otherwise hexadecimal zeros.
	VARCHAR(128) NOT NULL	Internal use only.
	CHAR(8) NOT NULL FOR BIT DATA	Internal use only.

9.1.4 Calculating the EDM statement cache hit ratio

The EDM storage statistics provide information that can help to determine how successful applications are at finding statements in the dynamic cache and in the SYSDYNQRY catalog table.

PREPARE REQUESTS (A) records the number of PREPARE requests.

FULL PREPARES (B) records the number of times that a statement was inserted into the cache after a full PREPARE to derive the access path and build the runtime structures.

LOAD FROM CATALOG (C) records the number of times that a statement was inserted into the cache via the process where a SYSDYNQRY record is used to achieve access paths stability, therefore bypassing the optimization and building of the runtime structure in a full PREPARE process altogether.

To determine how often the dynamic statement was used from the cache, check the value in CACHE HIT RATIO (D).

Procedure to calculate the EDM statement cache hit ratio

The following formula can be used to calculate the cache hit ratio which is the best performance path:

$$(\text{PREPARE REQUESTS} - \text{FULL PREPARES} - \text{LOAD FROM CATALOG}) / \text{PREPARE REQUESTS}$$

To find cache hit ratio and load from catalog hit ratio, the following formula can be used:

$$(\text{PREPARE REQUESTS} - \text{FULL PREPARES}) / \text{PREPARE REQUESTS}$$

Example 9-3 demonstrates that formula.

Example 9-3 The EDM storage statistics

DYNAMIC SQL STMT -----	QUANTITY -----	FORMULA -----
PREPARE REQUESTS A	210225	
FULL PREPARES B	42681	
SHORT PREPARES	167544	(A - B - C)
Short PREPARES based on cache	154592	(A - B - C - C)
Short PREPARES based on catalog C	12952	
CACHE HIT RATIO (%)	73.54	(A - B - C) / A
CATALOG+CACHE HIT RATIO (%)	79.70	(A - B) / A

In the example, **A**, **B**, and **C** are the EDM statistic counters QISED SG, QISED SI, and QISED PSF respectively.

9.1.5 Invalidation of stabilized dynamic statements

When a statement is captured into the SYSDYNQRY table, its COPY ID value is '0' and VALID = 'Y'.

Various actions and events can invalidate stabilized dynamic SQL statements. For example, changing objects that are referenced by the statement, such by issuing ALTER, REVOKE, and DROP statements can invalidate stabilized dynamic SQL statements. The VALID column of the SYSDYNQRY catalog table indicates the validity of stabilized dynamic statements. When these invalidation situations occur, DB2 marks the saved runtime structures for the stabilized access paths as invalid (VALID = 'N' with COPYID remains as '0').

The next time an invalidated statement is prepared, DB2 uses the full prepare process to generate new access paths. Such invalidated statements are not stabilized until another START DYNQUERYCAPTURE command is issued to stabilize them again. At that time, the invalidated query will have copy ID of '4', and the newly stabilized, valid copy is added with copy ID of '0'. The load stabilized dynamic statement process on the PREPARE request uses only the valid copy.

9.1.6 EXPLAIN changes

In DB2 12, cached queries in the DSC keep track of their stabilized statement IDs if a query was captured into or loaded from the SYSDYNQRY table. This ID is externalized in the EXPLAIN STATEMENT CACHE output and IFCID 316 trace record.

DB2 12 also adds a new option on the EXPLAIN statement, STABILIZED DYNAMIC QUERY STMTID, for the ability to retrieve the access path information saved with a specific captured statement.

Figure 9-2 shows the EXPLAIN statement's syntax diagram.

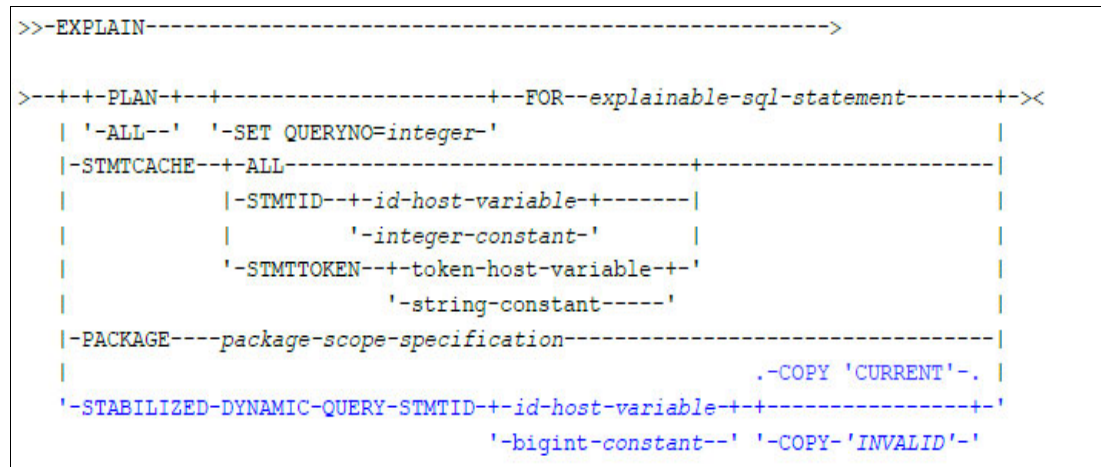


Figure 9-2 The EXPLAIN statement syntax diagram

A COPY 'CURRENT' option will retrieve the row in the SYSDYNQRY table with matching SDQ_STMTID and COPYID of 0 which is the copy usable on a load process. A COPY 'INVALID' option will retrieve the row in the SYSDYNQRY table with matching SDQ_STMTID and COPYID of 4 which is the copy not usable on a load process. This EXPLAIN statement is useful when an access path's comparison is needed such as after certain DDL activities (dropping an index).

Because no QUERYNO exists for this EXPLAIN request, the HINT_USED column in the PLAN_TABLE is populated with the string 'EXPLAIN SDQ: cid', and cid will be one of the following values:

- ▶ "0" is the current copyid.
- ▶ "4" is the invalid copyid.

The QUERYNO column of each EXPLAIN table record that is returned is set to the default value 0, and the COLLID column is set to DSNSTBLQRYEXPLAIN.

Various EXPLAIN tables in DB2 12 also have the new PER_STMT_ID column to correlate back to the SYSDYNQRY.SDQ_STMTID column.

9.1.7 The FREE STABILIZED DYNAMIC QUERY subcommand

As queries are captured into the persistent catalog, the user can monitor usage and remove the statements which are not loaded recently because they are not frequently executed. The SYSDYNQRY.LASTUSED column can be referenced for the above information. A new TSO

FREE STABILIZED DYNAMIC QUERY subcommand is supported to allow a user deleting the captured statement (or statements) either by the ID or stabilization group name.

Note: When freeing multiple statements in a stabilization group, DB2 issues a COMMIT after each statement free. If any of the specified queries are in the dynamic statement cache, **FREE STABILIZED DYNAMIC QUERY** purges them from the dynamic statement cache.

The PLANMGNTSCOPE option can be used to specify all copies or invalid copies (COPYID = '4') are to be freed. The INVALIDONLY option is provided to target the invalid copies (VALID = 'N').

Note: A row with the VALID column of 'N' might or might not have the COPYID of '4'. COPY ID of '4' is updated only when the statement is re-prepared, cached in the DSC and recaptured in the SYSDYNQRY table.

Figure 9-3 shows the **FREE** subcommand syntax.

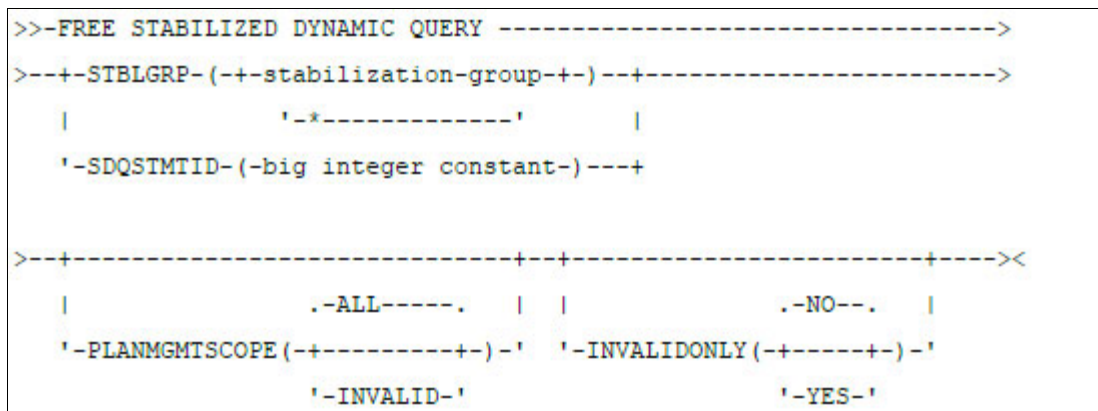


Figure 9-3 The **FREE** subcommand syntax diagram

Example 9-4 indicates how to free the stabilized dynamic query whose ID is 1234 and invalid with COPYID of '4'.

Example 9-4 Free the invalid stabilized dynamic query whose ID is 1234, copy id = '4'

```
FREE STABILIZED DYNAMIC QUERY SDQSTMTID (1234) PLANMGMTSCOPE(INVALID)
```

This new subcommand can also be revoked with the ADMIN_COMMAND_DSN stored procedure. Example 9-5 shows the stored procedure's invocation to free all stabilized dynamic queries for stabilization group APP01.

Example 9-5 The ADMIN_COMMAND_DSN invocation with Free stabilized dynamic queries

```
CALL SYSPROC.ADMIN_COMMAND_DSN('FREE STABILIZED DYNAMIC QUERY STBLGRP(APP01)', ?)
MSG:
ROWNUMText
1      DSNT340I      -DB2A FREE STABILIZED DYNAMIC QUERY
2      COMPLETED SUCCESSFULLY FOR 5 STATEMENTS.
"ADMIN_COMMAND_DSN" RETURN _STATUS: 0
```

9.1.8 Monitor for stabilization

Because the **STA DYNQRY MONITOR(NO)** command performs only a snapshot capture of stabilized dynamic queries, it does not capture all queries for the workload. You must periodically issue the command to capture more queries as they are executed and qualified. The **MONITOR(YES)** option avoids this unreliable work for you and allows DB2 to perform the work continuously without performance overhead to the mainline SQL execution or to overall system resource.

Start the monitor

When a **STA DYNQRY** command is issued with the **MONITOR(YES)** option, partially qualified statements can be monitored to be stabilized later. An example of a partially qualified statement is when the statement matches the **CURRENT SQLID** but the number of executions has not reached the specified threshold value. Several concurrent monitoring capture commands can be started, and DB2 assigns the stabilization group and command number to each partially qualified statement in the DSC. Additionally, as new statements are inserted into the DSC through the full **PREPARE** process, the stabilized dynamic statement monitor process can link them to an active monitor request (if the **CURRENT SQLID** matches).

At every minute interval, DB2 service task checks the DSC for monitored statements meeting all criteria (such as the execution threshold) and stabilizes them into the **SYSDYNQRY** table.

Note: The **STA DYNQRY MONITOR(YES)** command also includes the **MONITOR(NO)** work (taking the snapshot of the DSC and stabilized currently qualified statements) before beginning the monitoring work.

Display the monitors

The new **DISPLAY DYNQUERYCAPTURE** command (**DIS DYNQRY** for short) can be used to display all currently active dynamic query capture monitors. The **DSNX250I** message shows that **DIS DYNQRY** command started, and the display output is followed. The **DSNX260I** message indicates that a long display output continues from the previous output. Figure 9-4 shows the command syntax.

```
>>-DISPLAY DYNQUERYCAPTURE -----+-----+----->
                                     | .-*-----|
                                     'CNO(+---integer---+)-'
```

Figure 9-4 The **DISPLAY DYNQUERYCAPTURE** command syntax diagram

Stop the monitor

The new **STOP DYNQUERYCAPTURE** command stops the specified active dynamic query capture monitoring request (or requests). Figure 9-5 shows the command syntax.

```
>>-STOP DYNQUERYCAPTURE -----+-----+-----+----->
| ,-'-----, | | ,-'LOCAL-', |
| 'CNO(+---integer---+)-' | '-SCOPE-(+-----+)-'+
|                                     | '-GROUP-'
```

Figure 9-5 The **STOP DYNQUERYCAPTURE** command syntax

The CNO value can be obtained from the command number shown in the DSNX2211 message issued on the **STA DYNQRY** command or the **DIS DYNQRY** command output.

Use case

The following use case shows how the monitoring process works for multiple commands and how the **DIS DYNQRY** command can be used to track progress. Assume that two applications exist, application A and application B, with the dynamic SQL statements that are listed in Example 9-6.

Example 9-6 *Dynamic SQL statements for application A and application B*

```
Application A:  SET CURRENT SQLID = SCHEMA1
                SELECT * FROM SCHEMA1.T1
                INSERT INTO SCHEMA1.T1 SELECT FROM SCHEMA2.T2
Application B:  SET CURRENT SQLID = SCHEMA2
                UPDATE SCHEMA2.T2 SET C1 -1 WHERE C2 = 'ABC'
                DELETE FROM SCHEMA2.T2 WHERE C2 = 'ABC'
```

Assume also that applications A and B are executed twice, and the runtime structures of the statements in Example 9-6 are inserted into the DSC with a number of executions of 2.

Assume further that the following first stabilize command is issued to start the stabilization for dynamic query statement with monitoring capability:

```
- START DYNQUERYCAPTURE GROUP(GRPA) CURSQLID(SCHEMA1) THRESHOLD(2) MONITOR(YES)
```

With this command, the **SELECT** and **INSERT** statements from application A are stabilized into the DB2 catalog because they match both criteria **CURRENT SQLID** and execution threshold. Example 9-7 shows the output when a **DIS DYNQRY** command is issued after.

Example 9-7 *Output of issuing a DIS DYNQRY command*

```
-DISPLAY DYNQUERYCAPTURE CNO(*)
*** BEGIN DISPLAY DYNAMIC QUERY CAPTURE CNO(*)
=====
CNO          : 1
STBLGRP      : GRPA
SCHEMA       : SCHEMA1
THRESHOLD    : 2
STABILIZED   : 2
=====
*** END DISPLAY DYNAMIC QUERY CAPTURE
```

Next, the following second stabilize command is issued to start stabilization with monitoring capability:

```
- START DYNQUERYCAPTURE GROUP(GRPB) CURSQLID(SCHEMA2) THRESHOLD(10) MONITOR(YES)
```

On this command, DB2 links the UPDATE and DELETE statements in the DSC to the second monitor request. Example 9-8 shows the output when a **DIS DYNQRY** command is issued after.

Example 9-8 Output of issuing a DIS DYNQRY command

```
-DISPLAY DYNQUERYCAPTURE CNO(*)
*** BEGIN DISPLAY DYNAMIC QUERY CAPTURE CNO(*)
=====
CNO          : 1
STBLGRP     : GRPA
SCHEMA      : SCHEMA1
THRESHOLD   : 2
STABILIZED  : 2
-----
CNO          : 2
STBLGRP     : GRPB
SQLID       : SCHEMA2
THRESHOLD   : 10
STABILIZED  : 0
=====
*** END DISPLAY DYNAMIC QUERY CAPTURE
```

Assume that application B is running again and its UPDATE statement is executed 8 more times. The monitor service task finds that the UPDATE statement fully satisfies the filtering criteria of the second monitoring request since the number of executions is now 10 and stores the bound form of the UPDATE statement in the persistent catalog. The command in Example 9-9 can then be issued to display pending monitoring requests.

Example 9-9 Display pending monitoring requests

```
- DISPLAY DYNQUERYCAPTURE CNO(*)
*** BEGIN DISPLAY DYNAMIC QUERY CAPTURE CNO(*)
=====
CNO          : 1
STBLGRP     : GRPA
SCHEMA      : SCHEMA1
THRESHOLD   : 2
STABILIZED  : 2
-----
CNO          : 2
STBLGRP     : GRPB
SQLID       : SCHEMA2
THRESHOLD   : 10
STABILIZED  : 1
=====
*** END DISPLAY DYNAMIC QUERY CAPTURE
```

Note: For the command number (CNO) 2, the output now shows the progress of 1 (one) stabilized statement.

To stop a pending monitoring request, a **STOP DYNQUERYCAPTURE** command can be issued. For example, assume that at this time, a **STOP DYNQUERYCAPTURE CNO(2)** command is issued for the second monitoring command. In response, the monitoring request for SCHEMA2 is deleted and the link from application B's DELETE statement is also broken. The DELETE statement can be subsequently executed but it will not be stabilized.

Next, assume further that a new application C executes the following dynamic SQL statement twice:

```
INSERT INTO SCHEMA1.T1 SELECT * FROM SCHEMA1.T2
```

As this new statement is prepared and its runtime structures inserted into the DSC, a check is made against the monitoring requests, and a link is established with the monitoring request 1 because the new statement satisfies the CURRENT SQLID criteria. When the execution threshold of 2 is reached, DB2 automatically stabilizes this statement. Example 9-10 shows that the output displays the updated progress in the monitoring requests.

Example 9-10 Display the updated progress in monitoring requests

```
-DISPLAY DYNQUERYCAPTURE CNO(*)
*** BEGIN DISPLAY DYNAMIC QUERY CAPTURE CNO(*)
=====
CNO          : 1
STBLGRP     : GRPA
SCHEMA      : SCHEMA1
THRESHOLD   : 2
STABILIZED  : 3
=====
*** END DISPLAY DYNAMIC QUERY CAPTURE
```

9.1.9 DSNZPARM and installation panel

A new subsystem parameter, CACHEDYN_STABILIZATION, is introduced for specifying how DB2 should stabilize cached dynamic SQL. The CACHEDYN_STABILIZATION DSNZPARM is externalized as CACHE DYN STABILITY on the DSNTIP8 installation panel (Figure 9-6 on page 145).


```

DSNTIP8          INSTALL DB2 - PERFORMANCE AND OPTIMIZATION
===>

Enter data below:
 1 CACHE DYNAMIC SQL      ===> YES      NO or YES
 2 CACHE DYN STABILITY    ===> BOTH     CAPTURE, LOAD, BOTH, or NONE
 3 OPTIMIZATION HINTS     ===> NO      Enable optimization hints. NO or YES
 4 EVALUATE UNCOMMITTED   ===> NO      Evaluate uncommitted data. NO or YES
 5 SKIP UNCOMM INSERTS    ===> NO      Skip uncommitted inserts. NO or YES
 6 IMMEDIATE WRITE        ===> NO      NO or YES

 7 PLAN MANAGEMENT        ===> EXTENDED OFF, BASIC, EXTENDED
 8 PLAN MANAGEMENT SCOPE  ===> STATIC  ALL, STATIC, DYNAMIC
 9 PACKAGE RELEASE COMMIT===> YES      Permit BIND and REBIND at COMMIT time
                                on packages bound as RELEASE(DEALLOCATE)

10 RANDOMIZE XML DOCID    ===> NO      NO or YES
11 DISABLE EDM RTS        ===> NO      Disable EDM real time stats. NO or YES

PRESS:  ENTER to continue  RETURN to exit  HELP for more information

```

Figure 9-6 The DSNTIP8 installation panel

Consider the following information about the parameter:

- ▶ **CACHE DYN STABILITY:** Specify how DB2 is to stabilize cached dynamic SQL statements. When a statement is stabilized, the CURRENT SQLID, statement text, and runtime structures are written to catalog tables. When a dynamic SQL statement is not present in the dynamic SQL statement cache, DB2 will load the runtime structures if available from the SYSIBM.SYSDYNQUERY catalog table rather than performing a full prepare. This extends the stability and reliability of performance of dynamic SQL.
- ▶ **Range:** CAPTURE, LOAD, BOTH, NONE
 - **CAPTURE:** Statements may be stabilized through the **-START DYNQUERY** command with both **MONITOR(NO)** and **MONITOR(YES)**. DB2 will not load stabilized statements from SYSDYNQUERY.
 - **LOAD:** Statements may not be stabilized through any means. The **-START DYNQUERY** command will fail, and any **MONITOR(YES)** commands in progress will not stabilize statements, even if stabilization criteria are matched. During long prepare, DB2 will attempt to load stabilized statements from SYSDYNQUERY with which to run.
 - **BOTH:** This is the default setting. Statements may be stabilized through the **-START DYNQUERY** command through both **MONITOR(NO)** and **MONITOR(YES)**. During long prepare, DB2 will attempt to load stabilized statements from SYSDYNQUERY with which to run.
 - **NONE:** Statements may not be stabilized through any means. The **-START DYNQUERY** command will fail, and any **MONITOR(YES)** commands in progress will not stabilize statements, even if stabilization criteria are matched. DB2 will not load stabilized statements from SYSDYNQUERY.

- ▶ Default: BOTH
- ▶ Data sharing scope: All members should use the same setting
- ▶ Online changeable: Yes
- ▶ DSNZPxxx: DSN6SPRM.CACHEDYN_STABILIZATION

9.2 Resource limit facility for static SQL

Traditionally, the DB2 resource limit facility (RLF) is used to govern dynamic SQL statements so they do not unexpectedly consume too much resources such as locks, CPU, storage, I/O, and so on, due to unanticipated access path change, DASD hardware degradation, or poorly coded SQL. Abrupt poor performance when a static SQL statement in application runs away in DB2 and accumulates many resources is also an issue that could negatively impact online transactions in a production system.

Transaction Managers such as IBM CICS® and IMS have functions that control transactions consuming resources in an IMS or CICS database. However, when the CICS or IMS transaction invokes a static SQL statement running in DB2 for z/OS, the resource limit control function does not apply. In a high volume online transaction processing environment, the ability to proactively cancel poorly running DB2 threads can help avoid severe degradation of an LPAR.

DB2 12 extends the RLF functionality to support static SQL statements to avoid unanticipated application's elapsed time as well as to control system's resource consumption.

9.2.1 Reactive governing static SQL

You can define the RLF tables by using the unique naming convention DSNTRLST_{xx} and DSNTRLMT_{xx} where *xx* is an ID number. These RLF tables allow you to specify the limit amount of processor resources, in service units, used by dynamic SQL statements. This type of control function is called *reactive governing* because DB2 can interrupt the execution of the qualified SQL statement and return a negative SQLCODE when the limit is reached.

DB2 12 enhances the reactive governing function of the RLF tables so that you can limit resources used by static SQL statements too. To use RLF, the you insert rows into a resource limit table with values that identify the context of the governed statements, the type of governing, and threshold limits. The limits specified in the RLF tables apply to individual dynamic or static SQL statements that qualify for a defined scope. You can specify different function codes in the RLF tables to indicate whether static or dynamic SQL statements are to be governed with the limit when they are executed. To insert a row for reactive governing on static SQL statements, you can specify 'A' in the RLFFUNC column and the amount of processor limit in ASUTIME column. The PLANNAME column has to be blank.

Note: Other functions are provided with RLF too, such as the predictive governing function, query parallelism control function, and bind limit function. These function codes can be specified in the RLF tables and are still applicable to dynamic SQL statements only.

Similar to dynamic SQL, only the following static statements can be reactively governed:

- ▶ SELECT (cursor and singleton),
- ▶ INSERT
- ▶ UPDATE (search and position)
- ▶ MERGE
- ▶ TRUNCATE
- ▶ DELETE (search and position)

The change shown in Table 9-3 and Table 9-4 is applicable to the RLFFUNC column of the user DSNRLSTxx and DSNRLMTxx tables supplied by DB2.

Table 9-3 RLFFUNC column in RLF DSNRLSTxx table

Column name	Column type	Description
RLFFUNC	CHAR(1)	<p>Specifies how the row is used. These values have an effect:</p> <ul style="list-style-type: none"> ▶ '1' = The row reactively governs bind operations. ▶ '2' = The row reactively governs dynamic SELECT, INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statements by package or collection name. ▶ '4' = The row disables query CP parallelism. ▶ '7' = The row predictively governs dynamic SELECT, INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statements by package or collection name. ▶ 'A' = The row reactively governs static SELECT (cursor and singleton), INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statements by package or collection name. <p>All other values are ignored.</p>

Table 9-4 RLFFUNC column in RLF DSNRLMTxx table

Column name	Column type	Description
RLFFUNC	CHAR(1)	<p>Specifies how the row is used. These values have an effect:</p> <ul style="list-style-type: none"> ▶ '8' = The row reactively governs dynamic SELECT, INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statements by client information (RLEUID, RLFEUAN, RLFEUWN, and RLFIP). ▶ '9' = The row predictively governs dynamic SELECT, INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statements by client information (RLEUID, RLFEUAN, RLFEUWN, and RLFIP). ▶ 'B' = The row reactively governs static SELECT (cursor and singleton), INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statements by client information (RLEUID, RLFEUAN, RLFEUWN, and RLFIP). <p>All other values are ignored.</p>

After a **START RLF** command is issued referencing these tables, the changes to the resource limit table become effective immediately for all new threads.

While a thread is executing and using a limit, DB2 can detect that a filtering criteria can change and apply the corresponding limit:

- ▶ When the primary user is changed
- ▶ When the client information is changed
- ▶ When a package is loaded for execution

Also, while a thread is executing and using a limit, a new limit may be updated in the RLF tables. For the changed limit to be effective for such thread, RLF must be restarted.

When a row in the RLF table matches the currently executing static statement, the row's ASUTIME value is used to limit the statement's execution. The static statement's execution can be stopped with the SQLCODE -905 when the consumed resource exceeds the specified ASUTIME limit. Note that SQL requests related to a cursor such as OPEN, FETCH, CLOSE can accumulate CP resources on the same SELECT statement. When no row in the RLF table matches the currently executing static statement, DB2 uses the default limit value that is set in the RLST ACCESS ERROR DSNZPARM for static SQL on installation panel DSNTIPO4 which appears after panel DSNTIPO3.

9.2.2 Use cases

Two use cases are presented here.

Use case 1

These are the steps for using the DB2 resource limit facility to govern static SQL statements:

1. Insert rows into a DSNRLSTxx resource limit table with values that identify the context of the governed statements (which application the statements are in, which client connection the statements are from, which primary authorization ID executes the statements, and more), the type of governing, and CP limits. Assume the rows are inserted (Table 9-5).

Table 9-5 Rows in the DSNRLSTxx table

RLFFUNC	AUTHID	PLANNAME	RLFCOLLN	RLFPKG	LUNAME	ASUTIME
A	JOE	(blank)	COL1	(blank)	(blank)	(null)
2	JOE	(blank)	COL1	(blank)	(blank)	15000
A	(blank)	(blank)	(blank)	PKG2	PUBLIC	10000
2	(blank)	(blank)	(blank)	(PKG2)	PUBLIC	20000

The first row indicates that when user JOE runs any package in the collection COL1, at the local location, no limit restricts any static statement in the package because the ASUTIME column value is null.

The second row shows that when user JOE runs any package in the collection COL1, at the local location, each dynamic statement in the package is restricted to 15,000 SUs.

The third row reflects that when any user runs package PKG2, in any collection from any location in the network, including the local location, a processor limit of 10,000 SUs is applied for each static statement in the package.

The last row shows that when any user runs package PKG2, in any collection from any location in the network, including the local location, a processor limit of 20,000 SUs is applied for each dynamic statement in the package.

2. Issue the **START RLIMIT ID=xx** command, where *xx* is the two-character identifier that was specified when the RLF table was created. You can start and stop different resource limit tables at different times. However, only one resource limit table of each type (DNSRLMTxx or DSNRLSTxx) can be used at any given time.
3. When SQL statements are run in a package, DB2 uses the following search order:
 - a. Exact match
 - b. Authorization ID
 - c. Plan name, or collection name and package name
 - d. LU name
 - e. No row match

Note: A DB2 11 behavior is that the CP time for a dynamic cursor can include CP time incurred by a dynamic positioned UPDATE and DELETE (WCO) statement against the same cursor. A static cursor can also be used to update or delete a particular row with a static or dynamic positioned UPDATE or DELETE statement. When the ASUTIME limits for static and dynamic SQL specified in the RLF table differ, a dynamic positioned UPDATE and DELETE statement will use the dynamic limit, and a static positioned UPDATE and DELETE statement will use the static limit.

The following examples show the SQL statements in different packages that qualified for the previous DSNRLSTxx table example and their corresponding limits governed by DB2 (Table 9-6).

Table 9-6 Dynamic and static SQL in packages and RLF limits used

Package name	SQL request	Applied limit in service units
COL1.X	DCL CURSOR C1 FOR SELECT...	
	OPEN/FETCH C1	Infinite is used to govern (accumulate into C1)
	Static UPDATE WHERE CURRENT OF C1	Infinite is used to govern (accumulate into C1)
	Dynamic UPDATE WHERE CURRENT OF C1	15000 is used to govern (accumulate into C1)
X.PKG2	DCL CURSOR C2 FOR STMID PREPARE STMID FOR SELECT...	
	OPEN/FETCH C2	10000 is used to govern (accumulate into C2)
	Dynamic UPDATE WHERE CURRENT OF C2	10000 is used to govern (accumulate into C2)
X.PKG3	DCL CURSOR C3 FOR SELECT...	
	OPEN/FETCH C3	10000 is used to govern (accumulate)
	Static UPDATE WHERE CURRENT OF C3	10000 is used to govern (accumulate into C3)
	Dynamic UPDATE WHERE CURRENT OF C3	20000 is used to govern (accumulate into C3)

Use case 2

Assume that the following subsystem parameters for RLF are set as shown here:

- ▶ RLFENABLE = ALL, to govern both static and dynamic SQL
- ▶ RLFERRSTC = 5000, as the default limit for locally originated static SQL
- ▶ RLFERRDSTC = 8000, as the default limit for remotely originated static SQL
- ▶ RLFERR = 10000, as the default limit for locally originated dynamic SQL
- ▶ RLFERRD = NOLIMIT, as the default limit for remotely originated dynamic SQL

The DSNRLST01 table (Table 9-7) and DSNRLMT01 table (Table 9-8) are activated for RLF.

Table 9-7 DSNRLST01 table

RLFFUNC	AUTHID	PLANNAME	RLFCOLLN	RLFPKG	LUNAME	ASUTIME
	JOE	(blank)	COL1	PKG1	(blank)	2000
	JOE	(blank)	COL1	PKG1	(blank)	800
	(blank)	(blank)	(blank)	PKG2	PUBLIC	500
	DAVID	(blank)	(blank)	(blank)	PUBLIC	600

Note that in Table 9-7, the blank value of the LUNAME column is for locally originated SQL, and the PUBLIC value is for both locally and remotely originated SQL.

Table 9-8 DSNRLMT01 table

RLFFUNC	RLFEUAN	RLFEUID	RLFEUWN	RLFIP	ASUTIME
	APP1	(blank)	(blank)	(blank)	1500
	APP1	(blank)	(blank)	(blank)	700

The steps are as follows:

1. The following static query is issued from the locally executed package COL3.PKG3, by the authorization ID FRANK:

```
DCL CURSOR C1 FOR SELECT SALARY FROM DSN8C10.EMP;
OPEN C1;
FETCH C1;
```

The default limit, 5000 service unit, for the locally originated static SQL is applied for cursor C1 because no matching row is found for the thread from either the DSNRLMT01 table or the DSNRLST01 table.

2. The following static query is issued from the package COL4.PKG4, which is remotely executed in the middleware application, APP4, by the authorization ID FRANK:

```
DCL CURSOR C2 FOR SELECT SALARY FROM DSN8C10.EMP;
OPEN C2;
FETCH C2;
```

The default limit, 8000 service unit, for the remotely originated static SQL is applied for cursor C2 since there is no matching row found for the thread either from the DSNRLMT01 table or DSNRLST01 table.

3. The following static SQL, which updates a static positioned row, is issued from the locally executed package, COL1.PKG1, by the authorization ID JOE:

```
DCL CURSOR C3 FOR SELECT SALARY FROM DSN8C10.EMP;  
OPEN C3;  
FETCH C3;  
UPDATE DSN8C10.EMP SET SALARY = SALARY + (SALARY * 0.1) WHRE CURRENT OF C3;
```

The 800 service unit limit, is applied for the static UPDATE statement that references static positioned cursor C3 because the second row in the DSNTRLST01 table matches the thread for static SQL.

4. The following static SQL, which updates a dynamic positioned row, is issued from the locally executed package COL1.PKG1, by the authorization ID JOE:

```
DCL CURSOR C4 FOR S4;  
STMT4 = 'SELECT SALARY FROM DSN8C10.EMP';  
PREPARE S4 FROM :STMT4;  
OPEN C4;  
FETCH C4;  
UPDATE DSN8C10.EMP SET SALARY = SALARY + (SALARY * 0.1) WHRE CURRENT OF C4;
```

The 2000 service unit limit, is applied for the static UPDATE statement that references the dynamic positioned cursor C4 because the first row in the DSNTRLST01 table matches the thread for dynamic SQL.

Note: When a dynamic or static statement contains an external user-defined function, the execution time for the user-defined function is not included in the ASUTIME of the RLF-governed dynamic or static statement's execution. The ASUTIME for a user-defined function's execution is controlled based on the ASUTIME option specified for the user-defined function in the CREATE FUNCTION statement.

9.2.3 RLF DSNZPARMs and installation panels

Because DB2 12 supports different limits for governing dynamic and static SQL statements, you might want to activate RLF for dynamic as in DB2 11 only, or static only, or both. The new RLFENABLE DSNZPARM is introduced for the selection mentioned. Two new DSNZPARMs are also added so the user can specify the action taken in the case DB2 cannot access the RLF tables.

Two subsystem parameters for RLF are added to DSN6SYSP:

- ▶ RLFENABLE: Specifies the level of RLF governing when RLF is started.
- ▶ RLFERRSTC: Specifies what action DB2 is to take for static SQL statements when the Resource Limit Facility governor encounters a condition that prevents it from accessing the resource limit specification table. This setting also applies if DB2 cannot find an applicable row in the resource limit specification table. It is equivalent to the existing RLFERR parameter which pertains to dynamic SQL statements.

Also, a subsystem parameter for RLF is added to DSN6FAC:

- ▶ RLFERRDSTC: Specifies what action DB2 is to take for static SQL statements from a remote location when the Resource Limit Facility governor encounters a condition that prevents it from accessing the resource limit table. This setting also applies if DB2 cannot find an applicable row in the resource limit table. It is equivalent to the existing RLFERRD parameter of which pertains to dynamic SQL statements from a remote location.

A new installation panel, DSNTIPO4 is introduced specifically for RLF-related subsystem parameters. It appears after panel DSNTIPO3 (Figure 9-7).

```

DSNTIPO4          INSTALL DB2 - RESOURCE LIMIT FACILITY
===>

Enter data below:
 1 RLF AUTO START    ---> NO          Resource Limit Facility. NO or YES
 2 RLF SCOPE         ---> DYNAMIC     DYNAMIC only, STATIC only, or ALL
 3 RLST NAME SUFFIX ---> 01          Resource Limit Spec. Table (RLST)

Enter actions for RLST access errors:
 4 DYNAMIC SQL      ---> NOLIMIT     NOLIMIT, NORUN, or 1-5000000
 5 STATIC SQL       ---> NOLIMIT     NOLIMIT, NORUN, or 1-5000000

Enter actions for RLST access errors for workload from a remote location:
 6 REMOTE DYNAMIC SQL ---> NOLIMIT     NOLIMIT, NORUN, or 1-5000000
 7 REMOTE STATIC SQL  ---> NOLIMIT     NOLIMIT, NORUN, or 1-5000000

PRESS:  ENTER to continue  RETURN to exit  HELP for more information

```

Figure 9-7 New DSNTIPO4 installation panel for Resource Limit Facility

Summary of fields is as follows:

- ▶ The RLF AUTO START and RLST NAME SUFFIX fields were relocated directly from panel DSNTIPO. They correspond to the DSN6SYSP.RLF and DSN6SYSP.RLFTBL parameters.
- ▶ The DYNAMIC SQL field was also relocated from DSNTIPO where it was named RLST ACCESS ERROR. It corresponds to the DSN6SYSP.RLFERR parameter.
- ▶ The REMOTE DYNAMIC SQL field was relocated here from panel DSNTIPR (Distributed Data Facility panel 1) where it was named RLST ACCESS ERROR. It corresponds to the DSN6FAC.RLDERRD parameter.
- ▶ The RLF SCOPE field is new and corresponds to the DSN6SYSP.RLFENABLE parameter.
- ▶ The STATIC SQL field is also new and corresponds to the DSN6SYSP.RLFERRSTC parameter.
- ▶ The REMOTE STATIC SQL field is also new and corresponds to the DSN6FAC.RLFERRDSTC parameter.

Only the three following new fields are discussed next:

- ▶ RLF SCOPE
- ▶ STATIC SQL
- ▶ REMOTE STATIC SQL

RLF SCOPE

Specify the level of RLF governing when RLF is started.

- ▶ Range: DYNAMIC, STATIC, ALL
 - DYNAMIC, the default, means that RLF will govern only dynamic SQL statements.
 - STATIC means that it will govern only static SQL statements.
 - ALL means that it will govern both types.
- ▶ Default: DYNAMIC
- ▶ Data sharing scope: All members should use the same setting
- ▶ Online changeable: Yes
- ▶ DSNZPxxx: DSN6SYSP.RLFENABLE

STATIC SQL

Specify what action DB2 is to take for static SQL statements when the Resource Limit Specification governor encounters a condition that prevents it from accessing the Resource Limit Specification table (RLST). This setting also applies if DB2 cannot find an applicable row in the RLST.

- ▶ Range: NOLIMIT, NORUN, or an integer from 1 to 5000000
 - NOLIMIT, the default, means that RLF will allow all static SQL statements to run without limit.
 - NORUN means that RLF will terminate all static SQL statements immediately with an SQL error code. An integer setting of 1 - 5000000 specifies the number of service units that RLF will use as the default resource limit for all static SQL statements. If the limit is exceeded, the SQL statement is terminated.
- ▶ Default: NOLIMIT
- ▶ Data sharing scope: It is recommended that all members use the same setting
- ▶ Online changeable: Yes
- ▶ DSNZPxxx: DSN6SYSP.RLFERRSTC

REMOTE STATIC SQL

Specify what action DB2 is to take for static SQL statements from a remote location when the Resource Limit Specification governor encounters a condition that prevents it from accessing the Resource Limit Specification table (RLST). This setting also applies if DB2 cannot find an applicable row in the RLST.

- ▶ Range: NOLIMIT, NORUN, or an integer from 1 to 5000000
 - NOLIMIT, the default, means that RLF will allow all static SQL statements from a remote location to run without limit.
 - NORUN means that RLF will terminate all static SQL statements from a remote location immediately with an SQL error code. An integer setting of 1 - 5000000 specifies the number of service units that RLF will use as the default resource limit for all static SQL statements from a remote location. If the limit is exceeded, the SQL statement is terminated.
- ▶ Default: NOLIMIT
- ▶ Data sharing scope: It is recommended that all members use the same setting
- ▶ Online changeable: Yes
- ▶ DSNZPxxx: DSN6FAC.RLFERRDSTC

9.3 Column level deferred alter (pending alter column)

In DB2 11, the capability exists to alter a table column's attribute, such as its data type, precision, scale, or length through the ALTER TABLE ALTER COLUMN statement as an immediate alteration. However, this function impacts the system in several ways:

- ▶ Some column alterations result in indexes being placed in a restrictive status. If a unique index is placed in restrictive status, it results in an outage to the table.
- ▶ Column alterations invalidate dependent packages so the SQL referencing the columns can be rebound or autobound. When dependent packages are rebound or go through autobind, indexes in restrictive status are not candidates in selecting access path. Hence, a suboptimal access path may be used instead.
- ▶ After the column is altered, the column definition changes are immediately reflected in the catalog, but not in the data. Subsequent access to the column data results in data on the retrieved rows being converted from the old definition format to the new definition format, which incurs a performance overhead until the table is reorganized to convert all data to the new definition format.
- ▶ If pending alterations for the containing table space or table are not yet materialized, executing a subsequent immediate alteration (such as column alteration) fails. To resolve this situation, one of the following actions is needed:
 - A REORG utility must be run to materialize the pending changes first, followed by an immediate column alteration, followed by another REORG to convert to the new definition format.
 - The order of alterations must be changed so that the immediate alterations are performed prior to the pending alterations.

