IBM® Information Management Software

IBM DB2 11 for z/OS Technical Overview

- Understand the synergy with System z platform
- Enhance applications and avoid incompatibilities
- Run business analytics and scoring adapter

Paolo Bruni
Felipe Bortoletto
Ravikumar Kalyanasundaram
Sabine Kaschta
Glenn McGeoch
Cristian Molaro

Redbooks

ibm.com/redbooks

International Technical Support Organization

IBM DB2 11 for z/OS Technical Overview

December 2013

SG24-8180-00

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

First Edition (December 2013)

This edition applies to Version 11, Release 1 of DB2 for z/OS (program number 5615-DB2) and Version 11, Release 1 of DB2 Utilities Suite for z/OS (program number 5655-W87).

Note: This book is based on a pre-GA version of a product and may not apply when the product becomes generally available. We recommend that you consult the product documentation or follow-on versions of this book for more current information.

Contents

Figures

Tables

Examples

Notices

This information was developed for products and services offered in the U.S.A.

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, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: 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 states 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 supply in any way it believes appropriate without incurring any obligation to you.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

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.

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 the names and addresses used by an actual business enterprise 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.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

CICS®
DB2®
DB2 Connect™
Distributed Relational Database Architecture™
DRDA®
DS8000®
FICON®
FlashCopy®
IBM®
IMS™
InfoSphere®
iSeries®
Language Environment®
MQSeries®
MVS™
OMEGAMON®
Optim™
Parallel Sysplex®
pureQuery™
pureXML®
RACF®
Redbooks®
Redbooks (logo) ®
RETAIN®
RMF™
SPSS®
System i®
System Storage®
System z®
Tivoli®
VTAM®
WebSphere®
z/OS®
z10™
zEnterprise™

The following terms are trademarks of other companies:

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

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

UNIX is a registered trademark of The Open Group in the United States and other countries.

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

Summary of changes

This section describes the technical changes made in this edition of the book and in previous editions. This edition might also include minor corrections and editorial changes that are not identified.

Summary of Changes
for SG24-8180-00
for IBM DB2 11 for z/OS Technical Overview
as created or updated on May 5, 2016.

December 2013, First Edition

This revision of the first edition published December 2013, reflects the addition, deletion, or modification of new and changed information described below.

May 2014, First Update

Changed information

- Replaced Figure 4-9 on page 80.
- Removed an error message at “Problem determination” on page 254.
- Corrected parameter spelling of PROGAUTH keyword.

New information

- Updated RSU at Appendix B.1, “DB2 APARs” on page 438.

Preface

IBM® DB2® Version 11.1 for z/OS® (*DB2 11 for z/OS* or just *DB2 11* throughout this book) is the fifteenth release of DB2 for IBM MVS™. It brings performance and synergy with the IBM System z® hardware and opportunities to drive business value in the following areas:

- Unmatched reliability, availability, and scalability
  - Improved data sharing performance and efficiency
  - Less downtime by removing growth limitations
  - Simplified management, improved autonomics, and reduced planned outages
- Save money, save time
  - Aggressive CPU reduction goals
  - Additional utilities performance and CPU improvements
  - Save time and resources with new autonomic and application development capabilities
- Simpler, faster migration
  - SQL compatibility, divorce system migration from application migration
  - Access path stability improvements
  - Better application performance with SQL and XML enhancements
- Enhanced business analytics
  - Faster, more efficient performance for query workloads
  - Accelerator enhancements
  - More efficient inline database scoring enables predictive analytics

The DB2 11 environment is available either for new installations of DB2 or for migrations from DB2 10 for z/OS subsystems only.

This IBM Redbooks® publication introduces the enhancements made available with DB2 11 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 11.

Authors

This book was produced by a team of specialists from around the world working at the IBM Silicon Valley Lab, San Jose, California.

Paolo Bruni is a DB2 Information Management Project Leader at the International Technical Support Organization based in the Silicon Valley Lab. He has authored several IBM Redbooks publications about DB2 for z/OS and related tools, and has conducted workshops and seminars worldwide. During his years with IBM, in development and in the field, Paolo has worked mostly on database systems.

Felipe Bortoletto is a Certified IBM IT Specialist in information management and an IBM Certified DBA for DB2 for z/OS V7, V8, V9 and DB2 10. He has 18 years of experience in IT with 13 years of experience with DB2 for z/OS. He joined IBM 9 years ago and is currently a member of the IBM GBS in Brazil. He holds a degree in Computer Science from UNICAMP. Felipe co-authored *Securing and Auditing Data on DB2 for z/OS*, SG24-7720 and *DB2 10 for z/OS Performance Topics*, SG24-7942.

Ravikumar Kalyanasundaram is an IBM Certified Thought Leader in the I/T Specialist profession and a Distinguished I/T Specialist (Certified by The Open Group). He has more than 21 years of experience and currently working as a Senior Managing Consultant at IBM Software Group - IM Lab Services. He provides technical consulting services to clients world wide, utilizing specialized knowledge and skills in DB2 for z/OS database, Analytics Accelerator and Information Management tools. He provides a truly integrated set of high quality services with a focus on database performance management. He plays a direct role in increasing the long term strength and enhancing the market position and competitive posture of IBM database products and tools. He holds a Bachelors degree in Electrical and Electronics Engineering and a Masters degree in Business Administration (MBA). He is a detail oriented person, with outstanding project management, problem-solving, team-building and decision making skills. Ravi is a co-author of several IBM Redbooks publications, including *Optimizing Restore and Recovery Solutions with DB2 Recovery Expert for z/OS V2.1*, SG24-7606, *DB2 9 for z/OS: Resource Serialization and Concurrency Control*, SG24-4725, *DB2 10 for z/OS Performance Topics*, SG24-7942, *Optimizing DB2 Queries with IBM DB2 Analytics Accelerator for z/OS*, SG24-8005, *Hybrid Analytics Solution using IBM DB2 Analytics Accelerator for z/OS V3.1*, SG24-8151.

Sabine Kaschta is a DB2 Specialist working for the IBM Software Group in Germany. Currently, she works as a Segment Skills Planner for the worldwide curriculum for DB2 for z/OS training as well as IT consultant. She also works on course development and in her role as IT consultant enjoys teaching customized workshops for customers worldwide. Sabine has 22 years of experience working with DB2. Before joining IBM in 1998, she worked for a third-party vendor providing second-level support for DB2 utilities. She is experienced in DB2 system programming and client/server implementations in the insurance industry in Germany. She co-authored several IBM Redbooks publications, including *DB2 UDB for OS/390 and Continuous Availability*, SG24-5486, *Cross-Platform DB2 Distributed Stored Procedures: Building and Debugging*, SG24-5485, *IBM TotalStorage Migration Guide for the SAP User*, SG24-6400*, DB2 UDB for z/OS Version 8: Everything You Ever Wanted to Know, ... and More*, SG24-6079, *DB2 9 for z/OS Technical Overview*, SG24-7330*, DB2 9 for z/OS Stored Procedures: Through the CALL and Beyond*, SG24-7604*,* and *DB2 9 for z/OS: Deploying SOA Solutions*, SG24-7663*, DB2 10 for z/OS Technical Overview*, SG24-7892.

Glenn McGeoch is a Senior DB2 Consultant for the IBM DB2 for z/OS Lab Services organization based in San Francisco, CA, US. He has 36 years of experience in the software industry, with 28 years of experience in working with DB2 for z/OS. He holds a degree in Business Administration from the University of Massachusetts and an MBA from Rensselaer Polytechnic Institute. Glenn worked for 19 years as an IBM customer with a focus on IBM CICS® and DB2 application development, and spent the last 17 years with IBM assisting DB2 customers. His areas of expertise include application design and performance, stored procedures, and DB2 migration planning. He has presented to regional DB2 User Groups and to customers on various DB2 topics. Glenn co-authored several IBM Redbooks publications, including *DB2 for z/OS Stored Procedures: Through the CALL and Beyond*, SG24-7083, *DB2 9 for z/OS Stored Procedures: Through the CALL and Beyond*, SG24-7604, *DB2 10 for z/OS Performance Topics*, SG24-7942, and *Streamline Business with Consolidation and Conversion to DB2 for z/OS*, SG24-8044.

Cristian Molaro is an IBM Gold Consultant, an independent DB2 specialist, and an instructor based in Belgium. He has been recognized by IBM as an IBM Champion for Information Management in 2009, 2010, 2011, 2012, and 2013. His main activity is linked to DB2 for z/OS administration and performance. Cristian is co-author of several IBM Redbooks publications related to DB2. He holds a Chemical Engineering degree and a Masters degree in Management Sciences. Cristian was recognized by IBM as “TOP” EMEA Consultant at the IDUG EMEA DB2 Tech Conference Prague 2011.

Special thanks to people worldwide who have contributed to the preparation of the material for the Sequoia Introduction Program and the programming specifications. We have used that material as the basis for this book.

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

Bob Haimowitz
International Technical Support Organization

Jeff Berger
Mengchu Cai
Gayathiri Chandran
Ramani Croisettier
Janet Figone
Bill Franklin
Jeff Josten
Akiko Hoshikawa
Gopal Krishnan
Laura Kunioka-Weis
Allan Lebovitz
Chris Leung
Maggie Lin
Irene Liu
John Lyle
Jane Man
Bruce McAlister
Ka-Chun Ng
Jim Pickel
Emily Prakash
Terry Purcell
Jim Ruddy
John Tobler
Jay Yothers
Debbie Yu
IBM Silicon Valley Lab

Shirley Brost
IBM Information Management Lab Services

Robert Gensler
Glenn Wilcock
IBM Tucson Lab

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 publications

- Find us on Facebook:

  http://www.facebook.com/IBMRedbooks

- Follow us on Twitter:

  http://twitter.com/ibmredbooks

- Look for us on LinkedIn:

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

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

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

- Stay current on recent Redbooks publications with RSS Feeds:

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

1

DB2 11 for z/OS at a glance

DB2 11 for z/OS delivers key innovations that reduce your total cost of ownership and that increase availability, scalability, and security for your business-critical information. In addition, DB2 11 for z/OS offers improvements for analytics and makes installation and migration simpler and faster. DB2 11 for z/OS can also connect core operational data with big data to drive more business value and uses secure connections to support increasing mobile device requests.

This chapter provides a brief overview of the most important functions provided by IBM DB2 Version 11.1 for z/OS (also referred to as *DB2 11 for z/OS* or just *DB2 11* throughout this book). For the purposes of this discussion, these functions are divided into the following categories, which correspond to the parts of this book:

- Subsystem
- Application functions
- Operations and performance

1.1 Subsystem

As with all previous versions, DB2 11 for z/OS takes advantage of the latest improvements in the platform. DB2 11 increases the synergy with System z hardware and software to provide better performance, more resilience, and better function for an overall improved value.

DB2 11 benefits from advances in large real memory support, faster processors, and better hardware compression.

Additional RUNSTATS and LOAD workloads can now take advantage of zIIP engines, driving faster **LOAD** processing, improved performance, and CPU savings. With IBM zEnterprise™ EC12 (zEC12) and DB2 11, you can achieve additional CPU reductions by using pageable large (1 MB) page frames and Flash Express and by supporting 2 GB page frames.

In DB2 11, the relative byte address (RBA) and log record sequence number (LRSN) log records are expanded from basic 6-byte format to extended 10-byte format. You also can convert the RBA and LRSN to extended 10-byte format to avoid reaching the logging limits. Extending the RBA and LRSN to 10-byte format helps to avoid wrapping of the RBA and LRSN values, which can cause system problems. In addition, the increased precision of the 10-byte format for the LRSN provides performance improvements for data sharing environments.