To address the ways that the system is impacted, DB2 12 enhances the alter column attribute process to limit data unavailability, reduce impact to access paths, and allow ease of scheduling schema alterations and REORG activities. You have a choice to execute the ALTER TABLE ALTER COLUMN statement as an immediate or pending change. Other DDL pending changes have been supported since DB2 10 such as ALTER TABLE ALTER PARTITION, ALTER TABLE DROP COLUMN, ALTER TABLESPACE DSSIZE, ALTER INDEX BUFFERPOOL, and others. When the ALTER TABLE ALTER COLUMN statement is a pending change in DB2 12, the actual alteration will be done during the REORG utility execution, and therefore, the following benefits can be observed:

- ▶ The new column definition is reflected in both the catalog and data at the same time. There is no conversion needed when accessing data and hence no performance impact incurred as a result of such conversion.
- ▶ Indexes are no longer placed in any pending states. They will be rebuilt during the materializing REORG utility.
- ▶ Dependent packages are invalidated during the REORG SWITCH phase, after indexes have already been rebuilt with the new definition.
- ▶ All pending alterations can be grouped together to be materialized by a single REORG activity of the table space.

Example 9-11 shows a partitioned table created followed by a pending alteration of a column data type issued, and the resulting row in the SYSPENDINGDDL table (Figure 9-8).

Example 9-11 Pending alteration with the SYSPENDINGDDL row

```
CREATE TABLE SC.TB1
  (COLUMN1 INTEGER,
   COLUMN2 CHAR(100),
   COLUMN3 VARCHAR(100))
  IN DB1.TS1;
CREATE INDEX SC.IX1 ON SC.TB1(COLUMN1);
ALTER TABLE SC.TB1 ALTER COLUMN COLUMN1
  SET DATA TYPE BIGINT; -> SQLCODE +610
```

SYSPENDINGDDL Field	Value
DBNAME	'DB1'
TSNAME	'TS1'
DBID	<i>DBID of DB1</i>
PSID	<i>PSID of DB1</i>
OBJSHEMA	'SC'
OBJNAME	'TB1'
COLNAME	'COLUMN1'
OBJJOBID	<i>OBID of SC.TB1</i>
OBJTYPE	'T'
STATEMENT_TYPE	'A'
COLUMN_KEYWORD	'ALTER'
OPTION_KEYWORD	'SET DATA TYPE'
OPTION_VALUE	'BIGINT' (Note: data type spelling is normalized per specification in PFS)
STATEMENT_TEXT	'ALTER TABLE SC.TB1 ALTER COLUMN COLUMN1 SET DATA TYPE BIGINT'

Figure 9-8 SYSPENDINGDDL table

DDL_MATERIALIZATION, a new ZPARM, is introduced to indicate whether eligible column alterations are executed as immediate or as pending alterations.

Valid ZPARM values:

- ▶ ALWAYS_IMMEDIATE (default)
- ▶ ALWAYS_PENDING

This ZPARM only applies when all of the following conditions are met:

- ▶ The APPLCOMPAT bind option (for static ALTER) or the CURRENT APPLICATION COMPATIBILITY special register (for dynamic ALTER) is set to V12R1M500 or higher.
- ▶ The ALTER COLUMN SET DATA TYPE statement is altering the column's data type, length, precision, or scale. The following alter items are ineligible to be pending:
 - Altering the inline length of a LOB column
 - Altering the subtype of a column
 - Altering the XML type modifier of an XML column
- ▶ The underlying table space is a Universal Table Space (UTS) or pending alter exists to convert to UTS.
- ▶ The underlying table space data sets are defined.

Figure 9-9 shows the different behaviors between an immediate alteration and a pending alteration based on the zparm DDL_MATERIALIZATION.

Differences Between Immediate and Pending Alter Column		
Description (DDL_MATERIALIZATION Setting)	Immediate Alteration (ALWAYS_IMMEDIATE)	Pending Alteration (ALWAYS_PENDING)
Mixture of immediate and pending-eligible options in the same ALTER statement	The ALTER is executed as an immediate change.	The ALTER fails.
Unmaterialized pending changes exist for the table (and related objects) to be altered	If there are existing unmaterialized pending changes, the ALTER fails.	The ALTER is executed as a pending change.
Method of materialization	REORG TABLESPACE (SHRLEVEL NONE, REFERENCE, or CHANGE) or LOAD REPLACE can all be used to materialize the column alterations.	Only Online REORG (SHRLEVEL REFERENCE or CHANGE) can be used to materialize the column alterations.
Scope of materialization	Alteration can be materialized in the data by partition-level REORG.	Alteration can only be materialized by a tablespace-level REORG.
When the changes are materialized	Alteration is materialized in the catalog and OBD at time of ALTER. Alteration is materialized in the data at REORG time.	Alteration is materialized in the catalog, OBD, and data at REORG time.
Object status after the ALTER statement is processed	Depending on the type of alteration, the table space may be left in AREO* or REORP status, and affected indexes may be left in AREO* or RBDP/PSRBD status.	Table space is left in AREOR status.
Generation of schema versions for the table	ALTERs against tables within the same table space that are executed in the same commit scope receive a single new version number. ALTERs executed in different commit scopes receive different version numbers.	ALTERs against tables within the same table space that are materialized in the same REORG receive a single new version number. ALTERs materialized in different REORGs receive different version numbers.

Figure 9-9 Differences between an immediate alteration and a pending alteration

Figure 9-10 describes the status for the table space and its index when an allowed alteration (of data type or data length) is done as immediate or pending.

		ALWAYS_IMMEDIATE	ALWAYS_PENDING
Status quo COLUMN	ALTER COLUMN SET DATA TYPE	Object status	Object status
CHAR(20)	CHAR(30)	TS – AREO*, IX – AREO*	TS – AREOR, IX – nothing
VARCHAR(20) Padded IX	VARCHAR(30)	TS – nothing, IX – AREO* TS – AREO* (if already versioned)	TS – AREOR, IX – nothing
VARCHAR(20) Not padded IX	VARCHAR(30)	TS – nothing, IX – nothing TS – AREO* (if already versioned)	TS – AREOR, IX – nothing
CHAR(20)	VARCHAR(30)	TS – AREO*, IX – AREO*	TS – AREOR, IX – nothing
VARCHAR(20) Padded IX	CHAR(30)	TS – AREO*, IX – AREO*	TS – AREOR, IX – nothing
VARCHAR(20) Not padded IX	CHAR(30)	TS – AREO*, IX – AREO*, PSRBD	TS – AREOR, IX – nothing
VARCHAR(20) no index	VARCHAR(30)	TS – nothing TS – AREO* (if already versioned)	TS – AREOR
DEC(15, 0)	DEC (24, 0)	TS – AREO*, IX – PSRBD	TS – AREOR, IX – nothing
CHAR(20) FOR BIT DATA	BINARY(20)	TS – AREO*, IX – PSRBD	TS – AREOR, IX – nothing

Figure 9-10 Object status difference between immediate and pending alterations

Figure 9-11 shows how the order of pending alteration requests can make a difference (successful or not).

	ALWAYS_IMMEDIATE	ALWAYS_PENDING
ALTER TABLESPACE ... DSSIZE 16G	SQLCODE+610 AREOR	SQLCODE+610 AREOR
ALTER TABLE ... ALTER COLUMN SET DATA TYPE ...	SQLCODE-20385 REASON 2	SQLCODE+610 AREOR

Figure 9-11 Fail pending column alteration

The immediate column alteration request fails because the earlier ALTER TABLESPACE DSSIZE statement is a pending request (regardless of the zparm DDL_MATERIALIZATION value) and has not been materialized yet. Switching to pending request by setting the zparm DDL_MATERIALIZATION = ALWAYS_PENDING can fix this failure because both pending alterations are stackable.

However, when the order of the two alter requests are switched, both statements will be successful with the zparm DDL_MATERIALIZATION set to ALWAYS_PENDING or ALWAYS_PENALWAYS_IMMEDIATE. The ALTER TABLE ALTER COLUMN statement is done first under DDL_MATERIALIZATION = ALWAYS_IMMEDIATE and is an immediate alteration which does not impede the next pending ALTER TABLESPACE statement (Figure 9-12).

	ALWAYS_IMMEDIATE	ALWAYS_PENDING
ALTER TABLE ... ALTER COLUMN SET DATA TYPE ...	AREO*	SQLCODE+610 AREOR
ALTER TABLESPACE ... DSSIZE 16G	SQLCODE+610 AREO*, AREOR	SQLCODE+610 AREOR

Figure 9-12 Successful ALTER requests due to order of execution

The pending column alteration enhancement does not allow any new alterations, and only the possible existing ones are able to be deferred alterations. Some current restrictions still exist on column level alterations such as length reduction (conversion from BINARY to CHAR FOR BIT DATA data type), and change of a LOB column's inline length. Restrictions related to pending alters in general also apply to pending alter column.

New restrictions apply if alter column is executed as a pending change:

- ▶ An ALTER INDEX statement with the NOT PADDED clause where the index references a column with pending definition
- ▶ A CREATE TABLE or ALTER TABLE statement that specifies a FOREIGN KEY referencing a parent column with pending definition changes

9.3.1 Utility

The REORG materialization of alteration in the data can be done at a partition level for immediate alter (flagged by the AREO* status). However, the REORG SHRLEVEL REFERENCE or CHANGE utility to materialize alteration in the data must be done at a table space level for pending alter (flagged by the AREOR status).

The LOAD REPLACE utility does materialize changes for immediate alter (flagged by the AREO* status). This utility does not materialize changes for pending alter (flagged by the AREOR status). Online REORG on the complete table space must be run as indicated above.

The RECOVERY and RECOVERY to point-in-time utility time prior to an immediate column alteration are supported.

In addition to deferred ALTER processing (for example, cleanup AREOR), the online REORG utility also performs the following actions:

- ▶ Regenerates views, triggers, masks and permissions.
- ▶ Invalidates dependent package invalidation and dynamic statement cache.
- ▶ Updates versioning counter where only one new version number is generated.
- ▶ The DISCARD option discards rows in the materialized format with new schema definition.
- ▶ If the table space is in Basic Row Format (BRF), it will be converted to Reordered Row Format (RRF) regardless of ROWFORMAT keyword or the RRF ZPARM.
- ▶ If the STATISTICS keyword not specified, the default option used is STATISTICS TABLE ALL, INDEX ALL, UPDATE ALL, HISTORY ALL.

9.3.2 ALTER INDEX

An alteration of index to and from COMPRESS is a pending alteration independent of the zPARM DDL_MATERIALIZATION. Before DB2 12, the index is set to the PSRBD status. Such pending index change is materialized by the REORG, not by the REBUILD utility.

An alteration of index to NOT PADDED is not allowed if pending changes on column exists. The reason is that the materializing REORG might generate a key larger than 2000 bytes due to 2-byte length of NOT PADDED key entries. Without pending changes, the alteration to NOT PADDED is allowed but sets the PSRBD status on the index.

An alteration of index to PADDED is always allowed but sets the PSRBD status on the index. The reason this ALTER INDEX TO PADDED statement is allowed is because the keys are fully expanded and can only shrink by the 2-byte length of NOT PADDED key entries.

9.4 Insert partition

In DB2 10, partitions in a partitioned table may be rearranged by rotating them around using the ALTER TABLE ROTATE PARTITION 'n' TO LAST statement. This support provided some capability for user to modify a table's partition configuration. In addition, a user can also add a new partition as the last partition at the end of the table via the ALTER TABLE ADD PARTITION statement with ENDING AT x where x must be the highest limit key value.

DB2 12 extends the support to dynamically adding a partition in the middle of the table for greater flexibility so that the logical partition numbers are arranged as how the data with the limit key values are supposed to be stored. This functionality provides usability while maximizing availability of the altered object.

9.4.1 ALTER ADD PARTITION

Two SQL statements are available that can insert a new partition in the middle of the table. Both give the same behavior:

- ▶ The user specifies the high limit key (x) of the newly inserted partition:

```
ALTER TABLE
    ADD PARTITION ENDING AT x
```

This syntax is the same syntax as existed in DB2 10 when adding a partition at the end. With DB2 12, the limit key x value can be any value so the new partition can be added in the middle of the table.

- ▶ The user specifies the high limit key (x) of the newly inserted partition, along with the high limit key (y) of the existing partition (n as the physical partition number for the subsequent logical partition) that will be affected by the insert operation:

```
ALTER TABLE
    ADD PARTITION ENDING AT x
    ALTER PARTITION n ENDING AT y
```

This syntax is new in DB2 12 for family compatibility and ensures the user specifies the intended place where the new partition is to be added.

Inserting a partition in the middle of the table is supported when the APPLCOMPAT bind option (for static ALTER) or the CURRENT APPLICATION COMPATIBILITY special register (for dynamic ALTER) is set to V12R1M500 or greater.

The keyword INCLUSIVE is implied by default even if it is not stated explicitly. The partition specified in the ALTER PARTITION clause needs to be the very next logical partition to the partition being added in the ADD PARTITION clause. In addition, the high limit key value specified in the ALTER PARTITION clause must be the existing high limit key value for the very next logical partition. The high limit key value cannot be altered in the same statement while inserting a new partition.

A new first partition will be added if the specified high limit key for the inserted partition is lower than the existing lowest limit key in the table.

The newly inserted partition will be physically added at the end to obtain a new physical partition number. It will be assigned a new logical partition number based on the location inserted and all the logical partition numbers after that will be renumbered by incrementing by 1.

The adding partition request is considered a *pending* change when there is an existing pending change or when both of the following items are true:

- ▶ A partition is added in the middle of the table.
- ▶ The data sets of the table space are already created.

Otherwise, the change is considered an *immediate* change, for example, adding a partition at the end or the table space has the DEFINE(NO) attribute. Adding the last partition does not affect existing data in existing partitions. When the data sets for a table space that has not yet been defined, the table's redefinition does not need to wait either. If the change is considered an immediate change, the change to the description of the table takes effect immediately.

If the change is a pending change to the definition of the table, the changes are not reflected in the definition or data at the time the ALTER TABLE statement is issued. Instead, the affected partitions are placed in an advisory REORG-pending state (AREOR).

Figure 9-13 shows a table with four partitions ending at keys 250, 500, 750, and 1000. It shows how the physical and logical partitions are arranged if any of the following statements are executed:

- ▶ ALTER TABLE ADD PARTITION ENDING AT 1250 statement
- ▶ ALTER TABLE ROTATE PARTITION 2 TO LAST ENDING AT 1250 statement
- ▶ ALTER TABLE ADD PARTITION ENDING AT 400 statement

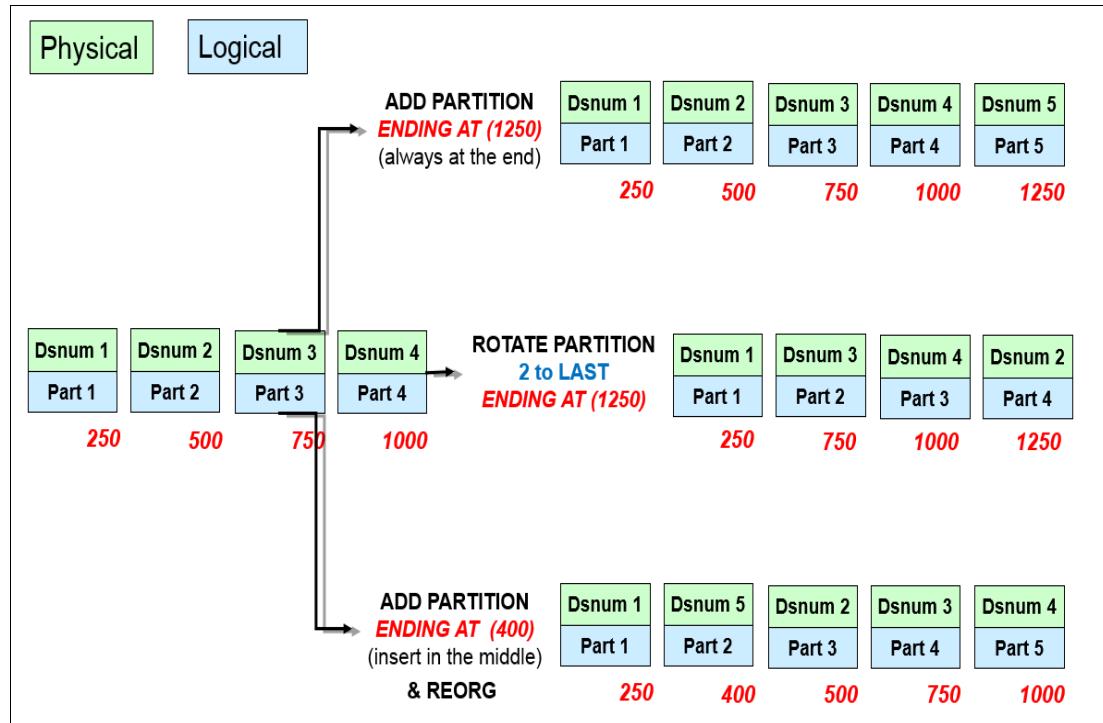


Figure 9-13 Adding a partition at the end or middle of the table, or rotating a partition

The following restrictions are implemented:

- ▶ The table must be a ranged-partition table (PBR and PBR Relative Page Number are both supported).
- ▶ The newly inserted partition's limit key value under the ADD PARTITION clause is not the same as an existing partition's limit key.
- ▶ The partition specified in the ALTER PARTITION clause must be the very next logical partition to the partition being added in the ADD PARTITION clause.
- ▶ The high limit key value specified in the ALTER PARTITION clause must be the existing high limit key value for the very next logical partition, which can be obtained from the catalog table. The high limit key value cannot be altered in the same statement while inserting a new partition.
- ▶ If any outstanding unmaterialized alter limit key pending definition changes exist on the last partition of the table, insert partition will not be allowed in the same table until the pending alter limit key changes are materialized by a REORG execution.
- ▶ Adding a partition to the end of the table is not allowed if any outstanding pending definition changes exist for the table space or objects within the table space.
- ▶ The table cannot contain a LOB column, a distinct type column that is based on a LOB data type, or an XML column.

Note that only one new partition to be inserted can be specified in a single ALTER TABLE statement. However, multiple insert partition requests can be submitted with multiple ALTER statements, one partition per statement, and all those pending requests will be materialized by a single materializing REORG utility execution (Example 9-12).

Example 9-12 Multiple pending insert partitions in the same table are materialized

```
ALTER TABLE T1 ADD PARTITION ENDING AT 15
ALTER TABLE T1 ADD PARTITION ENDING AT 25
REORG TABLESPACE SHARELEVEL REFERENCE
```

9.4.2 Utilities affected

REORG and RECOVER are affected utilities.

REORG

In order to materialize the pending definition change for inserting partition, a partition-level REORG utility for the affected partition (or partitions) must be executed. The REORG utility must be online (SHRLEVEL REFERENCE or SHRLEVEL CHANGE). The new SYSDDLPENDING.REORG_SCOPE_LOWPART and SYSDDLPENDING.REORG_SCOPE_HIGHPART columns represent the logical partition range of the object's definition prior to any pending definition changes being materialized. If there are multiple SYSPENDINGDDL entries for multiple insert partition pending definition changes and some of them get materialized first but not all of them, these two fields get updated during the previous materializing REORG processing to reflect the most current logical partition number range.

After converting this logical partition range to physical partition range, this is the partition range that the user must include in the partition-level REORG to materialize this particular pending definition change. If this partition range is not included in the partition-level REORG execution, the REORG processing will proceed and a new warning message DSNU2918I with return code 4 will be issued to indicate that the utility was run on a subset of affected partitions and not all pending definition changes on the partition-level are applied.

If any additional table space level pending definition changes were issued on the table space while some partitions are in advisory-REORG (AREOR) pending status due to inserting a new partition, a table space level REORG is required to materialize all the pending definition changes together at once.

If the partitions that are in pending status in the table space is due to insert partition pending definition changes only, a table space level REORG with SCOPE PENDING option could also be used to reorganize affected partitions and materialize the insert partition pending definition changes.

If adjacent partitions are affected by either alter limit key or insert partition partition-level pending definition changes, these adjacent partitions are required to be reorganized together in order to materialize the partition-level pending definition changes. If the affected adjacent partitions are not materialized together, none of the pending definition changes would be materialized. In addition, a table space level REORG with SCOPE PENDING option can also be used to reorganize affected partitions and materialize the pending definition changes.

While a partition-level REORG on the affected partitions can materialize insert partition pending definition changes, by either PART specification or SCOPE PENDING, the underlying REORG needs to quiesce the entire partitioned table space. The complete table space is drained in the LOG phase due to partition numbers and ordering changes. Also, packages and dynamic statements dependent on the table space will be invalidated.

The REORG utility may collect new statistics for the newly inserted and affected partitions. For the newly inserted partition (or partitions), statistics will be collected with STATISTICS TABLE ALL INDEX ALL UPDATE ALL HISTORY ALL options, unless you explicitly specify the STATISTICS keyword with different options.

For the affected partition (or partitions), it is recommended to collect all statistics inline and to have a profile defined for the table, and specify REORG STATISTICS TABLE USE PROFILE in the materializing REORG execution for the recollection of the complete set of statistics. If recalculation of statistics is not done, a warning message DSNU1166I with return code 4 can be issued to indicate that some partition statistics might no longer be accurate because they have become obsolete. The partition statistics that might be obsolete are COLGROUP statistics, KEYCARD statistics, HISTOGRAM statistics, frequency statistics with NUMCOLS > 1, and statistics for extended indexes where applicable. You should execute the RUNSTATS utility to collect the partition statistics again after the REORG utility.

RECOVER

The RECOVER utility to point-in-time cannot be run to a point before the materializing REORG. The error message DSNU556I with return code 8 will be issued if such time is specified. The REORGE utility can be executed to materialize the pending changes before recovery to a point in time. DB2 12 does insert the SYSOBDS records for point-in-time recovery in the future. A MODIFY RECOVERY utility does a cleanup of these records.

9.4.3 Catalog changes

Catalog changes are described in this section.

SYSIBM.SYSTABLESPACE

When a new partition is added at the end of the table, the existing table space PRIQTY and SECQTY attributes of the previous logical partition are used for the space attributes of the new partition. When a new partition is inserted in the middle of the table, the existing table space PRIQTY and SECQTY attributes from table space level stored in SYSTABLESPACE are used for the space attributes of the new partition. For this usage, DB2 12 added new columns in SYSIBM.SYSTABLESPACE.

In addition, for newly created objects, the values for other table space attributes specified on the CREATE TABLESPACE statement, or default values for unspecified options are saved in new SYSTABLESPACE columns. These values are saved in SYSTABLESPACEPART in DB2 11. In DB2 12, these values are also saved in SYSTABLEPART so that insert partition can inherit these table space values. The column values can also be populated when an ALTER TABLESPACE statement on these attributes occurs at the global level.

For migrated objects, values for these new columns, which are still NULL, will be populated when any DDL changes (insert partition, alter partition, rotate partition, conversion from index-controlled to table-controlled partitioning, table space type conversion, and others) occur on the table space, and it will inherit values from the last logical partition (same as the existing add partition behavior). The effect takes place immediately for immediate alters and during materialization for pending definition changes.

The following new columns are added:

- ▶ PQTY
- ▶ STORTYPE
- ▶ STORNAME
- ▶ VCATNAME
- ▶ FREEPAGE

- ▶ PCTFREE
- ▶ COMPRESS
- ▶ GBPCACHE
- ▶ TRACKMOD
- ▶ SECQTYI
- ▶ PCTFREE_UPD
- ▶ PCTFREE_UPD_CALC

SYSIBM.SYSINDEXES

The following new nullable columns will be added to the SYSINDEXES catalog table to store default values for partition attributes at the index space level. These columns also exist in the SYSINDEXPART catalog table in DB2 11. In DB2 12, the new columns added to SYSINDEXES have their data type and description identical to those in SYSINDEXPART except the new columns in SYSINDEXES are nullable.

The following new columns are added:

- ▶ PQTY
- ▶ STORTYPE
- ▶ STORNAME
- ▶ VCATNAME
- ▶ FREEPAGE
- ▶ PCTFREE
- ▶ GBPCACHE
- ▶ SECQTYI

SYSIBM.SYSPENDINGDDL

Two new nullable SMALLINT columns, REORG_SCOPE_LOWPART and REORG_SCOPE_HIGHPART, are added to the SYSPENDINGDDL catalog table to store the low logical partition range number and the high logical partition range number for which REORG needs to include in the partition-level REORG execution to materialize insert partition pending definition changes. In DB2 12, values for these two columns will be populated for all pending definition changes.

Figure 9-14 shows a table that has three partitions (L for logical partition and P for physical partition) ending at 20, 40, and 60 key values.

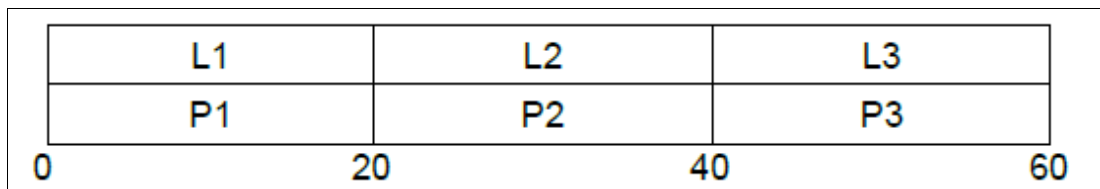


Figure 9-14 Table with 3 partitions

When an insert partition request with limit key of 30 is issued, the partitions and the SYSDDLPENDING row look like this example (see Figure 9-15 on page 165 and Figure 9-16 on page 165):

```
ALTER TABLE ADD PARTITON ENDING AT 30
```

L1	L2	L3	L4
P1	P4	P2	P3

0 20 30 40 60

Figure 9-15 Adding a logical partition with key 30

OBJNAME	OBJTYPE	OPTION_KEYWORD	OPTION_VALUE	PARTITION	PARTITION_KEYWORD	REORG_SCOPE_LOWPART	REORG_SCOPE_HIGHPART
TB01	T	ENDING AT	30	0	ADD	2	2

Figure 9-16 SYSDDLPENDING row

This row has the columns SYSDDLPENDING.REORG_SCOPELOWPART = 2 and SYSDDLPENDING.REORG_SCOPEHIGHPART = 2 because only the existing logical partition number 2 is affected (it will become logical partition 3 and rows will be moved to the new partition).

Then a REORG utility is executed with affected partition number 2 to materialize the pending change. Now the table has the new logical partition 3 with key values from 31 to 40. Next, another insert partition request with limit key of 35 is issued. The partitions and the SYSDDLPENDING row look like this example (see Figure 9-17 and Figure 9-18):

```
ALTER TABLE ADD PARTITON ENDING AT 35
-> SYSDDLPENDING.OPTION_VALUE = 35,
```

L1	L2	L3	L4	L5
P1	P4	P5	P2	P3

0 20 30 35 40 60

Figure 9-17 Adding logical partition with key 35

OBJNAME	OBJTYPE	OPTION_KEYWORD	OPTION_VALUE	PARTITION	PARTITION_KEYWORD	REORG_SCOPE_LOWPART	REORG_SCOPE_HIGHPART
TB01	T	ENDING AT	35	0	ADD	3	3

Figure 9-18 SYSDDLPENDING row

This row has the columns SYSDDLPENDING.REORG_SCOPELOWPART = 3 and SYSDDLPENDING.REORG_SCOPEHIGHPART = 3 because only the existing logical partition number 3 is affected (it will become logical partition 4 and rows will be moved to the new partition).

If no REORG utility was run to materialize the first insert partition request (where limit key was at 30), the two rows shown in Figure 9-19 would be inserted in the SYSDDLPENDING table.

OBJNAME	OBJTYPE	OPTION_KEYWORD	OPTION_VALUE	PARTITION	PARTITION_KEYWORD	REORG_SCOPE_LOWPART	REORG_SCOPE_HIGHPART
TB01	T	ENDING AT	30	0	ADD	2	2
TB01	T	ENDING AT	35	0	ADD	2	2

Figure 9-19 SYSDDLPENDING table

Only the original logical partition 2 is affected for both pending changes.



Security

This chapter covers the following DB212 security topics:

- ▶ Installation or migration without requiring SYSADM
- ▶ UNLOAD privilege
- ▶ Object ownership transfer

10.1 Installation or migration without requiring SYSADM

Up to DB2 11, only users with installation *SYSADM* authority can install new a DB2 subsystem or data sharing group or migrate DB2 to the new release. However, a system operator (with installation *SYSOPR* authority) usually is the person performing the installation or migration steps, which means he or she is granted *SYSADM* authority. *SYSADM* authority also includes access to all data. Certain government regulations and policies require that sensitive user data not be exposed to anyone, except the system administrator or data owner. Therefore, to help protect user data and comply with security regulations, DB2 12 provides a way to install or migrate a DB2 subsystem without having *SYSADM* authority.

With DB2 12 compatibility mode, a user with installation *SYSOPR* authority has the capability to install and migrate a subsystem with no access to user data. Enhancements are made so that *SYSOPR* authority can perform work necessary in the process such as these examples:

- ▶ Execute the **CATMAINT** utility to install or migrate to a new release.
- ▶ Issue the **ACTIVATE FUNCTION LEVEL** command.
- ▶ Access to all catalog tables and all tables in the system databases.
- ▶ Set current SQLID to *SYSINSTL*, regardless of *SEPARATE SECURITY DSNZPARM* setting.
- ▶ Use *BINDAGENT* privilege to specify any owner. The *BINDAGENT* privilege also gives the ability to free any package, and bind or free a plan.

When executing the migration or installation jobs, system objects in the DB2 catalog might need to be created by the user with installation *SYSOPR* authority, but these objects will be owned by *SYSINSTL*. For this to work, the current SQLID must be set to '*SYSINSTL*'. The runner with installation *SYSOPR* now has the authority to use *CREATE*, *ALTER*, and *DROP* on the following objects:

- ▶ Database
- ▶ Table space
- ▶ Table, auxiliary table, created global temporary table
- ▶ Index
- ▶ Stogroup
- ▶ Trigger
- ▶ Procedure, function including those in schema *SYSTOOLS* and *SYSFUN*

Note: If procedure and function have an associated package, then the *OWNER* keyword must be specified. If the stored procedures are created with the *SECURITY DEFINER* clause, *SYSINSTL* must be defined in Resource Access Control Facility (IBM RACF®).

Additionally, *SYSINSTL* with installation *SYSOPR* authority now have the ability to *CREATE*, *ALTER*, and *DROP* several objects without any additional privileges such as alias, distinct type, sequence, global variable. *SYSINSTL* can also grant privileges on the following system objects and resources:

- ▶ All database, table space, and table privileges on objects in the *DSNDB04* database, and *DSNRGFDB*, *DSNRLST*, *DSNOPTDB*, *DSNMDCDB*, *DSNADMDB*, *DSNATPDB*, *DSN5JSDB*, *DSNMQDB*, *SYSIBMTA*, *SYSIBMTS*, and *DSNXSR* system databases
- ▶ The *USE* privilege on buffer pool and storage group
- ▶ All privileges on plans that begin with '*DSN*'
- ▶ All privileges on packages where the collection-ID and package-name begin with '*DSN*'
- ▶ The execute privilege on system-defined routines

On the installation panel DSNTIPG shown in Figure 10-1, the ROUTINES CREATOR, SEC DEF CREATOR, and INSTALL SQL ID fields must be set to SYSINSTL. The INSTALL PKG OWNER field on this panel also must be set to an authorization ID (PKOWNER in this example) that has been granted system DBADM and DATAACCESS authorities. This is necessary because the **BIND** and **REBIND** commands issued during the installation and migration processes require an owner that has authorization to bind and execute all the SQL statements in the package.

```

DSNTIPG                MIGRATE DB2 - INSTALLATION PREFERENCES
===> _

Enter authorization IDs for installing DB2-supplied routines:
_ 1 ROUTINES CREATOR  ===> SYSINSTL  Authid to create and bind DB2 routines
_ 2 SEC DEF CREATOR   ===> SYSINSTL  Authid for routines w/ SECURITY DEFINER

Enter authorization IDs for other installation jobs:
_ 3 INSTALL SQL ID    ===> SYSINSTL  To process SQL in install jobs
_ 4 INSTALL PKG OWNER ===> PKOWNER   To own packages bound by install jobs
_ 5 INSTALL GRANTEE(S)===> PUBLIC    > To be granted access on objects
                                         created by install jobs

Enter authorization IDs for IVP jobs:
_ 6 IVP SQL ID        ===> SYSADM    To process SQL in IVP jobs
_ 7 IVP PACKAGE OWNER ===>          To own packages bound by IVP jobs
_ 8 IVP GRANTEE(S)    ===> PUBLIC    > To be granted access on objects
                                         created by IVP jobs

PRESS:  ENTER to continue  RETURN to exit  HELP for more information

```

Figure 10-1 DSNTIPG panel with Install SQL ID set to SYSINSTL

10.2 UNLOAD privilege

Up to DB2 11, the SELECT privilege is the lowest required privilege to execute the **UNLOAD** utility. Even with the SELECT privilege, if the table has column masks or row permission defined, a user might not be able to retrieve certain rows and columns by using an application that issues the SELECT SQL statement. However, such a user with the SELECT privilege has the ability to read all data in the table using the **UNLOAD** utility. Furthermore, the SQL SELECT statement, static or dynamic, can be governed by the DB2 resource limit facility (RLF) to control how many rows an application with those SQL statements can retrieve. RLF limits, row permissions, and column masks do not apply to utilities. Executing an **UNLOAD** utility with only the SELECT privilege can be considered a security vulnerability because a user can unload and have access to large amounts of data beyond the user's intended authorization. Also, when running the **UNLOAD** utility, a user with the SELECT privilege can leave a restricted state on the object in case the utility experiences an abend condition and is not terminated.

DB2 12 closes this security exposure by introducing the UNLOAD privilege, which is required when executing the **UNLOAD** utility. This DB2 UNLOAD privilege provides separation between the SQL **SELECT** and utility execution, thus the security administrator will have better control to grant the appropriate authorization to intended usage.

10.2.1 Enforcing new privilege

The UNLOAD privilege can be established for users starting in DB2 12 compatibility mode (or function level V12R1M100). However, this privilege is mandated for the **UNLOAD** utility only after new function mode is activated (function level V12R1M500 or greater).

The **SELECT** privilege can still be used for the **UNLOAD** utility access in DB2 12 after new function is activated, if AUTH_COMPATIBILITY DSNZPARM is set to the SELECT_FOR_UNLOAD option. The default value for this DSNZPARM is NULL. Prior to activating function level V12R1M500 or greater, the new serviceability trace record IFCID 404 may be enabled to collect the authorization IDs that use the **SELECT** privilege to execute an **UNLOAD** utility. This trace record is also retrofitted to DB2 11 so the auditing work can be done before migration to DB2 12 also. The preference is to activate IFCID 404 *prior to* migrating to DB2 12 with the PTF for APAR PI55706. Actions must be taken for those authorization IDs so the **UNLOAD** utility will work as intended in the DB2 12 new function mode.

10.2.2 Using DB2 security facility

The following SQL statement syntax diagrams show how to grant (Figure 10-2) the UNLOAD privilege to a user ID and revoke (Figure 10-3 on page 171) the privilege.

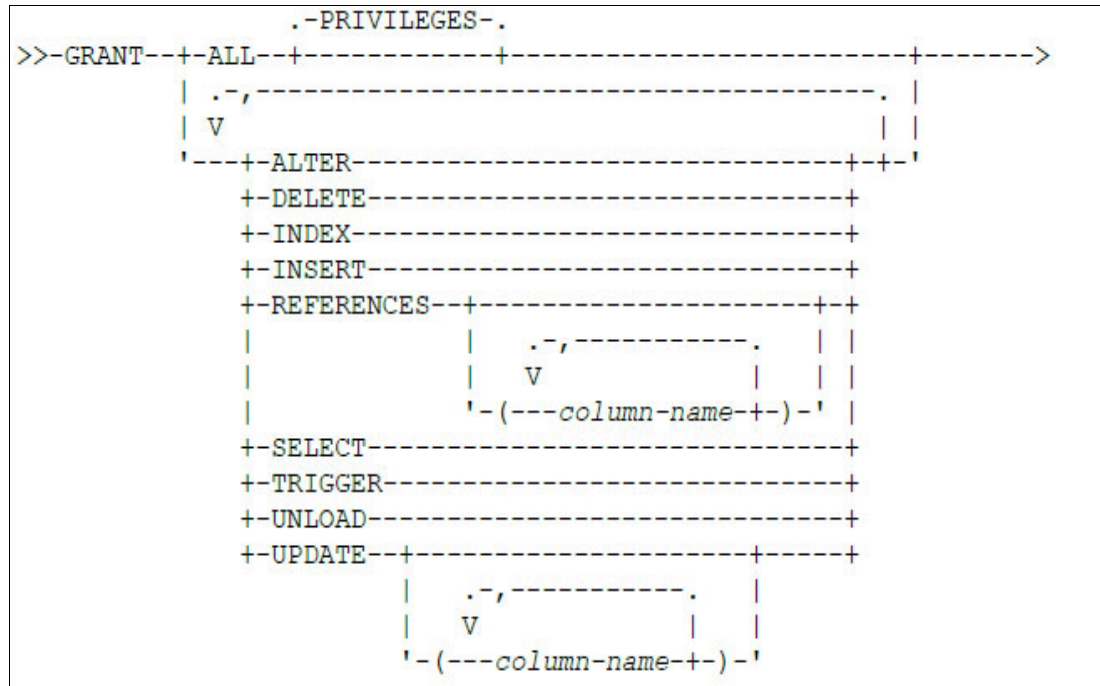


Figure 10-2 The GRANT statement syntax with UNLOAD privilege

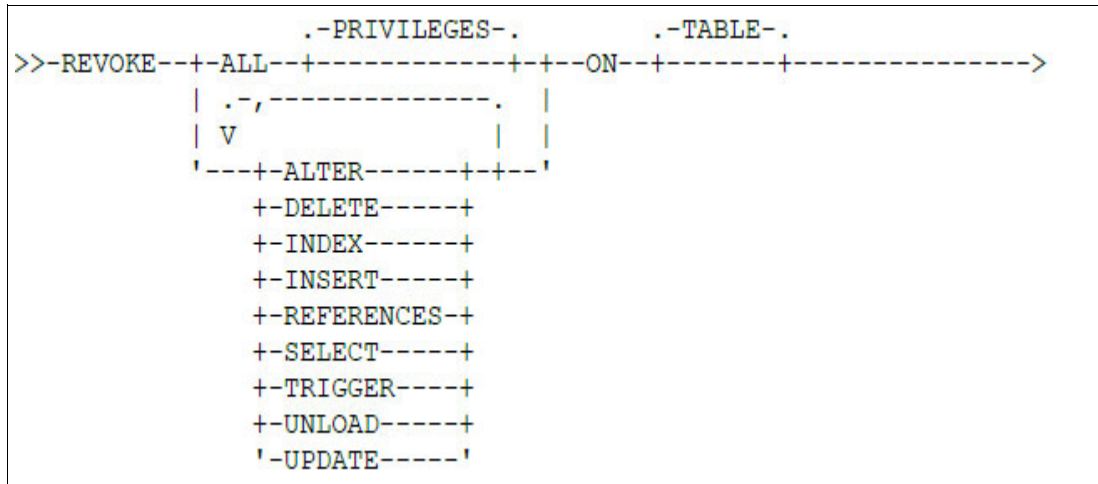


Figure 10-3 The REVOKE statement syntax with UNLOAD option

A new column UNLOADAUTH is added to the SYSIBM.SYSTABAUTH catalog table to record the successful result of the GRANT statement and the authorization ID having the UNLOAD privilege in the GRANTEE column (Example 10-1; the result is shown in Table 10-1).