In DB2 11 you can alter the limit keys for a partitioned table space without impacting the availability of the data. When you change the limit key values, the data remains available, and applications can continue to access the data.

Under some conditions, DB2 11 can improve performance, because there is less need to use a **REORG** utility. These improvements come in the following ways:

- By automatically cleaning up pseudo-deleted index entries.
- Second, by managing free space differently to reduce the number of indirect references when the row size varies.

1.2 Application functions

DB2 11 delivers several SQL enhancements that can help applications to ease development and porting. DB2 11 also provides several enhancements to the support of IBM pureXML®, stored procedures, and connectivity. Application-enabling infrastructure changes allow for intersection with big data, XML, and e-business.

In addition to providing more SQL aggregation improvements, the combination of z/OS Communication Server for TCP/IP, DB2 11 functions and the many new features in IBM DB2 Analytics Accelerator V4 improve the bandwidth of the hybrid system solution for OLTP and analytical workloads.

IBM also provides the InfoSphere® BigInsights solution, which brings the power of Hadoop to the enterprise. DB2 11 provides an efficient access to BigInsights data with the use of a “universal table” user-defined function that allows this data to be read in DB2. Alternatively, DB2 data can also be accessed from BigInsights using a Jaql JDBC driver.

In DB2 11, expanded support for SQL, XML, and temporal tables can result in improved application performance.

DB2 11 allows you to automatically insert rows that are deleted from one table into a separate table called an *archive table*. Archive tables provide the following benefits:

- DB2 can manage historical data for you. You do not have to manually move data to a separate table.
- Because rows that are infrequently accessed are stored in a separate table, you can potentially improve the performance of queries against the archive-enabled table.
- You can modify queries to include or exclude archive table data without having to change the SQL statement and prepare the application again. Instead, you can control the scope of the query with a global variable.
- You can store archive tables on a lower-cost device to reduce operating costs.

1.3 Operations and performance

With key enhancements in DB2 11, DB2 for z/OS and System z continue to lead the industry in security and auditing. With DB2 11 RACF® Exit enhancements, external security that is managed by RACF administrators can now fully handle access to DB2 objects. This function provides the following benefits:

- The use of the OWNER keyword is now acceptable by RACF. A new installation parameter allows the use of the package OWNER for static and dynamic SQL authorization.
- A refresh function for DB2's authorization cache is implemented, which allows RACF to dynamically notify changes and keep the security definitions synchronized.
- DB2 11 also removes some of the restrictions on the use of the **SQL GROUP BY**, **DISTINCT**, and **UNION** clauses when querying a masked table.

Utilities have enhancements that can significantly reduce execution time for **REORG**, more zIIP eligibility for the **RUNSTATS** utility. Point-in-time recovery is now allowed, which follows dynamic schema changes and can improve application availability and DBA productivity.

In DB2 11, simpler, faster migration results in a faster return on your investment. This version of DB2 for z/OS provides enhancements to the DB2 installation CLIST, ISPF panels, and jobs, and provides new installation verification procedures (IVPs). Also, a new feature helps to streamline the migration process by allowing an application with incompatible SQL or XML to continue running on DB2 11 without requiring code changes. You no longer need to wait for application changes to be planned and delivered for your business to realize the benefits of DB2 11.

Performance improvements in DB2 11 focus on optimizing query processing and reducing CPU processing time without causing significant administration or application changes. However, DB2 11 also offers a balanced approach to performance improvements across all types of workloads, whether your workloads are for online transaction processing (OLTP), batch, or utilities.

Early measurements show DB2 11 CPU savings of up to 10% for complex OLTP and update-intensive batch workloads when compared to DB2 10. Queries can see up to 25% DB2 CPU savings for uncompressed tables and up to 40% when running against compressed tables. Benefits can be achieved without any application changes, just binding. More CPU savings are possible for specific workloads, with application changes.

Part 1

Subsystem

DB2 continues to evolve by removing structural constraints to support its increasing use by concurrent workloads. Several improvements to the DB2 engine allow better and more use of System z hardware and software functions that provide growth, performance, and cost of ownership reduction.

The following chapters in this part describe functions generally related to the DB2 subsystem and the z/OS platform:

2

Synergy with System z

As with all previous versions, DB2 11 for z/OS takes advantage of the latest improvements in the platform. DB2 11 increases the synergy with System z hardware and software to provide better performance, more resilience, and better function for an overall improved value. In addition, DB2 11 benefits from advances in large real memory support, faster processors, and better hardware compression.

DB2 for z/OS is designed to take advantage of the System z platform to provide capabilities that are unmatched in other database software products. The DB2 development team works closely with the System z hardware and software teams to take advantage of existing System z enhancements and to drive many of the enhancements that are available on the System z platform.

This chapter describes the synergy between DB2 11 and the System z hardware and software that removes constraints for growth, improves reliability and availability, and continues to improve total cost of ownership and performance. It also outlines features and functions of the IBM zEnterprise platform and z/OS V2R1 that are expected to benefit DB2 for z/OS. It includes the following topics:

- Synergy with IBM zEnterprise System
- Synergy with IBM System z and z/OS
- Using zIIP speciality processors
- Reduced need for REORG
- DFSMS storage tiers
- Additional System z enhancements

2.1 Synergy with IBM zEnterprise System

The IBM zEnterprise System delivers unique value and industry-leading capabilities that allow you to maximize the business value of your unique information. The IBM zEnterprise EC12 (zEC12) is the cornerstone of the latest zEnterprise System and flagship of the IBM Systems portfolio. The superscalar design allows the zEC12 to deliver a record-level capacity. It is powered by 120 of the world's most powerful microprocessors that run at 5.5 GHz and is capable of executing more than 78,000 millions of instructions per second (MIPS).

DB2 for z/OS takes advantage of the following features available with the zEC12:

- Faster CPU speed speed
- More system capacity
- zEC12 hardware features

2.1.1 Faster CPU speed

The CPU speed of the zEC12 has been measured at 1.25 times the speed of the z196. The improved CPU speed of the zEC12 provides the following performance improvements over the z196 for DB2:

- 20-28% CPU reduction for OLTP workloads
- 25% CPU reduction for Query and Utility workloads
- 1-15% less compression overhead with DB2 data

2.1.2 More system capacity

The zEC12 provides up to 50% more total capacity than the z196. This increase capacity makes the zEC12 an excellent choice to grow either horizontally or vertically within one server. The zEC12 is a good choice if you are planning a large scale consolidation because of its ability to provide secure data serving and to support mission-critical transaction processing. DB2 11 provides scalability features, as described in Chapter 3, "Scalability" on page 23. The zEC12 provides the synergy to take advantage of these scalability enhancements in DB2 11.

2.1.3 zEC12 hardware features

DB2 11 takes advantage of the following hardware features of the zEC12.

Large frame area (LFAREA)

The large frame area is used for the fixed 1 MB large page frames and fixed 2 GB large page frames. Using large page frames can improve performance for some applications by reducing the overhead of dynamic address translation. This improvement is achieved by each large frame requiring only one entry in the translation lookaside buffer (TLB), as compared to the larger number of entries that are required for an equivalent number of 4 KB page frames. A single TLB entry improves TLB coverage for users of large page frames by increasing the hit rate and decreasing the number of TLB misses that an application incurs.

TLB buffer: Memory addresses that are referred to by a process are virtual addresses and require translation to the physical address. The TLB is a relatively small cache area that is used to perform this address translation.

Large pages are a performance improvement feature for some cases, but switching to large pages is not recommended for all workloads. Large pages provide performance value to a select set of applications that can generally be characterized as memory access-intensive and long-running. These applications meet the following criteria:

- They must reference large ranges of memory.
- They tend to exhaust the private storage areas that are available within the 2 GB address space (such as IBM WebSphere®), or they use private storage that is above the 2 GB address space (such as IBM DB2).

Flash memory and pageable 1 MB page frames

The zEC12 supports an optional hardware feature called *Flash Express memory cards*. These memory cards are supported in an I/O drawer with other I/O cards. The cards come in pairs for improved availability, and no HCD/IOCP definition is required. Flash memory is assigned to partitions the same way that main memory is assigned, and each partition's flash memory is isolated, similar to main memory. You can dynamically increase the maximum amount of flash memory on a partition, and you can dynamically configure flash memory into and out of the partition.

You can use flash memory to solve many different problems. Flash memory is much faster than spinning disk, but it is much slower than main memory. Flash memory takes less power to utilize than either option.

With the combination of Flash Express installed on a zEC12 and the pageable 1 MB large page frame support in z/OS V1R13, DB2 takes advantage of the large page frame support by allocating internal control blocks (PMBs) using 1 MB pageable storage. These large page frames can be paged to and from Flash Express, and performance might be improved due to a reduction in TLB misses and an increase in the TLB hit rate.

Flash memory can also be used to improve SVC dump data capture time. It removes the requirement for pageable link pack area (PLPA) and common page data sets when used for cold start IPLs.

This feature requires zEC12 (2827) hardware with Flash Express installed and z/OS V1R13 and above with requisite PTFs (FMID JBB778H). APARs PM85944 and PM90486 retrofit this feature to DB2 10 for z/OS.

2 GB large page frames

A 2 GB page frame is a memory page that is 2048 times larger than a 1 MB page and 524,288 times larger than the ordinary 4 KB base page. 2 GB large page frames allow for a single TLB entry to fulfill many more address translations than either a large page or an ordinary base page. 2 GB large page frames provide exploiters with much better TLB coverage and, therefore, potentially allow the following benefits:

- Better performance by decreasing the number of TLB misses that an application incurs
- Less time spent converting virtual addresses into physical addresses
- Less real storage used to maintain DAT structures

Note that 2 GB large pages require z/OS V2R1 and the hardware features of the zEC12.

The Buffer Manager component of DB2 uses a 2 GB frame size only when there are at least 2 GB of buffer storage to allocate and when the buffer pool is defined as long-term page fixed. For example, if you specify a small buffer pool size, such as **VPSIZE=20000**, a 2 GB frame is not used. If you specify **VPSIZE=524288** for a 4 KB buffer pool, you are requesting a buffer pool that can contain 524,288 pages that are 4 KB in size, for a total of 2,147,483,648 bytes, which is exactly 2 GB. In this case, you get exactly one 2 GB frame allocated. If you specify

VPSIZE=600000, you get one 2 GB frame, with the remainder of the buffer pool allocated in 1 MB frames up to the specified size.

For DB2 to take advantage of 2 GB large pages, the **ALTER BUFFERPOOL** command now includes the **FRAMESIZE** attribute. The valid values are 4 KB, 1 MB and 2 GB. Example 2-1 runs the **ALTER BUFFERPOOL** command to establish a page fixed buffer pool with 2 GB pages.

Example 2-1 ALTER BUFFERPOOL command to use 2 GB frame size

```
                              DB2 COMMANDS                    SSID: DB1D
===>

Position cursor on the command line you want to execute and press ENTER

Cmd 1 ===> -ALTER BUFFERPOOL(BP4) VPSIZE(600000) FRAMESIZE(2G) PGFIX(YES)
Cmd 2 ===>
Cmd 3 ===>
```

Example 2-2 shows the results of the **ALTER** command. The DSNB543I message shows that the **PGFIX** attribute is set to YES. The DSNB522I message shows that the **FRAMESIZE** attribute is set.