Example 10-1 Granting the UNLOAD privilege to USRT001 and the SELECT privilege to USRT002

```

GRANT UNLOAD ON TABLE T1 TO USRT001;
GRANT SELECT ON TABLE T1 TO USRT002;
SELECT GRANTEE, TTNAME, SELECTAUTH, UNLOADAUTH FROM SYSIBM.SYSTABAUTH WHERE
(GRANTEE = 'USRT001' OR GRANTEE = 'USRT002') AND TTNAME = 'T1';

```

Table 10-1 Result of query

GRANTEE	TTNAME	SELECTAUTH	UNLOADAUTH
USRT001	T1	Y	
USRT002	T1		Y

10.2.3 Using Resource Access Control Facility (RACF)

If RACF is used for security management, then a new profile must be added for the UNLOAD privilege in the RACF class. Example 10-2 shows a couple of RACF profiles defined for the UNLOAD privilege (given to user USER001) and SELECT privilege (given to both users USERT001 and USER002).

Example 10-2 Permitting UNLOAD and SELECT privileges to USER001 and only the SELECT privilege to USER002

```

RDEFINE MDSNTB DB2A.EMPLOYEE.TABLE01.UNLOAD UACC(NONE) -OWNER(DB2OWNER)
PERMIT DB2A.EMPLOYEE.TABLE01.UNLOAD ID(USER001) ACCESS(READ) - CLASS(MDSNTB)
RDEFINE MDSNTB DB2A.EMPLOYEE.TABLE01.SELECT UACC(NONE) -OWNER(DB2OWNER)
PERMIT DB2A.EMPLOYEE.TABLE01.SELECT ID(USER001 USER002) -ACCESS(READ)
CLASS(MDSNTB)
SETR RACLIST(MDSNTB) REFRESH

```

If a RACF access control module is written, the module should be updated to handle the new ULOADAUTT constant value (297 or x'129') in the XAPLPRIV field of the RACF parameter list. The RACF access control exit should have logic to handle whether the user or the role associated with the user owns the table:

- ▶ If *yes*, then XAPLUPRM must match the owner name passed from DB2 by the XAPLOWNR parameter when XAPLONRT indicates an authorization ID, or XAPLUCHK must match XAPLOWNR and XAPLUCKT must match XAPLONRT.
- ▶ If *no*, then the user must have sufficient authority to one of the resources listed in the corresponding class shown in Table 10-2.

Table 10-2 Resources and classes

One of these resources:	In this class:
DB2-subsystem.table-qualifier.table-name.UNLOAD	MDSNTB or GDSNTB
DB2-subsystem.database-name.DBADM	DSNADM
DB2-subsystem.SQLADM This check is bypassed for user tables	MDSNSM or GDSNSM
DB2-subsystem.SYSDBADM This check is bypassed for user tables	DSNADM
DB2-subsystem.DATAACCESS	DSNADM
DB2-subsystem.ACCESSCTRL This check is bypassed for user tables	DSNADM
DB2-subsystem.SYSCTRL This check is bypassed for user tables	DSNADM
DB2-subsystem.SYSADM	DSNADM
DB2-subsystem.SECADM This check is bypassed for user tables	DSNADM

10.3 Object ownership transfer

Another popular security item addressed in DB2 12 is the ability to change an object's ownership online. Ownership of a database object can be determined by several rules such as whether the DDL statement that defines the object is a static or dynamic SQL, executed under a trusted context with ROLE AS OBJECT OWNER or not, the privilege set for the DDL statement, and so on. The ownership belongs to either an authorization ID or a role. To comply with government and company regulations, the ownership of sensitive data must be transferable from one authorization ID or role to another authorization ID or role. In DB2 11, such a task can be accomplished only with DROP and CREATE DDL statements followed by reloading data in the object as needed. These activities are disruptive in a production system where access to the object is constantly acquired.

To provide the support for changing ownership while keeping the object available, DB2 12 introduces the TRANSFER OWNERSHIP SQL statement with the appropriate authorization. This SQL statement can be either static or dynamic. The TRANSFER OWNERSHIP statement is allowed when the application compatibility is V12R1M500 or greater. The package containing the TRANSFER OWNERSHIP statement can be bound to that application compatibility level after activating new function.

Figure 10-4 shows the syntax diagram for the TRANSFER OWNERSHIP statement.

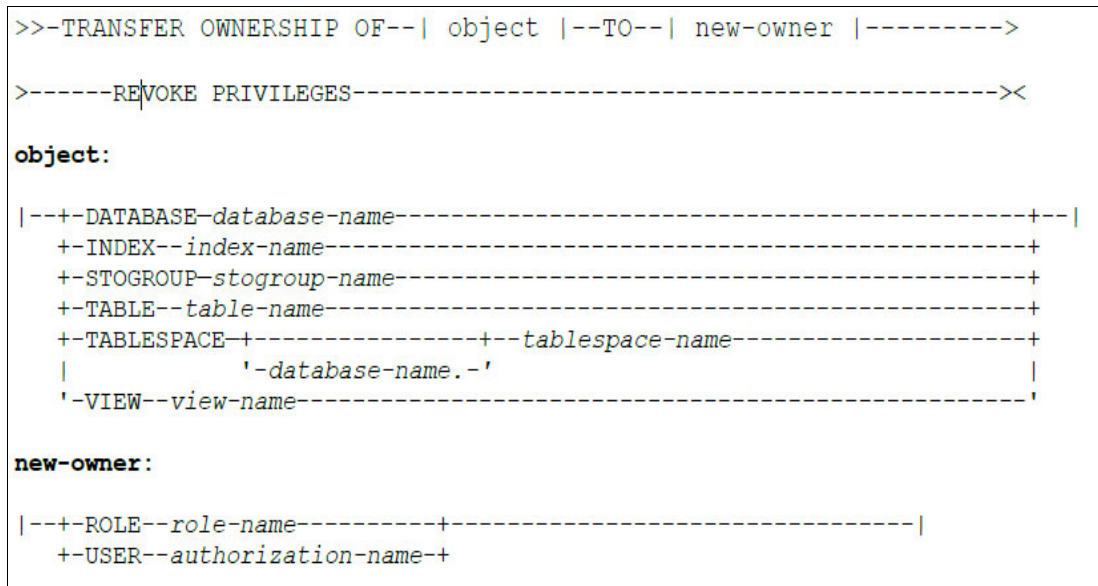


Figure 10-4 The TRANSFER OWNERSHIP statement syntax diagram

Note: Although the syntax diagram does not specify ALIAS, an alias for TABLE and VIEW can be specified on the TRANSFER OWNERSHIP statement and its based table or the view ownership will be transferred.

The CREATOR and CREATORTYPE columns in the following tables are updated with the new owner when the transfer is successful for the objects:

- SYSIBM.SYSDATABASE** If ownership of a *database* is transferred (ownership of the DSNDB01, DSNDB04, and DSNDB06 databases are not allowed to be transferred)
- SYSIBM.SYSSTOGROUP** If ownership of a *storage group* is transferred
- SYSIBM.SYSTABLESPACE** If ownership of a *table space* is transferred

The OWNER and OWNERTYPE columns in the following tables are updated with the new owner when the transfer is successful for the objects:

- SYSIBM.SYSTABLES** If ownership of a *table* is transferred
- SYSIBM.SYSINDEXES** If ownership of an *index* is transferred
- SYSIBM.SYSVIEWS** If ownership of a *view* is transferred

The example in Figure 10-5 on page 174 shows a table that is created and owned by an authorization ID and later, its ownership is transferred to another authorization ID by the SECADM authority.

```

ADMFO02:
CREATE TABLE SZI10T
...
SYSIBM.SYSTABLES                                SYSIBM.SYSTABAUTH
+-----+-----+-----+-----+-----+-----+-----+-----+
! CREATOR ! NAME ! CREATEDBY ! OWNER ! GRANTOR ! GRANTEE ! TCREATOR ! TTNAME !
+-----+-----+-----+-----+-----+-----+-----+-----+
! ADMFO02 ! SZI10T ! ADMFO02 ! ADMFO02 ! ADMFO02 ! ADMFO02 ! ADMFO02 ! SZI10T !
+-----+-----+-----+-----+-----+-----+-----+-----+

SECADM:
TRANSFER OWNERSHIP OF TABLE ADMFO02.SZI10T TO USER ADMFO03
REVOKE PRIVILEGES

SYSIBM.SYSTABLES                                SYSIBM.SYSTABAUTH
+-----+-----+-----+-----+-----+-----+-----+-----+
! CREATOR ! NAME ! CREATEDBY ! OWNER ! GRANTOR ! GRANTEE ! TCREATOR ! TTNAME !
+-----+-----+-----+-----+-----+-----+-----+-----+
! ADMFO02 ! SZI10T ! ADMFO02 ! ADMFO03 ! ADMFO03 ! ADMFO03 ! ADMFO02 ! SZI10T !
+-----+-----+-----+-----+-----+-----+-----+-----+

```

Figure 10-5 SECADM transfers table's SZI10T's ownership from ADMFO02 to ADMFO03

Note: The SYSTABLES.CREATEDBY column remains as is with the old owner while the OWNER column is updated with the new owner. The catalog table SYSTABAUTH is also updated with the appropriate table authorization's grantor and grantee.

10.3.1 Supported objects

The privilege set of the application executing the TRANSFER OWNERSHIP statement must be the owner of the object or SECADM authority. Even with the SEPARATE_SECURITY DSNZPARM set to NO, installed SYSADM and SYSADM authority is not sufficient to perform the ownership transfer. By using this SQL statement, ownership of the following objects can be transferred:

- ▶ Database
- ▶ Tablespace
- ▶ Table
- ▶ Index
- ▶ View
- ▶ Stogroup

Those objects must not be the system objects (owned by schemas that begin with SYS) and must exist at the current server where the statement is executing. Therefore, three-part name table or view, or alias created with the three-part name should have the location name resolved to the local server. When ownership of a database is transferred, the ownerships of other objects created in the same database such as tablespaces, tables, and views are not transferred. When ownership of a table is transferred, the ownership of the following dependent objects is also transferred:

- ▶ Index (if same owner)
- ▶ Implicitly created table space for this base table
- ▶ Implicitly and explicitly (same owner) created LOB objects (aux table, aux index, LOB table space)
- ▶ XML objects (table, index, table space)

10.3.2 New owner

The new owner can be an authorization ID, a role or the SESSION user (the primary authorization ID executing the application). The new owner's name and type are recorded in either the CREATOR/CREATOR TYPE or OWNER/OWNERTYPE columns in the appropriate DB2 catalog table where the object's definition is kept. The new owner is automatically granted the same privileges that the old owner obtained when the object was created. The new owner must have the set of privileges on the base objects, as indicated by the objects dependencies, that are required to maintain the objects existence, unchanged.

For example, TAMMIED is the owner of table T1 and TAMMIED ID also has DBADM authority. After executing the TRANSFER OWNERSHIP OF T1 TO ACACIO statement, ACACIO ID will also have DBADM authority.

The example in Figure 10-6 shows that TAMMIED creates a view with the CREATE VIEW V1 AS SELECT MYUDF(C1) FROM T1 statement. This view is updatable so TAMMIED ID, as the owner, has SELECT, INSERT, UPDATE, DELETE privileges on the T1 based table. TAMMIED should also have EXECUTE privilege on the MYUDF user-defined function. When owner TAMMIED or SECADM executes the TRANSFER OWNERSHIP OF T1 TO MBERNAL statement, DB2 requires that MBERNAL also has SELECT, INSERT, UPDATE, and DELETE privileges on the T1 based table, and EXECUTE privilege on the MYUDF user-defined function.

```
CREATE VIEW TAMMIED.SZI10V AS SELECT MYUDF(C1) FROM TAMMIED.SZI10T;  
TRANSFER OWNERSHIP OF VIEW TAMMIED.SZI10V TO USER MBERNAL  
REVOKE PRIVILEGES;  
  
SQLERROR ON TRANSFER COMMAND, EXECUTE FUNCTION  
RESULT OF SQL STATEMENT:  
DSNT408I SQLCODE = -20342, ERROR: AUTHORIZATION ID MBERNAL DOES NOT HAVE THE  
REQUIRED PRIVILEGE SELECT ON OBJECT TAMMIED.SZI10TOF TYPE TABLE FOR OWNERSHIP  
TRANSFER.
```

Figure 10-6 View's ownership transfer failure

10.3.3 Revoking privileges of current owner

In DB2 12, the REVOKE PRIVILEGES option on the TRANSFER OWNERSHIP statement must be specified to indicate that the current owner will not have any implicit privilege on the object after the transfer is completed (Example 10-3). The following objects are invalidated:

- ▶ Dynamic cached statements that are dependent on the current owner's privilege on the object
- ▶ Stabilized dynamic statements that are dependent on the current owner's privilege on the object

Example 10-3 Transfer ownership of table EMPLOYEE.BENEFIT to role BENEFIT_ADMINROLE and revoke current owner's privileges on this table

```
TRANSFER OWNERSHIP OF TABLE EMPLOYEE.BENEFIT TO ROLE BENEFIT_ADMINROLE REVOKE  
PRIVILEGES;
```

If any package exists that is dependent on the current owner's privilege on the object, the TRANSFER statement fails. This failure can be avoided by explicitly granting those privileges from another source (such as SECADM) or if the current owner has an administrative authority that allows access (such as DBADM on the database).

The query shown in Figure 10-7 can be used to identify those packages and the privileges associated with a table and schema.

```
SELECT B.GRANTEE AS PACKAGE,
       B.SELECTAUTH AS SELECT,
       B.INSERTAUTH AS INSERT,
       B.UPDATEAUTH AS UPDATE,
       B.DELETEAUTH AS DELETE
FROM SYSIBM.SYSTABLES A, SYSIBM.SYSTABAUTH B
WHERE A.CREATOR = 'table-creator '
      AND A.NAME = 'table-name '
      AND B.TCREATOR = A.CREATOR
      AND B.TTNAME = A.NAME
      AND B.GRANTOR = A.OWNER
      AND B.GRANTORTYPE = A.OWNERTYPE
      AND B.GRANTEETYPE = 'P'
      AND B.AUTHHOWGOT = ' ';
```

Figure 10-7 Query to find packages dependent on current owner's privilege on the table

For tables created prior to DB2 9, the query shown in Figure 10-8 can be used.

```
SELECT B.GRANTEE AS PACKAGE,
       B.SELECTAUTH AS SELECT,
       B.INSERTAUTH AS INSERT,
       B.UPDATEAUTH AS UPDATE,
       B.DELETEAUTH AS DELETE
FROM SYSIBM.SYSTABLES A, SYSIBM.SYSTABAUTH B
WHERE A.CREATOR = 'table-creator '
      AND A.NAME = 'table-name '
      AND B.TCREATOR = A.CREATOR
      AND B.TTNAME = A.NAME
      AND B.GRANTOR = A.CREATOR
      AND B.GRANTORTYPE = ' '
      AND B.GRANTEETYPE = 'P'
      AND B.AUTHHOWGOT = ' ';
```

Figure 10-8 Query to find packages dependent on current owner's privilege on table (before DB2 9)

The grant actions performed by the current owner are not changed when the transfer completes. Those privileges can be revoked separately after by using REVOKE statement with the BY clause.

Note: The clone table and the base table are considered unrelated objects with regard to access control. Therefore, transferring ownership of the clone table does not affect ownership of the base table and vice versa. Ownership of each table should be transferred separately as needed.

Also, that functionality does not apply to plan, packages, and security objects created in DB2. Ownership of an application plan and package can be transferred by using the **REBIND** subcommand with the existing **OWNER** option. The security objects, such as trusted contexts, roles, row permissions, column masks and their ownerships can be transferred with the DROP and CREATE DDL statements.



Utilities

This chapter covers the following enhancements for utilities:

- ▶ Backup and recovery enhancements
- ▶ RUNSTATS enhancements
- ▶ REORG enhancements
- ▶ LOAD and UNLOAD enhancements

11.1 Backup and recovery enhancements

DB2 12 introduces the following enhancements for backup and recovery utilities:

- ▶ Sequential image copy enhancements
- ▶ Copy support for FASTREPLICATION
- ▶ Alternate copy pools for system-level backups
- ▶ FLASHCOPY_PPRCP keyword option
- ▶ Point-in-time recovery enhancements
- ▶ MODIFY RECOVERY enhancements

11.1.1 Sequential image copy enhancements

DB2 uses &ICTYPE to refer the type of image copy taken. In DB2 11, &ICTYPE was set to 'C' for image data set name allocated through TEMPLATE for COPY utility with CHANGELIMIT. That way, users are unable to determine quickly and accurately whether an image copy data set is a full or incremental copy.

DB2 12 supports &ICTYPE on TEMPLATE to reflect the actual type of image copy even when CHANGELIMIT is specified for the COPY utility:

- ▶ &ICTYPE = 'F', when a full image copy will be generated
- ▶ &ICTYPE = 'I', when an incremental image copy will be generated

Figure 11-1 describes how to identify the type of image copy on DB2 11. The &ICTYPE is the same for both types of image copies (incremental and full image copy), the letter "C" is specified in the DSN value. The difference is found three lines above, which shows INCREMENTAL IMAGE COPY or FULL IMAGE COPY.

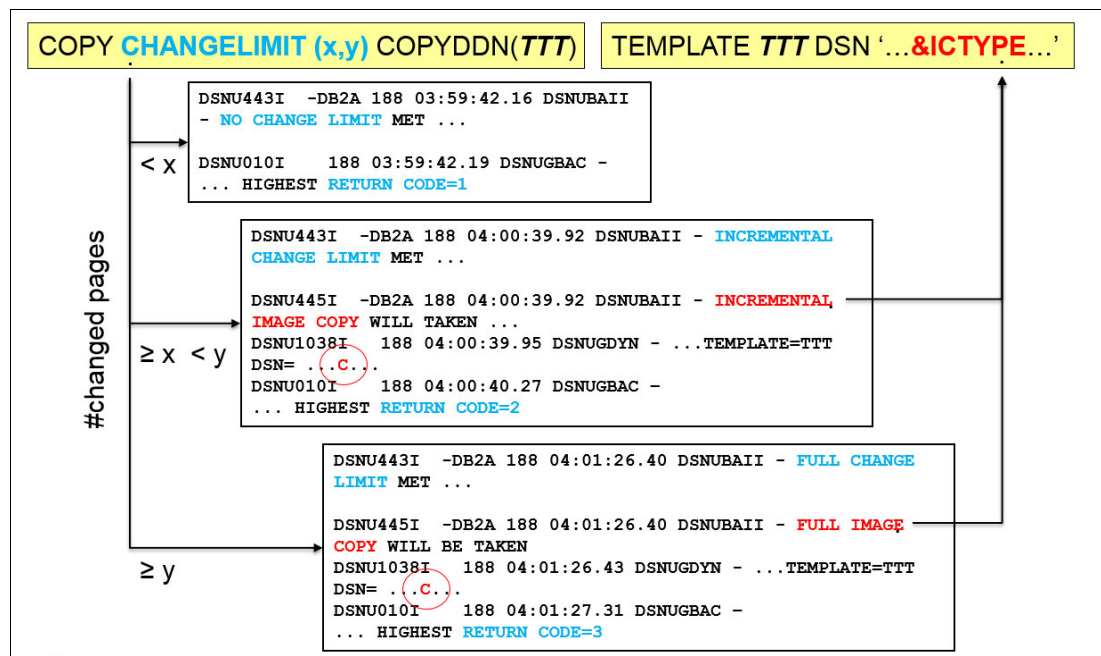


Figure 11-1 How to identify the type of image copy on DB2 11

Figure 11-2 describes how to identify the type of image copy on DB2 12. For incremental image copy, the &ICTYPE is specified with the letter “I” in the DSN value, and for full image copy, the &ICTYPE is specified with the letter “F” in the DSN value.

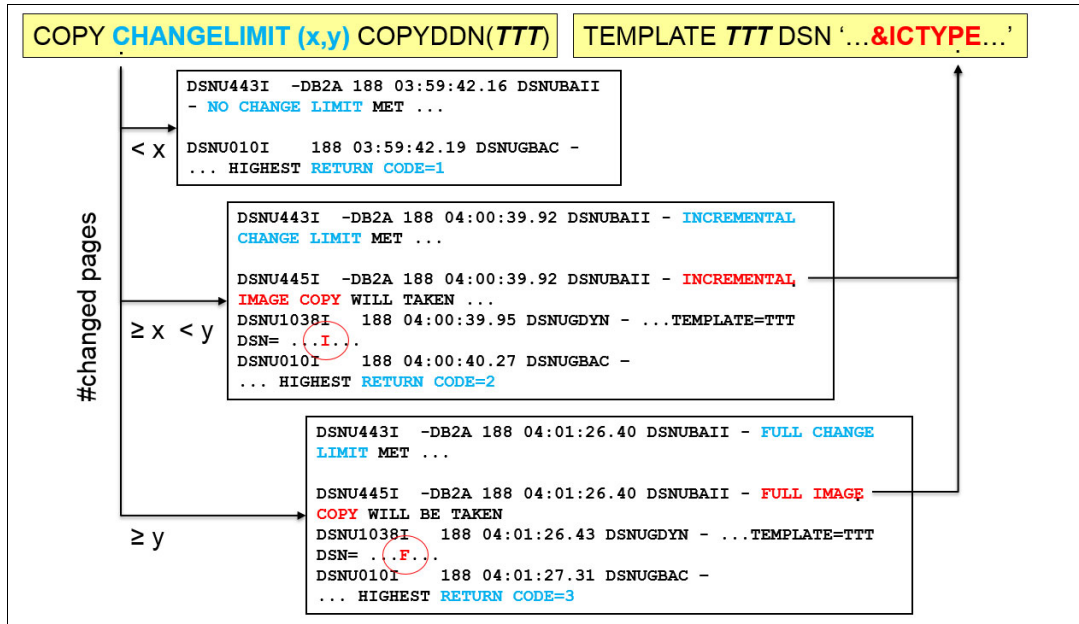


Figure 11-2 How to identify the type of image copy on DB2 12

Also an additional difference exists in the ways to identify the type of sequential image copy being copied from FlashCopy. In DB2 11, the &ICTYPE letter for full image copy is “C” in the DSN; in DB2 12, the letter “F” represents the &ICTYPE for full image copy taken from FlashCopy (Figure 11-3).

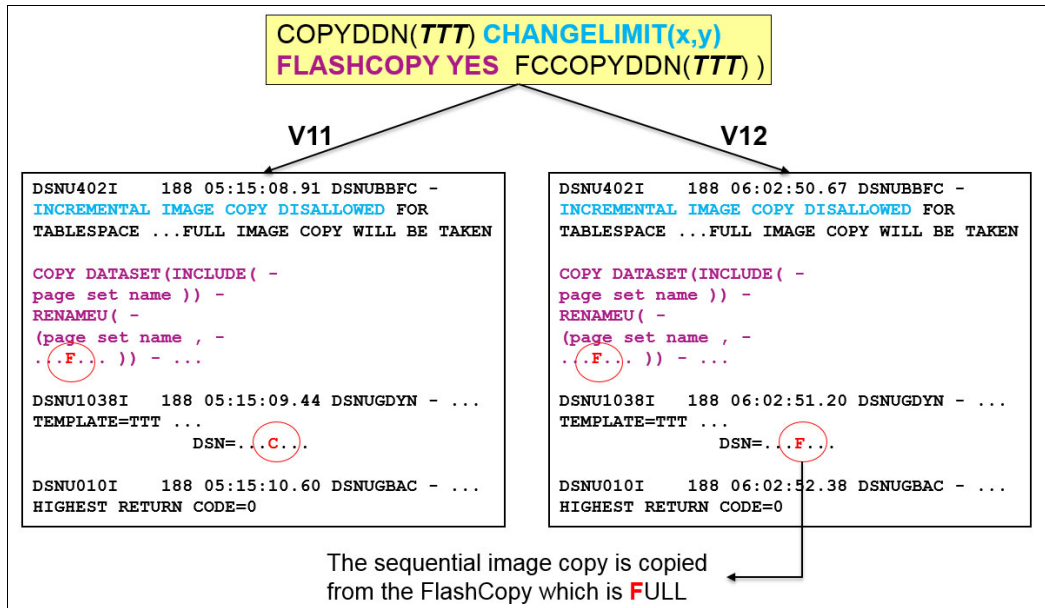


Figure 11-3 Identifying the type of sequential image copy being copied from FlashCopy

11.1.2 Copy support for FASTREPLICATION

DB2 12 includes COPY_FASTREPLICATION, a subsystem parameter to specify whether fast replication is required, preferred, or not needed during the creation of FlashCopy image copy by the COPY utility. This new parameter was necessary because the creation of FlashCopy image copy by the COPY utility used a default of FASTREP (PREF) (fast replication preferred) and no options to override existed.

Figure 11-4 shows that in DB2 11, the FASTREPLICATION option was not specified, so it depended on the customer's SMS default setting and the default for the SMS is PREFERRED.

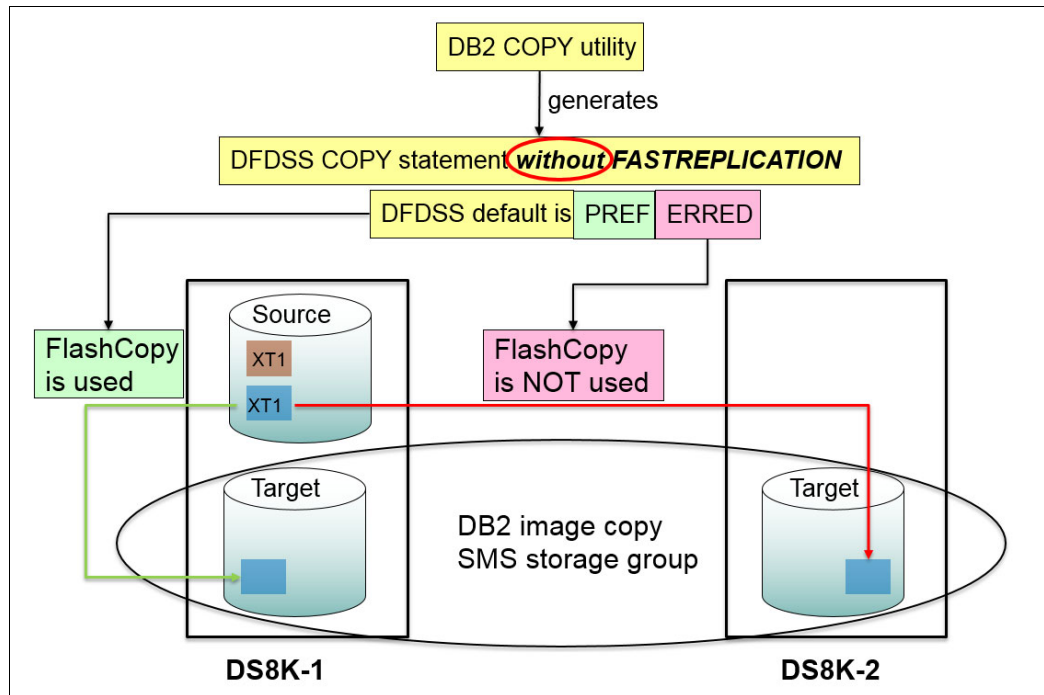


Figure 11-4 DB2 11 without FASTREPLICATION parameter

DB2 12 has the ability to control whether or not fast replication is used as Figure 11-5 represents.

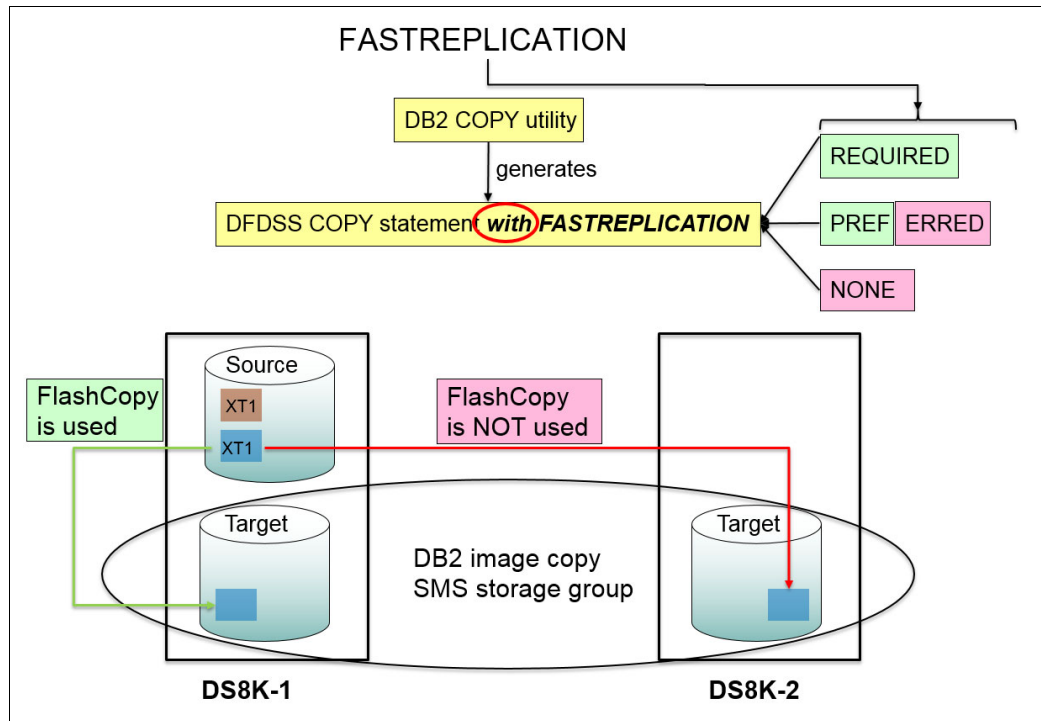


Figure 11-5 DB2 12 with FASTREPLICATION parameter

11.1.3 Alternate copy pools for system-level backups

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.

The naming conventions for the database copy pool and log copy pool are as follows:

- ▶ For the database copy pool: DSN\$1ocname\$DB
- ▶ For the log copy pool: DSN\$1ocname\$LG

11.1.4 FLASHCOPY_PPRCP keyword option

In DB2 12, the FLASHCOPY_PPRCP keyword is added to RESTORE SYSTEM and RECOVER utilities allowing 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 recovery base.

Control options for RECOVER from FlashCopy image copy are shown in Table 11-1.

Table 11-1 Control options for RECOVER from FlashCopy image copy

	Preserve mirror behavior	FlashCopy usage
zPARM	FLASHCOPY_PPRC	REC_FASTREPLICATION
Default	REQUIRED	PREFERRED
RECOVER keyword	FLASHCOPY_PPRCP	Not applicable
ADRDSU keyword	FCTOPPRCP	FASTREPLICATION

Control options for RECOVER from System Level Backup are shown in Table 11-2.

Table 11-2 Control options for RECOVER from System Level Backup

	Preserve mirror behavior	FlashCopy usage
zPARM	FLASHCOPY_PPRC	REC_FASTREPLICATION
Default	REQUIRED	PREFERRED
RECOVER keyword	FLASHCOPY_PPRCP	Not applicable
FRRECOV keyword	ALLOWPPRCP	FR (=FASTREPLICATION)
ADRDSU keyword	FCTOPPRCP	FASTREPLICATION
Control options before V12	ISMF COPYPOOL panels	SETSYS FASTREPLICATION (DATASETRECOVERY(...))

Control options for RESTORE SYSTEM are shown in Table 11-3.

Table 11-3 Control options for RECOVER from RESTORE system

	Preserve Mirror behavior	FlashCopy usage
zPARM	FLASHCOPY_PPRC	
Default	REQUIRED	
RESTORE SYSTEM keyword	FLASHCOPY_PPRCP	
FRRECOV keyword	ALLOWPPRCP	-
ADRDSU keyword	FCTOPPRCP	FASTREPLICATION
Control options before V12	ISMF COPYPOOL panels	

Example 11-2 demonstrates how PIT recovery before materializing REORG works for a LOB compression object and sets the object to REORG pending status. Also, VERIFY YES is enforced, so in this way, all involved objects must be set to this point.

Example 11-2 PIT recovery before materializing REORG for LOB compression: REORG pending

```

DSNU050I   300 08:42:39.92 DSNUGUTC - RECOVER TABLESPACE SZI10D.SZI10SA1
TOLOGPOINT X'00CFCC48468A97610400' VERIFYSET NO
DSNU124I  -DB2A 300 08:42:39.96 DSNUCAIN - VERIFYSET NO SPECIFICATION IS IGNORED
AND VERIFYSET YES IS IN EFFECT FOR CURRENT UTILITY EXECUTION
DSNU1316I -DB2A 300 08:42:39.96 DSNUCAIN - THE FOLLOWING TABLESPACES ARE MISSING
FROM THE RECOVERY LIST
                SZI10D.SZI10S PARTITION 00001
DSNU050I   308 03:15:31.79 DSNUGUTC - RECOVER TABLESPACE SZI10D.SZI10SA1
TABLESPACE SZI10D.SZI10S TOLOGPOINT X'00CFCC48468A97610400' VERIFYSET NO
DSNU124I  -DB2A 308 03:15:31.86 DSNUCAIN - VERIFYSET NO SPECIFICATION IS IGNORED
AND VERIFYSET YES IS IN EFFECT FOR CURRENT UTILITY EXECUTION
...
DSNU535I  -DB2A 308 03:15:32.94 DSNUCATM - FOLLOWING TABLESPACES RECOVERED TO A
CONSISTENT POINT
                SZI10D.SZI10S
                SZI10D.SZI10SA1
DSNU506I  -DB2A 308 03:15:32.98 DSNUGRSX - TABLESPACE SZI10D.SZI10SA1 IS IN REORG
PENDING STATE

```

Example 11-3 refers to the **-DISPLAY DATABASE** command showing the REORG pending restriction status of the object.

Example 11-3 Display for table space with restrictive status

```

DSNT397I  -DB2A
NAME      TYPE PART  STATUS          PHYERRLO PHYERRHI CATALOG  PIECE
-----
SZI10S    TS      0001  RW
SZI10S    TS
SZI10SA1  LS              RW,REORP

```

Recovering only necessary data set

DB12 introduces the SCOPE keyword, which 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, then 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, this 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.

The syntax for SCOPE UPDATED and SCOPE ALL are shown in Figure 11-6.

```

>>-RECOVER----->
>--+--LIST--listdef-name-----list-options-spec--+>
| | .-DSNUM--ALL----- | | |
| | V      .-DSNUM--integer----- | |
| | '---object-----' | |
| | | (1) | |
| | | '---DSNUM--integer-----' | |
+--recover-options-spec-----+
'-object--PAGE--page-number-----'
'-CONTINUE-'
.
.
.

>+-----+>
| | .-VERIFYSET--YES-. .-ENFORCE--YES-. .-SCOPE--UPDATED-. |
+--TORBA--X'byte-string'--+--VERIFYSET--NO--' '-ENFORCE--NO--' '-SCOPE--ALL-----' |
| | .-VERIFYSET--YES-. .-ENFORCE--YES-. .-SCOPE--UPDATED-. |
'-TOLOGPOINT--X'byte-string'--+--VERIFYSET--NO--' '-ENFORCE--NO--' '-SCOPE--ALL-----'
'-VERIFYSET--NO--' '-ENFORCE--NO--' '-SCOPE--ALL-----'

```

Figure 11-6 Syntax for SCOPE UPDATED and SCOPE ALL keywords

11.1.6 MODIFY RECOVERY enhancements

Two options are added to the MODIFY RECOVERY utility:

- ▶ DELETEDS
- ▶ NOCOPYPEND

DELETEDS option

The DELETEDS option is added. Its functionality is to delete cataloged image copy data sets on disk or migrated by DFSMSshsm tape when corresponding SYSCOPY records are deleted. The IDCAM program is invoked to perform the deletion with **DELETE** commands.

This is an optional feature because it increased 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, the image copy data sets that were not deleted in the first invocation will not be deleted.

NOCOPYPEND 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.

11.2 RUNSTATS enhancements

DB2 12 provides several statistics collection enhancements, including several of which improve SQL performance. These enhancements are described in the following topics:

- ▶ Specifying FREQVAL without the COUNT n keywords
- ▶ USE PROFILE support for Inline Statistics
- ▶ INVALIDATECACHE option
- ▶ RUNSTATS TABLESPACE LIST INDEX improvements
- ▶ New keyword REGISTER for RUNSTATS utility

11.2.1 Specifying FREQVAL without the COUNT n keywords

FREQVAL indicates, when specified with the COLGROUP option, that frequency statistics to be gathered for the specified group of columns. Up to DB2 11, a necessary step is to specify an integer number of how many frequently occurring values are collected from the specified column group; to get this number, DB2 users usually run queries simulating RUNSTATS collection. If the count was not specified, an error occurred when COLGROUP was specified.

Example 11-4 shows the syntax diagram DB2 11.

Example 11-4 FREQVAL syntax for DB2 V11

```
>>-+-----+----->
|          .-COUNT 10 MOST----- . |
'-FREQVAL -+-----+-----'
|          |          .-MOST-- . |
|          |          '-COUNT--integer--+-+-----+-'
|          |          +-BOTH--+
|          |          +-LEAST--+
```

Now on DB2 12, if FREQVAL is specified without the COUNT n keywords, DB2 automatically determines the appropriate number to collect and does not assume COUNT 10 MOST as in DB2 11 with the COLGROUP option (Example 11-5).

Example 11-5 FREQVAL syntax for DB2 V12

```
>>-+-----+----->
'-FREQVAL -+-----+-----'
|          |          (1) .-MOST-- . |
|          |          '-COUNT--integer--+-+-----+-'
|          |          +-BOTH--+
|          |          '-LEAST--'
```

11.2.2 USE PROFILE support for inline statistics

The USE PROFILE support allows inline statistics to employ a previously stored statistics profile to gather statistics for a table. The statistics profile is created by using the RUNSTATS options of SET PROFILE and is updated using the UPDATE PROFILE RUNSTATS option.

On DB2 12 profiles are supported with inline statistics to match RUNSTATS functional support. The latest PROFILE settings will be picked up so the appropriate statistics are gathered during LOAD and REORG, avoiding an additional execution of RUNSTATS.

Example 11-6 shows the support for the USE PROFILE keywords added when the STATISTICS keyword is used to specify the collection of inline statistics by REORG TABLESPACE STATISTICS.

Example 11-6 Inline statistics

```
LISTDEF LLLL
  INCLUDE TABLESPACES      DATABASE ... PARTLEVEL
REORG TABLESPACE LIST LLLL
SHRLEVEL CHANGE           COPYDDN(TTTT)
STATISTICS TABLE(ALL) USE PROFILE REPORT YES
```

In addition, support for the USE PROFILE keywords is added when the STATISTICS keyword is used to specify the collection of inline statistics by LOAD STATISTICS too.