Example 2-2 Results of ALTER BUFFERPOOL command to change FRAMESIZE

```
DSNB522I  -DB1D VPSIZE FOR BP4 HAS BEEN SET
DSNB543I  -DB1D THE PGFIX ATTRIBUTE IS ALTERED FOR
           BUFFER POOL BP4
           CURRENT ATTRIBUTE = YES
           NEW ATTRIBUTE = YES
           THE NEW ATTRIBUTE IS IN PENDING STATE.
DSNB522I  -DB1D FRAME FOR BP4 HAS BEEN SET
DSN9022I  -DB1D DSNB1CMD '-ALTER BUFFERPOOL' NORMAL COMPLETION
***
```

To validate that the frame size was set properly and that DB2 uses a 2 GB frame, Example 2-3 issues the **DISPLAY BUFFERPOOL** command.

Example 2-3 DISPLAY BUFFERPOOL command to show 2 GB frame size

```
                              DB2 COMMANDS                    SSID: DB1D
===>

Position cursor on the command line you want to execute and press ENTER

Cmd 1 ===> -DISPLAY BUFFERPOOL(BP4) DETAIL
```

Example 2-4 shows the results of the **DISPLAY** command. Note that the preferred frame size is 2 GB. However, no buffers have yet been allocated to the 2 GB frame because no DB2 workload has been run that uses this buffer pool since altering the size.

Example 2-4 Results of DISPLAY BUFFERPOOL command showing 2 GB frame defined

```
DSNB401I  -DB1D BUFFERPOOL NAME BP4, BUFFERPOOL ID 4, USE COUNT 0
DSNB402I  -DB1D BUFFER POOL SIZE = 600000 BUFFERS  AUTOSIZE = NO
           VPSIZE MINIMUM = 0 VPSIZE MAXIMUM = 0
             ALLOCATED       =        0   TO BE DELETED   =        0
             IN-USE/UPDATED  =        0
```

```
DSNB406I  -DB1D PGFIX ATTRIBUTE -
             CURRENT = YES
             PENDING = YES
            PAGE STEALING METHOD = LRU
DSNB404I  -DB1D THRESHOLDS -
             VP SEQUENTIAL    = 80
             DEFERRED WRITE   = 30   VERTICAL DEFERRED WRT  =  5,  0
             PARALLEL SEQUENTIAL =50   ASSISTING PARALLEL SEQT=  0
DSNB546I  -DB1D PREFERRED FRAME SIZE 2G
        0 BUFFERS USING 2G FRAME SIZE ALLOCATED
```

Next, a table is created with a long row (2021 bytes in this case) that was not compressed. For this example, 3.3 million rows are inserted into the table and then a **SELECT *** is issued on the table with no **WHERE** clause to ensure that all the rows and all the columns on each row are read.

After running the SQL statements, a **DISPLAY BUFFERPOOL** command is issued again. Example 2-5 shows the results.

Example 2-5 Results of DISPLAY BUFFERPOOL command showing 2 GB and 1 MB frame allocation

```
DSNB401I  -DB1D BUFFERPOOL NAME BP4, BUFFERPOOL ID 4, USE COUNT 1
DSNB402I  -DB1D BUFFER POOL SIZE = 600000 BUFFERS  AUTOSIZE = NO
            VPSIZE MINIMUM = 0 VPSIZE MAXIMUM = 0
              ALLOCATED       =   600000   TO BE DELETED   =        0
              IN-USE/UPDATED  =        0
DSNB406I  -DB1D PGFIX ATTRIBUTE -
             CURRENT = YES
             PENDING = YES
            PAGE STEALING METHOD = LRU
DSNB404I  -DB1D THRESHOLDS -
             VP SEQUENTIAL    = 80
             DEFERRED WRITE   = 30   VERTICAL DEFERRED WRT  =  5,  0
             PARALLEL SEQUENTIAL =50   ASSISTING PARALLEL SEQT=  0
DSNB546I  -DB1D PREFERRED FRAME SIZE 2G
        524288 BUFFERS USING 2G FRAME SIZE ALLOCATED
DSNB546I  -DB1D PREFERRED FRAME SIZE 2G
        19200 BUFFERS USING 1M FRAME SIZE ALLOCATED
DSNB546I  -DB1D PREFERRED FRAME SIZE 2G
        56512 BUFFERS USING 4K FRAME SIZE ALLOCATED
```

Note that the buffer pool is defined as 600,000 buffers, which is a little more than 2 GB. Because a large enough workload was run to use more than 2 GB of buffer pool storage, DB2 allocated 524,288 pages to a 2 GB frame, which amounts to exactly 2 GB of storage. DB2 then allocated 19,200 pages to 1 MB frames, which amounts to 75 1 MB frames. The remaining 56,512 pages were allocated to 4 KB frames.

You might have expected all the storage above 2 GB to be allocated to 1 MB frames. However, DB2 does use some internal calculations to allocate what is left over after the 2 GB allocation, and it does not always come out to exactly what the system has defined. For this example, the **DISPLAY VIRTSTOR,LFAREA** command was run to see the maximum possible allocation to each frame size.

Example 2-6 shows the results of the **DISPLAY** command. In this test case, a maximum of 100 page frames can be used for a frame size of 1 MB. Based on the internal calculation, DB2

allocated 75 page frames of 1 MB each (`19200 * 4096 / 1024 / 1024`), with the remaining 56,512 pages allocated using a 4 KB frame size.

Example 2-6 D VIRTSTOR command to show the maximum allocation of 2 GB and 1 MB frames

```
D VIRTSTOR,LFAREA
IAR019I  18.51.21 DISPLAY VIRTSTOR 200
 SOURCE =  00
 TOTAL LFAREA = 100M , 2G
 LFAREA AVAILABLE = 20M , 0G
 LFAREA ALLOCATED (1M) = 80M
 LFAREA ALLOCATED (4K) = 0M
 MAX LFAREA ALLOCATED (1M) = 80M
 MAX LFAREA ALLOCATED (4K) = 0M
 LFAREA ALLOCATED (PAGEABLE1M) = 0M
 MAX LFAREA ALLOCATED (PAGEABLE1M) = 0M
 LFAREA ALLOCATED NUMBER OF 2G PAGES = 1
 MAX LFAREA ALLOCATED NUMBER OF 2G PAGES = 1
```

2.2 Synergy with IBM System z and z/OS

This section discusses interfaces that are used by DB2 11 to take advantage of the synergy potential between the System z hardware and the z/OS operating system software. There are a number of features in DB2 11 that use features in different versions of the z/OS operating system. DB2 11 takes advantage of the following features available in z/OS:

- AUTOSIZE options VPSIZEMIN and VPSIZEMAX
- 1 MB page frames for DB2 execution code
- Improved performance of batch updates in data sharing
- Improved usability and consistency for security administration
- Log writing

2.2.1 AUTOSIZE options VPSIZEMIN and VPSIZEMAX

The **AUTOSIZE** attribute of the **ALTER BUFFERPOOL** command specifies whether DB2 uses Workload Manager (WLM) services, if available, to increase the buffer pool size automatically as appropriate.

The **VPSIZEMIN** and **VPSIZEMAX** attributes have been added to the **ALTER BUFFERPOOL** command to allow more control. These attributes specify the minimum and maximum size for a buffer pool when `AUTOSIZE(YES)` is in effect beyond the DB2 increase or decrease of the buffer pool size by +/-25%. They require z/OS V2R1.

2.2.2 1 MB page frames for DB2 execution code

In z/OS V2R1, the execution code for DB2 itself can be backed by 1 MB pageable page frames. It is available only with Flash Express configured, which can result in CPU reductions that are associated with loading the code.

2.2.3 Improved performance of batch updates in data sharing

z/OS V2.1 with IBM DB2 11 for z/OS running on zEC12 or zBC12, or later systems with CFLEVEL 18, is planned to take advantage of the function to allow batched updates to be written directly to disk without being cached in the coupling facility in an IBM Parallel Sysplex®. This function can help avoid application stalls that might sometimes occur during large concurrent batch updates.

When a page set is GBP-dependent, if **GBPCACHE CHANGED** is used, both COMMITs and DEREFFED WRITEs need to write the pages to the GBP. If the I/O subsystem is slower at casting out pages from the GBP to DASD (or to a remote site) than the rate at which deferred writes are filling up the GBP, COMMITS are suspended and cast out operations can free space in the GBP. Typically the deferred writes done on behalf of batch updates are the culprit. In effect, DB2 is thrashing the coupling facility, because there is no value in having the deferred writes be written to the GBP. DB2 11 solves this situation with the support of changes to z/OS.

The z/OS support for this function is also available on IBM zEnterprise 196 (z196) and zEnterprise 114 (z114) servers with CFLEVEL 17 and an MCL, and on z/OS V1.12 and z/OS V1.13 with the PTF for APAR OA40966.

This feature is described in more detail in 5.1, “Group buffer pool write-around protocol” on page 86.

2.2.4 Improved usability and consistency for security administration

DB2 11 for z/OS is designed to improve usability and consistency for security administration. z/OS V2.1 RACF, when used with DB2 11, is designed to provide consistency between DB2 and RACF access controls for bind and rebind under an owner’s authorization identifier, RACF security exit support for declared global temporary tables (DGTT), and support for automatic authorization statement cache refreshes when RACF profiles are changed. This is intended to make DB2 security administration easier.

Details on security enhancements can be found in Chapter 10, “Security” on page 239.

2.2.5 Log writing

As a performance improvement in DB2 11, log records are written without the need to first space switch to the xxxxMSTR address space. To support this change, log buffers must be moved from their current location in xxxxMSTR 31-bit private to common storage. Because the log buffers can be large, up to 400 MB, it is not practical to move the log buffers to ECSA because most systems would not have enough ECSA available for a single request of this size. The log buffers are moved to 64-bit common (HCSA).

The amount of HCSA used is roughly the size of the log buffers specified by the OUTBUFF parameter plus 15%. The SYS1.PARMLIB setting for HVCOMMON must be large enough to accommodate this size for each DB2 11 subsystem active on an LPAR. In addition, the buffers can reside in 1 MB page frames, if available. You might want to increase the SYS1.PARMLIB setting for LFAREA to allow for this allocation.

IFCID 225 includes statistics for the common storage used by log manager buffers and control structures.

This function is enabled in conversion mode (CM).

2.3 Using zIIP speciality processors

DB2 for z/OS began using zIIP specialty processors in V8 and continued to improve total cost of ownership (TCO) by further using zIIP engines in DB2 9 and DB2 10. DB2 11 continues this trend by providing additional zIIP workload eligibility, as described in this section.

zIIP is designed to help free general computing capacity and lower software costs for select DB2 workloads. The initial DB2 implementation of zIIP was targeted towards reducing the software costs for business intelligence (BI), enterprise resource planning (ERP), and customer relationship management (CRM) workloads on the mainframe. However, non-DB2 workloads can take advantage of zIIP as well.

The amount of redirect in each case varies based on workload characteristics.

The following DB2 11 for z/OS processing is authorized to execute on zIIP:[1]

- Asynchronous processing that is executed under enclave SRBs and that will be “charged” for CPU consumption purposes to a DB2 address space (rather than to user applications), with the exception of P-lock negotiation processing

  Such zIIP eligible processing includes:

  - Cleanup of pseudo deleted index entries as part of DB2 system task cleanup
  - Cleanup of XML multi-version documents (available in DB2 10 for z/OS through APAR PM72526)
  - Log write and log read
- The DB2 base LOAD, REORG, and **REBUILD INDEX** utility processing of inline statistics collection that DB2 directs to be executed under enclave Service Request Blocks (SRBs)[2]
- The DB2 base processing of the **RUNSTATS** utility Column Group Distribution statistics collection that DB2 directs to be executed under enclave SRBs[2]
- The DB2 base **LOAD** utility index management processing when running **LOAD REPLACE** that DB2 directs to be executed under enclave SRBs[2]

From the DB2 address space point of view:

- DBM1 address space
  - System task performing clean up of pseudo-deleted index entries
  - Portions of XML multi version documents cleanup processing (also available in DB2 10 through APAR PM72526)
  - System-related asynchronous SRB processing with the exception of P-lock negotiation processing
- MSTR address space

  System related asynchronous SRB processing, such as log write or log read
- Utilities

---

[1] This information provides only general descriptions of the types and portions of workloads that are eligible for execution on IBM Specialty Engines (for example, zIIPs, zAAPs, and IFLs). IBM authorizes customers to use IBM Specialty Engines only to execute the processing of eligible workloads of specific programs expressly authorized by IBM as specified in the “Authorized Use Table for IBM Machines” provided at:
http://www.ibm.com/systems/support/machine_warranties/machine_code/aut.html
No other workload processing is authorized for execution on a Specialty Engine. IBM offers Specialty Engine at a lower price than General Processors/Central Processors because customers are authorized to use Specialty Engines only to process certain types or amounts of workloads as specified by IBM in the Authorized Use Table.

[2] DB2 does not direct all such base utility processing to be executed under enclave SRBs.

- Portions of inline statistics gathering processing during **LOAD**, **REORG**, and **REBUILD** index processing
- Portions of **RUNSTATS** column group distribution statistics processing
- The work on elimination on NPSIs during **LOAD REPLACE PART** with dummy input

Refer to the IBM documentation for software and hardware requisites for zIIP at:

http://www.ibm.com/systems/z/hardware/features/ziip/about.html

zAAP on zIIP

IBM continues to support running IBM System z Application Assist Processor (zAAP) workloads on IBM System z Integrated Information Processor (zIIP) processors (zAAP on zIIP). z/OS V2.1 is designed to remove the restriction that prevents zAAP-eligible workloads from running on zIIP processors when a zAAP is installed on the server. This support is intended to help facilitate migration and testing of zAAP workloads on zIIP processors. This support is also available with the PTF for APAR OA38829 for z/OS V1.12 and z/OS V1.13.

IBM zEnterprise EC12 is planned to be the last high-end System z server to offer support for zAAP specialty engine processors. IBM intends to continue support for running zAAP workloads on zIIP processors (zAAP on zIIP).

2.4 Reduced need for REORG

Starting in 2009, several product enhancements emerged that improved performance for disorganized index and data. These enhancements provided less need to run expensive DB2 REORGs. DB2 11 continues this progress towards reducing the need for REORGs. This section reviews what has happened since 2009 and explains features in DB2 11 that help to further reduce the need for REORG.

In 2009 IBM and other vendors began to offer solid-state disks (SSD) for enterprise storage. SSD has no mechanical seeks or rotational delays that are associated with disorganized data, enabling the device to efficiently stream the data no matter how the data is organized. Random pages are still not streamed as fast as sequential pages, but the performance gap between random and sequential data is significantly reduced.

In 2011 IBM delivered High Performance FICON® (zHPF) support for DB2 list prefetch with its IBM System Storage® DS8000® Licensed Machine Code (LMC) level R6.2. This support also requires a z196 or zEC12 processor. In addition, IBM delivered FICON Express 8S channels for these two processors. FICON Express 8S is optimized for zHPF. Using zHPF, FICON Express 8S can read discontiguous pages faster.

R6.2 also introduced List Prefetch Optimizer to optimize the fetching of data from disk when DB2 is using list prefetch. List Prefetch Optimizer requires zHPF. List Prefetch Optimizer is optimal for both random pages (as is the case with a disorganized index scan) and skip-sequential scans (as is the case with a sorted RID list). List Prefetch Optimizer is especially good in conjunction with solid-state disks. For details, see *GPFS in the Cloud: Storage Virtualization with NPIV on IBM System p and IBM System Storage DS5300*, REDP-4682.

Furthermore, R7.2 recently delivered Flash Optimized Offering for the DS8870. It is expected that some disorganized index scans might benefit from this support when using SSD. See the recent announcement at:

http://www-01.ibm.com/common/ssi/cgi-bin/ssialias?infotype=AN&subtype=CA&htmlfid=897/ENUS113-174

The price of SSD is rapidly coming down and gaining market share. As cost reduction happens, DB2 10 and 11 are well positioned to take advantage of this hardware to further reduce the need for REORGs.

2.5 DFSMS storage tiers

The classic z/OS DFSMS storage hierarchy involves a three-level hierarchy. Data that is regularly accessed is maintained on a fast drive Primary Level (Level 0) that is managed by DFSMShsm. When the data is no longer accessed regularly, it is migrated to Migration Level 1 (ML1), which is typically on a less expensive disk drive. When the data has not been accessed for a longer period of time, then it is migrated to Migration Level 2 (ML2), which is typically on tape.

Figure 2-1 shows an example of the classic three-level storage hierarchy.

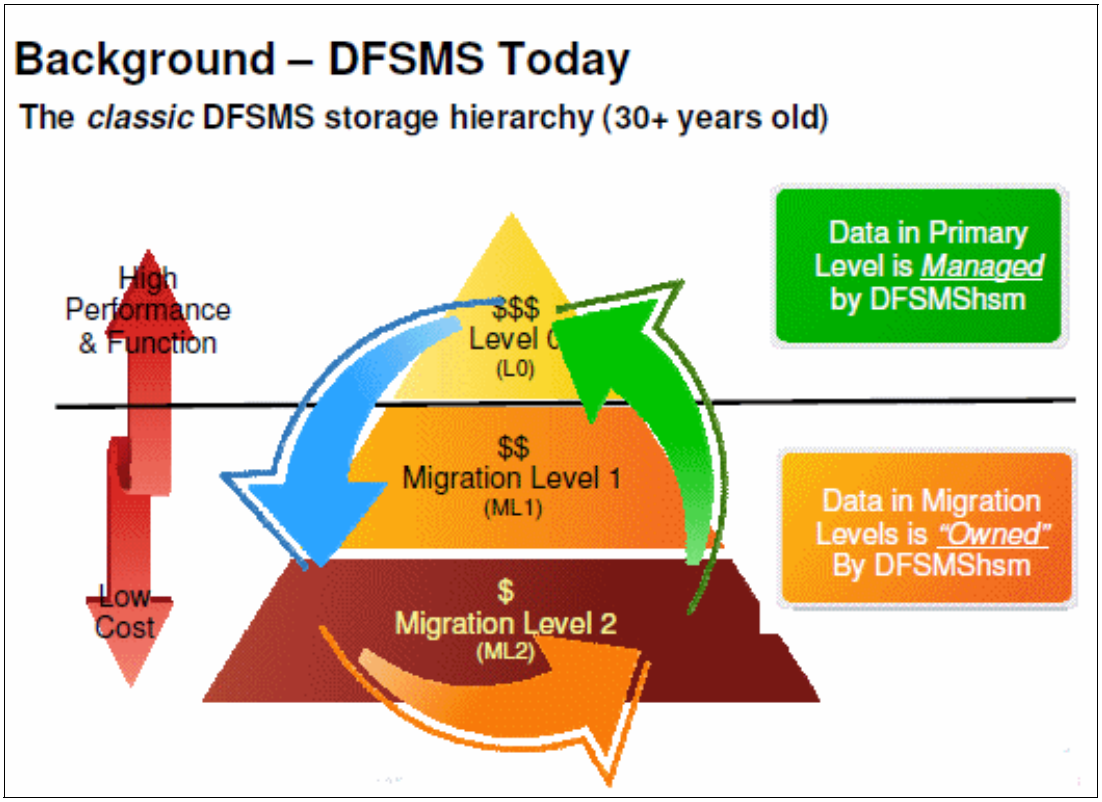


Figure 2-1 The classic DFSMS storage hierarchy

Over the years, typical configurations have changed to leave data on Level 0 longer and then migrate directly to ML2, bypassing ML1. When ML2 is a Virtual Tape Server (VTS), then VTS disk cache replaces the ML1 tier. The VTS disk cache implementation provides the following savings:

- Eliminates MIPS required for software compression to ML1
- Eliminates DFSMShsm ML1 to ML2 processing

Although the classic DFSMS storage hierarchy provides these benefits, this storage management solution also have the following shortcomings:

- There is no policy-based automation for moving data within the Primary Storage Hierarchy (Level 0)
- There is no policy-based management of Active (open) data

z/OS V2R1 DFSMS introduces a storage tiers solution. This solution provides an automated, policy-based space management that moves data from tier to tier within the Primary (Level 0) Hierarchy. The storage tiers solution provides the following benefits:

- It better aligns storage costs with changing business value.
- It minimizes the TCO for System z data by actively managing data on the lowest cost storage that meets the business needs of the data.

Within the storage tiers solution, automated movement of data is provided through the existing DFSMShsm Space Management function. Movement is referred to as a *class transition*. The data that is moved remains in its original format and can be accessed immediately after the movement is complete.

The storage tiers solution replaces ML1 with a Nearline level, which represents data that is not at the Enterprise (Level 0) level but still needs to be accessed relatively quickly. That is, it is not “hot” data, but it is not “frigid” data either. The data is either cool or cold, which means it is not accessed that frequently. The storage tiers solution allows this “cool” data to be transitioned from the Enterprise level (Level 0) to the Nearline level (Level 1) after some specified period of time. The data that is stored in the Nearline level is still stored on DASD and is still immediately accessible. The data is just transitioned to a different class of storage.

Figure 2-2 shows an overview of the storage tiers solution.

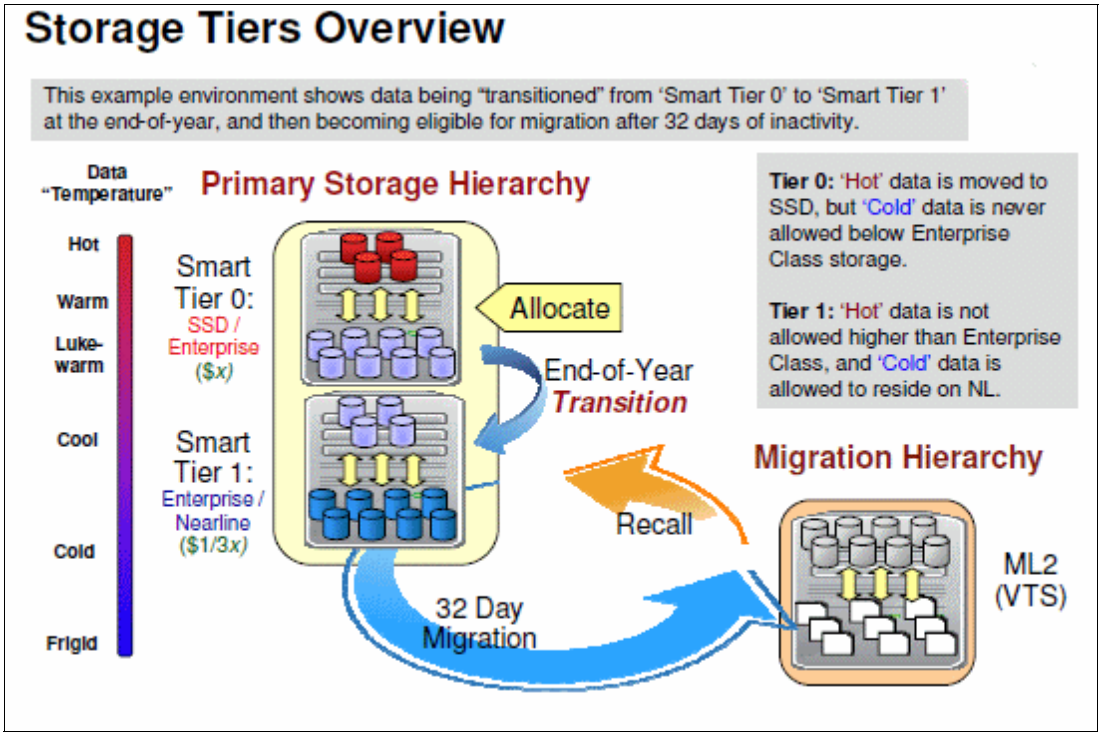


Figure 2-2 Storage tiers overview

The critical enterprise data, or “hot” data as it is sometimes called, is stored on Enterprise Level storage, or tier 0. The less critical, but still regularly accessed, data is stored on Nearline Level storage, or tier 1. Data on the Nearline level that is not accessed in 32 days is migrated to ML2 storage.