The statistics profile is created by using the RUNSTATS options of SET PROFILE and is updated using the UPDATE PROFILE RUNSTATS option (Example 11-7).

Example 11-7 RUNSTATS option

```
RUNSTATS TABLESPACE ... TABLE (...) SET PROFILE FROM EXISTING STATS
SELECT * FROM SYSIBM.SYSTABLES_PROFILES
PROFILE_TEXT
-+-----+-----+-----+---
COLUMN(..., ..., ...) INDEX(ALL)DEFAULT PROFILE
```

11.2.3 INVALIDATECACHE option

The INVALIDATECACHE option is introduced to control the invalidation of the dynamic statement cache. This allows you to control when to invalidate the cache while collecting statistics on objects.

With the INVALIDATECACHE YES/NO option for RUNSTATS, you can explicitly specify whether cached statements should be invalidated or not. The proposed default for RUNSTATS is INVALIDATECACHE NO.

The exceptions to the default for RUNSTATS are as follows:

- ▶ When RUNSTATS LIST REPORT NO UPDATE NONE is executed, the default value for the INVALIDATECACHE option will be YES. The dynamic statement cache is invalidated for the target objects.
- ▶ When RUNSTATS RESET ACCESSPATH is executed without the INVALIDATECACHE keyword, the default value for the INVALIDATECACHE option will be YES.

If you specify RUNSTATS RESET ACCESSPATH INVALIDATECACHE NO, then RUNSTATS will issue DSNU070I error message for the specification. The INVALIDATECACHE NO option is not supported when RESET ACCESSPATH option is specified.

When you use a RUNSTATS LIST or execute INLINE STATISTICS, the default for INVALIDATECACHE keyword is NO.

Invalidation by main utilities

When LOAD, REORG, or REBUILD INDEX is executed and when the object is in a restrictive state, the dynamic statement cache for the target objects will be invalidated by the main utility. For example, when the index is in restrictive rebuild pending state (RBDP), the statements in the dynamic cache are invalidated by the main utility. When there are pending alters, as in the case of REORG, the statements in the dynamic cache are invalidated by the main utility. The invalidation is done regardless of inline STATISTICS specification.

The INVALIDATECACHE keyword in the statistics-spec applies only to the statistics collection. It does not control the invalidation done by the main utility.

Figure 11-7 shows cache invalidation examples for LOAD, RUNSTATS, and REORG utilities.

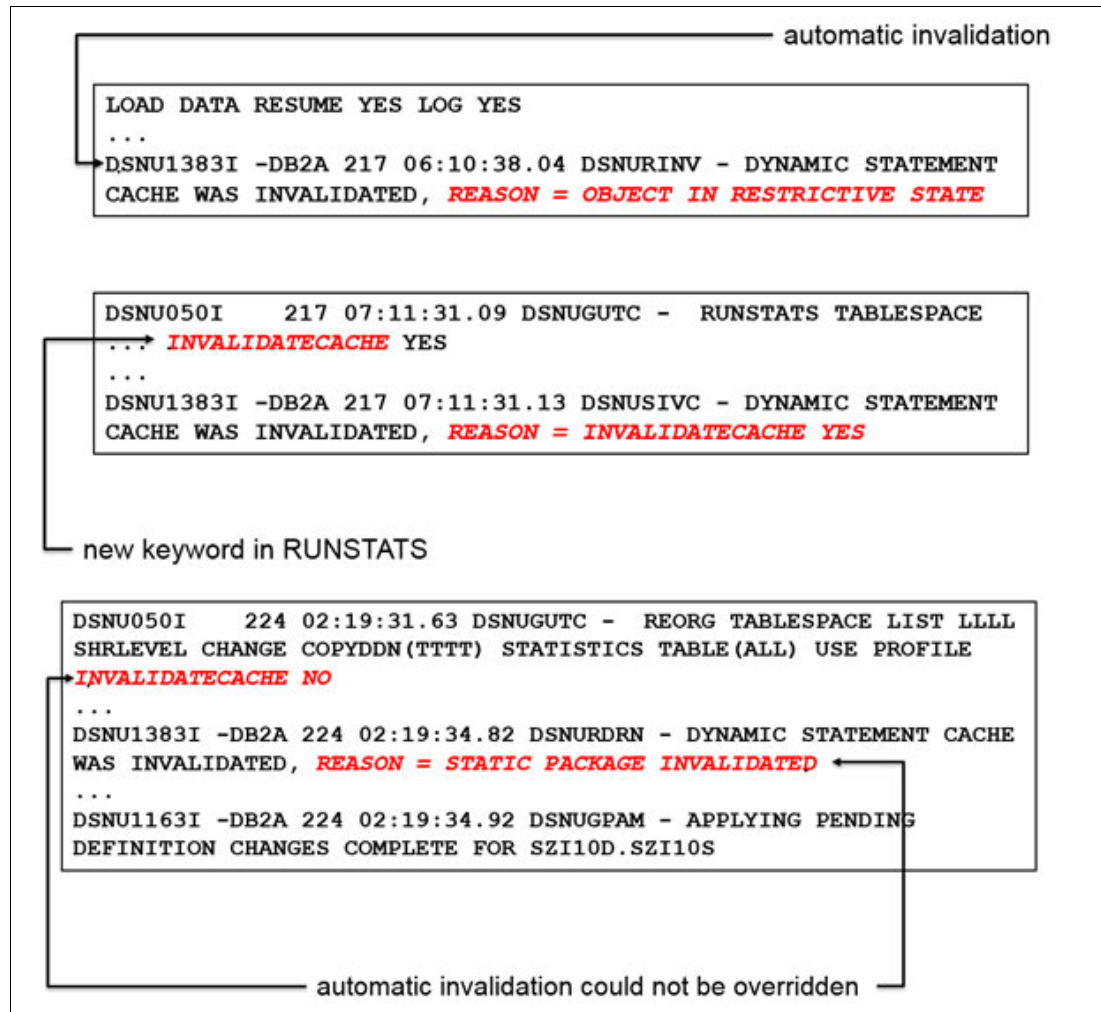


Figure 11-7 Cache invalidation examples

Figure 11-8 shows RUNSTATS and INVALIDATECACHE combinations.

	INVALIDATECACHE		
	Default	Possible	DSNU0701*
if RESET ACCESSPATH	YES	YES	NO
if REPORT NO UPDATE NONE	YES	YES	NO
if other specification	NO	NO/YES	-

Figure 11-8 RUNSTATS and INVALIDATECACHE combinations

11.2.4 RUNSTATS TABLESPACE LIST INDEX improvements

An improvement for RUNSTATS TABLESPACE LIST INDEX was developed because of an excessive elapsed time and contention during the update of catalog tables when a LIST was defined with PARTLEVEL keyword and the statement RUNSTATS TABLESPACE LIST INDEX was issued. For every partition, the statistics were collected on all corresponding indexes.

Now, in DB2 12, when a list is defined with the PARTLEVEL keyword and the statement RUNSTATS TABLESPACE LIST TABLE USE PROFILE is executed, for every tablespace part processed, index statistics are collected on the respective index part of a partitioned index or data partitioned index, no statistics on non-partitioned indexes (NPIs) are collected.

If the keyword INCLUDE NPI is specified, the statistics for the NPIs are collected for every part of the tablespace.

Figure 11-9 on page 190 represents the difference between the method adopted by DB2 11 and how DB2 12 works with this improvement. In the example, the corresponding statics from NPI that are relative to Partition 2 were not collected.

11.3 REORG enhancements

The following REORG enhancements are described in this section:

- ▶ Improved FlashCopy management
- ▶ Prevention of COPY-pending for a LOB table space during REORG of PBG
- ▶ Improved partition-level partition-by-growth (PBG) REORGs
- ▶ REORG option for empty PBG partitions deletion
- ▶ Support for the new COMPRESSRATIO catalog column
- ▶ Display claimer information on each REORG drain failure
- ▶ Additional REORG enhancements

11.3.1 Improved FlashCopy management

One of the image copy options available for REORG TABLESPACE is FlashCopy technology. The main advantages of this method against the traditional DB2 utility methods are the performance and the availability. The FlashCopy is faster and reduces the time of unavailable data.

The following scenario caused an extended application outage which is inconsistent with running online REORG mindset:

1. REORG ran and created an inline FlashCopy with no sequential inline image copy.
2. The FlashCopy failed but then REORG completed and left the pageset in a copy-pending state.

DB2 12 brings an improvement to this scenario, if FlashCopy has an unsuccessful execution, REORG completes with return code (RC) of 8 and does not leave object in a restricted status (Example 11-8).

Example 11-8 Online REORG failure scenario with unsuccessful FlashCopy image copy execution

```
DSNU050I    259 03:40:12.03 DSNUGUTC - REORG TABLESPACE LIST LLLL SHRLEVEL CHANGE
FCCOPYDDN(TTTT) FLASHCOPY CONSISTENT
...
DSNU385I    259 03:40:14.59 DSNURLGD - LOG PHASE COMPLETE, ELAPSED TIME = 00:00:00
DSNU421I    259 03:40:14.73 DSNUGFUM - START OF DFSMS MESSAGES
...
ADR709E (001)-ACS (01), AN ERROR OCCURRED IN THE STORAGE MANAGEMENT SUBSYSTEM
WHILE DETERMINING SMS CONSTRUCTS FOR
                                DATA SET DSNC910.DSNDBC.SZI10D.SZI10S.I0001.A001 WITH
NEWNAME
                                DSNC910.IC.FESWD20G.SZI10D.SZI10S.A001.F. SMS MESSAGES
FOLLOW.
  IGD01014I DATA SET ALLOCATION REQUEST FAILED -
  SPECIFIED MGMTCLAS MGMTCL1 DOES NOT EXIST
...
DSNU422I    259 03:40:16.83 DSNUGCFD - END OF DFSMS MESSAGE
DSNU2908I  259 03:40:16.83 DSNURSWT - FLASHCOPY IMAGE COPY CREATION IS
UNSUCCESSFUL
DSNU012I    259 03:40:18.47 DSNUGBAC - UTILITY EXECUTION TERMINATED, HIGHEST
RETURN CODE=8
```

11.3.2 Preventing COPY-pending on a LOB table space during REORG of PBG

In the previous DB2 version, REORG of a PBG table space with a LOB column required a new PBG partition to be created in the LOG phase. That new LOB table space had COPY-pending status due to REORG's inability to allocate a new image copy dataset at that time.

A feature in DB2 12 prevents COPY-pending restriction status on a LOB table space during REORG of PBG. An inline image copy is allocated for the new LOB table space created.

Example 11-9 shows output for REORG TABLESPACE of a PBG with a LOB column required, in which inline image copies were not created for auxiliary table spaces and left in COPY-pending in DB2 11.

Example 11-9 COPY-pending on a LOB table space during REORG of partition-by-growth in V11

```
DSNU050I    265 08:50:02.56 DSNUGUTC - REORG TABLESPACE SZI10D.SZI10S SHRLEVEL
CHANGE COPYDDN(TTTT) AUX YES
DSNU1155I -DB2A 265 08:50:03.80 DSNURFIT - AUXILIARY TABLESPACE SZI10D.SZI10SA1
WILL BE REORGANIZED IN PARALLEL WITH BASE TABLESPACE
...
...
DSNU1155I -DB2A 265 08:51:14.46 DSNURLOG - AUXILIARY TABLESPACE SZI10D.LM70M5MI
WILL BE REORGANIZED IN PARALLEL WITH BASE TABLESPACE
...
DSNU1157I    265 08:51:18.64 DSNURSWT - INLINE IMAGE COPIES ARE NOT CREATED FOR
AUXILIARY TABLE SPACES REORGANIZED AND ARE LEFT IN COPY PENDING
DSNU381I    -DB2A 265 08:51:21.82 DSNUGSRX - TABLESPACE SZI10D.LM70M5MI IS IN COPY
PENDING
```

DB2 12 prevents the COPY-pending on a LOB table space during REORG of PBG. Example 11-10 shows REORG for this case, and image copy is successful.

Example 11-10 Preventing COPY-pending on a LOB table space during REORG of PBG in DB2 12

```
DSNU050I    266 01:56:04.72 DSNUGUTC - REORG TABLESPACE SZI10D.SZI10S SHRLEVEL
CHANGE COPYDDN(TTTT) AUX YES
DSNU1155I -DB2A 266 01:56:06.08 DSNURFIT - AUXILIARY TABLESPACE SZI10D.SZI10SA1
WILL BE REORGANIZED IN PARALLEL WITH BASE TABLESPACE...
...
DSNU1155I -DB2A 266 01:57:32.17 DSNURLOG - AUXILIARY TABLESPACE SZI10D.LM81BRVC
WILL BE REORGANIZED IN PARALLEL WITH BASE TABLESPACE
...
DSNU428I    266 01:57:33.09 DSNURSWT - DB2 IMAGE COPY SUCCESSFUL FOR TABLESPACE
SZI10D.LM81BRVC
```

11.3.3 Improved partition-level PBG REORGs

DB2 12 improves the REORG TABLESPACE utility to support creation of a new PBG partition for overflow rows during a partition-level REORG. The new partitions are automatically created up to MAXPARTITIONS.

Without this feature, the REORG of a subset of PBG partitions failed because of the inability to fit the data back into the required parts. In this way, more REORG executions were required for the partitions than intended or even were necessary for a REORG of the entire table space, resulting in higher CPU, memory, and disk space utilization.

Figure 11-11 shows a scenario comparison of REORG partitions on DB2 11 and DB2 12. In this scenario, data does not fit in the partitions on DB2 11, and on DB2 12 new partitions are created to hold the data.

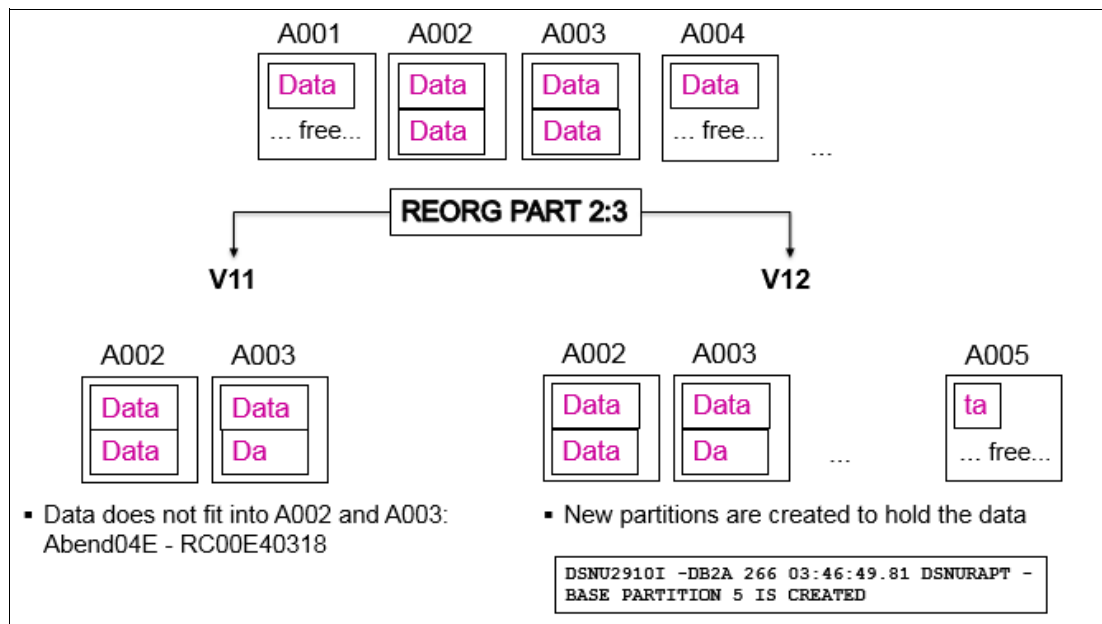


Figure 11-11 Scenario comparison of REORG partitions on DB2 11 and DB2 12

Example 11-12 shows a compression report in which the percent number of the bytes are saved from compressed data rows.

Example 11-12 Compression report

```
DSNU231I  -DB2A 004 01:14:27.02 DSNURBDC - DICTIONARY WITH 4096 ENTRIES HAS BEEN
SUCCESSFULLY BUILT FROM 4603 ROWS FOR TABLE SPACE SZI10D.SZI10S
```

```
DSNU234I  -DB2A 004 01:16:26.10 DSNURWT - COMPRESSION REPORT FOR TABLE SPACE
SZI10D.SZI10S
```

```
58333 KB WITHOUT COMPRESSION
15396 KB WITH COMPRESSION
74 PERCENT OF THE BYTES SAVED FROM COMPRESSED DATA ROWS
```

```
99 PERCENT OF THE LOADED ROWS WERE COMPRESSED
```

```
...
SELECT * FROM SYSIBM.SYSTABLEPART
WHERE DBNAME = 'SZI10D' ;
```

```
-----+
! COMPRESSRATIO !
-----+
!      74      !
-----+
-----+
-----+
-----+
```

11.3.6 Display claimers information on each REORG drain failure

Up to DB2 11, REORG was not able to display the blocking claimers information on drain failures except on the last drain attempt. It caused REORG termination with no possible action taken.

DB2 12 displays the blocking claimers on each drain attempt failure in the utility job output for REORG SHRLEVEL REFERENCE or CHANGE.

As Figure 11-13 shows, that information will be put out for each drain failure. The default is six attempts, which is why Figure 11-13 indicates this is the first drain attempt that fails and the second attempt will be made in five seconds.

```
DSNU1122I -DB2A 004 01:35:07.57 DSNURLOG - JOB SYSADM1 PERFORMING
REORG
      WITH UTILID SZI10UTIL UNABLE TO DRAIN SZI10D.SZI10S.
      RETRY 2 OF 6 WILL BE ATTEMPTED IN 5 SECONDS
DSNU590I  -DB2A 004 01:35:17.58 DSNURDRN - RESOURCE NOT AVAILABLE, REASON=X'00C200EC',
ON SZI10D.SZI10S PROHIBITS PROCESSING
DSNT360I  -DB2A *****
DSNT361I  -DB2A * DISPLAY DATABASE SUMMARY
          * GLOBAL CLAIMERS
DSNT360I  -DB2A *****
DSNT362I  -DB2A DATABASE = SZI10D STATUS = RW
          DBD LENGTH = 4028
DSNT397I  -DB2A
NAME      TYPE PART  STATUS          CONNID  CORRID  CLAIMINFO
-----+-----+-----+-----+-----+-----+
SZI10S   TS          RW,UTRW        TSO     SYSADM  (CS,C)
-                AGENT TOKEN 8
-                MEMBER NAME DB2B
***** DISPLAY OF DATABASE SZI10D ENDED *****
DSN9022I  -DB2A DSNTDDIS 'DISPLAY DATABASE' NORMAL COMPLETION
```

Figure 11-13 DB2 12 blocking claimers display

11.3.7 Additional REORG enhancements

This section covers several additional REORG enhancements:

- ▶ New mapping table format support
- ▶ Permit REORG against RO objects
- ▶ Additional offload to zIIP

Mapping table format support

A mapping table format supports 7-byte RIDs needed with range-partitioned (PBR) relative page numbering. For more information about PBR relative page numbering, see Chapter 3, “Scalability” on page 33.

Permit REORG against read-only objects

Up to DB2 11, the REORG utility failed when executed against read-only objects, because this option was not allowed by DB2. To correct this situation, DB2 12 allows REORG utility execution against read-only objects.

Additional offload to zIIP

The IBM z Systems® Integrated Information Processor (zIIP) is enabled for REORG RELOAD phase in DB2 12.

The objective of the enhancement is to reduce the cost of running REORG by offloading CPU time to the zIIP processor. The phases of the utilities have been enabled over time:

V8	Index build in LOAD, REORG, and REBUILD
V9	REORG UNLOAD
V10	RUNSTATS
V11	Inline stats
V12	LOAD and REORG RELOAD

The zIIP is used as long as Workload Manager (WLM) tells the utility that zIIP is configured; there are no changes to the utility statement.

Measurements show CPU time reduction in the RELOAD phase in LOAD being offloaded to the zIIP and also for the offload for the REORG RELOAD phase.

11.4 LOAD and UNLOAD enhancements

LOAD and UNLOAD are also enhanced:

- ▶ LOAD enhancements
- ▶ UNLOAD enhancements

11.4.1 LOAD enhancements

These are the LOAD enhancements:

- ▶ LOAD PARALLEL with COLGROUP
- ▶ LOAD PART REPLACE skipping NPSI scan if empty partition-by-range (PBR) partition is empty.
- ▶ PREFORMAT support for auxiliary table spaces
- ▶ Additional zIIP offload

LOAD PARALLEL with COLGROUP

The use of parallelism can reduce the elapsed time for loading a considerable amount of data and improve the performance for the LOAD utility. DB2 11 introduced the use of the PARALLEL option for LOAD SHRLEVEL CHANGE | NONE, but it did not support inline COLGROUP statistics.

The advantage of the COLGROUP option is to collect frequency and cardinality statistics on any column group. This way determines a better access plan and improves query performance. All the COLGROUP statistics can be collected during the LOAD invocation by specifying FREQVAL and HISTOGRAM keywords.

LOAD PART REPLACE skipping NPSI scan if empty partition-by-range (PBR) partition is empty

DB2 12 reduces elapsed and CPU time with this enhancement on LOAD PART REPLACE. It skips the non-partitioned secondary index (NPSI) scan if partition-by-range (PBR) is empty.

PREFORMAT support for auxiliary table spaces

The PREFORMAT option supports auxiliary table spaces on DB2 12. With this option, the need for DB2 to preformat new pages in a table space during the execution time is eliminated.

Additional zIIP offload

The zIIP is enabled for LOAD RELOAD phase in DB2 12. This enhancement reduces the cost of running LOAD by offloading CPU time to the zIIP.

11.4.2 UNLOAD enhancements

DB2 12 provides two UNLOAD enhancements:

- ▶ The REGISTER keyword
- ▶ An UNLOAD privilege

The REGISTER keyword

The REGISTER keyword can be used if SHRLEVEL CHANGE ISOLATION UR is specified. It controls whether to register pages read by the UNLOAD utility with SHRLEVEL CHANGE ISOLATION UR to the coupling facility in a data sharing environment (Figure 11-14).

```

      .-SHRLEVEL--CHANGE--ISOLATION--CS--+-----+-----+.
      |                                     '-SKIP LOCKED DATA-' |
>-----+-----+-----+-----+-----+-----+----->
      |                                     .-REGISTER NO--.    |
      |                                     '-REGISTER YES-'   |
      |                                     '-REFERENCE-----' |

```

Figure 11-14 REGISTER keyword syntax example for UNLOAD

When REGISTER NO is specified, pages that are read by the UNLOAD utility are not registered with the coupling facility. Pages that are resident with the coupling facility are not processed by the UNLOAD utility. This option reduces data sharing overhead.

The REGISTER NO option is not supported for base table spaces with LOB or XML columns. If the REGISTER NO option is not specified explicitly for a base table space with LOB or XML columns, DB2 sets the REGISTER value to YES. If REGISTER NO is explicitly specified for a base table space with LOB or XML columns, DB2 issues an error.

When REGISTER YES is specified, pages that are read by the UNLOAD utility are registered with the coupling facility.

New UNLOAD privilege

For new UNLOAD privilege, see Chapter 10, “Security” on page 167.



Installation and migration

This chapter provides information to help you evaluate the changes in DB2 12 for z/OS and to plan for a successful installation of or migration to DB2 12 for z/OS.

This chapter covers the following topics:

- ▶ Prerequisites for DB2 12
- ▶ Single-phase migration and function level
- ▶ Installing a new DB2 12 system
- ▶ Subsystem parameters
- ▶ Installation or migration without requiring SYSADM
- ▶ Installation with z/OS Management Facility (z/OSMF)
- ▶ Temporal catalog

12.1 Prerequisites for DB2 12

This section describes the prerequisite requirements for hardware and software to successfully install and work with DB2 12 for z/OS.

12.1.1 Data sharing

Data sharing requires the latest coupling facility (CF) level recommended for your processor:

<http://www.ibm.com/systems/z/advantages/psocftable.html>

12.1.2 Processor requirements

These are the processor requirements:

- ▶ z196, or later, processors running z/OS V2.1, or later
- ▶ The processors must have enough real storage to satisfy these combined requirements:
 - DB2 12 for z/OS
 - z/OS
 - The appropriate DFSMS storage management subsystem components, access methods, telecommunications, batch requirements, and other customer required applications

12.1.3 Software requirements

These are the software requirements:

- ▶ z/OS V2.1 Base Services (5650-ZOS), at minimum
- ▶ DFSMS V2 R1: DB2 Catalog is SMS managed
- ▶ IBM Language Environment® Base Services
- ▶ z/OS Version 2 Release 1 Security Server (RACF)
- ▶ IRLM Version 2 Release 3 (included with DB2 12 for z/OS)
- ▶ z/OS Unicode Services and appropriate conversion definitions

12.1.4 DB2 Connect prerequisites

You can make use of any version of IBM DB2 Connect™ to exploit most DB2 12 features (such as new SQL syntax, utilities that work in function level V12R1M500, and so forth). For continuous availability during the migration process, the minimum recommended level is DB2 V9.7 FP6 or 10.2 FP2 or later.

Some enhancements require a minimum DB2 Connect version and fix pack, which is available after DB2 12 General Availability:

- ▶ A new client API to provide fast loading of tables from mobile devices.
- ▶ A new client API to determine the DB2 function level.
- ▶ Support for continuous delivery
- ▶ Improved security and scalability for client sysplex workload balancing
- ▶ Improved sysplex support for distributed global transactions
- ▶ Support for preserving prepared dynamic statements after a rollback
- ▶ Improved client separability aids

The suggestion is to move from the client or runtime client packages toward using the data server driver.

12.1.5 Programming language requirements, minimum levels

The following application development programming languages can be used to build applications for DB2 12:

- ▶ Building applications using a DB2 precompiler:
 - Assembler: High Level Assembler, part of the System Services element of z/OS
 - Fortran: VS Fortran V2.6 (5668-806, 5688-087, 5668-805); new data types and new SQL functions are not supported since DB2 9 for z/OS
- ▶ Building applications using a DB2 precompiler or coprocessor:
 - C/C++: C/C++ (without Debug Tool), which is an optional priced feature of z/OS
 - COBOL (one of the following):
 - Enterprise COBOL for z/OS, V3.4 (5655-G53)
 - Enterprise COBOL for z/OS, V4.1 (5655-S71), or later
 - Enterprise COBOL for z/OS, V5.1 (5655-W32)
 - PL/I (one of the following):
 - Enterprise PL/I for z/OS, V3.9 (5655-H31)
 - Enterprise PL/I for z/OS, V4.1 (5655-W67), or later

12.1.6 Minimum configuration (IEASYSxx)

- ▶ 1 TB of contiguous shared private per DB2 – HVSHARE:
 - Default is 510 TB.
 - Article on HVSHARE.
- ▶ 6 GB of contiguous 64-bit (HVCOMMON) per DB2:
 - Same as DB2 11, with a default of 66 GB
- ▶ Configure additional megabytes of 1 MB LFAREA for maximum benefit:
 - Large frame area.
 - See z/OS APAR OA34024 for LFAREA sizing information.
- ▶ PDSEs:
 - Required for SDSNLOAD, SDSNLOD2, ADSNLOAD, ADSNLOD2 (same as DB2 11).

12.2 Single-phase migration and function level

DB2 12 eliminates multiple migration modes that existed in DB2 11, such as CM, CM*, ENFM, ENFM*, NFM. The CM or compatibility mode is equivalent to function level V12R1M100 (before new function activation, BNFA). The new function mode (NFM) is now equivalent to function level V12R1M500 or greater (after new function activation, ANFA). The CM* mode is now called a *star function level*. For more information about function level, see Chapter 2, “Continuous delivery” on page 7.

The **DSNTIJTC (CATMAINT)** migration job performs all changes necessary for the DB2 catalog and directory objects. These objects can be handled by DB2 12 before and after new function activation as well by DB2 11 with the fallback SPE applied. Fallback to DB2 11 (non-data sharing) or coexist with DB2 11 members (in data sharing) is possible after the catalog level has been changed to DB2 12 format (V12R1M500 catalog level).

There is no job to enable NFM. Instead, the DB2 **ACTIVATE** command is used to activate new functions. Because only one step is necessary to update the catalog during migration to new function, there is less impact to applications accessing the catalog thus improving catalog availability.

Figure 12-1 shows important steps during migration to DB2 12 with some jobs being removed from the DB2 11 migration process.

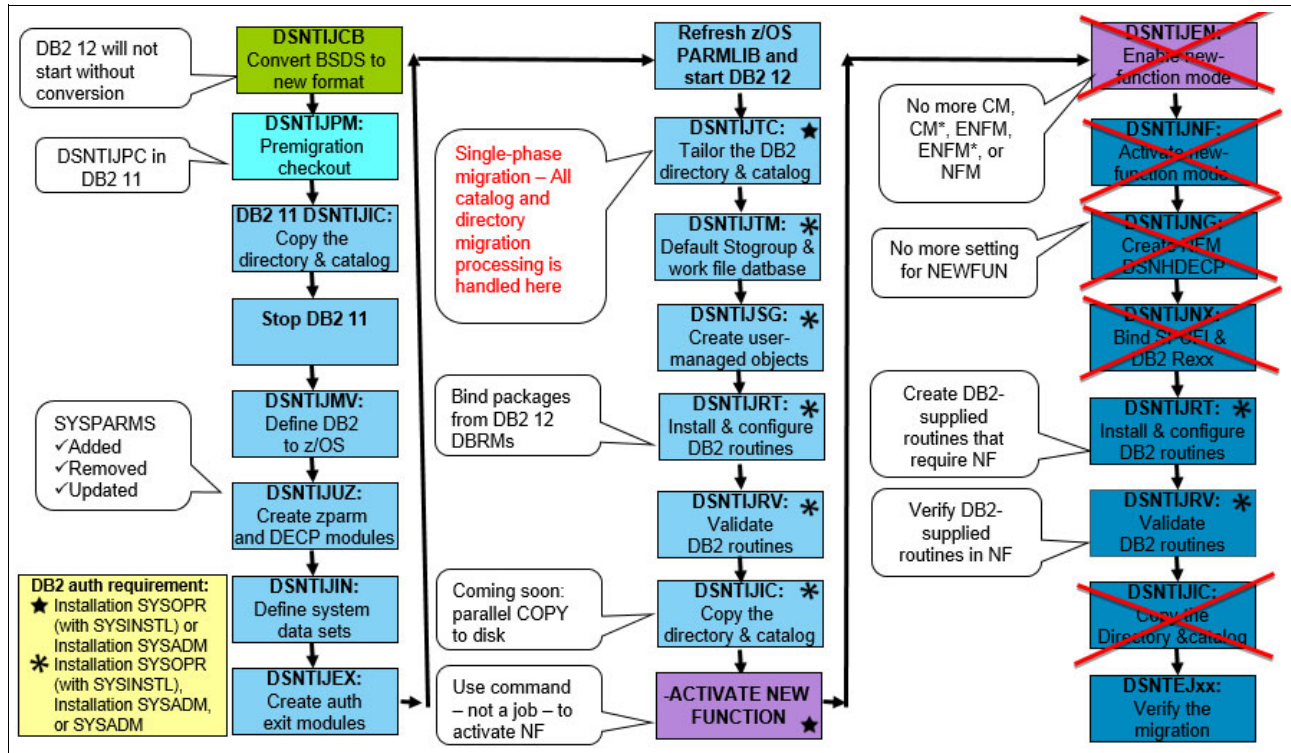


Figure 12-1 The migration process

12.2.1 Fallback SPE

The fallback SPE APAR PI33871 and any prerequisite fixes must be applied on DB2 11 before an attempt to start it with DB2 12 code level. If this is not done, DB2 12 restart will terminate with message DSNR045I (Example 12-1).

Example 12-1 DSNR045I message

```
DSNR045I -DB2A DSNRRPRC DB2 SUBSYSTEM IS STARTING 795
AND
```

```
IT WAS NOT STARTED IN A
PREVIOUS RELEASE WITH THE FALLBACK SPE APPLIED.
FALLBACK SPE APAR: UNKNOWN
NEW RELEASE LEVEL: E5F1F2D9
KNOWN LEVEL(S): 0000D780000D782000D785000D79000000000
```

In data sharing, all other active DB2 11 members must have the fallback SPE PI33871 applied. Message DSNX209E is issued if one member has the SPE on but another active member does not (Example 12-2).

Example 12-2 DSNX209E message

```

DSNX209E  -DB2A DSNGEDLC MEMBER DB2B IS AT A CODE 264
LEVEL WHICH IS NOT COMPATIBLE WITH THIS STARTING OR MIGRATING MEMBER.
  REQUIRED LEVEL: E5F1F2D9F1D4F5F0F0
  KNOWN LEVELS: 0000D7850000D79000000000

```

DB2 12 will not be creating or maintaining an INFO APAR. APARs are marked with appropriate fix categories and that is what should be used to identify necessary maintenance. A fix category is an identifier used to group and associate PTFs to a particular category of software fixes. You can use the following fix categories to check for migration fallback and coexistence maintenance:

- ▶ IBM.Migrate-Fallback.DB2.V12
Fixes that allow prior releases of DB2 to migrate to or fall back from DB2 12.
- ▶ IBM.Coexistence.DB2.SYSPLEXDataSharing
Fixes that enable DB2 releases to coexist when in data sharing mode.

12.2.2 EARLY code

If DB2 11 is at the prerequisite maintenance level prior to migration, its early code is upward compatible with DB2 12. The DB2 12 early code is downward compatible with DB2 11 also.

After applying the appropriate early code maintenance in prefix.SDSNLINK data set, the early code can be activated by either an initial program load (IPL) of the system or issuing the **MODIFY LLA, REFRESH** command followed by the **DB2 REFRESH DB2, EARLY** command. The **REFRESH** command displays the early code maintenance levels so you can check the output to be sure that the correct early code was applied (Figure 12-2).

DSN3117I DB2A MAINTENANCE LEVELS					
CSECT	DATE	APAR	CSECT	DATE	APAR
OSN3UR00	11/23/15	18.06	DSNAET03	01/04/16	11.22
OSNAET04	11/23/15	17.54	DSNAPRHO	11/23/15	17.54
OSNAPRHX	12/10/15	09.23	DSNVRMTR	11/23/15	18.01
OSNVRSRB	11/23/15	18.01	DSNZPARS	11/23/15	18.04
OSN3AC0X	11/23/15	18.05	DSN3CL0X	11/23/15	18.05
OSN3DEQ0	11/23/15	18.05	DSN3ECMS	11/23/15	18.05
OSN3EC0X	11/23/15	18.05	DSN3ENQ0	11/23/15	18.05
OSN3EST0	11/23/15	18.05	DSN3RDMP	11/23/15	18.05
OSN3RIB	11/23/15	18.05	DSN3RRSX	12/03/15	19.00
OSN3RRXF	11/23/15	18.06	DSN3RS0X	11/23/15	18.06
OSN3RTR0	11/23/15	18.06	DSN3SPRX	11/23/15	18.06

Figure 12-2 The DB2 -REFRESH command output

12.2.3 Pre-migration checkout

The DB2 BSDS data set must be converted to extended format using the **DSNTIJCB** job. If the BSDS data sets are not in extended format, the DB2 12 restart will fail with the DSNJ157J message. The preference is for this step to be done in DB2 11 NFM.

The **DSNTIJPM** and **DSNTIJPC** jobs are useful to identify actions needed prior to migration:

- ▶ Identify unsupported and deprecated Resource Limit Facility tables, Explain tables, utilities mapping tables, and Unicode columns in EBCDIC tables that must be converted to DB2 12 format after new function mode activation.
- ▶ Simple table spaces to be converted.
- ▶ Objects that must be dropped.
- ▶ DB2 12 supports application plans and packages bound in DB2 10 or DB2 12 only. Plans and packages bound in DB2 9 or earlier releases will be auto-rebound on their first execution on DB2 12. The pre-migration job **DSNTIJPM** identifies these objects. If they are executed in DB2 12, they will be auto-rebound and a later execution on a coexist DB2 11 member will result in another autobind (because DB2 11 cannot execute a plan or package bound in a later release). In addition, no PLANMGMT support is available when autobind occurs. Therefore, be sure to rebind them while in DB2 11 to avoid autobind risk during online migration.

To check DB2 11 data consistency and integrity on different objects, use the following utilities:

- ▶ DBD integrity for user data bases in catalog and directory:
 - REPAIR DBD TEST DATABASE dbname
 - REPAIR DBD DIAGNOSE DATABASE dbname
- ▶ DB2 referential integrity and table check constraint violations and consistency between base table and corresponding LOB/XML table spaces:
 - CHECK DATA TABLESPACE dbname.tsname part SCOPE ALL
- ▶ Consistency between table spaces and indexes:
 - CHECK INDEX ALL TABLESPACE or CHECK INDEX LIST (with appropriate LISTDEF)
- ▶ Invalid LOB values or structural problems with the pages in a LOB table space:
 - CHECK LOB TABLESPACE dbname.tsname for each LOB table space
- ▶ Integrity of the DB2 catalog:
 - IBM provided DSNTESQ queries (should always return zero rows)
- ▶ Options to check each page of a table space for consistency:
 - DSN1COPY with **CHECK** option

Another set of objects that should be checked prior to migration to DB2 12 is the expression-based indexes. When these indexes are created, DB2 saves the bound form (runtime structure) of the expression in the directory along with the index descriptor. If the index was created in DB2 9, its runtime structure cannot be executed on DB2 12, thus DB2 12 will implicitly alter/regenerate the index when it is referenced in a SQL statement as an access path. Similar to the recommendation to rebind packages bound prior to DB2 10, be sure to explicitly alter/regenerate the expression-based indexes created on DB2 9 before migrating to DB2 12. This avoids the implicit DDL activity while a DML application is executing on DB2 12 later. This suggestion is even more important if there is a DB2 11 coexist member in the data sharing group.

After the index is regenerated on DB2 12 for execution, if the DML application executes on a DB2 11 member, the index will be regenerated there again (because DB2 11 does not understand DB2 12 runtime structure). The query shown in Example 12-3 can be used to find the expression-based indexes created in DB2 9.

Example 12-3 Query to find the expression-based indexes created in DB2 9

```
SELECT NAME, RELCREATED
FROM SYSIBM.SYSINDEXES
WHERE (IX_EXTENSION_TYPE = 'S' OR IX_EXTENSION_TYPE = 'T') AND
      RELCREATED = 'M'
```

12.2.4 Creating DSNZPARM and DECP modules

The DSNTIJUZ creates the DSNZPxxx and DSNHDECP load modules. Note that the APPLCOMPAT DSNZPARM is set to V11R1 if the installation CLIST is run in MIGRATE mode. The preference is for this DSNZPARM to remain as is on migration because it is the default for the package's bind option APPLCOMPAT.

12.2.5 Creating and verifying routines supplied by DB2

DSNTIJRT must be run before new function activation and again after new function activation:

- ▶ *Before activation* to bind packages from DB2 12 DBRMs for existing DB2 supplied routines.
- ▶ *After activation* to install and configure new DB2 supplied routines:
 - SYSPROC.DSNUTILV
Similar to DSNUTILU but supports a utility statement of up to 2 GB in length.
 - SYSIBMADM.CREATE_WRAPPED
For creating obfuscated native SQL routines.