2.5.1 Use cases for storage tiers

The following data set examples benefit from the storage tiers solution.

One case that benefits is where the data sets are not currently eligible for migration because they always need to be immediately accessible. In this situation, a delay while waiting for the data set to be recalled is unacceptable. These data sets can be allocated on a particular class of storage and then later transitioned to a less expensive class of storage for permanent retention.

A second case that might benefit is when there are data sets that are eligible for migration today, but there is a benefit to keeping them online for a longer period of time. In this case, it makes sense to convert the migration of data sets to transition to a lower cost storage and then to increase the number of days that the data sets must be unreferenced before migrating directly to ML2.

Note that there is a difference between the HSM migrate/recall functions and class transitions. When a data set is recalled, it is returned to the class of storage as directed by the automatic class selection (ACS) routines, which typically is higher than where a data set resides after a transition. When a data set transitions to a lower class of storage, it remains there until it is transitioned again or until it migrates.

2.5.2 Setup and invocation of storage tiers

DFSMShsm Space Management processing uses policy-based automation to ensure that volumes within the Primary Storage Hierarchy have enough free space for new data and to ensure that data is stored at the lowest acceptable tier in the Storage Hierarchy. This function is accomplished through the following processes:

- Data set expiration
- Migration of unreferenced data to the Migration Hierarchy
- “Class Transitions” within the Primary Hierarchy

Class Transition processing is new and is used by the storage tier solution. This processing is integrated into the following existing DFSMShsm Space Management functions:

- Primary Space Management
- On-Demand Migration, which is a new function introduced in V1R13

  This function performs space management on a volume as soon as it goes over its high threshold. It is a replacement for on-the-hour Interval Migration processing.
- Interval Migration

  When a volume is selected for space management processing due to being over a threshold, in addition to existing expiration and migration checking, space management functions determines if a data set is eligible to be transitioned, based on management class criteria.

SMS Management Class

The SMS Management Class provides the Class Transition policies, which include the following components:

- Class Transition Criteria
- Serialization Error Exit
- Transition Copy Technique

Each of these policies is discussed in more detail in the sections that follow.

Class Transition Criteria

This criteria determines if and when a data set should be transitioned. This criteria includes information about how long since the data set was created and how long since the data set was used. In addition, there is a periodic setting that specifies that a data set should be transitioned monthly, quarterly, or annually, regardless of the usage of the data set.

Serialization Error Exit

This exit indicates what type of special processing occurs if the data set cannot be serialized, meaning that the data set is open and it cannot be moved. The following serialization error exit setting options are available:

- NONE
- DB2
- CICS
- zFS
- EXIT

If the setting is DB2, the exit invokes DB2 to close and unallocate the object. If this operation is successful, the object is serialized and moved, and DB2 is invoked to reopen the object. For DB2 data, the data set can always be open, and special processing might be needed to transition the data at any time. Because it is expected that data sets can be open, the default is to *not* issue an error message if a data set cannot be exclusively serialized; it is just skipped, which is similar to migration processing.

Transition Copy Technique

This technique setting indicates which copy technique is used to move the data set. The following techniques are available:

- *Standard* uses standard I/O, which is the default.
- *Fast Replication Preferred* prefers Fast Replication. If it cannot be used, standard I/O is used.
- *Fast Replication Required* requires Fast Replication. If it cannot be used, fail the data movement. This technique requires the target volume to be in the same storage controller.
- *Preserve Mirror Preferred* prefers to use Preserve Mirror. This technique indicates that a Metro Mirror primary volume is allowed to become an IBM FlashCopy® target volume. If Preserve Mirror cannot be used, FlashCopy or standard I/O can be used.
- *Preserve Mirror Required* requires Preserve Mirror. The transition is performed only if the Metro Mirror primary target volume does not go duplex pending. This parameter has no affect if the target volume is not a Metro Mirror primary volume.

Storage Group Processing Priority

In addition to the class transition policies, the new Storage Group Processing Priority specifies the relative order in which storage groups are processed during Primary Space Management. To help ensure that the “receiving” storage groups have enough space for the

data sets that are moved to them, a new storage group Processing Priority is provided. These storage groups are assigned a higher priority. Storage Groups are processed in the order of their priority. A higher value means a higher priority. The valid values are 1 to 100, with a default of 50.

After DFSMShsm determines that a data set has met the Class Transition criteria specified by the Management Class, it invokes the ACS routines to determine what the transition should be. The ACS routines are invoked with the new ACS environment (the `&ACSENVIR` variable) value of `SPMGCLTR`, for "space management class transition." The following ACS routines are invoked in the order shown:

- Storage Class
- Management Class
- Storage Group

Any or all policies can be transitioned.

The Storage Class indicates the "preferred" class of storage to which the data set is allocated. If the storage class changes but the storage group remains the same and if a device matching the new storage class attributes cannot be selected, the data set is not moved.

When a new management class is assigned, DFSMShsm begins using the newly assigned policies to manage the data set. If only the management class changes, the data set is altered to assign it to the new management class, and no data movement is performed.

During processing of the Storage Group routine, from 1 to 15 storage groups can be returned. The storage administrator ensures that a different storage group name provides a meaningful transition.

When DFSMShsm determines that a data set should be moved for a Class Transition, DFSMSdss is invoked to perform a Logical COPY with the **`DELETE`** command. In this case, DFSMSdss is the full data mover, unlike migrate/recall and backup/recover processing where DSS is only the half data mover. DFSMSdss handles the Copy Technique and Exit processing. After the movement, the data set retains all existing attributes and can be immediately accessed.

The ICF catalog is updated as a part of the movement. No new DFSMShsm control data set records are created for transitions; however, new functional statistics record (FSR) type 24 is created for reporting purposes.

2.5.3 Use cases for DB2

Possible use cases for storage tiers for DB2 data are cases where the data is partitioned and the data is date or time dependent and the latest data is always added to the end. For example, if a table is defined as partitioned by range (PBR) and the partitioning key is defined as a date or time stamp, you can design it such that each partition held one month's worth of data. If the most frequently accessed data is data within the last 60 days, you can set up storage tiers such that the partition that contains the current month's data is on Primary storage, and the two partitions that contain the prior two month's data are on Nearline storage. All data prior to those months are on Migration Level 2 storage.

A similar scenario can be made for partition by growth (PBG) table spaces, with the assumption that newly added partitions contain the data for which there is the most interest.

2.6 Additional System z enhancements

The following additional enhancements to the System z hardware and software platform also provide benefits to DB2 for z/OS.

2.6.1 Enhancing DB2 BACKUP SYSTEM solution

DB2 11 enables recovery of single pageset from DB2 system-level backup even if original volume does not have sufficient space and enables exploitation of FlashCopy consistency group for DB2 BACKUP SYSTEM. It also enables restoration of a pageset to a different name.

FRBACKUP COPYPOOL with consistency allows you to create a backup of the log copypool with consistency. Prior to DB2 11, you need a conditional restart of DB2 with a log truncation point that corresponds to the data complete LRSN of the system-level backup. The conditional restart is needed to compensate for the fuzziness of the backup of the log copypool. If the backup of the log copy pool is taken with consistency, you no longer need to do a conditional restart of DB2.

You can use the FlashCopy Consistency Group function to minimize application impact when making consistent copies of data spanning multiple volumes. The procedure consists of freezing the source volume during each volume copy operation and thawing the frozen volumes using the **CGCREATED** command after a FlashCopy Consistency Group is formed. During the time period between the first and the last volumes are frozen, no dependent write updates occur, which allows a consistent copy of logically related data that spans multiple volumes.

2.6.2 z/OS DFSMS VSAM RLS for z/OS catalog support

In a Parallel Sysplex environment, z/OS V2.1 extends support for the VSAM record-level sharing (RLS) environment to catalogs to allow improvements to both single-system and shared catalog performance.

DB2 9 and above can see improved DB2 data set open/close performance.

2.6.3 DDF Synchronous Receive support

DB2 10 currently uses Asynchronous Receive, which requires extra SRB dispatching. DB2 11 uses the z/OS 1.13 Communication Server services for synchronous receive. The benefits are reduced CPU for DIST address space, especially for high performance DBATs or long running transactions.

No application changes or binds are required.

2.6.4 zEnterprise Data Compression

zEnterprise Data Compression (zEDC) for z/OS V2.1, a priced optional feature of z/OS that runs on zEC12 and zBC12 systems with the zEDC Express adapter, is designed to support a new data compression function. This facility is designed to provide high-performance, low-latency compression without significant CPU overhead. Initially, z/OS allows you to specify that SMF data written to log streams be compressed, which is expected to reduce disk storage requirements for SMF data and reduce SMF and System Logger CPU

consumption for writing SMF data. For more information about this function, see *Subsystem and Transaction Monitoring and Tuning with DB2 11 for z/OS*, SG24-8182.

Further support for zEDC is also planned. Corresponding support in the SMF dump program IFASMFDL is designed to support both hardware-based and software based decompression, and software-based decompression support is available on z/OS V1.12 and z/OS V1.13 with the PTF for APAR OA41156. This function allows higher write rates for SMF data when hardware compression is enabled. IBM RMF™ support for hardware compression includes SMF Type 74 subtype 9 records and a Monitor I PCIE Activity report that provides information about compression activity on the system.

In addition, plans are to make the BSAM and QSAM access methods available by the end of the first quarter of 2014. These functions can help you save disk space, improve effective channel and network bandwidth without incurring significant CPU overhead, and improve the efficiency of cross-platform data exchange.

Plans are also to provide support for DFSMSdss to take advantage of zEDC by the end of the third quarter 2014. This function is designed to be available for dumping and restoring data and also when DFSMShsm uses DFSMSdss to move data. This function can provide efficient compression with lower CPU overheads than the processor- and software-based compression methods currently available.

3

Scalability

Business resiliency is a key component of the value proposition of DB2 for z/OS and the System z platform, supporting your efforts to keep your business running even when the workload keeps growing or you need to make changes. DB2 11 innovations drive new value in resiliency through scalability improvements and fewer outages—planned or unplanned.

The most important scalability enhancement delivers the ability to switch to a longer log RBA and LRSN value. This allows for logging capability that should cover you for the next decades even with growth at more than historic rates.