DSNTIJRV must also be run before new function activation and again after new function activation:

- ▶ *Before activation* to verify migration of existing DB2 supplied routines.
- ▶ *After activation* to verify installation and configuration of new DB2 supplied routines:
 - SYSPROC.DSNUTILV
Similar to DSNUTILU but supports a utility statement of up to 2 GB in length.
 - SYSIBMADM.CREATE_WRAPPED
For creating obfuscated native SQL routines

12.2.6 REBIND at each new release

After DB2 is migrated to DB2 12 function level V12R1M100, you should rebind existing packages for the following reasons:

- ▶ Expose applications with static SQL to DB2 12 while fallback is possible.
- ▶ Take advantage of improved performance from new runtime structure (bound static SQL).
- ▶ Avoid SPROCs disabled and “puffing” required when executing prior release packages.
- ▶ Expose to new query optimization and runtime enhancements.
- ▶ Expose to new access path choices.
- ▶ Reduce exposure to problems with migrated packages from earlier releases.
- ▶ Avoid application’s incorrect outputs and thread abnormally ended situations.
- ▶ Prepare for further usage of plan management in DB2 12 and beyond.

12.2.7 Activating new function level

The new **ACTIVATE** command is used to move to function level V12R1M500 where new functions are allowed. This is a **SCOPE (GROUP)** command. All members in a data sharing group must be DB2 12. When this command is successful, a DB2 11 member cannot restart in the same data sharing group. Fallback to DB2 11 is not allowed. Application compatibility can be set to V12R1M500, V12R1M100, V11R1, or V10R1.

The **ACTIVATE** command with **TEST** option can be issued to check whether all requirements are met to support the specified function level. After, the **DISPLAY GROUP** command can be used to verify more information and results.

For syntax of the **ACTIVATE** command, see Chapter 2, “Continuous delivery” on page 7.

12.2.8 Deprecated in earlier releases and removed in DB2 12

Consider the following information:

- ▶ Query I/O parallelism is no longer supported: query will be executed with sequential access mode.
- ▶ Resource limit tables (DSNRLMTxx) with formats earlier than DB2 1 are deprecated:
 - **START RLIMIT** command will issue **DSNT732I** message.
 - Tables are unusable.
- ▶ Resource limit tables (DSNRLSTxx) and related index with formats earlier than DB2 Version 8 are not supported:
 - The **DSNT731I** message is issued during the **START RLIMIT** command to inform that the **DSNRLSTxx** table is not used for RLF.
 - The **DSN9023I** message is issued, the **START RLIMIT** command fails if a supported **DSNRLMTxx** table is not available.

- ▶ Basic row format is deprecated:

All new tablespaces are created in reordered row format. Newly added partitions are created in reordered row format, unless the tablespace contains a table with an **EDITPROC**. The **LOAD REPLACE** or **REORG TABLESPACE** utility can be executed on the basic row format table spaces to convert them to reordered row format.

- ▶ Explain tables:
 - Error SQLCODE -20008 is returned to the application if the explain tables are created before DB2 11.
 - Warning SQLCOD +20520 is returned to the application if the explain tables are created in DB2 11.

These tables can be converted to DB2 12 format by either the ADMIN_EXPLAIN_MAINT stored procedure or the DSNTIJXA batch job.

- ▶ Utility mapping table:

The TARGET_XRID column in static mapping tables should be changed from CHAR(9) to CHAR(11) by using either the ALTER or DROP followed by CREATE statement. Enforcement begins after new function activation. The REORG utility ignores down-level static mapping tables, creates a temporary implicit mapping table, and issues informational message DSNU2900I and **DSNU2901I** (Figure 12-3).

```

DSNU2900I -DB2A 004 17:44:28.85 DSNURMAP - MAPPING TABLE IS SPECIFIED WITH A NON-
EXPANDED RID COLUMN
DSNU2901I -DB2A 004 17:44:29.80 DSNURMAP - MAPPING TABLE
SYSADM.REORG_MAPTABLE_DSNTX_0000
AND MAPPING INDEX SYSADM.REORG_MAPINDEX_DSNTX_0000
CREATED IN DSN8D12P.RM393E0D
  
```

Figure 12-3 Messages on REORG for old mapping table

12.3 Installing a new DB2 12 system

Figure 12-4 shows the DB2 12 installation process changes.

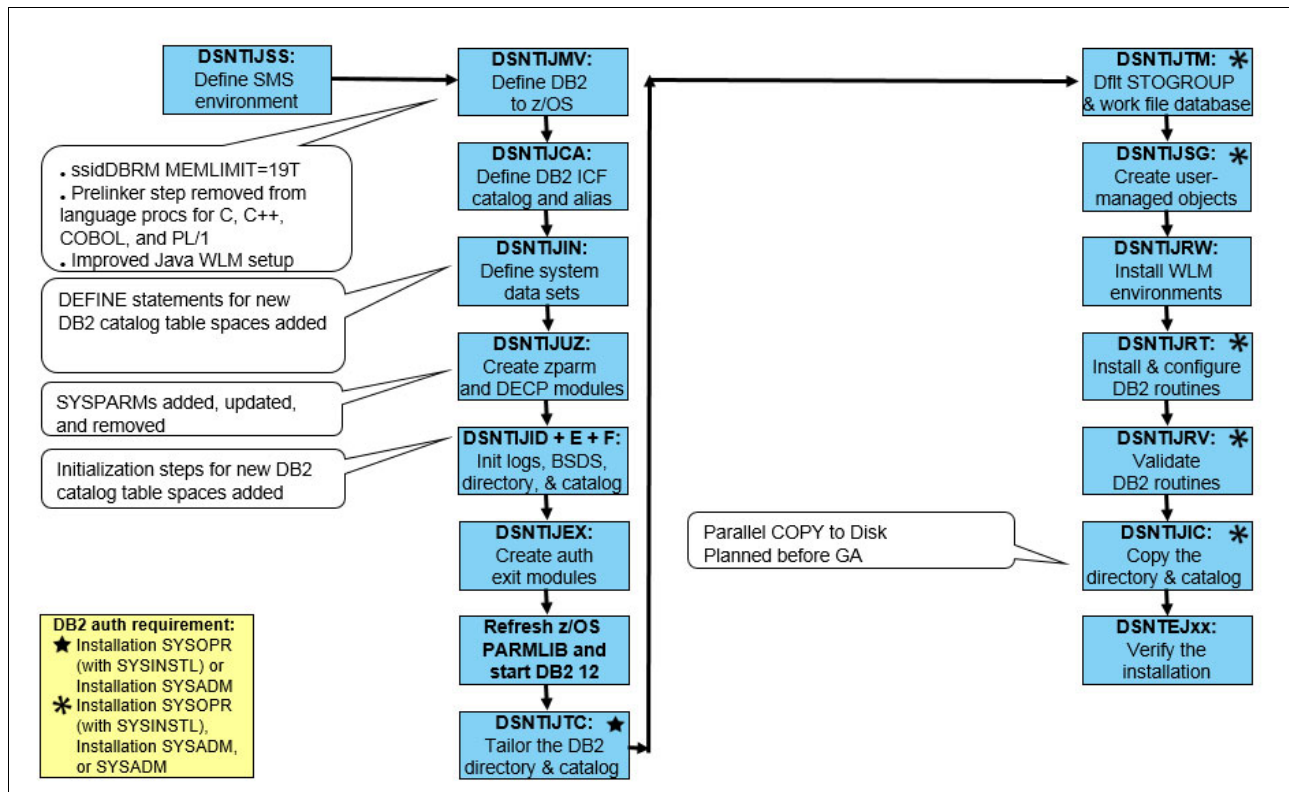


Figure 12-4 The installation process

12.3.1 Defines DB2 to z/OS

DSNTIJMV is used to define DB2 to the z/OS operating system. MEMLIMIT for ssidDBM1 address space has been changed from 4 TB to 19 TB to support new 16 TB limit for all buffer pool storage. MEMLIMIT is the limit on the total size of usable virtual storage above the bar in a single address space.

This panel also has improved setup for the DB2 supplied Java WLM environment (DSNWLM_JAVA):

- ▶ JAVAENV content is now minimal _CEE_ENVFILE, MSGFILE, and XPLINK.
- ▶ CEE ENVFILE now specifies DB2_BASE, JCC_HOME, JAVA_HOME, and JVMPROPS.
- ▶ JVMPROPS now documents use of parms.
- ▶ In all files, content no longer has trailing blanks.

12.4 Subsystem parameters

New DB2 releases typically have new and removed system parameters (DSNZPARMs) and also changes in their default values. This section describes the following information:

- ▶ New subsystem parameters
- ▶ Removed subsystem parameters
- ▶ Install Parameters Default Changes
- ▶ Deprecated system parameters

12.4.1 New subsystem parameters

The following list describes the parameters:

- ▶ **ALTERNATE_CP**: Specifies the name of an alternate SMS copy pool for the DB2 BACKUP SYSTEM utility.

Range:

- 1 – 14 alphanumeric: Indicates what BACKUP SYSTEM will use as the name of the alternate pool backup storage for system level backups and what RESTORE SYSTEM will use as the name of the alternate pool backup storage group for system level recoveries.
- Blank: Means that BACKUP SYSTEM will use the name specified in the ALTERNATE_CP keyword (if present) of the utility control statement as an alternate pool backup storage group when making the system level backup.

Default: blank

- ▶ **AUTH_COMPATIBILITY**: A list of options for overriding certain default authorization checks.

Range:

- **SELECT_FOR_UNLOAD**: If this option is specified, the UNLOAD utility checks whether the requester has the SELECT privilege on the target table. Otherwise, the UNLOAD utility checks whether the requester has the UNLOAD privilege on the specified table.
- Blank

Default: blank

- ▶ **CACHEDYN_STABILIZATION**: Specifies how DB2 is to stabilize cached dynamic SQL statements.

Range:

- **CAPTURE**: Statements can be stabilized through the **-START DYNQUERY** command by both **MONITOR(NO)** and **MONITOR(YES)**. DB2 will not load stabilized statements from SYSDYNQUERY.
- **LOAD**: Statements cannot be stabilized by any means. The **-START DYNQUERY** command will fail, and any **MONITOR(YES)** commands in progress will not stabilize statements, even if stabilization criteria are matched. During long prepare, DB2 will attempt to load stabilized statements from SYSDYNQUERY with which to run.
- **BOTH**: Statements can be stabilized through the **-START DYNQUERY** command by both **MONITOR(NO)** and **MONITOR(YES)**. During long prepare, DB2 attempts to load stabilized statements from SYSDYNQUERY with which to run.

- NONE: Statements cannot be stabilized by any means. The **-START DYNQUERY** command will fail, and any **MONITOR(YES)** commands in progress will not stabilize statements, even if stabilization criteria are matched. DB2 will not load stabilized statements from SYSDYNQUERY.

Default: BOTH

- ▶ **COMPRESS_DIRLOB**: Specifies whether LOB table spaces in the DB2 directory are to be compressed.

Range:

- NO: Means that LOB table spaces in the DB2 directory are not compressed.
- YES: Means LOB table spaces in the DB2 directory are compressed the next time they are reorganized.

Default: NO

- ▶ **COPY_FASTREPLICATION**: For the COPY utility, specify whether FlashCopy fast replication is PREFERRED, REQUIRED, or NONE for the creation of the FlashCopy image copy.

Range:

- PREFERRED
- REQUIRED
- NONE

Default: PREFERRED

- ▶ **DDL_MATERIALIZATION**: Specifies when DB2 should materialize changes to the definition of an object.

Range:

- ALWAYS_IMMEDIATE: For applicable requests, changes are materialized at the time the request is executed and the containing table space is placed in AREO* / REBUILD pending state. If there are any existing unmaterialized pending changes, the request fails. This is the existing behavior for ALTER TABLE ALTER COLUMN SET DATA TYPE.
- ALWAYS_PENDING: For applicable requests, changes are not materialized at the time the request and the affected objects are available until it is convenient to implement the changes. The containing table space is placed in AREOR state. If any immediate options specified in same statement, the change request fails. If any subsequent immediate changes are executed before a pending change is materialized, those subsequent immediate changes fail.

Default: ALWAYS_IMMEDIATE

- ▶ **DEFAULT_INSERT_ALGORITHM**: Specifies the default algorithm for inserting data into table spaces.

Range:

- 1: The basic insert algorithm is used.
- 2: The fast insert algorithm is used.

Default: 2

- ▶ **INDEX_MEMORY_CONTROL**: Specifies the amount of memory that DB2 should allocate for fast traversing of DB2 indexes.

Range:

- AUTO: Specifies that DB2 sets the upper limit of the storage to 20% of currently allocated buffer pools (as opposed to the defined buffer pool size).

- DISABLE: Specifies that DB2 returns any existing storage allocated for fast index traversal and allocates no further storage for the purpose.
- 500 - 200000: Indicates the storage limit in megabytes for fast index traversal.

Default: AUTO

- ▶ PAGEABLE_1MB_FOR_THREADS: Specifies whether DB2 may use 1 MB pageable storage when allocating HVSHARED above-the-bar object storage for thread pools and thread STACK.

Range:

- NO: DB2 will not use 1 MB pageable storage for these objects.
- YES: DB2 may use 1 MB pageable storage for these objects. YES requires a level of z/OS that supports 1 MB Pageable objects for HVSHARED. Sufficient real storage should be available to avoid performance issues related to paging. PAGEABLE_1MB_FOR_THREADS behavior is enabled only if REALSTORAGE_MANAGEMENT = OFF.

Default: YES

- ▶ PAGESET_PAGENUM: Specifies whether table spaces range-partitioned table spaces will be created to use absolute page numbers across partitions or relative page numbers.

Range:

- ABSOLUTE: Means that PBR table spaces and associated indexes are created with the same format and use of page number as DB2 11.
- RELATIVE: Means that PBR table spaces will be created so that PGNUM in the page header has no partition number, and the partition number is only contained in the header page for the partition.

Default: ABSOLUTE

- ▶ PEER_RECOVERY: Specifies whether a data sharing member is to participate in data sharing peer recovery.

Range:

- NONE: This member is not to participate in peer recovery. Use this option if you configured z/OS Automatic Recovery Manager (ARM) to restart failed DB2 members.
- RECOVER: This member should be recovered by a peer member in case it fails.
- ASSIST: This member should attempt to initiate peer recovery for other failed members. When this member detects a failure, it will attempt to initiate a LIGHT(YES) restart for the failed member if it hasn't already been initiated to recover the retained locks.
- BOTH: Both RECOVER and ASSIST options are to be activated for this member.

Default: NONE

- ▶ PROFILE_AUTOSTART: Specifies whether the **START PROFILE** command processing is to be automatically initiated as part of DB2 startup processing.

Range:

- NO: Indicates no automatic start behavior.
- YES: Specifies that **START PROFILE** command processing is to be automatically initiated when DB2 is started. This option is ignored if DB2 is started with an ACCESS(MAINT) or LIGHT(YES) specification.

Note: This option can impose a delay of DB2 availability when DB2 is started.

Default: NO

- ▶ **RLFENABLE:** Specifies the level of RLF governing when the resource limit facility (RLF) is started.

Range:

- **DYNAMIC:** Govern only dynamic SQL statements.
- **STATIC:** Govern only static SQL statements.
- **ALL:** Govern both dynamic SQL and static SQL statements.

Default: DYNAMIC

- ▶ **RLFERRSTC:** Specifies what action DB2 is to take for static SQL statements when the Resource Limit Specification governor encounters a condition that prevents it from accessing the resource limit specification table.

Range:

- **NOLIMIT:** Allows all static SQL statements to run without limit.
- **NORUN:** Terminates all static SQL statements immediately with an SQL error code.
- **1 - 5000000:** Specifies the number of service units to use as the default resource limit for all static SQL statements. If the limit is exceeded, the SQL statement is terminated.

Default: NOLIMIT

- ▶ **RLFERRDSTC:** Specifies what action DB2 is to take for a static SQL query from a remote location when the Resource Limit Specification governor encounters a condition that prevents it from accessing the resource limit specification table.

Range:

- **NOLIMIT:** Allows the remote static query to run without limit.
- **NORUN:** Terminates the remote static query immediately with an SQL error code.
- **1 - 5000000:** Specifies the number of service units to use as the default resource limit for a remote static query. If the limit is exceeded, the query is terminated.

Default: NOLIMIT

- ▶ **SQLLEVEL:** Specifies the DB2 function level to be used by the precompiler and coprocessor in the absence of the SQLLEVEL option.

Range:

- **V10R1:** The precompiler and coprocessor will accept only SQL that is valid in DB2 10 new function mode.
- **V11R1, V12R1M100:** The precompiler and coprocessor will accept only SQL that is valid in DB2 11 new function mode.
- **Function level:** The precompiler and coprocessor will accept only SQL that is valid in the specified function level.

Default: The maximum supported DB2 function level (on installation) or V12R1M100 (on migration)

- ▶ **STATFDBK_PROFILE:** Specifies whether statistics recommendations identified during query optimization should result in modifications to statistics profiles.

Range:

- **YES:** Means that DB2 will modify statistics profiles based on statistics recommendations during query optimization.
- **NO:** Means that DB2 will not modify statistics profiles based on statistics recommendations during query optimization. DB2 might still write the recommendations to the SYSIBM.SYSSTATFEEDBACK catalog table depending on the value of the subsystem parameter STATFDBK_SCOPE.

Default: YES

- ▶ UTIL_DBBSG: Specifies the name of a backup SMS storage group to be used by the DB2 BACKUP SYSTEM utility for the DB copy pool.

Range:

- Blank: Means that BACKUP SYSTEM will have HSM use the COPY POOL BACKUP storage group that is associated with each storage group specified for the copy pool.
- Valid SMS storage group name: Permitted only when ALTERNATE_CP is non-blank. This storage group must have been defined with the COPY POOL BACKUP attribute.

Default: blank

- ▶ UTIL_LGBSG: Specifies the name of a backup SMS storage group to be used by the DB2 BACKUP SYSTEM utility for the LOG copy pool.

Range:

- Blank: Means that BACKUP SYSTEM will have HSM use the COPY POOL BACKUP storage group that is associated with each storage group specified for the copy pool.
- Valid SMS storage group name: Permitted only when ALTERNATE_CP is non-blank. This storage group must have been defined with the COPY POOL BACKUP attribute.

Default: Blank

- ▶ UTILS_HSM_MSGDS_HLQ: Specifies 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).

Range:

- Blank: Means that utilities do not receive these messages from HSM and DFDSS and do not include them in diagnostics.
- A valid data set qualifier of 1 to 6 characters: A high-level qualifier that must also be registered in HSM through a SETSYS command. Data sets that use this high-level qualifier will be 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 will then delete the data set. These data sets will not include messages from DUMP processing because control is returned to DB2 before dump processing is complete.

Default: Blank

12.4.2 Removed subsystem parameters

These parameters are removed:

- ▶ EDMPOOL
- ▶ LOBVALA
- ▶ LOBVALS
- ▶ SQWIDSC
- ▶ XMLVALA
- ▶ XMLVALS
- ▶ RRF
- ▶ ALCUNIT
- ▶ CATALOG
- ▶ CACHE_DEP_TRACK_STOR_LIM
- ▶ CACHEDYN_FREELocal
- ▶ CHECK_SETCHKP
- ▶ CONSTOR

- ▶ MINSTOR
- ▶ DB2SORT
- ▶ INDEX_IO_PARALLELISM
- ▶ LEMAX
- ▶ SQWIDSC
- ▶ UTSORTAL

12.4.3 Install Parameters Default Changes

Changes are as follows:

- ▶ DSN6SPRM.APPLCOMPAT: From V11R1 to V12R1M500
- ▶ DSN6SPRM.EDM_SKELETON_POOL: From 10240 to 51200
- ▶ DSN6SPRM.PREVENT_NEW_IXCTRL_PART: From NO to YES
- ▶ DSN6ARVP.PRIQTY: From 4320 (blocks) to 125 (cylinders)
- ▶ DSN6ARVP.SECQTY: From 540 blocks to 15 cylinders
- ▶ DSN6SYSP.SMFACCT: From 1 to 1,2,3,7,8

12.4.4 Deprecated system parameters

Deprecated parameters are as follows:

- ▶ Subsystem parameter NEWFUNC is deprecated: Use SQLLEVEL instead.
- ▶ MATERIALIZE_NODET_SQLTUDF: In later DB2 releases, user-defined SQL table functions that are defined with NOT DETERMINISTIC always behave as if MATERIALIZE_NODET_SQLTUDF is set to YES.

12.5 Installation or migration without requiring SYSADM

Up to DB2 11, only users with installation SYSADM authority can install new a DB2 subsystem or data sharing group or migrate DB2 to the new release. However, a system operator (with installation SYSOPR authority) usually is the person performing the installation or migration steps which means he or she would be granted SYSADM authority. SYSADM authority also include access to all data. Certain government regulations and policies require that sensitive user data is not exposed to anyone, except the system administrator or data's owners. Therefore, to help protect user data and comply to security regulations, DB2 12 provides a way to install or migrate a DB2 subsystem without having SYSADM authority.

In DB2 12 compatibility mode, a user with installation SYSOPR authority can install and migrate a subsystem with no access to user data. Enhancements are made so that SYSOPR authority can perform works necessary in the process. For more information, see Chapter 10, "Security" on page 167.

12.6 Installation with z/OS Management Facility (z/OSMF)

With the goal of streamlining some areas in the z/OS system management, the z/OS Management Facility was introduced as a web-based user interface to perform traditional tasks and automate system-oriented tasks that run on the mainframe z/OS systems. By logging on to z/OSMF through a web browser, a user, with proper security defined on the z/OS system, can communicate to the z/OS system from anywhere, from any computer connected to the Internet to perform work.

The z/OSMF web-based interface can improve administrator productivity by allowing a single point of control for tasks such as managing z/OS software, tools, performance monitoring, problem diagnostics, and more.

See the following z/OS documentation for more detail about the z/OSMF product:

- ▶ z/OSMF installation and configuration:
IBM z/OS Management Facility Configuration Guide, SA38-0657-04
- ▶ z/OSMF workflow artifacts editing:
IBM z/OS Management Facility Programming Guide, SA32-1066-04

The z/OS Management Facility requires these items:

- ▶ z/OS Communications Server
- ▶ Security definitions (SAF)
- ▶ Other components are required for specific z/OSMF plug-ins
- ▶ IBM 64-bit SDK for z/OS Java Technology Edition V7
- ▶ You can use z/OSMF to modernize and automate various tasks required in the installation and migration processes to DB2 11 (Figure 12-5 on page 215). In DB2 12, enhancements were also made to the DB2 Installation CLIST (DSNTINST) and panels to generate z/OSMF workflow artifacts. These artifacts are used to automate the DB2 installation and migration tasks in z/OSMF. Then, the z/OSMF tasks can be scheduled for execution on the z/OS system to migrate or install DB2 12 with status monitoring, all from an easy-to-integrate web-based application.

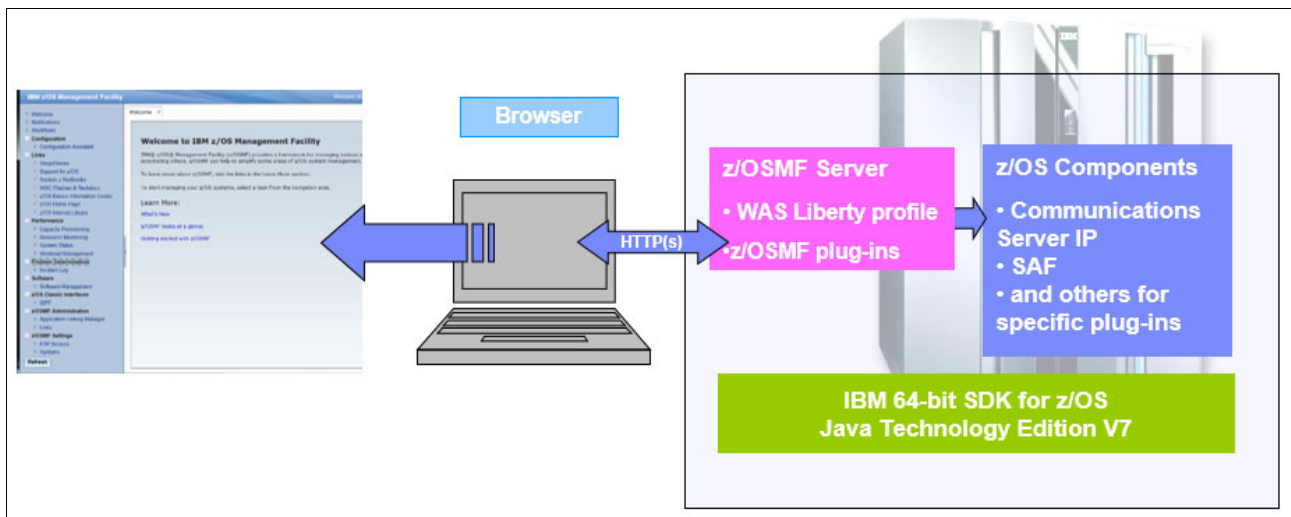


Figure 12-5 z/OS Management Facility

Figure 12-6 shows how the workflow artifacts are used by z/OSMF to interface to DB2 on the z/OS system. These artifacts (HLQ.SDSNMFSA, HLQ.SDSNSAM2, input variable files) are generated by the DB2 installation CLISTS.

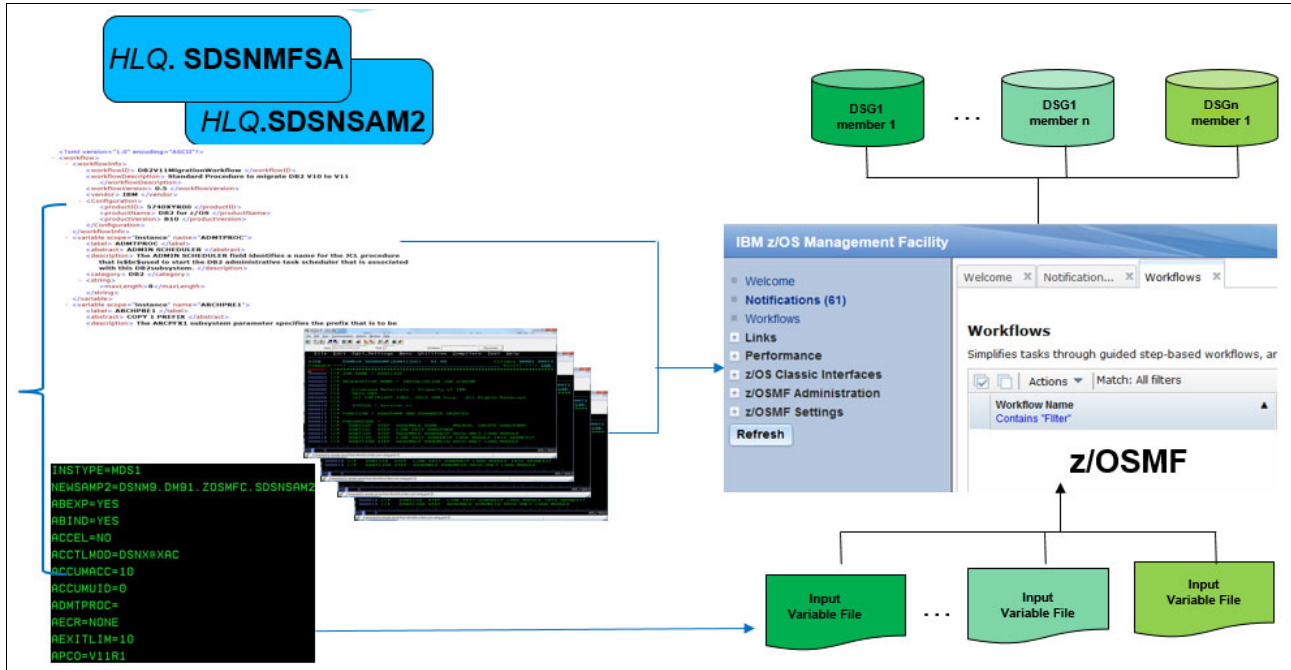


Figure 12-6 z/OSMF workflow generated for installing/migrating DB2

12.6.1 How to use DB2 installation CLIST and panels to generate z/OSMF artifacts

The new sample CLIST, DSN8IDB2, can be executed. It dynamically allocates to ISPF the data sets needed to run the DB2 installation CLIST, then starts the DB2 installation CLIST, and finally frees those data sets allocated. The following TSO command can be used to run this CLIST:

```
TSO EXEC 'prefix.SDSNCLST(DSN8IDB2)
```

The following figures (Figure 12-7 through Figure 12-10 on page 220) show installation panels and the input needed to create workflow artifacts (members in the HLQ.SDSNMFSA and HLQ.SDSNSAM2 data sets).


```

DSNTILA1  DB2 VERSION 12 INSTALL, UPDATE, AND MIGRATE - MAIN PANEL
===>

Will the installation/migration be performed with z/OSMF?
  USE Z/OSMF WORKFLOW  ===> YES      Yes or No

Check parameters and reenter to change:
  1 INSTALL TYPE        ===> MIGRATE  Install, Migrate, or Update
  2 DATA SHARING       ===> YES      Yes or No (blank for Update)

For migration only: Enter the data set and member name with the settings from
the previous Installation/Migration:
  3 DATA SET(MEMBER) NAME ===> DSNM9.SDSNSAMP.V11R1(M91TIDX1)
For DB2 SMP/E libraries (SDSNLOAD, SDSNMACS, SDSNSAMP, SDSNCLST, etc.), enter:
_ 4 LIBRARY NAME PREFIX  ===> DSNM9
_ 5 LIBRARY NAME SUFFIX  ===> V12R1.APAR
For install data sets (NEW.SDSNSAMP, NEW.SDSNCLST, RUNLIB.LOAD, etc.), enter:
_ 6 DATA SET NAME PREFIX ===> DSNM9.DM91.ZOSMFOU
_ 7 DATA SET NAME SUFFIX ===>
Enter to set or save panel values (by reading or writing the named members):
  8 INPUT MEMBER NAME    ===> DSNTIDXA  Default parameter values
  9 OUTPUT MEMBER NAME   ===> M91TIDX2  Save new values entered on panels
PRESS: ENTER to continue  RETURN to exit  HELP for more information

```

Figure 12-7 Creating workflow artifacts using DB2 installation panels

```

DSNTIP8          MIGRATE DB2 - PERFORMANCE AND OPTIMIZATION
===>

Enter data below:
_ 1 CACHE DYNAMIC SQL      ===> YES    NO or YES
_ 2 CACHE DYN STABILITY   ===> BOTH   CAPTURE, LOAD, BOTH, or NONE
_ 3 OPTIMIZATION HINTS    ===> NO    Enable optimization hints. NO or YES
_ 4 EVALUATE UNCOMMITTED  ===> NO    Evaluate uncommitted data. NO or YES
_ 5 SKIP UNCOMM INSERTS   ===> NO    Skip uncommitted inserts. NO or YES
_ 6 IMMEDIATE WRITE       ===> NO    NO or YES

_ 7 PLAN MANAGEMENT       ===> EXTENDED OFF, BASIC, EXTENDED
_ 8 PLAN MANAGEMENT SCOPE ===> STATIC  STATIC
_ 9 PACKAGE RELEASE COMMIT===> YES    Permit BIND and REBIND at COMMIT time
                                     on packages bound as RELEASE(DEALLOCATE)
_ 10 RANDOMIZE XML DOCID   ===> NO    NO or YES
_ 11 REAL TIME STATS      ===> 30    RTS time interval in minutes. 1-1440
_ 12 DISABLE EDM RTS      ===> NO    Disable EDM realtime stats. NO or YES
_ 13 STATISTICS FEEDBACK  ===> ALL   Scope of statistics feedback: ALL,
                                     DYNAMIC, NONE, or STATIC
_ 14 STATS PROFILE FEEDBACK===> YES    Statistics profile feedback: NO or YES

PRESS:  ENTER to continue  RETURN to exit  HELP for more information

```

Figure 12-8 Panel DSNTIP8 to generate variables for external DSNZPARM

```

DSNTIP3      Z/OSMF WORKFLOW ARTIFACTS GENERATION
===>

Type 's' in front of the subsystem parameters to be set as variables:
MACRO DSN6SPRM
  s ACCELMODEL                s AUTCMPAT
  s COMCRIT                   s EN_PJSJ
  s HONOR_KEEPCDICTIONARY     s INLISTP
  s MATERIALIZE_NODET_SQLTUDF s MAX_CONCURRENT_PKG_OPS
  s MAX_OPT_STOR              s MINDVSCL
  s MINRBLK                   s MXTBJOIN
  s NPGTHRSH                  s OPTXQB
  s OPT1ROWBLOCKSORT          s REALSTORAGE_MAX
  s REORG_DROP_PBG_PARTS      s REORG_LIST_PROCESSING
  s REORG_MAPPING_DATABASE    s REORG_PART_SORT_NPSI
  s RESTRICT_ALT_COL_FOR_DCC  s SECADM1_INPUT_STYLE
  s SECADM2_INPUT_STYLE       s SIMULATED_CPU_COUNT
  s SIMULATED_CPU_SPEED       s SJTABLES
  s SUBQ_MIDX                 s SUPPRESS_HINT_SQLCODE_DYN
  s UNION_COLNAME_7           s XML_RESTRICT_EMPTY_TAG
  s ZOSMETRICS                s CHECK_FASTREPLICATION
  s LIKE_BLANK_INSIGNIFICANT s PREVENT_NEW_IXCTRL_PART
PRESS: ENTER to continue  RETURN to exit  HELP for more information

```

Figure 12-9 Panel DSNTIP3 to generate variables for opaque DSNZPARM

```
DSNTIPM5          Z/OSMF JCL TEMPLATE CUSTOMIZATION
===>

Actions to complete before migration (DSNTIJPM):
  s THE SQL ID TO EXECUTE QUERIES IN DSNTIJPM ===> DB2ADM

Stop DB2 Version 11 activity (DSNTIJSO) and
Start DB2 Version 12 (DSNTIJSJA):
  s CONSOLE NAME                               ===> DB2ADM
      (CONSOLE NAME IS AN ID WITH MASTER AUTHORITY IN RACF)

Convert EXPLAIN tables to the current format (DSNTIJXA):
  s THE SQL ID TO EXECUTE QUERIES IN DSNTIJXA ===> DB2ADM
  s EXPLAIN TABLES SCHEMA                     ===> DB2ADM

The storage groups used for objects created by DSNTRIN in DSNTIJRT:
  s STORAGE GROUP FOR DATABASE AND TABLESPACE ===> SYSDEFLT
  s s STORAGE GROUP FOR INDEX                 ===> SYSDEFLT

PRESS:  ENTER to continue  RETURN to exit  HELP for more information
```

Figure 12-10 Panel DSNTIPM5 to stop DB2 11 and restart DB2 12

Those panels (Figure 12-7 on page 217 through Figure 12-10) will create XML files as members in the HDQ.SDSNMFS. They are workflow input variables. The panels also generate XML files as members in the HDQ.SDSNSAM2 data sets, which are workflow definition files.

Those members are listed in the next figures (Figure 12-11 on page 221 and Figure 12-12 on page 221).

Menu Functions Confirm Utilities Help						
ISRUDSM BROWSE	DSNM9.DM91.ZOSMFC.SDSNSMFA				Row 0000001 of 0000004	
Command ==>						Scroll ==> CSR
	Name	Prompt	Size	Created	Changed	ID
_____	DSNTIVMN					
_____	DSNTIVMS					
_____	DSNTIWMN					
_____	DSNTIWMS					
	End					

Figure 12-11 The created members in SDSNMFSA data sets

Menu Functions Confirm Utilities Help						
ISRUDSM BROWSE	DSNMZ.SMZ1.SDSNSAM2				Row 0000001 of 0000032	
Command ==>						Scroll ==> CSR
	Name	Prompt	Size	Created	Changed	ID
_____	DSNTEJ0					
_____	DSNTEJ1					
_____	DSNTEJ1L					
_____	DSNTEJ2A					
_____	DSNTIJCB					
_____	DSNTIJCI					
_____	DSNTIJCX					
_____	DSNTIJEX					
_____	DSNTIJIB					
_____	DSNTIJIC					
_____	DSNTIJIN					
_____	DSNTIJMV					
_____	DSNTIJNG					
_____	DSNTIJPM					
_____	DSNTIJRT					
_____	DSNTIJRV					
_____	DSNTIJRW					
_____	DSNTIJSA					
_____	DSNTIJSB					
_____	DSNTIJSG					
_____	DSNTIJSM					

Figure 12-12 The created members in SDSNSAM2 data sets

The following list describes sample workflow definition files (DSNTIWxx) provided by DB2 and their associated input variable files (DSNTIVxx):

DSNTIWMS, DSNTIVMS	For migrating a non-data sharing DB2 subsystem or the first member of a data sharing group
DSNTIWMD, DSNTIVMD	For migrating a subsequent member of a data sharing group
DSNTIWMN, DSNTIVMN	For enabling V12 code or V11 NFM
DSNTIWIN, DSNTIVIN	For installing a DB2 subsystem or the first member of a data sharing group
DSNTIWIA, DSNTIVI	For adding a DB2 subsystem to a data sharing group

12.6.2 Feeding the generated artifacts to z/OSMF

After generating the workflow definition files as artifacts through the install and migrate CLIST, you can use those artifacts to create a workflow instance on the web-based z/OS Management Facility. z/OSMF creates a workflow with a list of tasks. The tasks in the workflow are steps to install or migrate DB2 and can be assigned to a different user who is authorized in z/OSMF. The owner receives notifications for the tasks assigned. Then, the workflow creator or owner can review the steps created for the workflow. Subsequently, the workflow can be scheduled for automatic execution on z/OS, and progress can be monitored from the web-based window.

That process to migrate a non-data sharing subsystem to DB2 12, using the sample DSNTIWMS and DSNTIVMS files, is described in the following steps:

1. Create a workflow to install DB2 on z/OS.
2. Check list of tasks generated for the workflow.
3. Automate execution of the workflow which is the actual DB2 migration process on z/OS.
4. Monitor status of the workflow's tasks.

Figure 12-13 and Figure 12-14 on page 223 show the Create Workflow window on z/OSMF to create a workflow named DM91_Migrate DB2 11 to V12_012616 to migrate a non-data sharing member to DB2 12.

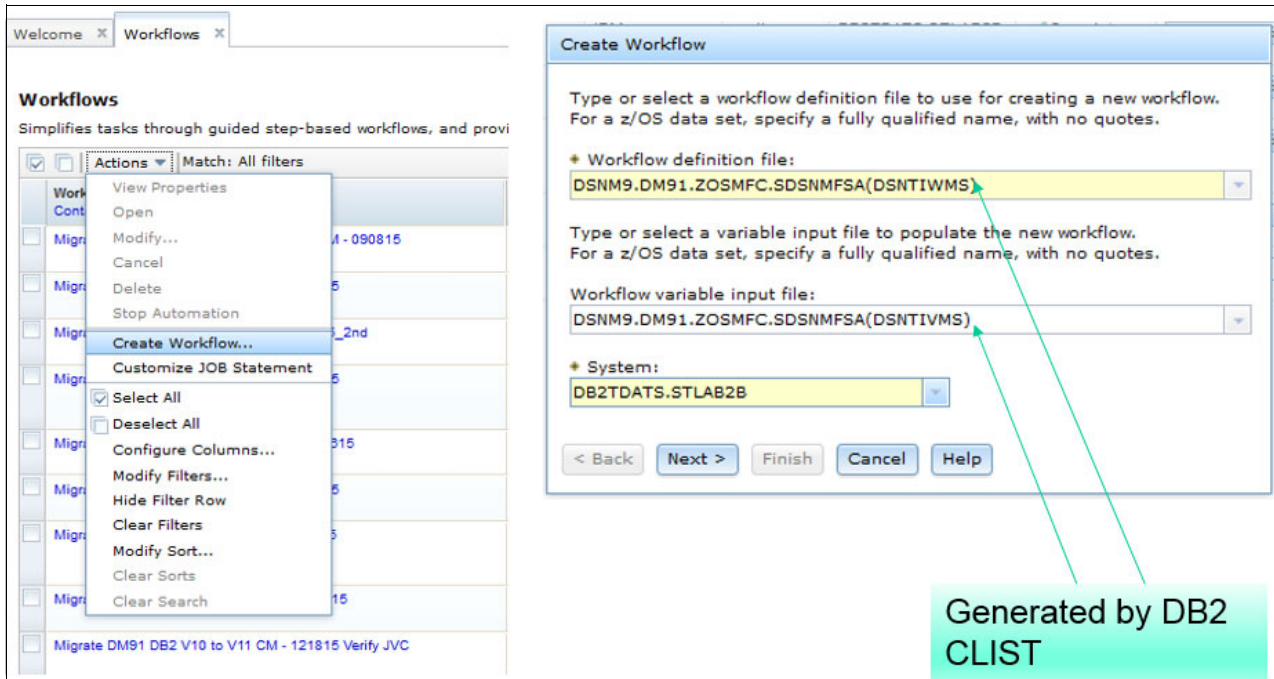


Figure 12-13 Create workflow dialog

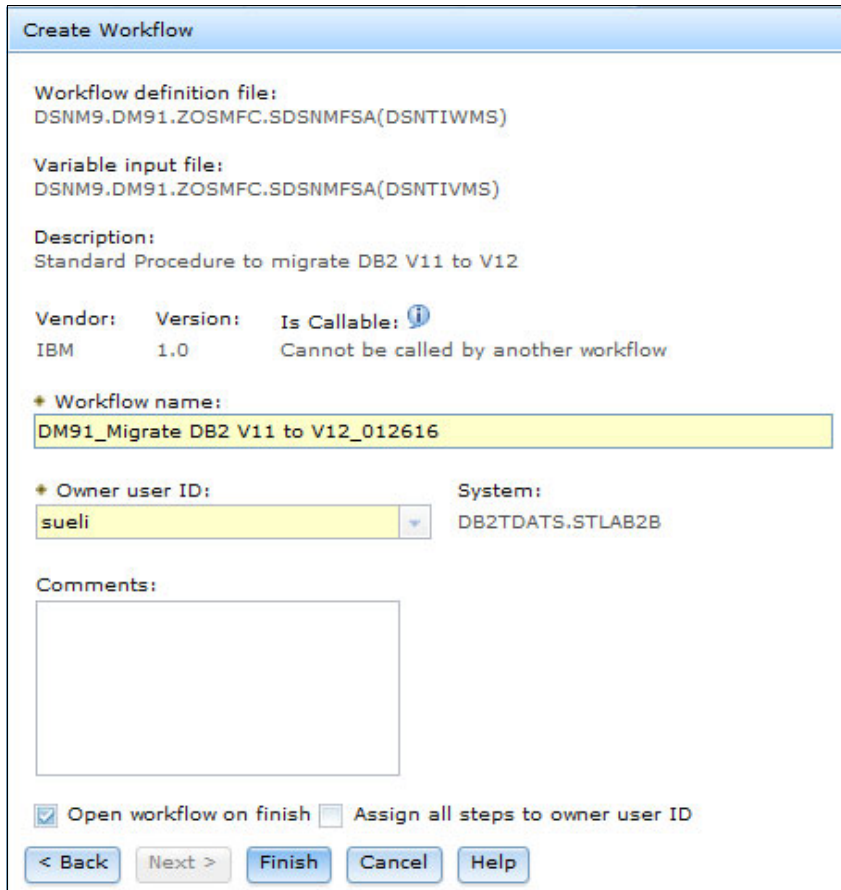


Figure 12-14 Creating a workflow named DM91_Migrate DB2 11 to V12_012616 to migrate a non-data sharing member to DB2 12

Figure 12-15 shows a z/OSMF window to check the steps created for the workflow DM91_Migrate DB2 11 to V12_012616.

Note: These steps are similar to the jobs listed in Figure 12-1 on page 202.

Welcome x Workflows x

Workflows > DM91_Migrate DB2 V11 to V12_012616

DM91_Migrate DB2 V11 to V12_012616

Description: Standard Procedure to migrate DB2 V11 to V12
 Owner: sueli
 System: DB2DATS.STLAB2B
 Is Callable: Cannot be called by another

Percent complete: 0%
 Steps complete: 0 of 20
 Status: In Progress

Workflow Steps

State Filter	No. Filter	Title Filter	CalledWorkflow Filter	Automated Filter	Owner Filter	Skill Category Filter	Assignees Filter
<input type="checkbox"/> Ready	1	Actions to complete before migration(DSNTUPM)		Yes	sueli	Database Administrator	sueli
<input type="checkbox"/> Not Ready	2	Stop DB2 Version 11 activity(DSNTJSO)		Yes	sueli	Database Administrator	sueli
<input type="checkbox"/> Not Ready	3	Convert BSDS records to extended format(DSNTJCB)		Yes	sueli	Database Administrator	sueli
<input type="checkbox"/> Not Ready	4	Start DB2 Version 11(DSNTUSB)		Yes	sueli	Database Administrator	sueli
<input type="checkbox"/> Not Ready	5	Take image copies of the directory and catalog(DSNTJIB)		Yes	sueli	Database Administrator	sueli
<input type="checkbox"/> Not Ready	6	Make DB2 CLISTS available to TSO and batch users(DSNTJVC)		Yes	sueli	Database Administrator	sueli
<input type="checkbox"/> Not Ready	7	Stop DB2 Version 11 activity(DSNTJSO)		Yes	sueli	Database Administrator	sueli
<input type="checkbox"/> Not Ready	8	Define DB2 initialization parameters(DSNTJUJZ)		Yes	sueli	Database Administrator	sueli
<input type="checkbox"/> Not Ready	9	Define system data sets(DSNTJIN)		Yes	sueli	Database Administrator	sueli
<input type="checkbox"/> Not Ready	10	Start DB2 Version 12 in MAINT mode(DSNTJSM)		Yes	sueli	Database Administrator	sueli
<input type="checkbox"/> Not Ready	11	Tailor DB2 Version 12 catalog(DSNTJTC)		Yes	sueli	Database Administrator	sueli
<input type="checkbox"/> Not Ready	12	Stop DB2 Version 12 activity(DSNTJSO)		Yes	sueli	Database Administrator	sueli
<input type="checkbox"/> Not Ready	13	Start DB2 Version 12(DSNTJSA)		Yes	sueli	Database Administrator	sueli
<input type="checkbox"/> Not Ready	14	Ensure that the catalog has no problems(DSNTJCX)		Yes	sueli	Database Administrator	sueli
<input checked="" type="checkbox"/> Not Ready	15	Prepare dynamic SQL		Yes	sueli	Database Administrator	sueli

Figure 12-15 Checking steps in the workflow DM91_Migrate DB2 11 to V12_012616

Figure 12-16 shows a window in z/OSFM used to perform automated execution of the steps in the workflow DM91_Migrate DB2 V11 to V12_012616.

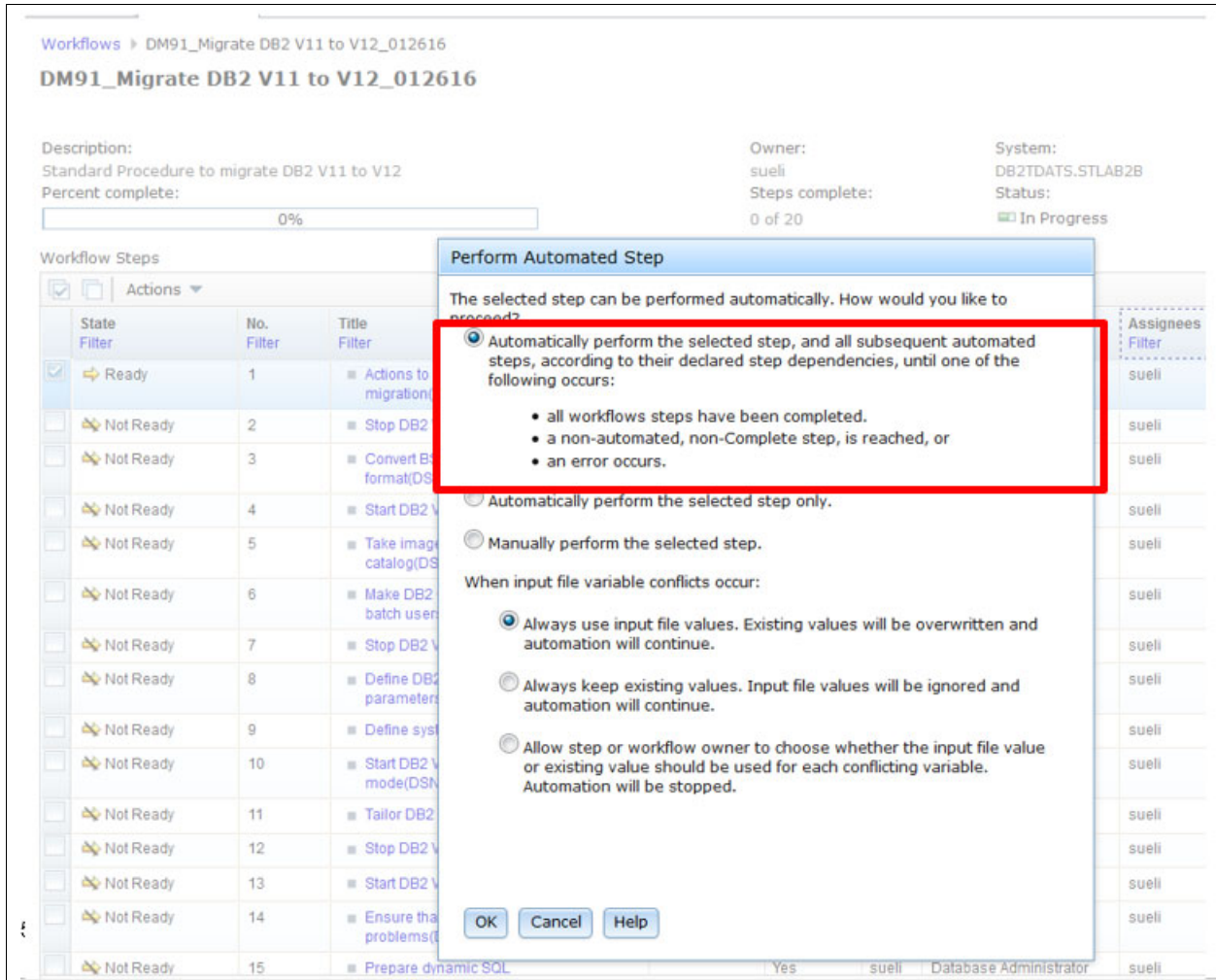


Figure 12-16 Automating execution for workflow DM91_Migrate DB2 V11 to V12_012616

Figure 12-17 shows a window to track the execution status of the steps in the workflow DM91_Migrate DB2 V11 to V12_012616.

Workflows > DM91_Migrate DB2 V11 to V12_012616

DM91_Migrate DB2 V11 to V12_012616

Description:
Standard Procedure to migrate DB2 V11 to V12

Percent complete:

Owner: sueli
Steps complete: 9 of 20

System: DB2TDATS.STLAB2B
Status: ■ In Progress

Workflow Steps

State Filter	No. Filter	Title Filter	CalledWorkflow Filter	Automated Filter	Owner Filter	Skill Category Filter	Assignees Filter
<input type="checkbox"/> Complete	1	■ Actions to complete before migration(DSNTJPM)		Yes	sueli	Database Administrator	sueli
<input type="checkbox"/> Complete	2	■ Stop DB2 Version 11 activity(DSNTJJO)		Yes	sueli	Database Administrator	sueli
<input type="checkbox"/> Complete (Override)	3	■ Convert BSDS records to extended format(DSNTJCB)		Yes	sueli	Database Administrator	sueli
<input type="checkbox"/> Complete	4	■ Start DB2 Version 11(DSNTJSB)		Yes	sueli	Database Administrator	sueli
<input type="checkbox"/> Complete	5	■ Take image copies of the directory and catalog(DSNTJIB)		Yes	sueli	Database Administrator	sueli
<input type="checkbox"/> Complete	6	■ Make DB2 CLISTS available to TSO and batch users(DSNTJVC)		Yes	sueli	Database Administrator	sueli
<input type="checkbox"/> Complete	7	■ Stop DB2 Version 11 activity(DSNTJJO)		Yes	sueli	Database Administrator	sueli
<input type="checkbox"/> Complete	8	■ Define DB2 initialization parameters(DSNTJUJZ)		Yes	sueli	Database Administrator	sueli
<input checked="" type="checkbox"/> Complete	9	■ Define system data sets(DSNTJIN)		Yes	sueli	Database Administrator	sueli
<input type="checkbox"/> Ready	10	■ Start DB2 Version 12 in MAINT mode(DSNTJSM)		Yes	sueli	Database Administrator	sueli
<input type="checkbox"/> Not Ready	11	■ Tailor DB2 Version 12 catalog(DSNTJTC)		Yes	sueli	Database Administrator	sueli
<input type="checkbox"/> Not Ready	12	■ Stop DB2 Version 12 activity(DSNTJJO)		Yes	sueli	Database Administrator	sueli
<input type="checkbox"/> Not Ready	13	■ Start DB2 Version 12(DSNTJSA)		Yes	sueli	Database Administrator	sueli
<input type="checkbox"/> Not Ready	14	■ Ensure that the catalog has no		Yes	sueli	Database Administrator	sueli

Figure 12-17 Tracking status of workflow DM91_Migrate DB2 11 to V12_012616 - the migration process from z/OSMF

Note: Because z/OSMF also supports the Representational State Transfer (REST) API, you can also use those interfaces to execute the workflow definition files and their associated input variable files directly.

12.7 Temporal catalog

DB2 collects real time statistics (RTS) during different activities such as SQL INSERT, UPDATE, DELETE, utilities, DDL statement, and so on. DB2 always generates in-memory statistics for each table space and index space in your system, including catalog objects. DB2 periodically writes these real-time statistics to the SYSIBM.SYSTABLESPACESTATS and SYSIBM.SYSINDEXSPACESTATS catalog RTS tables at a specified interval. These real-time statistics help to determine when objects require maintenance by utilities such as REORG, RUNSTATS, or COPY.

Real-time statistics data is valuable to observe recent changes in size and organization of table space partitions and index space partitions. Many DB2 customers have had concerns with the lack of historical trending for real-time statistics data in the DB2 catalog objects over years. This lack of functionality prevents customers from timely taking important actions such as computing rate of change, and smarter automated management of the DB2 catalog indexes and table spaces. From such analysis on historical information, customers may be able to automate responses for near out-of-space conditions, or reorganize the objects as needed.

Additionally, the system auditors might require the ability to determine who had which authority at which time. DB2 11 can store the current state of authorization and preserve authorization history on the user tables. However, this functionality is not available on the DB2 catalog tables. Consequently, customers need to use log analysis tools to investigate changes in authorization.

DB2 10 introduced the temporal table feature, which keeps historical information for the table's data and activities on the table. During the migration to DB2 12 process, DB2 begins to take advantage of this feature and apply it on some catalog tables to address the concerns. New history tables for catalog objects are added during DB2 12 migration.

The DB2 12 CATMAINT process issues the following DDL statements:

- ▶ ALTER TABLE ADD column statement to add new SYS_START as row-begin, SYS_END as row-end, and TRANS_START as transaction-start-ID columns in 16 base access control tables and 2 RTS tables.
- ▶ ALTER TABLE ADD PERIOD statement to add a SYSTEM_TIME period to 16 base access control tables and 2 RTS tables.
- ▶ CREATE TABLE statements to create 18 new catalog tables as history tables for the 16 base access control tables and 2 RTS tables. The history tables are created in table spaces with **DEFINE(NO)** to avoid creating them for those customers that will never use system-period data versioning capabilities in the catalog.

Note: These definitions are done by migration and installation, but actual system-period data versioning enablement is to be done by the user.

The following access control tables and RTS tables are affected by the DDL statements in the previous list during CATMAINT:

- ▶ SYSAUDITPOLICIES
- ▶ SYSCOLAUTH
- ▶ SYSCONTEXT
- ▶ SYSCONTEXTAUTHIDS
- ▶ SYSCONTROLS
- ▶ SYSCTXTTRUSTATTRS
- ▶ SYSDBAUTH
- ▶ SYSINDEXSPACESTATS
- ▶ SYSPACKAUTH
- ▶ SYSPLANAUTH
- ▶ SYSRESAUTH
- ▶ SYSROUTINEAUTH
- ▶ SYSSCHEMAAUTH
- ▶ SYSSEQUENCEAUTH
- ▶ SYSTABAUTH
- ▶ SYSTABLESPACESTATS
- ▶ SYSUSERAUTH
- ▶ SYSVIEWDEP

Their corresponding history tables are created with the same column names, data types, and attributes (Figure 12-18). These history tables can be updated with proper authorization.

Catalog table name	Table space	Description
SYSAUDITPOLICIES_H	SYSTSADH	SYSAUDITPOLICIES history table
SYSCOLAUTH_H	SYSTSFAH	SYSCOLAUTH history table
SYSCONTEXT_H	SYTSCNH	SYSCONTEXT history table
SYSCONTEXTAUTHID_H	SYTSCXH	SYSCONTEXTAUTHIDS history table
SYSCONTROLS_H	SYTSCTH	SYSCONTROLS history table
SYSCTXTTRUSTATTR_H	SYTSTA_H	SYSCTXTTRUSTATTRS history table
SYSDBAUTH_H	SYTSDBH	SYSDBAUTH history table
SYSINDEXSPACESTATS_H	SYTSSIH	SYSINDEXSPACESTATS history table
SYSPACKAUTH_H	SYTSPKH	SYSPACKAUTH history table
SYSPLANAUTH_H	SYTSP_LH	SYSPLANAUTH history table
SYSRESAUTH_H	SYSGPAUH	SYSRESAUTH history table
SYSROUTINEAUTH_H	SYTSSRAH	SYSROUTINEAUTH history table
SYSSCHEMAAUTH_H	SYTSSCH	SYSSCHEMAAUTH history table
SYSSEQUENCEAUTH_H	SYTSSAH	SYSSEQUENCEAUTH history table
SYSTABAUTH_H	SYTSTBH	SYSTABAUTH history table
SYSTABLESPACESTATS_H	SYTSTSH	SYSTABLESPACESTATS history table
SYSUSERAUTH_H	SYTSUAH	SYSUSERAUTH history table
SYSVIEWDEP_H	SYTSDVH	SYSVIEWDEP history table

Figure 12-18 New catalog history tables

12.7.1 System-period data versioning for two RTS catalog tables

You can issue the ALTER TABLE statement with the ADD VERSIONING clause to enable system-period data versioning on the SYSIBM.SYSTABLESPACESTATS and SYSIBM.SYSINDEXSPACESTATS catalog tables. The alters can be done after new function activation. DB2 will then manage the corresponding history tables, SYSIBM.SYSTABLESPACESTATS_H and SYSIBM.SYSINDEXSPACESTATS_H, as system temporal tables. Data in these history tables can help to determine historical trending for every table space and index space's real-time statistics in the system.

The temporal relationship can be eliminated with the ALTER TABLE DROP VERSIONING statement as needed.

Such temporal relationships might be supported in the future on the access control history tables in the DB2 catalog, introduced by DB2 12 migration, to preserve authorization history. At this time, nn SQLCODE -607 message is issued for attempt to do so.

12.7.2 Real-time statistics externalization during migration

DB2 12 migration process is also enhanced to have the ability to disable RTS externalization during migration processing. This ability can be useful in many CATMAINT operations where RTS updates might interfere with CATMAINT type operations.

At the beginning of migration processing, RTS externalization will be disabled and when all necessary processing is completed, it will be re-enabled. New message DSNT537I will be issued when RTS externalization is enabled and disabled. In addition, the RTS Daemon will automatically enable externalization after 3 hours to ensure an abend or termination in CATMAINT processing does not leave RTS externalization permanently disabled.

After RTS statistics externalization is re-enabled at the end of migration, processing statistics are not immediately externalized. Instead, they are externalized at the next regular statistics externalization interval.



Performance

This chapter describes the DB2 12 enhancements that provide many types of improvements in reducing elapsed and CPU time. These added values improve processing and render better use of resources. An important aspect to mention is that this chapter discusses the results of IBM early observations and feedback from the Early Support Program (ESP).

This chapter covers the following topics:

- ▶ Performance expectations
- ▶ In-memory buffer pool
- ▶ In-memory index optimization
- ▶ Improved insert performance for non-clustered data
- ▶ Query performance enhancements

13.1 Performance expectations

This topic describes the performance expectations based in the measurement evaluations for the DB2 12 performance enhancements.

Online transaction processing (OLTP) achieved a CPU utilization reduction without an index memory feature and also by exploiting an index in-memory feature. Further reduction is possible with contiguous buffer pools, the RELEASE(DEALLOCATE) bind option, or both.

The insert improvements for non-clustered data achieved CPU reduction with throughput improvement if the current bottleneck is from space, search, or page contentions.

Query performance has a wide range of improvement, achieving CPU reduction for query workloads and improving efficiency by reducing other resource consumption.

13.2 In-memory buffer pool

Up to DB2 11, when the buffer pool was not large enough to contain the object, page-stealing might have occurred, so pages that did not fit within the size of the buffer pool were managed by the first-in, first-out (FIFO) algorithm.

In DB2 12, the updated PGSTEAL(NONE) option of the **ALTER BUFFERPOOL** command (which makes possible the assigning of objects to in-memory buffer pools, in which the overflow area size is automatically determined by DB2), generally 10% of the VPSIZE value in the range of 50 - 6400 buffers and the creation of overflow happens at the time that the buffer pool is allocated. If the buffer pool is not large enough to support the objects, the pages that do not fit in the main part of the buffer pool are placed in the overflow area, where in page stealing occurs. Figure 13-1 shows a syntax diagram for the **ALTER BUFFERPOOL** command using the PGSTEAL(NONE) option.

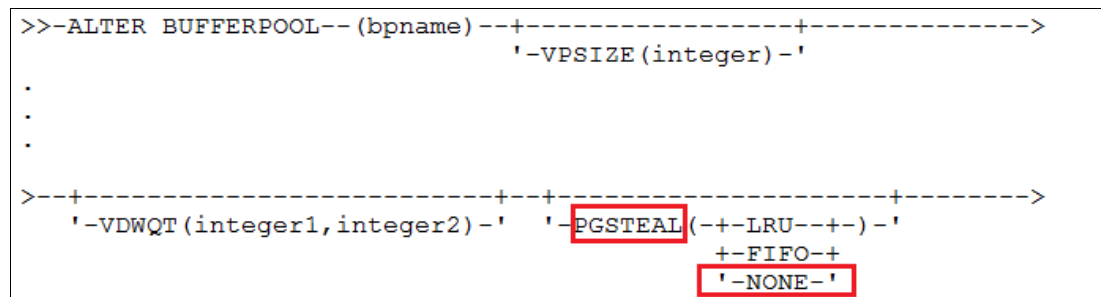


Figure 13-1 Syntax diagram for ALTER BUFFERPOOL command using PGSTEAL(NONE) option

This new feature gets direct row access, avoiding getpage overhead and providing CPU reduction measured for OLTP.

13.3 In-memory index optimization

DB2 12 introduces index Fast Traverse Block (FTB) with the Index Manager to optimize the memory structure for fast index lookups and improving random index access.

Up to DB2 11, Index Manager maintained transversal information in index lookaside, which keeps track of pages visited on the way to access the required index leaf page and then this

information was kept from one access to the next. So, this method is good for sequential access of an index, but thinking about random access of the index, several getpages are required if the look aside does not get a parent of the leaf. Therefore, FTB is a complement for index lookaside.

The use of FTBs is supported only on UNIQUE, PADDED, and NOT PADDED indexes with a key size of 64 bytes or less. The best candidates for using FTB are indexes that support heavy read access, indexes on tables with a random insert or delete pattern, and indexes with high PCTFREE.

A new storage group (acomidmgrcl20) is assigned to maintain the FTBs that are created. The storage group characteristics for FTB are as follows:

- ▶ Minimum of 10 MB
- ▶ Upper limit of 200 GB
- ▶ Default value of 20% of currently allocated buffer pool storage or 10 MB

In addition, FTB storage is not part of buffer pool allocation.

A daemon process monitors index usage and allocates FTB storage to the indexes that can benefit from this feature. FTB allocation is not permanent, so if the daemon determines that FTB storage is not being used, FTB storage can be removed.

Figure 13-2 exemplifies the usage of random keyed access.

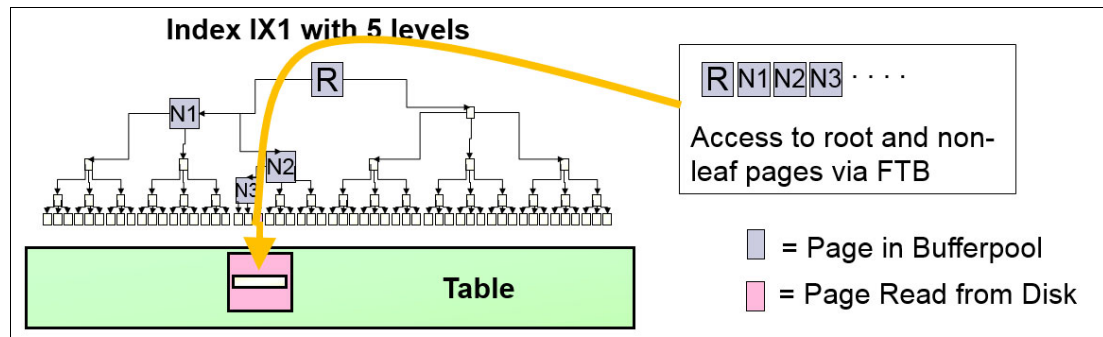


Figure 13-2 Random keyed access usage

To accommodate the fast transverse feature, a catalog table (SYSIBM.SYSINDEXCONTROL) was created, SYSINDEXCONTROL, which specifies time windows to control the use of memory allocated for an index.

Table 13-1 demonstrates how the data is stored in SYSIBM.SYSINDEXCONTROL.

Table 13-1 SYSIBM.SYSINDEXCONTROL example

SSID	Partition	IX Name	IX Creator	TYPE	ACTION	MONTH_WEEK	MONTH	DAY	FROM_TIME	TO_TIME
DB2A	12	IXABC	DBDA	F	D	W		7		
DB2B		IXZ12	DBDA	F	D	M		1	0000	1200
	23	IXDEF	DBDA	F	F	M				

Table 13-1 on page 233 shows the following information:

- ▶ On member DB2A, disable FTB on partition 12 of index IXABC every Sunday.
- ▶ On member DB2B, disable FTB on all partitions of index IXZ12 until noon on the first day of every month.
- ▶ On all members, force FTB on Partition 23 of index IXDEF at all times.

Figure 13-3 illustrates a syntax diagram to display for index memory usage.

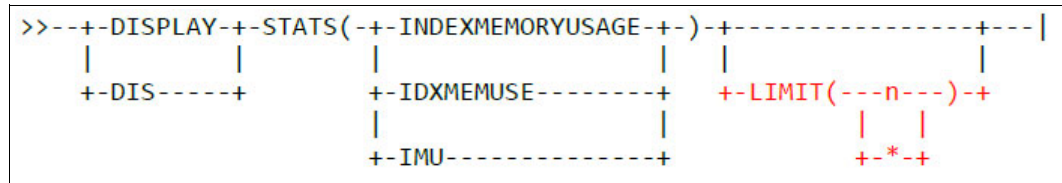


Figure 13-3 Display syntax diagram for index memory usage

13.4 Improved insert performance for non-clustered data

DB2 12 introduces an insert algorithm that eliminates page contention and false leads. This algorithm is called *fast insert*, which brings the concept of multiple algorithms for insert. The fast insert algorithm is the default algorithm for MEMBER CLUSTER universal table spaces (UTS) because universal table spaces are strategic and MEMBER CLUSTER addresses cross member space map contention. Tables defined as APPEND and NON-APPEND can use fast insert too.

Fast insert is available on DB2 12 after activation of new function because new log records are introduced. For information about activating new function, see Chapter 2, “Continuous delivery” on page 7.

A new subsystem parameter (ZPARM) is provided to define system-wide default. It is used if a need exists to change the DB2 default insert algorithm level. DDL keywords are provided and can be used when a specific insert algorithm level is needed.

Figure 13-4 shows that DB2 achieved three times the response time improvement of CPU reduction using new DB2 12 insert algorithm.

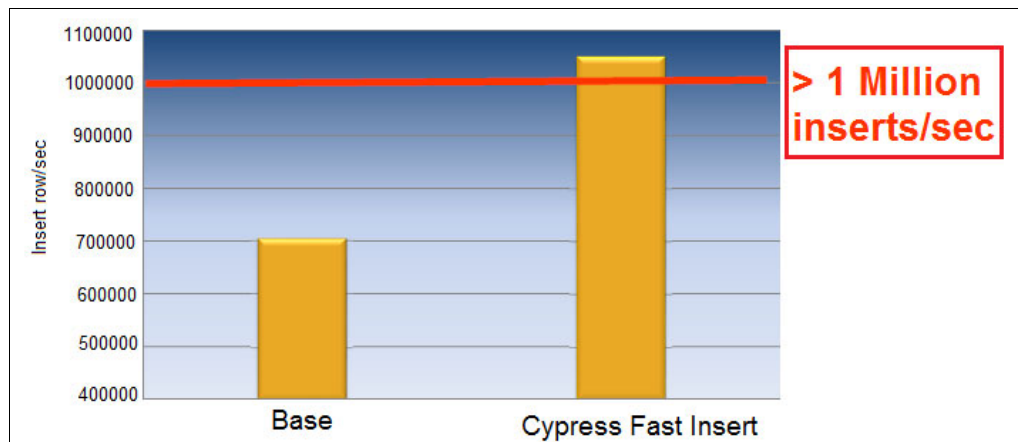


Figure 13-4 Fast insert algorithm CPU reduction representation

The required external controls for fast insert algorithm enhancement are described next.

13.5 Query performance enhancements

DB2 12 provides query performances enhancements that result in CPU and elapsed-time savings. Figure 13-5 illustrates some query performance measurement results that are covered in this topic.

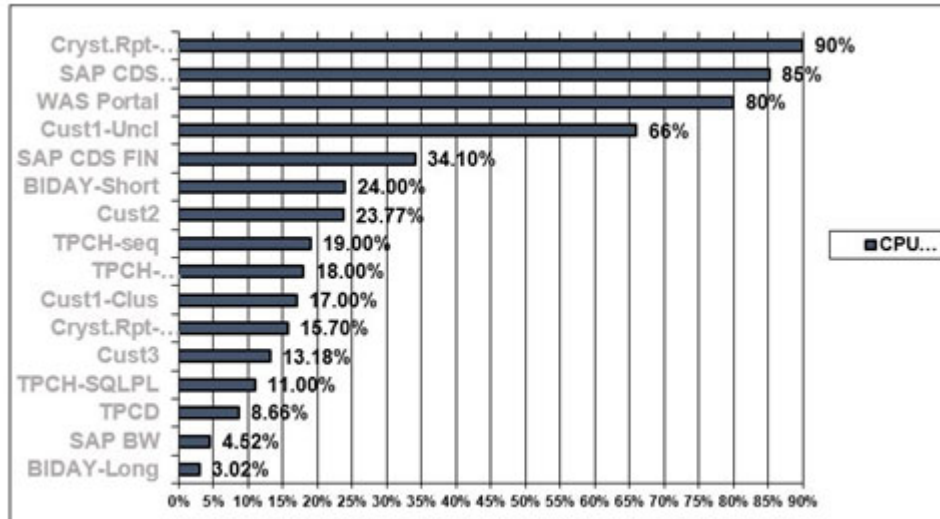


Figure 13-5 Query performance measurements results

The following topics explain and exemplify how these numbers were achieved:

- ▶ UNION ALL and Outer Join enhancements
- ▶ Sort improvements
- ▶ Predicate optimization
- ▶ Execution time adaptive index

13.5.1 UNION ALL and Outer Join enhancements

The query patterns UNION ALL and Outer Join have similar issues with materialization (workfile) usage and inability to apply filtering early.

DB2 12 introduces the following high-level solutions:

- ▶ Reorder outer join tables to avoid materializations
- ▶ UNION ALL and Outer Join predicate pushdown
- ▶ Avoid workfile for outer materialization
- ▶ Push predicates inside UNION ALL legs or outer join query blocks
- ▶ Sort outer into table expression order
- ▶ Enable sparse index for inner table/view expression
- ▶ Pruning unused columns from materialized result
- ▶ Extended LEFT JOIN table pruning

Reorder outer join tables to avoid materializations

Allowing DB2 to internally reorder the outer join tables within the query overcomes a limitation that can be exposed when combining outer and inner joins in the same query. In certain instances, DB2 11 and previous releases materialized some tables that can result in local or join filtering not being applied before the materialization. Some users who have been exposed to this performance challenge have rewritten their queries to ensure that all inner

joins appear before outer joins in the query, if possible. Rewriting a query is often difficult given the proliferation of generated queries and applications being deployed without thorough performance evaluation. Therefore, minimizing exposure to this limitation in DB2 12 provides a valuable performance boost for affected queries.

UNION ALL and Outer Join predicate pushdown

When joining a table expression or an undistributed UNION ALL, DB2 considers whether to push the join predicate into the table expression or UNION ALL legs.

The decision was cost-based because the decision whether to push the join predicate into the view must be based on whether the query will run faster by pushing down the predicate, so if the cost of the query is estimated to be smaller when the join predicate is pushed down, then DB2 pushes down that join predicate.

In this way, choosing the best performing access is possible. It also improves archive transparency queries that internally transform a base table into a materialized table expression. For more information about archive transparency, see *Managing Ever-Increasing Amounts of Data with IBM DB2 for z/OS: Using Temporal Data Management, Archive Transparency, and the DB2 Analytics Accelerator*, SG24-8316.

Avoid workfile for outer materialization

DB2 12 can “pipeline” the rows from the first UNION ALL (on the left side of the join). Transferring rows from one query block to another can require the results to be written (materialized) to an intermediate work file. Pipeline means to pass the rows from one query block to the next, without writing the intermediate result to a work file.

The work file avoidance for outer materialization with correlation predicate has the objective of improving the performance.

Up to DB2 11, one work file was created to materialize the result of inner table expression for each outer row; also, each work file was only allocated and read once, then was deallocated. This method applied a reduction of allocation and deallocation overhead by reusing the same work file, and in addition it applied only when the workfile fits a 32K page and the inner guaranteed to return one row for each probe.

DB2 12 is able to change its execution time logic to call the UNION ALL processing directly to avoid work file usage. Using this new method, DB2 enables the leading table expression pipeline join to subsequent tables without materializing the leading table expression, saving the cost of work file allocation and deallocation.

The results for this enablement are as follows:

- ▶ For outer pipelining (for the outer table of a join):
 - CPU time savings for the queries with access path change from ACCESTYPE 'R' to 'O'
 - Workfile get page counts reduced due to the avoidance of workfile materialization.
- ▶ For inner pipelining (for the inner table of the join):
 - Internal DB2 workloads achieved CPU reduction in WF getpages
 - CPU and elapsed reduction for best case

Push predicates inside UNION ALL legs or outer join query blocks

Figure 13-6 demonstrates performance challenges when UNION ALL is combined with an outer join. The original query is shown at the top of the figure as a simple two-table (left outer) join of T1 to T2. In this example, both T1 and T2 have archive enabled (which refers to the DB2 11 transparent archive feature), thus DB2 will rewrite the query to include the active and archive tables—with T1 circled and the arrow pointing to the first UNION ALL on the left side of the join, and T2 circled with arrow pointing to the second UNION ALL on the right side of the join. This representation is true of any UNION ALL within a view, where the view definition is replaced within the query where it is referenced.

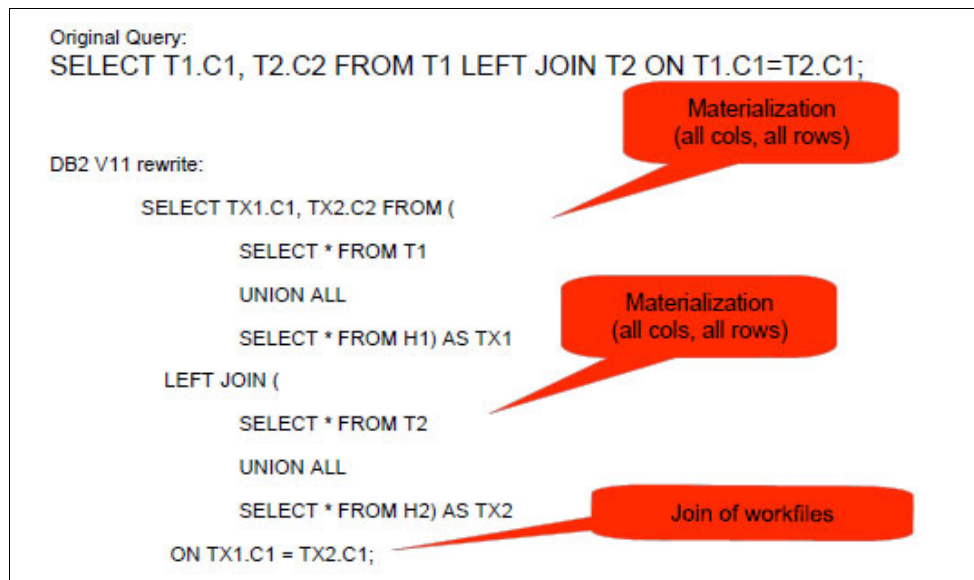


Figure 13-6 DB2 11 left outer join query with transparent archive tables (or any UNION ALL views)

The performance challenge for this query example is that DB2 will execute each leg of the UNION ALL separately and combine the results from each side of the first UNION ALL, and then combine each side of the second UNION ALL, before joining the two results together as requested in the outer query block. In the V11 rewrite of the query, there are no join predicates or any filtering within the UNION ALL legs. The term *combining* means that DB2 will return all columns and all rows from T1 in the first UNION ALL and all columns and all rows from H1 and materialize those rows to a work file. The work file will then be sorted in preparation for the join. This is repeated for the second UNION ALL—all columns and all rows from T2 and H2 are materialized into a work file and sorted in preparation for the join on column C1 from both work files.

The performance of this query will depend heavily on the size of the tables involved, with very large tables consuming significant CPU and work file resources to complete.

In using the same example (from Figure 13-6), the next figure (Figure 13-7 on page 239) explain how several of the UNION ALL enhancements in DB2 12 can improve the performance of this query. While the internal rewrite of the tables to the UNION ALL representation remains the same, DB2 12 adds the ability for the optimizer to make a cost-based decision as to whether to push the join predicates into the UNION ALL legs.

Figure 13-7 demonstrates the example where the optimizer chose to push the join predicates inside each UNION ALL leg. The join predicates occur on only the right side of the join because a join is FROM the left TO the right. The TABLE keyword is required externally for this example to be syntactically valid because pushing down the predicates results in the query appearing as a correlated table expression.