This chapter describes the following:

- Extended RBA and LRSN
- NOT LOGGED for declared global temporary tables
- More open data sets (DSMAX)
- PBG mapping tables to lift the 64 GB limit

3.1 Extended RBA and LRSN

Since the initial version of DB2, the relative byte address (RBA) of log records has been 6 bytes. This size provides 256 TB of log record addressing capacity over the life of a DB2 non-data sharing subsystem. In DB2 environments with high logging volumes, there have been several cases where users have exhausted DB2's logging capacity. When this happens, DB2 initially issues DSNJ032E and DSNJ033E warning messages and eventually terminates if you do not address the situation.

The following sections describe the current, basic format, RBA, and the enabling of the DB2 11, extended format RBA/LRSN:

- Reaching the end of the basic RBA
- The new 10 byte RBA and LRSN
- Considerations before converting to extended format
- Steps for enabling the extended RBA/LRSN format
- Converting the BSDS
- NOT LOGGED for declared global temporary tables
- Converting data from 6 byte to 10 byte RBA/LRSN or vice versa
- Additional considerations regarding utilities

3.1.1 Reaching the end of the basic RBA

When the end of the log RBA range is reached, you must take manual recovery actions to reset the RBA back to zero.

You also need to reset the PGLOGRBA values in every page for all objects. This change necessitates an extended outage.

For data sharing, the process is less intrusive, and involves shutting down the affected member and starting a new member to take its place. Data sharing is less intrusive because PGLOGRBA contains an log record sequence number (LRSN), which is a 6-bite value derived from the TOD clock value to be used for all members of the data sharing group. This value will not run out until the year 2042.

Data sharing customers who have disabled and re-enabled data sharing can have LRSN values "in the future" with respect to the TOD clock. This is, because during the re-enable process, DB2 records an LRSN "delta" value in the BSDS which gets added to the TOD clock value to derive the LRSN.

Another way that you end up with an LRSN value that is "in the future" is when you enable data sharing for a subsystem that has a RBA value that is numerically greater than the current TOD clock value. This might be done when a non data sharing system is approaching the end-of-range of its RBA, so enabling data sharing can yield a few extra years before the problem must be dealt with.

Therefore, some customers will run out of LRSN range well before 2042.

Note: The LRSN delta only occurs if the RBA of the originating member is higher than the STCK value at the moment data sharing is turned on. The determined delta is common to all members of a data sharing group. Its purpose is to ensure that an LRSN value is always larger than any of the RBAs used on any member. This delta is kept in each member's BSDS and in the SCA.[a]

a. The shared communications area (SCA) is a list structure in the coupling facility.

Figure 3-1 shows an example of how an LRSN delta is calculated and applied to find the real LRSN value that is associated with the various log records. The example also shows how the LRSN of this system would progress from August 5, 2013 to April 04, 2018 due to the necessary delta.

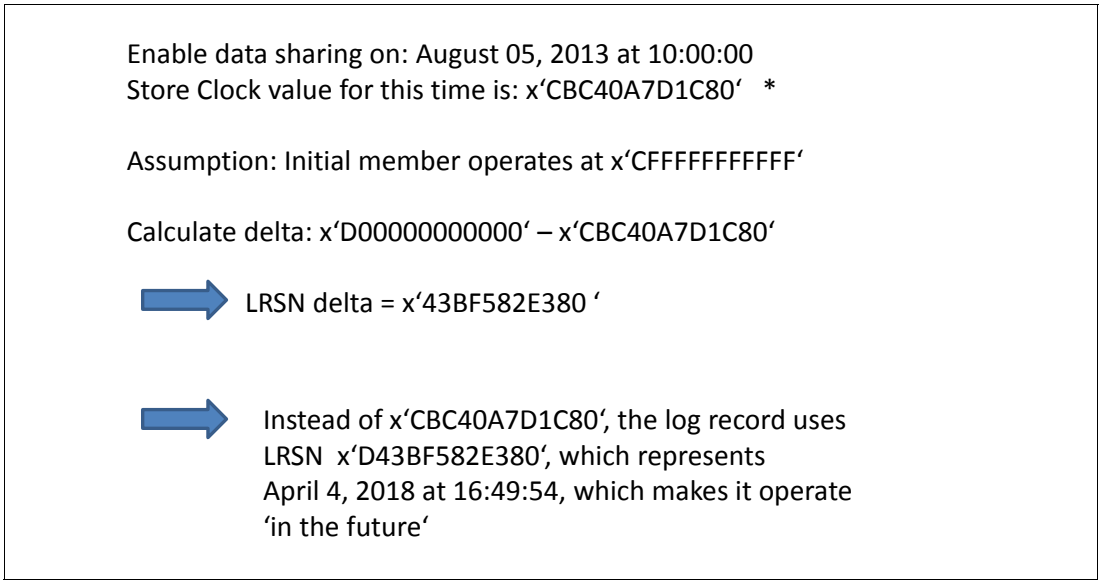


Figure 3-1 LRSN delta explanation

Important: If you use a tool, such as the IBM z/OS Store Clock converter, keep in mind that you have to add two bytes x'0000' to the LRSN value that you see.

The 6-byte LRSN provides a granularity of 16 microseconds, On faster machines, many consecutive log records can have duplicate LRSN values. DB2 9 and DB2 10 provide enhancements so that DB2 no longer has to 'spin' in the Log Manager to avoid duplicate LRSNs for most cases, but there are still some cases where CPU spinning is necessary. This adds considerable overhead. Extending the LRSN to use more of the TOD clock precision eliminates the need to "spin" to obtain a unique value, which improves data sharing performance.

To address these problems, the RBA and LRSN are being expanded to 10 bytes.

The LRSN is expanded allowing for over five orders of magnitude greater granularity and over 30,000 years before the end of range is encountered. The 10 byte RBA value should be large enough that it will take many decades to exhaust even if hardware speeds continue to increase at their historic rates.

3.1.2 The new 10 byte RBA and LRSN

RBA and LRSN values have both been extended to 10 bytes from the previous six byte size. For an RBA, a six byte value is converted to a ten byte value by adding zeroes to the four most significant bytes, to the left side of the value.

The new RBA limits

For an LRSN, the existing value has one zero byte added to the left side and three bytes added to the right side. The three bytes on the right side can be zero or `x'FF'`, depending on the usage. In particular, zero padding applies to new log records that are generated, and `x'FF'` padding is used for existing LRSN records that are generated. Figure 3-2 visualizes this example.

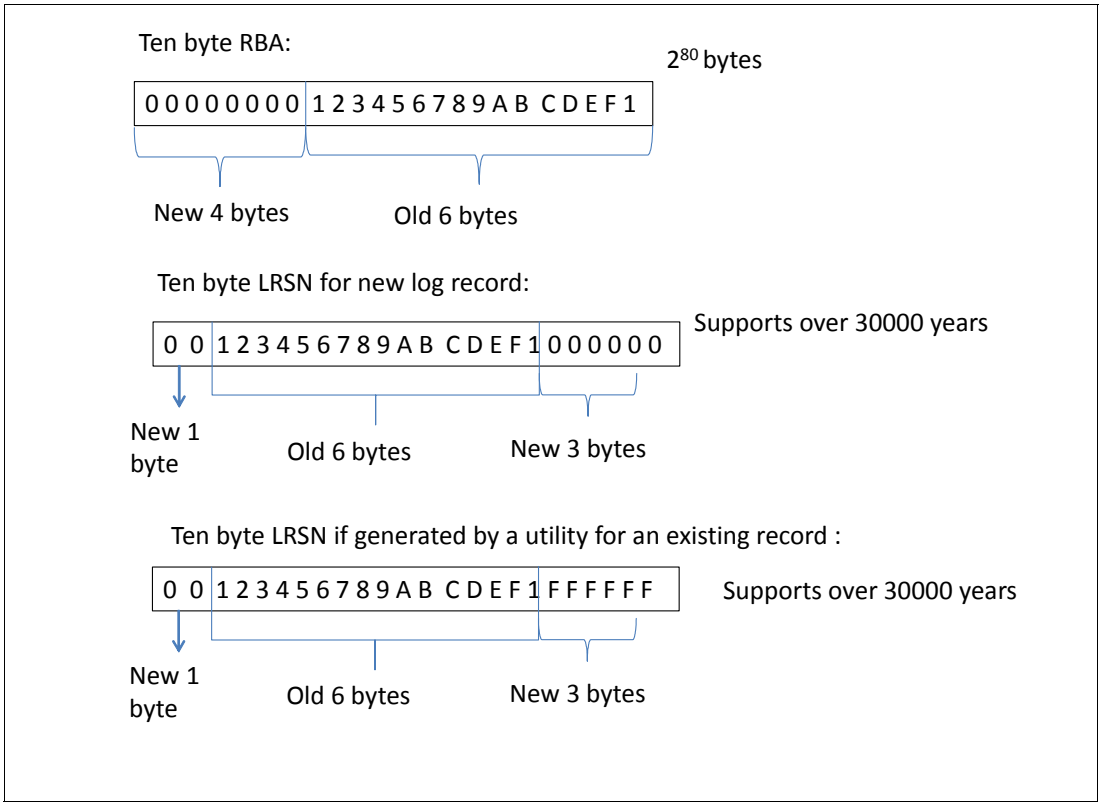


Figure 3-2 10-byte RBA/LRSN formats

Even if you do nothing with regards to larger RBAs and LRSNs, when starting DB2 11 in conversion mode (CM), these longer records are displayed everywhere in DB2.

As shown in Example 3-1, which lists the output of a **`-DIS GROUP`** command, subsystem DB0B is currently running in DB2 11 CM.

Example 3-1 Output of DISPLAY GROUP command

```
DSN7100I  -DB0B DSN7GCMD
 *** BEGIN DISPLAY OF GROUP(........) CATALOG LEVEL(111) MODE(CM  )
                   PROTOCOL LEVEL(2)  GROUP ATTACH NAME(....)
 ------------------------------------------------------------------------
 DB2                                    DB2 SYSTEM    IRLM
 MEMBER   ID  SUBSYS CMDPREF   STATUS   LVL NAME      SUBSYS IRLMPROC
 -------- --- ----   --------  -------- --- --------  ----   --------
 ........   0 DB0B   -DB0B     ACTIVE   111 SC63      IDOB   DB0BIRLM
 ------------------------------------------------------------------------
 SPT01 INLINE LENGTH:        32138
 *** END DISPLAY OF GROUP(........)
 DSN9022I  -DB0B DSN7GCMD 'DISPLAY GROUP ' NORMAL COMPLETION
***
```

All messages, log records, and so on are already handling the RBA in the 10 byte, the *extended format*.

Example 3-2 shows the longer RBA values in the messages of the starting MSTR address space.