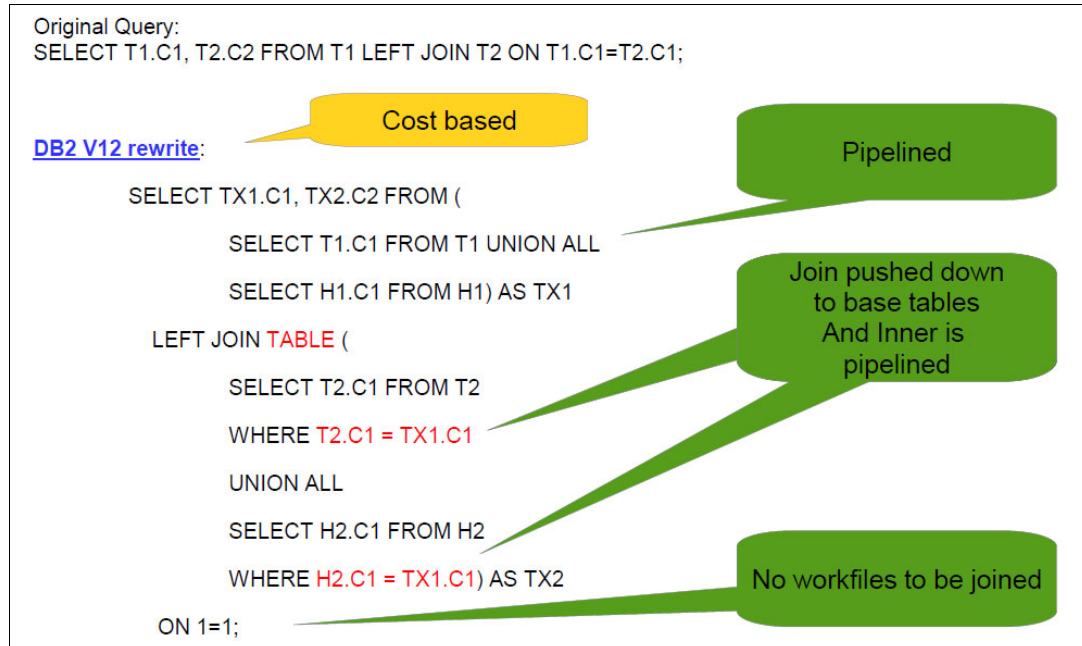


Figure 13-7 DB2 12 left outer join query with transparent archive tables (or any UNION ALL views)

Having the join predicates in each UNION ALL leg allows the join to “look up” each leg of the UNION ALL for every row coming from the outer, rather than sort the full results for the join. An additional optimization, to reduce work file usage and materialization, DB2 12 can “pipeline” the rows from the first UNION ALL (on the left side of the join). Transferring rows from one query block to another can require the results to be written (materialized) to an intermediate work file as in the DB2 11 example. However, DB2 12 can “pipeline” the result from the first UNION ALL to the join without requiring this materialization.

The result for this query in DB2 12, given the (cost based) join predicate pushdown, is that the work file/materializations are avoided, and available indexes can be exploited for each leg, as shown in Figure 13-7.

Sort outer into table expression order

Figure 13-8 on page 240 demonstrates additional DB2 12 enhancements that can improve performance closer to queries that do not use the UNION ALL infrastructure. One option that the optimizer has for improving join performance between two tables sequentially is for the optimizer to introduce a sort of the outer (composite—denoted as SORTC_JOIN='Y' in the EXPLAIN). If data from the outer table is accessed in a different sequence than the index used for the join to the inner, DB2 can choose to sort the outer into the sequence of the inner—allowing the access to the inner to occur in order. This approach is extended in DB2 12 to joins to UNION ALL views/table expressions and is a cost-based choice for the optimizer.

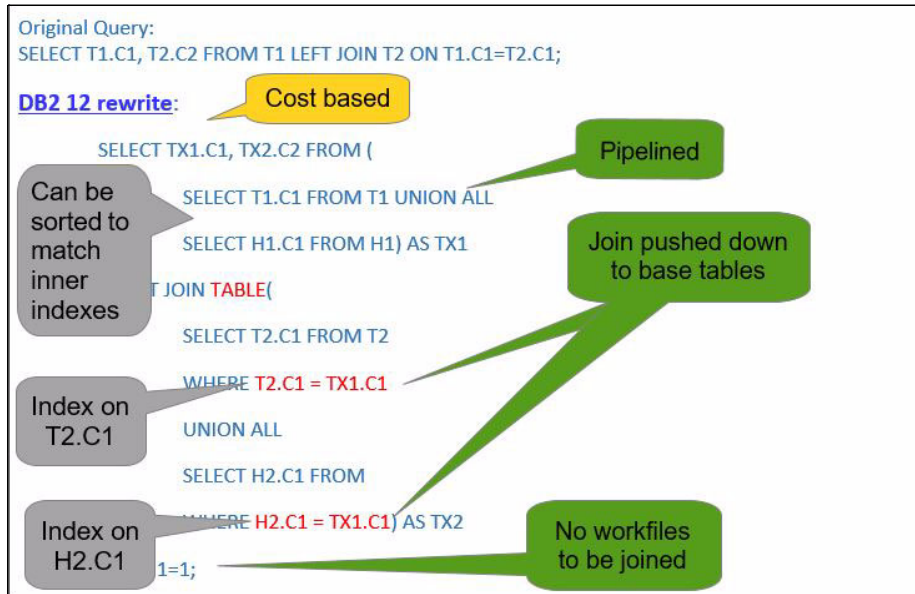


Figure 13-8 DB2 12 left outer join UNION ALL query with sort of outer

Enable sparse index for inner table/view expression

Up to DB2 11, a supporting index was required for optimal performance and executed like a correlated subquery for table/view expression with or without UNION ALL coded with correlation predicates.

DB2 12 supports sparse index creation on the correlated table/view expression, so it avoids the worst case of executing a correlated subquery without a supporting index.

Figure 13-9 on page 241 demonstrates another variation of the same query where there are no supporting indexes for the join to T2.C1 or H2.C1. DB2 12 allows a sparse index to be built on an individual leg of a UNION ALL. This applies to one or both legs, and is also cost-based and thus can be combined with the other UNION ALL-focused enhancements.

Note: If sparse index is chosen, then no need exists to sort the outer into join order, as depicted in Figure 13-8, because a sparse index can use hashing if the result can be contained within memory and thus no concerns exist regarding random I/O.

Remember H1 is the archive table for T1, and H2 is the archive table for T2, as used in previous examples.

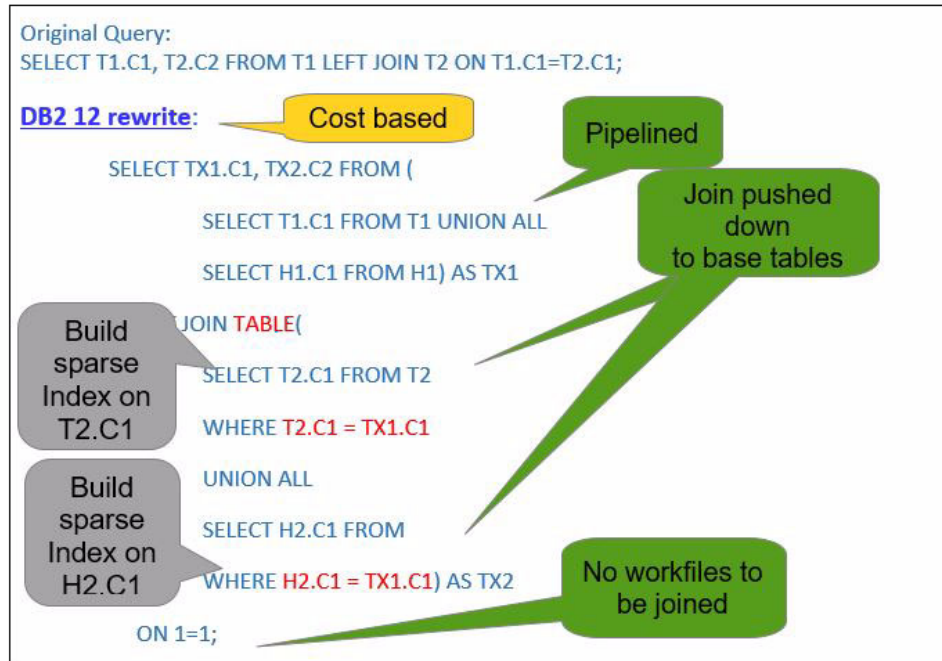


Figure 13-9 DB2 12 left outer join UNION ALL query without supporting join indexes

Pruning unused columns from materialized result

The select list pruning is extended, pruning unused columns from a materialized result when table expressions, views, and table functions are involved.

This extension offers the following benefits:

- ▶ Reduction of the size of the intermediate result with less work file consumption, smaller sort row, and so on.
- ▶ Enablement of more outer join, which means outer join tables that do not return columns for the result may be pruned.

Example 13-2 involves a LEFT OUTER JOIN to a UNION ALL table expression. In the select list for each leg of the UNION ALL is SELECT *, which means returning all columns from that table. However, the referencing SELECT only requires P2.P_PARTKEY (for the SELECT list and ON clause). Given the materialization of the UNION ALL table expression in DB2 11, all columns from PART table will be accessed and materialized to the work file. DB2 11 processes all columns in materialized view/table expression.

Example 13-2 DB2 11 processing all columns in materialized view/table expression

```

  SELECT P.P_PARTKEY,P2.P_PARTKEY
  FROM TPCH30.PART AS P LEFT JOIN
  (SELECT *
  FROM TPCH30.PART
  UNION ALL
  SELECT *
  FROM TPCH30.PART) P2
  ON P.P_PARTKEY = P2.P_PARTKEY;
  
```

DB2 12 will prune the unnecessary columns and only require P_PARTKEY to be returned from each UNION ALL leg, whereas DB2 11 returned all columns.

If the UNION ALL table expression is materialized, then only P_PARTKEY is retrieved and written or materialized to a work file compared with all columns retrieved in the DB2 11 example. And if P_PARTKEY is indexed, the optimizer might choose a non-matching index scan in DB2 12 rather than table space scan in DB2 11. Similarly, if the join predicates were pushed down in DB2 12 and matching index access was chosen on P_PARTKEY, then index-only would now be possible because only P_PARTKEY is required.

Example 13-3 refers to unreferenced columns with optional join pushdown being pruned by DB2 12.

Example 13-3 DB2 12 pruning unreferenced columns with optional join pushdown

```

SELECT P.P_PARTKEY,P2.P_PARTKEY
FROM TPCH30.PART AS P LEFT JOIN
(SELECT P_PARTKEY
FROM TPCH30.PART
UNION ALL
SELECT P_PARTKEY
FROM TPCH30.PART) P2
ON P.P_PARTKEY = P2.P_PARTKEY; <- Cost based pushdown available
to UA legs

```

It achieved the following performance result:

- ▶ CPU and elapsed reduction when no access path change.
- ▶ CPU reduction for access path change and when tables are pruned. If columns that are not needed are pruned by the query, the result might be that the query does not need any columns from a table. And if that table is on the right side of a LEFT OUTER JOIN, and the query will not return duplicates from that table, then the table can be pruned.

Extended LEFT JOIN table pruning

DB2 10 introduced LEFT OUTER JOIN table pruning when right table guaranteed not to return duplicates (due to unique index or DISTINCT/GROUP BY) and no columns were required for the final result, as shown in Figure 13-10.

<pre> SELECT T1.* FROM T1 LEFT JOIN T2 ON T1.C1 = T2.UNIQUE_COL; </pre>	<pre> SELECT DISTINCT T1.* FROM T1 LEFT JOIN T2 ON T1.C1 = T2.C1; </pre>
---	--

Figure 13-10 DB2 10 left outer join table pruning

Figure 13-11 on page 243 represents DB2 11 LEFT OUTER JOIN query with materialization and an unnecessary join. Similar to prior UNION ALL examples, the UNION ALL will retrieve all columns and all rows from T1 and T2 and materialize these to a work file, which will be sorted for the join. T3 is then joined to this materialized result and a sort to remove duplicate C1 values (given the DISTINCT). Because no columns were required from the UNION ALL of T1 and T2, and the DISTINCT would remove any duplicates that were introduced, this join is unnecessary.

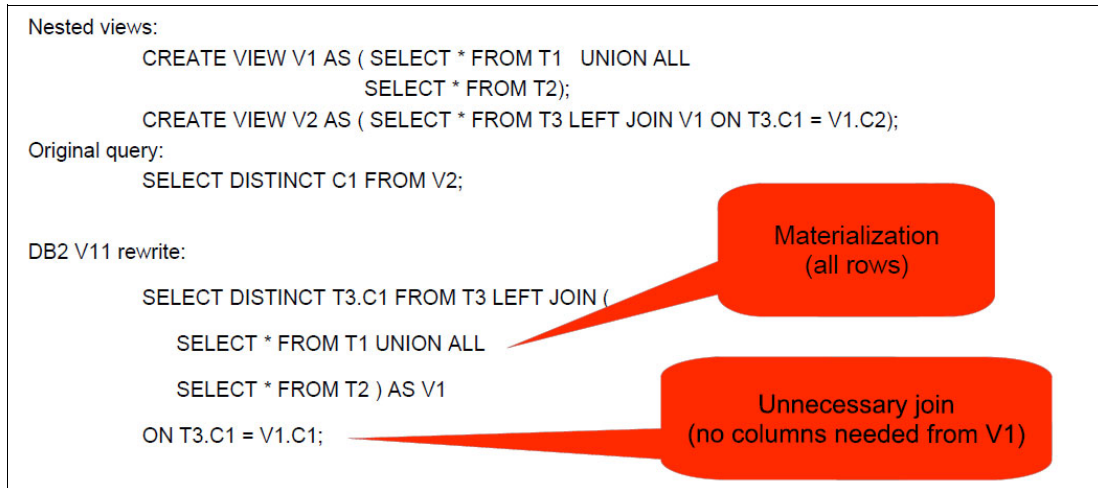


Figure 13-11 DB2 11 LEFT OUTER JOIN to unnecessary table expression

DB2 12 extends table pruning to views and table expressions where no columns are required and no duplicates are returned and also provides a simple rewrite as shown in Figure 13-12. The view definitions and the query against the views are the same between the two figures. What differs is how DB2 12 is able to prune out the table expression that contains the UNION ALL. The result is simply a SELECT DISTINCT requiring access only to T3.

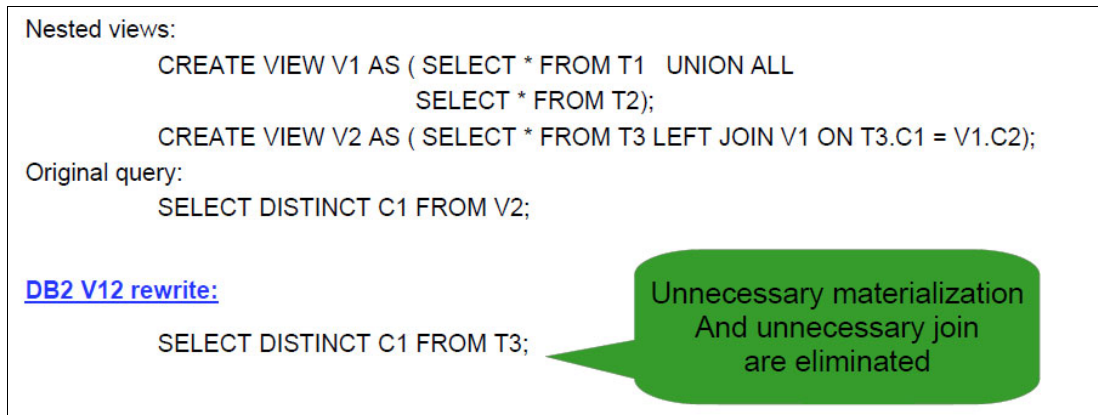


Figure 13-12 DB2 12 LEFT OUTER JOIN to unnecessary table expression

13.5.2 Sort improvements

The following topics cover sort, work file, and sparse index improvements:

- ▶ Sort minimization for partial order with FETCH FIRST
- ▶ Sort avoidance for OLAP functions with PARTITION BY
- ▶ Reducing sort row length
- ▶ Improve GROUP BY/DISTINCT sort performance and In-memory sort exploitation
- ▶ Sort workfile impacts
- ▶ Sparse index improvements

Sort minimization for partial order with FETCH FIRST

Up to DB2 11, a sort could be avoided and only “n” rows processed but only if an index was chosen that completely avoided the ORDER BY sort. In many situations if a sort is still required, seeing all the data is not necessary.

DB2 12 reduces the number of rows fetched or processed when there is no index that avoids the sort, if ordering by more than one column and only the leading column has an index.

Figure 13-13 shows how the sort minimization for a partial order with FETCH FIRST works:

```

SELECT * FROM T1
ORDER BY C1, C2
FETCH FIRST 10 ROWS ONLY

INDEX1 (C1)
Index entries: 1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4, 5.....999999

```




Figure 13-13 Sort minimization for partial order with FETCH FIRST

In the previous DB2 versions, the scenario was fetching all rows and sort into C1, C2 sequence, so in this way, millions of rows in the example above.

In DB2 12, when the tenth row is reached, then fetch until C1 changes. Using this method, 13 rows are fetched and 12 rows are sorted.

Sort avoidance for OLAP functions with PARTITION BY

Sort avoidance is also extended to online analytical processing (OLAP) functions that combine PARTITION BY and ORDER BY. Although DB2 11 already supports sort avoidance, if an index matches the ORDER BY clause with an OLAP function (such as RANK), that did not apply for sort avoidance when PARTITION BY was involved. Example 13-4 highlights an appropriate index that can be used in DB2 12 to avoid the sort for this SQL statement.

Example 13-4 Sort avoidance for OLAP functions with PARTITION BY clause

```

CREATE INDEX SK_SD_1 ON LINEITEM(L_SUPPKEY, L_SHIPDATE);
SELECT L_SUPPKEY, L_SHIPDATE,
       RANK() OVER(PARTITION BY L_SUPPKEY
                  ORDER BY L_SHIPDATE) AS RANK1
FROM LINEITEM;

```

Reducing sort row length

A common behavior is that predicates coded in the WHERE clause are redundantly included in the SELECT list and any redundancy in the sort key or data row has a negative impact on sort performance and resource consumption. ORDER BY sort will remove columns from the sort key if covered by the equals predicates in the WHERE clause.

DISTINCT or GROUP BY already removes redundant columns for sort avoidance. But if a sort is required for DISTINCT or GROUP BY, such redundant columns remain until DB2 12, when they are removed from the sort key, as shown in Example 13-5. Also, because C1 has an equals predicate in the WHERE clause, all sorted rows are guaranteed to contain that same value, and thus only C2 is needed for the sort.

Example 13-5 Redundant columns in the sort key

```

SELECT DISTINCT C1, C2
FROM TABLE
WHERE C1 = ?

```

Up to DB2 11, for a SELECT (like SELECT C1, C2....ORDER BY C1, C2), columns C1 and C2 are duplicated as the sort key. With DB2 12, the sort does not duplicate if the sort key is equal to the leading SELECT list columns.

This situation is applied for only fixed length columns (not VARCHAR, VARGRAPHIC, and others).

Improve GROUP BY/DISTINCT sort performance and In-memory sort exploitation

When sort cannot be avoided, exploiting memory to process the sort and reducing the length of the sort can result in that sort being contained in-memory or at a minimum to reduce the number of work file resources needed to complete the sort.

Prior to DB2 12, the maximum number of nodes of the sort tree was 32,000, and less for longer sort keys that were limited by ZPARM SRTPOOL.

DB2 12 enables sort tree and hash entries growth to 512,000 nodes (non-parallelism sort) or 128,000 (parallel child task sort). In addition, default 10MB SRTPOOL can support 100,000 nodes for 100 byte rows.

Also, DB2 9 added hashing as input to GROUP BY/DISTINCT sort; in DB2 the number of hash entries is tied to the number of nodes of the sort tree. Therefore, increasing the number of nodes can result in higher cardinality GROUP BY/DISTINCT results, consolidating the groups as rows are input to sort. This can increase the chance that the sort can be contained in memory or at least reduce the amount of work file space required to consolidate duplicates during the final sort merge pass.

DB2 12 can use the memory up to the ZPARM SRTPOOL value that might not have been exploited in the previous DB2 versions.

As result, DB2 12 performance improvements for GROUP BY/DISTINCT provides a CPU savings when sort can be contained in memory.

Sort workfile impacts

The large sorts may use the 32K page size regardless of row length on DB2 12. That is a big difference from DB2 9, in which this page size was limited to less than 100-byte rows.

This increase can result in more 32K page size data sets for sort. However, it was developed for other performance advantages that DB2 12 provides such as increased in-memory sorts, increased tree and hash size, reduced materialization, reduced sort keys size, and sort avoidance.

Sparse index improvements

DB2 12 provides sparse indexes support for the VARGRAPHIC data type. With this support, the memory allocation was improved when multiple sparse indexes in query. Also, the sort component improves its algorithms to adjust the type of sparse index that is built to optimize memory and also to reduce getpages when a sparse index must overflow to the work file.

When building a sparse index, sort also attempts to trim the information that must be stored.

Sparse index is able to avoid duplicate key information when key equals the data and fixed length keys, in addition, trimming trailing blanks for VARCHAR/VARGRAPHIC and prefix if all keys have the same prefix.

13.5.3 Predicate optimization

The following DB2 12 predicate optimizations are covered in this topic:

- ▶ Sort for stage 2 join expressions
- ▶ User-defined table function predicate optimizations
- ▶ VARBINARY data type indexability
- ▶ Row permission to correlated subquery indexability
- ▶ Additional IN-list performance enhancements

Sort for stage 2 join expressions

Expressions as join predicates are often stage 2, unless the exception for expression on the outer (without sort) for nested loop joins.

DB2 12 allows resolution of expression before sort to support join support for expression on inner and sparse index without sort on outer. See Example 13-6.

Example 13-6 Stage 2 join predicate

```
SELECT ...  
FROM T1, T2  
WHERE T1.col1 = T2.col1 and  
      T1.col2 = SUBSTR(T2.col2,1,10)
```

DB2 12 can improve the performance of stage 2 join predicates by allowing sort to evaluate the function, and allowing sparse index to be built on the result for the join—which becomes an optimal candidate on the inner table when the result can be contained in-memory, or when there is no other viable index to support the filtering of the join. Alternatively, if the table with the join expression is the outer table of the join, a sort for join order can allow access to the inner table to be performed sequentially.

Stage 2 join predicates are often observed if tables are not designed with consistent data types for joined columns, which might occur if applications are integrated at a later date, or if business information is embedded within columns, or if timestamp columns are used within each table and the join is by the consistent date portion of those columns (for example, insert timestamps do not match between two tables). DB2 12 can therefore improve performance significantly for these situations without requiring a targeted index on expression to be built.

User-defined table function predicate optimizations

User-defined table functions (also known as table UDFs, table functions, or TUDFs) were initially targeted to allow an application program to be called from within an SQL statement. This provided the flexibility to access objects that are not DB2 and represent them as a table within SQL to be joined with DB2 tables. Inline table UDFs were a further extension to DB2 support, allowing the definitions to contain native SQL. An increase in table functions has occurred as an alternative to views because of the capability to create a table function with input parameters, whereas parameterized views are not supported in DB2 for z/OS.

Although prior releases provided similar merge (and thus materialization avoidance) capabilities for table functions that were syntactically equivalent to views, DB2 12 improves both the merge of deterministic table functions with input parameters and also improves indexability of input parameters as predicates within the table function, as demonstrated in Figure 13-14 on page 247.

```

create function tpchudf(in_date1 CHAR(10), in_date2 CHAR(10))
...
SELECT O_ORDERPRIORITY, COUNT(*)
FROM ORDERS
WHERE O_ORDERDATE >= DATE(in_date1) <<< Was STAGE 2. Now indexable
      AND O_ORDERDATE < DATE(in_date2) <<< Was STAGE 2. Now indexable
SELECT * FROM table(tpchudf('1993-07-01', '1993-10-01')) as TAB1;

```

Figure 13-14 Table function with input variables

VARBINARY data type indexability

Up to DB2 11, support was limited for BINARY and VARBINARY predicate indexability when the lengths of the operands of the predicates did not match. DB2 12 implicitly adds CAST expressions on the VARBINARY and BINARY predicates when the length of the operands does not match. Figure 13-15 compares the pre DB2 12 predicate as stage 2 of that with a CAST added in DB2 12 to support indexability of mismatched length VARBINARY.

```

SELECT A.C_CUSTKEY, HEX(A.C_NAME)
FROM CUSTOMER A, CUSTNULL B
WHERE A.C_NAME > B.C_NAME;

CUSTOMER.C_NAME: VARBINARY(25) NOT NULL
CUSTNULL.C_NAME: VARBINARY(30)

Pre DB2 12: A.C_NAME > B.C_NAME (stage 2)
DB2 12     : CAST(A.C_NAME AS VARYING BINARY(30))>B.C_NAME (Indexable)

```

Figure 13-15 Pre-DB2 12 predicate as stage 2 of that with a CAST added in DB2 12 comparison

Although many clients might identify that VARBINARY or BINARY data types are not exploited within their environments, improving indexability is important because DB2 scalar functions can return a result as BINARY or VARBINARY. These improvements to the underlying support of BINARY and VARBINARY indexability were necessary to allow indexing on expressions to be built on those scalar functions and to be exploited for matching index access.

Example 13-7 shows a scalar function as an index on expression that is indexable in DB2 12. This example demonstrates the COLLATION_KEY scalar function with a parameter tailored to German.

Example 13-7 Index on expression for VARBINARY-based expression

```

CREATE INDEX EMPLOYEE_NAME_SORT_KEY ON EMPLOYEE
(COLLATION_KEY(LASTNAME, 'UCA410_LDE', 600));
SELECT *
FROM EMPLOYEE
WHERE COLLATION_KEY(LASTNAME, 'UCA410_LDE', 600) = < ?

```

Row permission to correlated subquery indexability

In versions prior to DB2 12, the correlation predicates in child-correlated subquery were stage 2 on row permissions for insert and update and DB2 12 provides support indexability for correlated subqueries on row permissions for insert and update, benefitting efficiency of security validation. Figure 13-16 shows a correlated subquery example of a row permission.

```
CREATE PERMISSION RP1 ON LINEITEM T1 FOR ROWS
WHERE EXISTS
(SELECT 1 FROM ORDER T2
WHERE T1.ORDERKEY=T2.ORDERKEY) ENFORCED FOR ALL ACCESS
ENABLE
- Insert row into table LINEITEM
- UPDATE LINEITEM SET COMMENT='?' WHERE ORDERKEY = ?;
```

Figure 13-16 Correlated subquery predicates in a row permission

Additional IN-list performance enhancements

DB2 10 added IN-list table, where the optimizer could choose list prefetch for matching IN-list access. The execution was one list prefetch request per IN-list element, so performed poorly if low number of duplicates per element existed.

DB2 12 introduces an improvement to matching IN-list performance for poor clustering index, allowing the accumulation of 32 RIDs per list prefetch request. Also, DB2 12 removes a limitation that IN-lists cannot be used with index screening predicate for range-list access.

13.5.4 Execution time adaptive index

Execution time adaptive index is provided in DB2 12 as a solution for generic search queries that used to be a challenge for query optimizers. They were considered a challenge for these reasons:

- ▶ Filtering that could change each execution, so that choosing the one best access path was impossible.
- ▶ Fields *not* searched by the user will populate with the whole range:
LIKE '%' or BETWEEN 00000 AND 99999.
- ▶ Fields searched will use their actual values:
LIKE 'SMITH' or BETWEEN 95141 AND 95141.

Runtime adaptive index solution has the following benefits:

- ▶ Allows list prefetching-based plans (single or multi-index) to quickly determine index filtering.
- ▶ Adjusts at execution time based on determined filtering:
 - Does not require REOPT(ALWAYS).

- For list prefetching or multi-index OR:
 - Earlier opportunity exists to fall back to tablespace scan if large percentage of table is to be read.
- For multi-index AND:
 - Reorder index legs from most to least filtering.
 - Early-out for non-filtering legs, and fallback to table space scan if no filtering.
- Quick evaluation is done based on literals that are used (for example LIKE ‘%’).
- Further costlier evaluation of filtering is deferred until after one RID block is retrieved from all participating indexes:
 - Provides better optimization opportunity while minimizing overhead for short running queries.
- ▶ IFCID 125 is enhanced to track this feature.
- ▶ Solution is not limited to the search screen challenge:
 - Any query where high uncertainty in the optimizer’s estimates exists:
 - Range predicates.
 - JSON, Spatial, and Index on expression.

The performance measurements evaluated to runtime adaptive index are as follows:

- ▶ List prefetching CPU reduction (when failover to table space scan is needed).
- ▶ Multi-index OR CPU reduction (when failover to table space scan is needed).
- ▶ Multi-index AND CPU reduction for re-ordering to put most filtering leg first.



Appendixes

This part contains the following appendix sections:

- ▶ Appendix A, “Information about IFCID changes” on page 253
- ▶ Appendix B, “Additional material” on page 279



Information about IFCID changes

This appendix provides information about the new or changed instrumentation facility component identifiers (IFCIDs) discussed in previous chapters of this book. See the following resources:

- ▶ For more information about IFCIDs, see *DB2 12 for z/OS What's New?*, GC27-8861.
- ▶ For collecting accounting and statistics, see *Subsystem and Transaction Monitoring and Tuning with DB2 11 for z/OS*, SG24-8182.

You can find up-to-date mappings of IFCIDs in the SDSNMACS data set that is delivered with DB2.

This appendix includes the following topics:

- ▶ IFCID header changes
- ▶ New IFCIDs
- ▶ Application compatibility IFCID changes
- ▶ Dynamic SQL plan stability IFCID changes
- ▶ Fast INSERT IFCID changes
- ▶ Lift partition limits IFCID changes limits
- ▶ Large object (LOB) compression IFCID changes
- ▶ Transfer ownership IFCID changes
- ▶ UNLOAD privilege for UNLOAD utility IFCID changes
- ▶ Additional changed IFCIDs

IFCID header changes

The standard header (QWHS) used by all trace records is modified to include three new fields:

- ▶ QWHS_MOD_LVL: Function level of DB2 in the form VvvRrrMmmm, where vv is the version, rr is the release and mmm is the modification level.
- ▶ QWHS_REC_INCOMPAT: Incompatible change counter.
- ▶ QWHS_REC_COMPAT: Compatible change counter.

Example A-1 shows the updated QWHS header.

Example A-1 Updated QWHS

QWHS_MOD_LVL DS	CL10	MODIFICATION LEVEL FOR
*		CONTINUOUS DELIVERY
QWHS_REC_INCOMPAT DS	XL2	Incompatible change value
*		incremented each time an
*		incompatible change occurs,
*		such as changing the size of
*		existing fields in a record
*		or removing fields no longer
*		being set which causes the
*		offset to other fields to
*		change is made
QWHS_REC_COMPAT DS	XL2	Compatible change value.
*		Incremented each time a
*		compatible change occurs,
*		such as adding a new field in a
*		reserved area no longer setting
*		an existing field or increasing
*		the size of a record to add a
*		new field is made

New IFCIDs

DB2 12 introduces the following new IFCIDs:

- ▶ IFCID 382: Begin parallel task synchronization suspend
- ▶ IFCID 383: End parallel task synchronization suspend
- ▶ IFCID 389: Fast index traversal
- ▶ IFCID 404: Serviceability trace record for new AUTH_COMPATIBILITY subsystem parameter
- ▶ IFCID 413: Begin of pipe wait for fast insert
- ▶ IFCID 414: End of pipe wait for fast insert
- ▶ IFCID 477: Fast index traversal

IFCID 382: Begin parallel task synchronization suspend

The IFCID 382 trace records the beginning of a suspend for parallel task synchronization.

Example A-2 shows the IFCID 382 indicating whether or not the task suspended was for a parent or child.

Example A-2 New IFCID 382 indicating the type of task suspended

```
*****
* BEGIN Suspend for parallel task synchronization *
*****QW0382
DSECT          IFCID(QWHS0382)
QW0382ST      DS  CL1          Type of task suspended.
QW0382PT      EQU C'P'        Task suspended is a parent
QW0382CT      EQU C'C'        Task suspended is a child
```

IFCID 383: End parallel task synchronization suspend

The IFCID 383 trace records the ending of a suspend for parallel task synchronization.

Example A-3 shows the IFCID 383 indicating whether or not the task suspended was for a parent or child.

Example A-3 New IFCID 383 indicating the type of the task suspended

```
*****
* END Suspend for parallel task synchronization *
*****
QW0383      DSECT          IFCID(QWHS0383)
QW0383RT    DS  CL1          Type of task resumed.
QW0383PT    EQU C'P'        Task resumed is a parent
QW0383CT    EQU C'C'        Task resumed is a child
```

IFCID 389: Fast index traversal

The IFCID 389 trace records information about indexes that have structures allocated for fast index traversal.

Example A-4 shows the IFCID 389 indicating which indexes have fast traversal blocks (FTBs) associated with them.

Example A-4 New IFCID 389 for indexes with fast traversal blocks (FTBs)

```
*****
* IFCID 0389 to record all indexes with FTBs. Each trace * * record can
  contain information about up to 500 indexes. *
*****
*
QW0389      DSECT          IFCID(QWHS0389)
QW0389H     DS   OCL8      HEADER RECORD
QW0389NU    DS   H         NUMBER OF INDEXES WITH FTBs
              DS   CL6      RESERVED
QW0389AR    DS   OCL16     (S)
*
```

QW0389DB DS	CL2	DATA BASE ID
QW03890B DS	CL2	INDEX PAGE SET ID
QW0389PT DS	CL2	PARTITION NUMBER
QW0389LV DS	CL2	NUMBER OF INDEX LEVELS IN FTB
QW0389SZ DS	XL4	SIZE OF FTB IN BYTES
	DS	CL4
		RESERVED

*

IFCID 404: Serviceability trace record for new subsystem parameter AUTH_COMPATIBILITY

A new UNLOAD authorization has been added to the UNLOAD utility. With the enhancement is a new AUTH_COMPATIBILITY subsystem parameter. The IFCID 404 provides serviceability information to be used by the DB2 development team.

Example A-5 shows the IFCID 404 providing serviceability information for the AUTH_COMPATIBILITY subsystem parameter.

Example A-5 New IFCID 404

```

*****
* IFCID 0404 to service authorization compatibility *
* settings *
*****
QW0404          DSECT
QW0404T0        DS CL1          (S)
                DS CL3          (S)
QW0404NM        DS CL16         (S)
QW0404PR        DS H           Privilege checked
QW04040T        DS CL1          Object type
QW0404AT        DS CL1          Authid type
*               ' ' - Authorization ID
*               'L' - Role
QW0404F1_Off    DS H           Offset from QW0404 to authid or a role
QW0404F2_Off    DS H           Offset from QW0404 to schema name
QW0404F3_Off    DS H           Offset from QW0404 to object name
                DS CL2          (S)
*
QW0404F1_D      DSECT
QW0404F1_Len    DS H           Length of the following field
QW0404F1_Var    DS OCL128      %U Authid or role
*
QW0404F2_D      DSECT
QW0404F2_Len    DS H           Length of the following field
QW0404F2_Var    DS OCL128      %U Schema name
*
QW0404F3_D      DSECT
QW0404F3_Len    DS H           Length of the following field
QW0404F3_Var    DS OCL128      %U Object name
*
QW0404CM        DC CL1'C'
QW0404SQ        DC CL1'S'
QW0404UL        DS CL1'U'
* (S) = FOR SERVICEABILITY

```


IFCID 413: Begin of pipe wait for a fast insert

IFCID 413 records the beginning of a wait for a pipe for a fast INSERT.

Example A-6 shows the IFCID 413 indicating the resource of the pipe wait for a fast insert.

Example A-6 New IFCID 413

```
*****
*   IFCID 0413 THE BEGINNING OF A WAIT FOR A PIPE SUSPEND   *
*****
QW0413   DSECT           IFCID(QWHS0413)
QW0413PN DS    CL8      PROC NAME
QW0413RN DS    OCL6     RESOURCE NAME
QW0413DB DS    XL2      DATA BASE ID
QW0413PS DS    XL2      PAGE SET ID
QW0413PT DS    XL2      PARTITION NUMBER
          DS    CL2      RESERVED
QW0413DMS DS    XL4      (S)
QW0413CNT DS    XL2      (S)
QW0413LMT DS    XL2      (S)
QW0413FL DS    XL2      (S)
          DS    XL2      (S)
*   (S) = FOR SERVICEABILITY
```

IFCID 414: End of pipe wait for a fast insert

IFCID 414 records the end of a wait for a pipe for a fast INSERT.

Example A-7 shows the IFCID 414 indicating the reason for the pipe wait resume.

Example A-7 New IFCID 414

```
*****
*   IFCID 0414 RECORDS THE END OF THE WAIT FOR PIPE SUSPEND *
*****
QW0414   DSECT           IFCID(QWHS0414)
QW0414R  DS    CL1      REASON FOR PIPE WAIT RESUME
QW0414FL DS    XL2      (S)
          DS    XL2      (S)
          DS    CL1      (S)
*   (S) = FOR SERVICEABILITY
```

IFCID 477: Fast index traversal

IFCID 477 records the allocation and deallocation of structures for fast index traversal.

Example A-8 shows the IFCID 477 indicating the whether or not the task suspended was for a parent or child.

Example A-8 New IFCID 477 to indicate fast traversal blocks (FTBs) allocation or deallocation

```
*****
*   IFCID 0477 trace record for each allocated or deallocated *
*   Index Fast Traverse Block (FTB). If action is "create FTB" *
*****
```

```

*   then QW0477CO bit is ON. If action is "FREE FTB" then      *
*   QW0477CO bit is OFF.                                       *
*****
QW0477  DSECT                IFCID(QWHS0477)
QW0477DB DS   CL2            DATA BASE ID
QW0477OB DS   CL2            INDEX PAGE SET ID
QW0477PT DS   CL2            PARTITION NUMBER
QW0477LV DS   CL2            NUMBER OF INDEX LEVELS IN FTB
QW0477SZ DS   XL4            SIZE OF FTB IN BYTES
QW0477FL DS   X              IFCID477 FLAGS
QW0477CO EQU  X'80'          THIS BIT IS ON IF ACTION -CREATE FTB
                               DS   CL3            AVAILABLE
*

```

Application compatibility IFCID changes

At times, DB2 might need to deliver changes that can have an effect on how your applications execute. If DB2 made a change in its behavior, you can run a trace record to identify which applications might be affected. Prior to DB2 12, IFCID 366 and IFCID 376 recorded such changes in your applications. Starting in DB2 12, IFCID 376 now records this information.

IFCID 366: Deprecated application compatibility trace

IFCID 366 is deprecated in DB2 12. DB2 no longer accepts trace commands for IFCID 366 and does not write out this trace record. Use Existing IFCID 376 to identify incompatibilities.

IFCID 376: Application compatibility trace

Constants resided in IFCID 366 but are now in IFCID 376. The constants indicate which incompatibility changes your application is exposed to. There are no new incompatibilities introduced in DB2 12.

Example A-9 shows the updated IFCID 376 containing constants indicating incompatibility types.

Example A-9 Updated IFCID 376

```

QW0376      DSECT
QW0376FN    DS F              Incompatible change indicator
*.....QW0376FN CONSTANTS.....
C_QW0376_CHAR    EQU 0001    V9 SYSIBM.CHAR(decimal-expr)
*                               function
C_QW0376_VCHAR   EQU 0002    V9 SYSIBM.VARCHAR(decimal-expr)
*                               function
*                               CAST (decimal as VARCHAR or CHAR)
C_QW0376_TMS     EQU 0003    Unsupported character string
*                               representation of a timestamp
C_QW0376_IMPCAST EQU 0007    Use the pre-v10 server compatibility
*                               behavior which is not to implicitly
*                               cast input host variables during
*                               server host bind-in processing
C_QW0376_SPPARMS EQU 0008    data types of the returned output

```

```

*          data match the data types of the
*          corresponding CALL statement
*          arguments
C_QW0376_IGNORETZ EQU 0009 Use the pre-V10 server compatibility
*          behavior which is to ignore time
*          zone information when bind in TMSTZ
*          hostvar to TMS target
C_QW0376_TRIM      EQU 0010 V9 RTRIM, LTRIM and STRIP functions
C_QW0376_XMLINS   EQU 1101 Insert into an XML column without
*          XMLDOCUMENT function
C_QW0376_XPATHERR EQU 1102 XPath evaluation error
C_QW0376_RLF      EQU 1103 RLF governing
C_QW0376_CLIENAC  EQU 1104 Long CLIENT_ACCTNG Special Reg value
C_QW0376_CLIENAP  EQU 1105 Long CLIENT_APPLNAME Special Reg
*          value
C_QW0376_CLIENUS  EQU 1106 Long CLIENT_USERID Special Reg value
C_QW0376_CLIENWK  EQU 1107 Long CLIENT_WRKSTNNAME Special Reg
*          value
C_QW0376_CLIENSR  EQU 1108 Long client Special Reg value for
*          RLF
C_QW0376_TMSCAST  EQU 1109 CAST(string AS TIMESTAMP)
C_QW0376_SPACEINT EQU 1110 SPACE integer argument greater than
*          32764
C_QW0376_VCHARINT EQU 1111 VARCHAR int argument greater than
*          32764
C_QW0376_XMLEMPT  EQU 1112 XML_RESTRICT_EMPTY_TAG ZPARM is used
*          and empty XML element is serialized
*          to <X></X>
*          .....

```

Dynamic SQL plan stability IFCID changes

Several IFCIDs were changed in support of the dynamic SQL plan stability enhancement:

- ▶ IFCID 002: RDS statistics block
- ▶ IFCID 002: EDM pool statistics block
- ▶ IFCID 021: Lock types
- ▶ IFCID 029: EDM request begin identifier and new block
- ▶ IFCID 030: EDM request end identifier and new block
- ▶ IFCID 106: New subsystem parameter
- ▶ IFCID 316: Stabilization and hash ID information

IFCID 002: RDS statistics block

The QXSTSFND field was added to the RDS statistics block (QXST) to indicate the number of times a PREPARE request was satisfied.

Example A-10 shows a snippet of the IFCID 002 indicating the new field.

Example A-10 Changed IFCID 002

```

QXSTSFND    DS  D    # of times a PREPARE request was satisfied+
*          by making a copy from the stabilized      +
*          statement in SYSIBM.SYSDYNQRY catalog      +

```

```

*          table. The stabilized statement search is +
*          done only when no matching statement was +
*          found in the prepared statement cache.   +

```

IFCID 002: EDM pool statistics block

Several fields are added to the EDM pool statistics block (QISE) to record the number of requests to look for dynamic SQL plan stability and to record the number of certain matches.

Example A-11 shows the changes to the QISE for the dynamic SQL plan stability enhancement.

Example A-11 Changed QISE

```

QISEDPSL DS    D    /* # of requests to look for DPS      */
QISEDPSC DS    D    /* # of times possible row found      */
QISEDPSM DS    D    /* # of times match thrgh text/bind opt*/
QISEDPSF DS    D    /* # of times match found             */

```

IFCID 021: Lock types

Two new lock types are added to IFCID 21 to record concurrent access control on a stabilized query in the SYSDYNQRY table.

Example A-12 shows the new lock types, QW0021HI and QW0021SG, that were added to the IFCID 021 for dynamic SQL plan stability.

Example A-12 Changed IFCID 021

```

QW0021HI EQU   X'42'  *          SYSDYNQRY HASH_ID lock      *
QW0021SG EQU   X'43'  *          SYSDYNQRY STBLGRP lock    *

```

IFCID 029: EDM request begin identifier and new block

A new identifier (ID) value, DY, is added to the IFCID 029 to indicate the object is for a table for dynamic SQL plan stability. Also, a new section is added that is specific for dynamic SQL plan stability.

Example A-13 shows the new ID value plus the new block dedicated to dynamic SQL plan stability information.

Example A-13 Changed IFCID 029

```

QW0029ID DS    CL2    DB=DBDID, CT=CURSOR TABLE, PT=PACKAGE TABLE
*              DY=DPS TABLE
. . .
          ORG    QW0029DB  DY DPS TABLE MAPPING FOLLOWS
QW0029SC DS    CL18    %U SCHEMA SHORT NAME
*              Truncated if QW0029SC_Off=0
QW0029QH DS    CL16    HASHID
QW0029CP DS    H       COPY ID
QW0029QD DS    XL8     SDQE_STMTID
QW0029QC DS    XL4     RESERVED
QW0029SC_Off  DS H (FIXED 15)

```

```

QW0029RB DS    CL1      release bound
QW0029FL DS    XL1      RESERVED
. . .
*
QW0029SC_D   Dsect      Use if QW0029SC_Off=0
QW0029SC_Len DS    H      Length of the following field
QW0029SC_Var DS    0CL128 %U SCHEMA

```

IFCID 030: EDM request end identifier and new block

The same information described above for IFCID 029 will also be added to the IFCID 030. A new identifier (ID) value, DY, is added to the IFCID 029 to indicate the object is for a table for dynamic SQL plan stability. Also, a new section is added that is specific for dynamic SQL plan stability.

Example A-14 shows the new ID value plus the new block dedicated to dynamic SQL plan stability information.

Example A-14 Changed IFCID 030

```

QW0030ID DS    CL2      DB=DBDID, CT=CURSOR TABLE, PT=PACKAGE TABLE
*
. . .
      ORG   QW0030DB   DY DPS TABLE MAPPING FOLLOWS
QW0030SC DS    CL18     %U SCHEMA SHORT NAME
*
      Truncated if QW0030SC_Off=0
QW0030QH DS    CL16     HASHID
QW0030CP DS    H        COPY ID
QW0030QD DS    XL8      SDQE_STMTID
QW0030QC DS    XL4      Number of records read
QW0030SC_Off DS    H (FIXED 15)
QW0030RB DS    CL1      release bound
QW0030FL DS    XL1      FLAG
. . .
QW0030SC_D   Dsect      Use if QW0030C_Off=0
QW0030SC_Len DS    H      Length of the following field
QW0030SC_Var DS    0CL128 %U SCHEMA

```

IFCID 106: New subsystem parameter

The QWP4CDST field is added to trace the internal setting of the new subsystem parameter CACHEDYN_STABILIZATION.

Example A-15 shows the field in IFCID 106.

Example A-15 Changed IFCID 106

```

QWP4CDST DS    CL1      CACHEDYN_STABILIZATION          s17830
*
      B = BOTH          s17830
*
      C = CAPTURE      s17830
*
      L = LOAD         s17830
*
      N = NONE         s17830

```

IFCID 316: Stabilization and hash ID information

Several fields have been added to the IFCID 316 to contain identifiers for the stabilized statement and hash plus the stabilization group name.

Example A-16 shows the new fields in IFCID 316.

Example A-16 Changed IFCID 316

QW0316_SDQ_STMTID	DS	XL8	Stabilized statement ID
QW0316_QUERY_HASH_ID	DS	CL16	Query's hash ID
QW0316_QUERY_HASH_VER	DS	F	Version of query's hash ID
QW0316_STBLGRP_Off	DS	H	Offset from QW0316 to
*			stabilization group
. . .			
QW0316_STBLGRP_D	Dsect		Use if QW0316STBLGRP_Off=0
QW0316_STBLGRP_Len	DS	H	Length of the next field
QW0316_STBLGRP_Var	DS	OCL128	%U Stabilization group name

Fast INSERT IFCID changes

Additional instrumentation was added to track the time spent in waiting for formatting pages in DB2. Tracking will be by package, statement, and plan accounting. The following IFCIDs are changed in support of the fast INSERT enhancement:

- ▶ IFCID 002: Package level pipe wait information
- ▶ IFCID 002: Data Manager statistics block
- ▶ IFCID 003: Accounting control block
- ▶ IFCID 018: End of inserts and scans
- ▶ IFCID 058: Accumulated pipe wait time
- ▶ IFCID 106: New subsystem parameter
- ▶ IFCID 316: Statement level pipe wait information
- ▶ IFCID 401: Accumulated pipe wait time
- ▶ IFCID 413: Begin pipe wait
- ▶ IFCID 414: End pipe wait

IFCID 002: Package level pipe wait information

Two new fields, QPAC_PIPE_WAIT and QPAC_PIPEWAIT_COUNT, are added to the accounting control block (QPAC) to indicate pipe wait information for the package.

Example A-17 shows the new fields to IFCID 002 for package pipe wait details.

Example A-17 Changed IFCID 002

QPAC_PIPE_WAIT	DS	XL8	/* accumulated wait time for a pipe */
*			/* while executing this package */
QPAC_PIPEWAIT_COUNT	DS	F	/* number of wait trace events */
*			/* processed for waits for a pipe */
*			/* while executing this package */

IFCID 002: Data Manager Statistics block

Two new counts, QISTINPA and QISTINPD, have been added to the Data Manager Statistics Block (DSNDQIST) of IFCID 002.

Example A-18 shows the new counters in IFCID 002.

Example A-18 Changed IFCID 002

QISTINPA	DS D	/* Number of DM Fast Insert	@fil*/
*		/* (Insert Algorithm Level 2)	@fil*/
*		/* pipes allocated since DB2	@fil*/
*		/* restart.	@fil*/
QISTINPD	DS D	/* Number of DM Fast Insert	@fil*/
*		/* (Insert Algorithm Level 2)	@fil*/
*		/* pipes disabled since DB2	@fil*/
*		/* restart.	@fil*/

IFCID 003: Accounting control block

Two new fields, QWAX_PIPE_WAIT and QWAX_PIPEWAIT_COUNT, are added to the accounting control block (QWAX) to indicate pipe wait information.

Example A-19 shows the new fields in IFCID 003 for pipe wait details.

Example A-19 Changed IFCID 003

QWAX_PIPE_WAIT	DS CL8	/* Accumulated wait time for pipe	
*		wait	*/
*			
QWAX_PIPEWAIT_COUNT	DS F	/* Number of wait trace events	
*		processed for pipe wait	*/

IFCID 018: End of inserts and scans

Four new counts are added to IFCID 018 to various fast INSERT processing information. In addition, the QW0018FI field was removed because it is no longer used by DB2.

Example A-20 shows the new counters in IFCID 018.

Example A-20 Changed IFCID 018

QW0018FI	DS XL8	ROWS INSERTED VIA FAST INSERT	@KS
*			@KS
QW0018FS	DS XL8	ROWS COULD NOT USE FAST INSERT	@KS
*			@KS
QW0018FA	DS XL8	NBR TIMES FAST INSERT PIPEREFILLD	@KS
*			@KS
QW0018FW	DS XL8	NBR TIMES DB2 WAITED FOR FAST INS	@KS

IFCID 058: Accumulated pipe wait time

A new timer, QW0058PW, has been added to the IFCID 058 to record the accumulated pipe wait time.

Example A-21 shows the new timer in IFCID 058.

Example A-21 Changed IFCID 058

QW0058PW	DS	CL8	Accumulated wait time for pipe
----------	----	-----	--------------------------------

IFCID 106: New subsystem parameter

The QWP4DINA field is added to trace the internal setting of the new subsystem parameter DEFAULT_INSERT_ALGORITHM.

Example A-22 shows the new subsystem parameter in IFCID 106.

Example A-22 Changed IFCID 106

QWP4DINA	DS	H	DEFAULT_INSERT_ALGORITHM	s17836
----------	----	---	--------------------------	--------

IFCID 316: Statement level pipe wait information

A new field, QW0316_PIPE_WAIT, is added to IFCID 316 to record the pipe wait for a statement.

Example A-23 shows the new statement level pipe wait time in IFCID 316.

Example A-23 Changed IFCID 316

QW0316_PIPE_WAIT	DS	wait time for pipe wait
------------------	----	-------------------------

IFCID 401: Accumulated pipe wait time

A new field, QW0401WH, is added to IFCID 401 to record the accumulated pipe wait time.

Example A-24 shows the new accumulated time for pipe wait in IFCID 401.

Example A-24 Changed IFCID 401

QW0401WH	DS	CL8	Accumulated wait for pipe wait
----------	----	-----	--------------------------------

IFCID 413: Begin pipe wait

IFCID 413 is new to DB2 12 and is used to record the beginning of the pipe wait processing. The serviceability IFCID is used by the DB2 development team.

Example A-25 shows the resource name in IFCID 413.

Example A-25 Changed IFCID 413

```
*****
*   IFCID 0413 THE BEGINING OF A WAIT FOR A PIPE SUSPEND   *
*****
QW0413   DSECT           IFCID(QWHS0413)
QW0413PN DS   CL8       PROC NAME
QW0413RN DS   OCL6      RESOURCE NAME
QW0413DB DS   XL2       DATA BASE ID
QW0413PS DS   XL2       PAGE SET ID
```



```

QW0413PT DS XL2 PARTITION NUMBER
          DS CL2 RESERVED
QW0413DMS DS XL4 (S)
QW0413CNT DS XL2 (S)
QW0413LMT DS XL2 (S)
QW0413FL DS XL2 (S)
          DS XL2 (S)
* (S) = FOR SERVICEABILITY

```

IFCID 414: End pipe wait

Like IFCID 413, IFCID 414 is new to DB2 12 and is used to record the end of the pipe wait processing. IFCID 414 is also a serviceability IFCID used by the DB2 development team.

Example A-26 shows the new accumulated time for pipe wait in IFCID 414.

Example A-26 Changed IFCID 414

```

*****
* IFCID 0414 RECORDS THE END OF THE WAIT FOR PIPE SUSPEND *
*****
QW0414 DSECT IFCID(QWHS0414)
QW0414R DS CL1 REASON FOR PIPE WAIT RESUME
QW0414FL DS XL2 (S)
          DS XL2 (S)
          DS CL1 (S)
* (S) = FOR SERVICEABILITY

```

Lift partition limits IFCID changes limits

DB2 currently stores absolute page numbers in certain IFCID trace records with the enhancements to lift partition limits; new fields will be added to store the 6-byte absolute page numbers and new subsystem parameter values. The following IFCIDs are changed in support of the enhancement to lift partition limits:

- ▶ IFCID 006: Pre-read page number flag and partition number
- ▶ IFCID 007: Post-read page number flag and partition number
- ▶ IFCID 021: Resource name
- ▶ IFCID 106: New subsystem parameter
- ▶ IFCID 124: Page number within pageset
- ▶ IFCID 127: Agent suspend
- ▶ IFCID 128: Agent resume
- ▶ IFCID 150: Resource name
- ▶ IFCID 172: Resource name
- ▶ IFCID 196: Resource name
- ▶ IFCID 198: Page numbering flag
- ▶ IFCID 223: Identifier, new constant, and partition number
- ▶ IFCID 226: Page numbering flag
- ▶ IFCID 227: Page numbering flag
- ▶ IFCID 255: Partition number and relative page number
- ▶ IFCID 259: Partition number and relative page number
- ▶ IFCID 305: Table space partition number and type

IFCID 006: Pre-read page number flag and partition number

IFCID 006 is modified to contain a flag indicating whether the page number is relative or absolute and includes the partition number, when applicable.

Example A-27 shows the new flag and partition number field in IFCID 006.

Example A-27 Changed IFCID 006

QW0006P	DS	X	FLAGS
QW0006P1	EQU	X'80'	1 = Relative page number in QW0006PG
*			0 = Absolute page number in QW0006PG
	DS	CL2	unused
QW0006PT	DS	F	Partition number or 0 if non-
*			partitioned

IFCID 007: Post-read page number flag and partition number

Like IFCID 006, IFCID 007 is modified to contain a flag that indicates whether the page number is relative or absolute and includes the partition number, when applicable.

Example A-28 shows the new flag and partition number field in IFCID 007.

Example A-28 Changed IFCID 007

QW0007P	DS	X	Flags
QW0007P1	EQU	X'80'	1 = Relative page number in QW0007PF
*			0 = Absolute page number in QW0007PF
	DS	CL1	unused
QW0007PT	DS	F	Part number or 0 if non-partitioned

IFCID 021: Resource name

The resource name has been added to IFCID 021 to indicate resources using relative page numbers.

Example A-29 shows the new resource name information in IFCID 021.

Example A-29 Changed IFCID 021

	ORG	QW0021KR		
QW0021KE	DS	0CL7	* ID of small resource when QW0021KL=16	*
QW0021KF	DS	XL2	* partition number	*
QW0021KG	DS	CL4	* page number	*
QW0021KH	DS	XL1	* record id within page	*
	DS	CL17	* insure that offset of QW0021FC is	*
*			* 24 bytes from QW0021KR	*

IFCID 106: New subsystem parameter

The QWP4PSPN field is added to trace the internal setting of the new subsystem parameter PAGESET_PAGENUM.

Example A-30 shows the new subsystem parameter in IFCID 106.

Example A-30 Changed IFCID 106

QWP4PSPN DS	CL1	PAGESET_PAGENUM:
*		A=ABSOLUTE, R=RELATIVE

IFCID 124: Page number within pageset

IFCID 124 has been modified to include the page number within the pageset, QW01244N.

Example A-31 shows the new field for the page number in IFCID 124.

Example A-31 Changed IFCID 124

QW01244N DS	XL6	! PAGE NUMBER WITHIN PAGESET
*		! For RPN obj=2 bytes of part#
*		! 4 bytes of page#
*		! For Non-RPN obj= ignore first
*		! 2 bytes, next 4 bytes are
*		! absolute page number
	DS CL2	! unused
*		

IFCID 127: Agent suspend

Like IFCID 006 and 007, IFCID 127 is modified to contain a flag indicating whether the page number is relative or absolute and includes the partition number, when applicable. IFCID 127 is also a serviceability IFCID used by the DB2 development team.

Example A-32 shows the new flag and partition number field in IFCID 127.

Example A-32 Changed IFCID 127

QW0127P DS	X	Flags
QW0127P1 EQU	X'80'	1 = Relative page number in QW0127PG
*		0 = Absolute page number in QW0127PG
	DS CL2	unused
QW0127PT DS	F	Part number or 0 if non-partitioned
	DS CL4	unused
*	(S)	= FOR SERVICEABILITY

IFCID 128: Agent resume

Like IFCID 127, IFCID 128 is modified to contain a flag indicating whether the page number is relative or absolute and includes the partition number, when applicable. Additionally, IFCID 128 records the page number. IFCID 128 is also a serviceability IFCID used by the DB2 development team.

Example A-33 shows the new flag and partition number field in IFCID 128.

Example A-33 Changed IFCID 128

QW0128P	DS	X	Flags
QW0128P1	EQU	X'80'	1 = Relative page number in QW0128PG
*			0 = Absolute page number in QW0128PG
	DS	CL1	unused
. . .			
QW0128PG	DS	F	PAGE NUMBER
*			Based on QW0128P1, either absolute or
*			relative page number is stored here,
*			partition# can be found in QW0128PT
QW0128PT	DS	F	Part number or 0 if non-partitioned
*	(S)		= FOR SERVICEABILITY

IFCID 150: Resource name

Like IFCID 021, IFCID 150 is updated to indicate the resource name using relative page numbers.

Example A-34 shows the new resource name information in IFCID 150.

Example A-34 Changed IFCID 150

	ORG	QW0150KR	*
QW0150KE	DS	OCL7	* ID of small resource when QW0150KL=16
QW0150KF	DS	XL2	* partition number
QW0150KG	DS	CL4	* page number
QW0150KH	DS	XL1	* record id within page

IFCID 172: Resource name

Like IFCID 021 and 150, IFCID 172 is updated to indicate the resource name using relative page numbers.

Example A-35 shows the new resource name information in IFCID 172.

Example A-35 Changed IFCID 172

	ORG	QW0172KR	*
QW0172KE	DS	OCL7	* ID of small resource when QW0172RL=16
QW0172KF	DS	XL2	* partition number
QW0172KG	DS	CL4	* page number
QW0172KH	DS	XL1	* record id within page
	DS	CL17	REST OF SPACE FOR 28 BYTE RESOURCE NAME

IFCID 196: Resource name

IFCID 196 is also updated to indicate the resource name using relative page numbers.

Example A-36 shows the new resource name information in IFCID 196.

Example A-36 Changed IFCID 196

ORG	QW0196KR	*	
QW0196KE DS	OCL7	*	ID of small resource when QW0196RL=16
QW0196KF DS	XL2	*	partition number
QW0196KG DS	CL4	*	page number
QW0196KH DS	XL1	*	record id within page
DS	CL17		REST OF SPACE FOR 28 BYTE RESOURCE NAME

IFCID 198: Page numbering flag

IFCID 198 is also modified to contain a flag indicating whether the page number is relative or absolute, and the partition number and page number.

Example A-37 shows the new fields in IFCID 198.

Example A-37 Changed IFCID 198

QW0198PN DS	F		PAGE NUMBER
*			Based on QW0198P1, either absolute or
*			relative page number is stored here,
*			the partition# can be found in QW0198PT
. . .			
QW0198P DS	X		Flags
QW0198P1 EQU	X'80'		1 = Relative page number in QW0198PN
*			0 = Absolute page number in QW0198PN
DS	XL2		unused
QW0198PT DS	F		Partition number or 0 if non-partitioned
*			

IFCID 223: Identifier, new constant, and partition number

IFCID 223 is modified to contain a resource identifier, a new constant indicating a partitioned table space with relative page numbers, and a partition number.

Example A-38 shows the new fields in IFCID 223.

Example A-38 Changed IFCID 223

QW0223KY DS	OCL5		ID of small resource when QW0223TY='L' or
*			'R'
. . .			
QW0223TR EQU	C'R'		PBR UTS that uses relative page numbers
QW0223PT DS	XL2		1-based partition number if partitioned
*	(S)		= FOR SERVICEABILITY

IFCID 226: Page numbering flag

IFCID 226 is also modified to contain a flag indicating whether the page number is relative or absolute, and the partition number and page number. IFCID 226 is also a serviceability IFCID used by the DB2 development team.

Example A-39 shows the new fields in IFCID 226.

Example A-39 Changed IFCID 226

QW0226PG DS	F	PAGE NUMBER TO READ/WRITE
*		Based on QW0226P1, either absolute or
*		relative page number is stored here,
*		partition# can be found in QW0226PT
. . .		
QW0226P DS	X	Flags
QW0226P1 EQU	X'80'	1 = Relative page number in QW0226PG
*		0 = Absolute page number in QW0226PG
	DS XL2	unused
QW0226PT DS	F	Partition number or 0 if non-partitioned
*	(S) =	FOR SERVICEABILITY

IFCID 227: Page numbering flag

Like IFCID 226, IFCID 227 is also modified to contain a flag indicating whether the page number was relative or absolute, and the partition number and page number. IFCID 227 is also a serviceability IFCID used by the DB2 development team.

Example A-40 shows the new fields in IFCID 227.

Example A-40 Changed IFCID 227

QW0227PG DS	F	PAGE NUMBER TO READ/WRITE
*		Based on QW0227P1, either absolute or
*		relative page number is stored here,
*		partition# can be found in QW0227PT
. . .		
QW0227P DS	X	Flags
QW0227P1 EQU	X'80'	1 = Relative page number in QW0227PG
*		0 = Absolute page number in QW0227PG
	DS XL2	unused
QW0227PT DS	F	Partition number or 0 if non-partitioned
*	(S) =	FOR SERVICEABILITY

IFCID 255: Partition number and relative page number

IFCID 255 has two new fields for the partition number and relative page number.

Example A-41 shows the new fields in IFCID 255.

Example A-41 Changed IFCID 255

QW0255P1 DS	CL2	2-byte Pageset piece/partition number
QW0255P2 DS	CL4	4-byte Relative page# (within the piece)

IFCID 259: Partition number and relative page number

IFCID 259 has two new fields for the partition number and relative page number.

Example A-42 shows the new fields in IFCID 255.

Example A-42 Changed IFCID 255

QW0259K4	DS	CL4	4-byte Relative page number for RPN object
*			For QW0259G1, relative pg# is stored here
	DS	CL16	16 BYTES OF ZEROS
. . .			
	ORG	QW0259K4	
QW0259KQ	DS	CL3	Relative page number for non-RPN object
	DS	CL1	reserved
	DS	CL16	16 bytes of zeros

IFCID 305: Table space partition number and type

IFCID 305 is modified to record the partition number for a table space that uses relative page numbers in addition to introducing a new table space type value.

Example A-43 shows the new fields in IFCID 305.

Example A-43 Changed IFCID 305

QW0305PT	DS	XL2	Partition number.
*			Valid when QW0305TY='R'
. . .			
QW0305TY	DS	CL1	TABLE SPACE TYPE
*			POSSIBLE VALUES ARE :
*			'N' - NON LARGE TABLE SPACE
*			'L' - NON-EA 5-BYTE RID TABLE SPACE
*			'V' - EA 5-BYTE RID TABLE SPACE
*			'R' - PBR UTS that uses relative
*			page numbers

Large object (LOB) compression IFCID changes

DB2 12 introduces LOB compression capabilities with the new IBM zEnterprise® Data Compression (zEDC) hardware. IFCID 003 and IFCID 106 are modified to trace LOB compression times and waits plus the new subsystem parameter internal setting.

IFCID 003: Accounting control block

Two new fields, QWAX_LOBCOMP_WAIT and QWAX_LOBCOMP_COUNT, are added to the accounting control block (QWAX) to indicate the accumulated wait time and the number of wait trace events processed for LOB compression.

Example A-44 shows the new fields in IFCID 003.

Example A-44 New fields for LOB compression in IFCID 003

QWAX_LOBCOMP_WAIT DS	CL8	/* Accumulated wait time for LOB	
*		compression	*/
QWAX_LOBCOMP_COUNT DS	F	/* Number of wait trace events	
*		processed for LOB compression	*/

IFCID 106: New subsystem parameter

The QWP4CDRL field is added to trace the internal setting of the new subsystem parameter COMPRESS_DIRLOB.

Example A-45 shows the new field in IFCID 106.

Example A-45 Changed IFCID 106

QWP4MISD DS	X		
*	EQU	X'80'	Not available e17790
QWP4CDRL EQU	X'40'	COMPRESS_DIRLOB	s10853

Transfer ownership IFCID changes

DB2 12 provide support to allow you to alter object owners with the new SQL statement TRANSFER OWNERSHIP. The following IFCIDs are modified:

- ▶ IFCID 002: RDS statistics block
- ▶ IFCID 062: Statement type
- ▶ IFCID 140: Source object owner and name
- ▶ IFCID 361: Source object owner and name

IFCID 002: RDS statistics block

The QXTRNOWN field is added to the RDS statistics block (QXST) to indicate the number of times that ownership was transferred.

Example A-46 shows a snippet of the IFCID 002 indicating the new field.

Example A-46 Changed IFCID 002

QXTRNOWN	DS	D	# of TRANSFER OWNERSHIP
----------	----	---	-------------------------

IFCID 062: Statement type

A new statement type of X'AB' is added to indicate the statement was a TRANSFER OWNERSHIP statement in IFCID 062.

Example A-47 shows the new field in IFCID 062.

Example A-47 Changed IFCID 062

QW0062TO EQU	X'AB'	TRANSFER OWNERSHIP
--------------	-------	--------------------

IFCID 140: Source object owner and name

The QW0140SC and QW0140SN fields, which describe the source object owner and name, are updated in IFCID 140.

Example A-48 shows the new fields in IFCID 140.

Example A-48 Changed IFCID 140

```
QW0140SC DS   CL8   %U SOURCE OBJECT OWNER: Three cases
*
*           1) If object type not equal User Auth -
*             Qualifier of the object against
*             which authorization was checked.
*             Valid for qualifiable objects.
*           2) If object type equals User Auth -
*             Qualifier of ALIAS being created.
*             Valid for CREATE ALIAS privilege.
*           3) If object type equals User Auth -
*             Qualifier of the object being
*             transferred. Valid for TRANSFER
*             OWNERSHIP statement.
*             Truncated if QW0140SC_Off=0
*
QW0140SN DS   CL18  %U SOURCE OBJECT NAME: Three cases
*
*           1) If object type not equal User Auth -
*             Name of the object against which
*             authorization was checked.
*           2) If object type equals User Auth -
*             Name of the object being created. Valid
*             when privilege is CREATE ALIAS,
*             CREATEDBA, CREATEDBC, or CREATE
*             STOGROUP.
*           3) If object type equals User Auth -
*             Name of the object being transferred.
*             Valid for TRANSFER OWNERSHIP statement.
*             Truncated if QW0140SN_Off=0
*
```

IFCID 361: Source object owner and name

The descriptions for the QW0361SC_Var and QW0361SN_Var fields for the source object owner and name are updated in IFCID 361.

Example A-49 shows the new fields in IFCID 361.

Example A-49 Changed IFCID 361

```
QW0361SC_Var DS  OCL128 %U SOURCE OBJECT QUALIFIER/OWNER
*
*           If object type equals User Auth -
*           Qualifier of the object being
*           transferred. Valid for TRANSFER
*           OWNERSHIP statement.
*
* . . .
QW0361SN_Var DS  OCL128 %U Source object Name
*
*           If object type equals User Auth -
*           Name of the object being transferred.
*           Valid for TRANSFER OWNERSHIP statement.
*
```

UNLOAD privilege for UNLOAD utility IFCID changes

A new UNLOAD privilege can be specified for the UNLOAD utility through the new AUTH_COMPATIBILITY subsystem parameter. IFCID 404 is introduced as a serviceability trace and is described in “New IFCIDs” on page 254. In addition, IFCID 106 is modified to trace the internal setting of the parameter.

IFCID 106: New subsystem parameter

The QWP4AUTC field is added to trace the internal setting of the new subsystem parameter AUTH_COMPATIBILITY. The QWP4AUTCSU field indicates the SELECT_FOR_UNLOAD option of AUTH_COMPATIBILITY has been specified.

Example A-50 shows the new fields in IFCID 106.

Example A-50 Changed IFCID 106

QWP4AUTC DS	XL1	AUTH_COMPATIBILITY	s20166
QWP4AUTCSU EQU	X'80'	- SELECT_FOR_UNLOAD	s20166

Additional changed IFCIDs

In addition to the IFCID changes, DB2 12 introduces changes to the following IFCIDs:

- ▶ IFCID 106: Modifications and enhancements
- ▶ IFCID 125: Adaptive index processing
- ▶ New QWAC_WORKFILE_MAX and QWAC_WORKFILE_CURR fields

IFCID 106: Modifications and enhancements

IFCID 106 is modified to remove fields no longer used in DB2 12:

- ▶ The QWP4RIFS field is removed. The QWP4RIFS field was used to trace the internal setting of the REORG_IGNORE_FREESPACE subsystem parameter but that parameter was deprecated in DB2 10 and in DB2 11 and is now eliminated in DB2 12.
- ▶ Starting in DB2 12, the storage management of LOB and XML data is managed by DB2 therefore the LOBVALA, LOBVALS, XMLVALA, and XMLVALS subsystem parameters are no longer recorded.

Enhancements to IFCID 106 are as follows:

- ▶ Eight new fields to trace internal settings of three new subsystem parameters are introduced for resource limit facility (RLF) for static SQL enhancements:
 - Fields for subsystem parameter RLFERRSTC:
 - QWP1RLFFS
 - QWP1RLFUS
 - QWP1RLFNS
 - Fields for subsystem parameter RLFENABL:
 - QWP1RLFDNEN
 - QWP1RLFSTEN

- Fields for subsystem parameter RLFERRDSTC:
 - QWP9RLFERS
 - QWP9RLFNS
 - QWP9RLFSL
 - ▶ The field QWP1PFASY is added to trace the internal setting of the new subsystem parameter PROFILE_AUTOSTART.
 - ▶ The field QWP4RSO is added to trace the internal settings of the new subsystem parameter RETRY_STOPPED_OBJECTS.
 - ▶ The field QWP4MNSU is added to trace the internal setting of the MATERIALIZE_NODET_SQLTUDF subsystem parameter.
 - ▶ The field QWP4ERTS is added to trace the internal settings of the new subsystem parameter RENAMETABLE.
 - ▶ For online backup and recovery enhancements, the QWP4CYFR field is added for the COPY support added to FASTREP(REQ).
- Also, four new subsystem parameters are introduced and their internal settings are being recorded:
- QWP4BSACP for the ALTERNATE_CP parameter
 - QWP4UDBSG for the UTIL_DBBSG parameter
 - QWP4ULBSG for the UTIL_LGBSG parameter
 - QWP4UHMDH for the UTILS_HSM_MSGDS_HLQ parameter
- ▶ The QWP4DDL field is added to trace the internal setting of the new subsystem parameter DDL_MATERIALIZATION.
 - ▶ The QWPAPERREC is added to trace the internal setting of the new subsystem parameter PEER_RECOVERY.
 - ▶ The QWP4SFPR field is added to trace the internal setting of the new subsystem parameter STATFDBK_PROFILE.

Another subsystem parameter, QWP4TPTM, is introduced in DB2 12 and is retrofit to DB2 11 and DB2 10. The QWP4TPTM is added to trace the internal setting of the new subsystem parameter TEMPLATE_TIME.

Example A-51 shows the modified IFCID 106 for the items described above.

Example A-51 Changed IFCID 106

QWP1RLFFS	EQU	X'10'	IF 1 INDICATE NOLIMIT (STATIC)	S20000
QWP1RLFUS	EQU	X'08'	IF 1 INDICATE NORUN (STATIC)	S20000
QWP1RLFDNEN	EQU	X'04'	1 when RLFENABLE=DYNAMIC or ALL	S20000
QWP1RLFSTEN	EQU	X'02'	1 when RLFENALBE=STATIC or ALL	S20000
. . .				
QWP1PFASY	EQU	X'10'	PROFILE_AUTOSTART=YES	dp1897
. . .				
	DS	CL4	Do not reuse	s22058
QWP1LVS	DS	F	(s)	s22058
. . .				
	DS	CL4	Do not reuse	s22058
QWP1XVS	DS	F	(s)	s22058
. . .				
QWP1RLFNS	DS	F	RLF static limit in SU's	S20000
. . .				
QWP4MS4D	DS	X		

*		X'80'		Not available	s2593
. . .					
QWP4RSO	EQU	X'01'		RETRY_STOPPED_OBJECTS	DM1851
. . .					
QWP4MNSU	EQU	X'80'		MATERIALIZE_NODET_SQLTUDF	DM1912
. . .					
QWP4ERTS	EQU	X'40'		RENAMETABLE	DN1840
*				'0' = DISALLOW_DEP_VIEW_SQLTUDF	DN1840
*				'1' = ALLOW_DEP_VIEW_SQLTUDF	DN1840
. . .					
QWP4BSACP	DS	CL16		ALTERNATE_CP	s12997
QWP4UDBSG	DS	CL8		UTIL_DBBSG	s12997
QWP4ULBSG	DS	CL8		UTIL_LGBSG	s12997
QWP4CYFR	DS	CL1		COPY_FASTREPLICATION	s23187
QWP4DDL	DS	CL1		DDL_MATERIALIZATION	n22751
*				I=ALWAYS_IMMEDIATE	n22751
*				P=ALWAYS_PENDING	n22751
. . .					
QWP4UHMDH	DS	CL8		UTILS_HSM_MSGDS_HLQ	s17863
. . .					
QWP4SFPR	EQU	X'20'		STATFDBK_PROFILE	s24345
. . .					
QWP9RLFLS	EQU	X'20'		If 1, indicate NOLIMIT (static)	S20000
QWP9RLFRS	EQU	X'10'		If 1, indicate NORUN (static)	S20000
. . .					
QWP9RLFNS	DS	F		RLF static limit in SU's	s20000
. . .					
QWPAPEERREC	DS	CL1		PEER_RECOVERY	DP1857
*				'N' = NONE	DP1857
*				'R' = RECOVER	DP1857
*				'A' = ASSIST	DP1857
*				'B' = BOTH	DP1857

IFCID 125: Adaptive index processing

The following fields are added to IFCID 125. The fields record the RID list processing statistics for adaptive index processing and also details about the adaptive index processing that was performed.

- ▶ QW0125TI
- ▶ QW0125QI
- ▶ QW0125_TRSN
- ▶ QW0125_PRSN
- ▶ QW0125_ORSN
- ▶ QW0125_TRSN

Example A-52 shows those new fields in IFCID 125 for adaptive index processing.

Example A-52 IFCID 125 new fields

QW0125TI	DS	XL8	index probing estimate: total number of
*			RIDs in the index (set to MAX BIGINT for
*			full leg)
QW0125QI	DS	XL8	index probing estimate: number of RIDs
*			within the keyrange, adjusted for filter

```

*
* factor
QW0125_TRSN DS CL1 reason leg was terminated
* F: leg was marked 'full'
* T: leg with < 32 RIDs
QW0125_PRSN DS CL1 reason leg not probed
* A: all legs fetched all RIDs
* B: this leg fetched all RIDs (<1 RIDblock)
* E: probing failed
* F: leg was marked 'full'
* K: cannot probe - missing high/low key
* M: mix of 'R'/'I'/'U' entries does not get
* reordered
* O: APS indicated to not probe
* S: earlier leg of index AND-ing was
* 'likely' very filtering
* V: leg 'likely' very filtering
QW0125_ORSN DS CL1 reason leg was reodered
* V: leg 'likely' very filtering
* P: probing
QW0125_FRSN DS CL1 reason leg was marked 'full'
* L: non-filtering LIKE
* R: range predicate (non-LIKE)
* P: if OR-ing and est. # of RIDs > 30%
* or if AND-ing and est. # of RIDs > 50%
* of the table
* M: if not reason 'P', but est. # of RIDs
* > 'ridlist logical limit'
* G: aggressive termination
* T: most selective leg of AND-ing
* with FF >= 35%

```

New QWAC_WORKFILE_MAX and QWAC_WORKFILE_CURR fields

Two fields, QWAC_WORKFILE_MAX and QWAC_WORKFILE_CURR, are added to the QWAC control block to provide thread-level information about the workfile usage.

Example A-53 shows changes to the QWAC.

Example A-53 Changed QWAC

```

QWAC_WORKFILE_MAX DS XL8 /* Maximum number of workfile blocks */
* /* being used by this agent at any */
* /* given point in time (traditional */
* /* workfile use, DGTT and DGTT */
* /* indexes) */
QWAC_WORKFILE_CURR DS XL8 /* Current number of workfile blocks */
* /* being used by this agent */
* /* (traditional workfile use, DGTT */
* /* and DGTT indexes) */

```



Additional material

This book refers to additional material that can be downloaded from the Internet as described in the following sections.

Locating the web material

The web material associated with this book is available in softcopy on the Internet from the IBM Redbooks web server:

<ftp://www.redbooks.ibm.com/redbooks/SG248383>

Alternatively, you can go to the IBM Redbooks website:

ibm.com/redbooks

Search for SG248383, select the title, and then click **Additional materials** to open the directory that corresponds with the IBM Redbooks form number, SG248383.

Downloading and extracting the web material

Create a subdirectory (folder) on your workstation, and extract the contents of the web material .zip file into this folder.

Redbooks

IBM DB2 12 for z/OS Technical Overview

SG24-8383-00

ISBN 0738442305



(0.5" spine)

0.475" x 0.873"

250 <-> 459 pages



SG24-8383-00

ISBN 0738442305

Printed in U.S.A.

Get connected