Example 3-2 Ten byte RBA in MSTR in CM

```
.  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
   Display  Filter  View  Print  Options  Search  Help
 ------------------------------------------------------------------------------
 SDSF OUTPUT DISPLAY DBOBMSTR STC03760  DSID     2 LINE 26      COLUMNS 21- 100
 COMMAND INPUT ===>                                            SCROLL ===> CSR
DSNY001I  -DBOB SUBSYSTEM STARTING
DSNJ127I  -DBOB SYSTEM TIMESTAMP FOR BSDS= 13.190 06:32:16.83
DSNJ001I  -DBOB DSNJW007 CURRENT COPY 1 ACTIVE LOG  425
DATA SET IS DSNAME=DBOBL.LOGCOPY1.DS01,
STARTRBA=0000000000002C4C0000,ENDRBA=0000000000002E67FFFF
DSNJ001I  -DBOB DSNJW007 CURRENT COPY 2 ACTIVE LOG  426
DATA SET IS DSNAME=DBOBL.LOGCOPY2.DS01,
STARTRBA=0000000000002C4C0000,ENDRBA=0000000000002E67FFFF
DSNJ099I  -DBOB LOG RECORDING TO COMMENCE WITH  427
STARTRBA=0000000000002C539000
S DBOBDBM1
S DBOBDIST
DSNR001I  -DBOB RESTART INITIATED
DSNR003I  -DBOB RESTART...PRIOR CHECKPOINT RBA=0000000000002C5339EE
DSNR004I  -DBOB RESTART...UR STATUS COUNTS  442
IN COMMIT=0, INDOUBT=0, INFLIGHT=0, IN ABORT=0, POSTPONED ABORT=0
DSNR005I  -DBOB RESTART...COUNTS AFTER FORWARD  443
  F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=IFIND     F6=BOOK
  F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
.....
```

Example 3-3 shows an extract of the archive log. The RBA and LRSN hexadecimal values appear in the expanded format.

Example 3-3 Log record in CM

```
DSN1LPRT UR  CONNID=DBOB      CORRID=021.OPNLGR00  AUTHID=SYSOPR    PLAN=SYSTEM
    START DATE=00.162 TIME=21:40:02  DISP=COMMITTED           INFO=COMPLETE
    STARTRBA=0000000000003DB1827A   ENDRBA=0000000000003DB18658
    STARTLRSN=00CBC49885F1B4000000  ENDLRSN=00CBC49885F1D6000000
    NID=* LUWID=USIBMSC.SCPDBOB.CBC49885F1A9.0001
    COORDINATOR=* PARTICIPANTS=*
    DATA MODIFIED:
         DATABASE=0001=DSNDB01   PAGE SET=00CF=SYSLGRNX
         DATABASE=0001=DSNDB01   PAGE SET=0087=DSNLLX01
         DATABASE=0001=DSNDB01   PAGE SET=0086=DSNLLX02
```

You see all these extended format RBAs and LRSNs but you are not yet actually using its functionality.

The new page format for larger RBA and LRSN

Because the RBA for non-data sharing or the LRSN for data sharing that correspond to the last logged update to a page (table or index page) are stored in a fixed six byte location in

each page (**PGLOGRBA**), the format of the pages needs to change to accommodate the new, larger value. This format is supported in New Function Mode (NFM), ENFM*, and CM*, that is, any mode that does not support coexistence with or fall back to an earlier release.

The terms *basic format* and *extended format* refer to objects that are in the 6 byte and 10 byte **PGLOGRBA** formats, respectively. An object can be converted to and from **EXTENDED** format by using the **REORG**, **REBUILD**, or **LOAD REPLACE** utilities.

Refer to Example 3-4 for the changed layout of the header page.

Example 3-4 DSN1PRNT of a header page in extended format

```
PAGE: # 00000000
-------------------------------------------------------------------------------------------------
 HEADER PAGE:  PGCOMB='10'X  PGBIGRBA='0000000000014AC0F7C7'X  PGNUM='00000000'X  PGFLAGS='38'X
               HPGOBID='01340005'X  HPGHPREF='00000002'X  HPGCATRL='00'X  HPGREL='D7'X  HPGZLD='00'X
               HPGCATV='00'X  HPGTORBA='000000000000'X  HPGTSTMP='20130725163247762845'X
               HPGSSNM='DB1A'  HPGFOID='0004'X  HPGPGSZ='1000'X  HPGSGSZ='0004'X  HPGPARTN='0000'X
               HPGZ3PNO='000000'X  HPGZNUMP='00'X  HPGTBLC='0001'X  HPGROID='0006'X
               HPGZ4PNO='00000000'X  HPGMAXL='0059'X  HPGNUMCO='000E'X  HPGFLAGS='0008'X
               HPGFLAGS2='00'X  HPGFLAGS3='80'X  HPGCONTM='20130807162852039670'X
               HPGSGNAM='SYSDEFLT'  HPGVCATN='DB1AD   '  HPGRBRBA='000000000000'X
               HPGLEVEL='000000000000'X  HPGPLEVL='000000000000'X  HPGCLRSN='000000000000'X
               HPGSCCSI='0025'X  HPGDCCSI='0000'X  HPGMCCSI='0000'X  HPGDSSZ='00200000'X
               HPGFLAG2='00'X  HPGEPOCH='0000'X  HPGRBLP='000000000000'X  HPGDNUMB='1F'X
               HPGDNUMC='0007'X  HPGDFSG='00000000'X  HPGDLSG='00000000'X  HPGSISP='00000000'X
               HPGBIGTORBA='00000000000000000000'X  HPGBIGRBRBA='0000000000014ABD9979'X
               HPGBIGLEVEL='00000000000000000000'X  HPGBIGPLEVL='00000000000000000000'X
               HPGBIGCLRSN='0000000000014AC0F7C7'X  HPGBIGRBLP='00000000000000000000'X  FOEND='52'X
```

For your convenience, Example 3-5 also shows the header page of a table space that is still in basic format.

Notice the difference highlighted in blue in both outputs. The extended format header page has seven new fields at the bottom of the page, which all contain the three letters BIG in the field names. In addition, **PGLOGRBA** is changed to **PGBIGRBA**, which can now hold the 10 byte RBA.

Also notice that the **FOEND** field at the end of the page contains a character if the page set is still stored and worked within basic format. If it is converted to extended format, it contains a hexadecimal value.

Example 3-5 DSN1PRNT of a header page in basic format

```
PAGE: # 00000000 -------------------------------------------------------------------------------
 HEADER PAGE:  PGCOMB='00'X  PGLOGRBA='0000044F9766'X  PGNUM='00000000'X  PGFLAGS='38'X
               HPGOBID='01120002'X  HPGHPREF='0000001A'X  HPGCATRL='00'X  HPGREL='D6'X  HPGZLD='00'X
               HPGCATV='00'X  HPGTORBA='000000000000'X  HPGTSTMP='20110428113926841184'X
               HPGSSNM='DBOB'  HPGFOID='0001'X  HPGPGSZ='1000'X  HPGSGSZ='0004'X  HPGPARTN='0000'X
               HPGZ3PNO='000000'X  HPGZNUMP='00'X  HPGTBLC='0002'X  HPGROID='0000'X
               HPGZ4PNO='00000000'X  HPGMAXL='0000'X  HPGNUMCO='0000'X  HPGFLAGS='0008'X
               HPGFLAGS2='00'X  HPGFLAGS3='00'X  HPGCONTM='20110428114327048498'X
               HPGSGNAM='DSN8G100'  HPGVCATN='DBOBD   '  HPGRBRBA='0000044FBDE4'X
               HPGLEVEL='0000044FBDE4'X  HPGPLEVL='000000000000'X  HPGCLRSN='0000044F9766'X
               HPGSCCSI='0025'X  HPGDCCSI='0000'X  HPGMCCSI='0000'X  HPGDSSZ='00200000'X
               HPGFLAG2='00'X  HPGEPOCH='0000'X  HPGRBLP='000000000000'X  HPGDNUMB='1F'X
               HPGDNUMC='0007'X  HPGDFSG='00000000'X  HPGDLSG='00000000'X  HPGSISP='00000000'X
               FOEND='E'
```

Other types of pages, such as data or dictionary pages, are different but have the same BIG fields. You can search for **PGBIG** or simply check out the **FOEND** information. If **FOEND** appears in hexadecimal, you are currently looking at a page set that has been converted to **EXTENDED** format.

3.1.3 Considerations before converting to extended format

Partitioned tables and indexes can be converted one partition at a time. You do not need to convert all partitions at the same time, except for PBG hashed table spaces that require the entire table to be **REORG**ed when being converted. XML tables with 8 byte version IDs must also have all partitions converted at the same time. Also, **CLONE** tables cannot be converted until the clone is dropped. The **SYSINDEXPART** and **SYSTABLEPART** catalog tables include an indicator of what format the partition is expected to be in.

This value is not guaranteed to be accurate because **DSN1COPY** can cause the format to change without updating the catalog. Therefore, this value is for informational purposes only, such for determining what objects might or might not have been converted to the new format. You can use the **REPAIR** utility to correct the format column, if it is incorrect. Utilities that change the format of the object also update the catalog columns.

The soft and hard limit

There are two logging limits that impact SQL and utility processing. These log limits are expressed as RBAs in non-data sharing and as LRSNs time-derived values in data sharing. For data sharing, the RBA value triggers warning messages from the log manager, but it does not affect utilities behavior because the LRSN value is used instead of the RBA value in database objects.

Warning messages already existed in DB2 10 for a subsystem that is approaching the end of its RBA range. When the subsystem reaches the critical threshold, the subsystem is terminated and can be restarted only in LIGHT mode or with **ACCESS(MAINT)**. The same applies when approaching the end of the LRSN range.

A warning message is issued starting approximately one year before the LRSN range will be exhausted and is reissued for every log switch. At about two months before the end of the LRSN range, the soft limit processing begins (as described in the next section). At about two weeks before the end of the LRSN range, the subsystem terminates and can be started only in LIGHT mode or with **ACCESS(MAINT)** until the BSDS is converted. If the BSDS is not converted to the new format and if the LRSN has exceeded the 6 byte maximum, the subsystem is not allowed to restart until the BSDS is converted.

The soft limit

This limit occurs at RBA `'FFF800000000'x` or at an LRSN approximately two months before the 6 byte capacity is exhausted. This *advisory limit*, also known as the *soft limit*, marks the beginning of new utility behaviors for any utility that updates a **BASIC** format object. The new behaviors at this limit are as follows:

- Objects in basic 6 byte format are available for read-only access. Attempts to update these objects are rejected. If you need to update table spaces and indexes that have reached the soft limit, you need to convert them to extended 10 byte format.
- As a general rule, with exceptions noted in the final item in this list, utilities that attempt to log an update of a BASIC page set fail with a DSNT500-style resource unavailable error and the `'00C2026D'x` reason code.

- The following catalog table spaces, and their indexes, are heavily used by utilities. If these table spaces are in BASIC format at the soft limit, utilities are significantly impacted and in many cases unusable.
  - DSNDB01.SYSUTILX
  - DSNDB06.SYSTSCPY
  - DSNDB01.SYSLGRNX

  Important: When your DB2 subsystem hits the soft limit and the DSNDB06.SYSTSCPY table space is still in **BASIC** format, you cannot run the **RECOVER** utility for any table space any longer. The **RECOVER** command always records its execution in SYSIBM.SYSCOPY, which is stored in this table space.

- If you cannot use a utility while in this phase but if this utility did not change the format of the page set from **BASIC** to **EXTENDED** (that is, left it in **BASIC** format), the page set is available only for read-only access.
- The following utilities are not affected by the restrictions that are put in place after the soft limit is reached:
  - Utilities that do not have a target object, such as **STOSPACE**
  - Utilities that do not update the target object, such as **REPORT**
  - Utilities that are in the process of converting page sets to the **EXTENDED** format
  - Utilities that are invoked against **EXTENDED** format page sets
  - Utilities that open the output object as non-recoverable

The hard limit

The actual logging limit occurs when the RBA or LRSN no longer fits in 6 bytes. At this time, the soft limit restrictions remain in place. In addition, you must convert the BSDS to start DB2. If BSDS is not converted, the attempt to start DB2 fails. As a consequence, you cannot use any online utilities.

New subsystem parameters for extended RBA

The following subsystem parameters are related to the extended RBAs and LRSNs when dealing with DB2 objects:

- **OBJECT_CREATE_FORMAT**
- **UTILITY_OBJECT_CONVERSION**

OBJECT_CREATE_FORMAT

The **OBJECT_CREATE_FORMAT** subsystem parameter specifies whether DB2 creates new table spaces and indexes to use a basic or extended log record format.

The acceptable values are **BASIC** and **EXTENDED**. The default value depends on whether a subsystem is newly installed in DB2 11 or migrated from DB2 10. The default value is **EXTENDED** for the newly installed DB2 and **BASIC** for migration.

- **BASIC**

  New table spaces and indexes are created with a maximum of 256 TB of log record addressing capacity over the life of a DB2 subsystem, or a maximum log record sequence number (LRSN) of 2**48 over the life of a DB2 data-sharing group. Use **BASIC** if you intend to use this instance of DB2 to copy or recover data from an instance of DB2 that does not support the **EXTENDED** format. However, after the 6 byte logging limit is exceeded, all new objects are created in **EXTENDED** format.

- **EXTENDED**

  New table spaces and indexes are created with a maximum of 1 YB (yottabyte[1]) of log record addressing capacity over the life of a DB2 subsystem or a maximum LRSN of 2**80 over the life of the DB2 data-sharing group. This setting is the default. Use this setting in either of the following situations:

  – When DB2 is likely to exhaust the basic log format. This setting is required to update database objects if the DB2 log RBA exceeds 2**48 for non-data-sharing environments or if the LRSN exceeds `x'FFFFFFFFFFFF'` for data-sharing environments.

  – In data-sharing environments only, when duplicate LRSN values occur because the processing speed exceeds the precision of the traditional log addressing format.

Important: In conversion mode, basic 6 byte format is used regardless of the setting of this parameter.

There is no way of overriding this setting with any keyword in the DDL syntax.

Important: The setting for this system parameter is ignored after you hit the hard limit. Objects are always created in **EXTENDED** format in this situation.

UTILITY_OBJECT_CONVERSION

The value of the **UTILITY_OBJECT_CONVERSION** parameter specifies whether DB2 utilities that accept the **RBALRSN_CONVERSION** option convert existing table spaces and indexes to 6 byte page format, to a 10 byte page format, or perform no conversion.

Note: The **RBALRSN_CONVERSION** keyword is available for the following utility control statements:

- **REORG TABLESPACE**
- **REORG INDEX**
- **REBUILD INDEX**
- **LOAD**

When you specify this keyword in the utility control statement, it generally overrides the current setting of the subsystem parameter unless set to **NOBASIC** (see Table 3-1 on page 37).

The following values are acceptable for this parameter:

- **BASIC**

  Existing table spaces and indexes that use extended 10-byte page format are converted to basic 6-byte page format. The **BASIC** option is allowed only if the **OBJECT CREATE FORMAT** field is also set to **BASIC**.

- **EXTENDED**

  Existing table spaces and indexes that use 6-byte page format are converted to extended 10-byte page format. The **EXTENDED** option is allowed only if the **OBJECT CREATE FORMAT** field is also set to **EXTENDED**.

- **NOBASIC**

  Existing table spaces and indexes that use 6-byte page format are converted to extended 10-byte page format. Table spaces and indexes that already use extended 10-byte page

[1] The byte units are: Kilobyte ·Megabyte ·Gigabyte ·Terabyte ·Petabyte ·Exabyte ·Zettabyte ·Yottabyte

format cannot be returned to the 6-byte page format. When this setting is in effect, utilities that specify the **RBALRSN_CONVERSION** keyword with **BASIC** fail. In addition, utilities that specify **RBALRSN_CONVERSION** keyword with **NONE** when the object is in 6-byte page format fail. The **NOBASIC** value is allowed in this field only if the **OBJECT CREATE FORMAT** field is set to **EXTENDED**.

Important: If you use the **NOBASIC** value, you prevent the user from overriding the system parameter setting with the utility control statement.

- **NONE** (default)

  No conversion is performed. This option is the default setting of this parameter. The **NONE** option is allowed regardless of the **OBJECT CREATE FORMAT** setting.

Conversion from six to 10 byte RBA and LRSN and vice versa is only possible in NFM.

3.1.4 Steps for enabling the extended RBA/LRSN format

Important: After migrating to NFM, check the RBA situation and decide if migration to extended RBS/LRSN is needed.

To fully enable the extended RBA/LRSN format, you need complete the following important tasks *in NFM* for DB2 11. Although you should run the BSDS conversion first, to improve performance, you can complete the other tasks in any order.

- Convert the BSDS records to support **EXTENDED** format (install job DSNTIJCB), as described 3.1.5, “Converting the BSDS” on page 32.

  This conversion causes outage for non-data-sharing subsystems. You need to plan for stopping DB2 and running the **DSNTIJCB**. It is a fast execution.

  In data sharing, you can convert a member at the time and avoid outage.
- Convert the DB2 catalog and directory table spaces and indexes to extended format, as described in 3.1.6, “Converting DB2 catalog and directory” on page 35, and then install the **DSNTIJCV** job.

  This task is mainly a **REORG SHRLEVEL CHANGE**, because it causes no outage. Furthermore, you can break the **DSNTIJCV** job logically by object and execute it a bit at the time if necessary.
- Convert the user data, as described in “The new page format for larger RBA and LRSN” on page 27 and 3.1.7, “Converting data from 6 byte to 10 byte RBA/LRSN or vice versa” on page 36.

You also need to decide which **DSNZPARM** option to use. See “New subsystem parameters for extended RBA” on page 30. In addition, keep an eye on the extended RBA disk space increase.

3.1.5 Converting the BSDS

The first step towards being able to use the extended RBA/LRSN format is to run the **DSNJCNVT** stand-alone utility for BSDS conversion. Running **DSNJCNVT** is optional and can be done any time after migrating to DB2 11 New Function Mode (NFM) if you are not approaching the end of the 6 byte range. The conversion is required during the DB2 installation if the RBA or LRSN is approaching the end of the 6 byte range.

For a data-sharing installation, if the LRSN is approaching the end of the 6 byte range, the BSDS of each member can be converted one at a time. If the RBA or LRSN are approaching the end of the 6 byte range, you need to convert the database objects also. They become read-only when the end of the range is reached.

Considerations for running DSNJCNVT

Keep in mind the following considerations when running the **DSNJCNVT** stand-alone utility:

- You must stop the DB2 subsystem that owns the BSDSs that are to be converted. **DSNJCNVT** is a stand-alone utility.
- In a data-sharing environment, allow DB2 utilities that read the logs of peer members to finish before converting the BSDSs.
- In a data-sharing environment, stop data replication products before the conversion to ensure that the old BSDSs can be successfully renamed and replaced by the converted BSDSs. The preferred procedure is to stop the replication product first and then stop the DB2 system that is to have its BSDSs converted. This procedure allows sharing systems to deallocate the BSDSs when the state of the member changes to inactive.
- The RACF user ID that is running **DSNJCNVT** must have read/write access to the new BSDSs and read access to the old BSDSs.
- The DB2 subsystem that owns the BSDS that is to be converted must start after the data sharing group was migrated to DB2 11 NFM.
- Conversion to the new BSDS format is required to write new format log records and remove the 6 byte RBA and LRSN limits.

Sample DSNJCNVT JCL

The statements in Example 3-6 specify that **DSNJCNVT** stand-alone utility is to convert the BSDS to support 10 byte RBA and LRSN fields.

Example 3-6 DSNJCNVT control statement

```
//CONVERT EXEC PGM=DSNJCNVT,REGION=64M
//SYSUT1   DD  DSN=DB2A.OLD.BSDS01,DISP=SHR
//SYSUT2   DD  DSN=DB2A.OLD.BSDS02,DISP=SHR
//SYSUT3   DD  DSN=DB2A.BSDS01,DISP=OLD
//SYSUT4   DD  DSN=DB2A.BSDS02,DISP=OLD
//SYSPRINT DD  SYSOUT=*
```

DSNJU004 to check if conversion has run

You can use the Print Log Map (**DSNJU004**) stand-alone utility to check if the conversion for a specific subsystem or member was performed. See the JCL shown in Example 3-7.

Example 3-7 DSNJU004 JCL

```
//DSNTLOG EXEC PGM=DSNJU004
//STEPLIB  DD  DISP=SHR,DSN=your.SDSNEXIT
//         DD  DISP=SHR,DSN=your.SDSNLOAD
//SYSUT1   DD  DISP=SHR,DSN=your.BSDS01
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//SYSIN    DD  *
//*
```

The resulting messages are shown in Example 3-8. The line marked in bold in this example indicates that for the DB2 subsystem the conversion has not been run yet.

Example 3-8 DSNJU004 output showing if DSNJCNVT has run

```
DSNJCNVB CONVERSION PROGRAM HAS RUN   DDNAME=SYSUT1
DSNJCNVT CONVERSION PROGRAM HAS NOT RUN   DDNAME=SYSUT1
   LOG MAP OF BSDS DATA SET COPY 1, DSN=DBOBB.BSDS01
   LTIME INDICATES LOCAL TIME, ALL OTHER TIMES ARE GMT.
         DATA SHARING MODE IS OFF
         SYSTEM TIMESTAMP   - DATE=2013.217  LTIME=12:27:14.79
         UTILITY TIMESTAMP  - DATE=2013.178  LTIME= 8:13:30.76
         VSAM CATALOG NAME=DBOBD
         HIGHEST RBA WRITTEN       00000000000003DB21472  2013.217
         HIGHEST RBA OFFLOADED     00000000000003DB21FFF
         RBA WHEN CONVERTED TO V4  00000000000000000000
```

Important: if you want the new BSDS format, you need to convert to it in NFM regardless of whether you installed or migrated to DB2 11.

Note: Looking at the **DSNJU004** output, RBAs and LRSNs already are shown in a 10 byte format. The **DSNJU004** stand-alone utility, as most DB2 externalization functions, uses the extended format, but DB2 still stores the RBAs and LRSNs as 6 byte values until the conversion is completed. The conversion changes the CI size for existing BSDSs from 4 KB to 8 KB so that it can accommodate the larger record sizes.

Install job DSNTIJCB

The **DSNTIJCB** new installation job applies the needed changes to the BSDS. It includes the following job steps:

DSNTDEF Activates **IDCAMS** to define new BSDS data sets with **CLUSTER**, **DATA**, and **INDEX** components under the temporary names `prefix.BSDS01.NEW` and `prefix.BSDS02.NEW`.

DSNTCNVT Activates **DSNJCNVT** to read records from the existing BSDS, converts them to support the extended RBA and LRSN format, and writes them into the `prefix.BSDS01.NEW` and `prefix.BSDS02.NEW` data sets.

DSNTRENO Renames the existing BSDS data sets to backup names, `prefix.BSDS01.OLDFMT` and `prefix.BSDS02.OLDFMT`.

DSNTRENN Renames `prefix.BSDS01.NEW` and `prefix.BSDS02.NEW` to `prefix.BSDS01` and `prefix.BSDS02`.

The job completes with return code zero. The following message displays in the job log:

```
IDC0002I IDCAMS PROCESSING COMPLETE. MAXIMUM CONDITION CODE WAS 0
DSNJ200I  DSNJCNVT CONVERT UTILITY PROCESSING COMPLETED SUCCESSFULLY FOR MEMBER
DB1A
```

After the BSDS is converted, the physical log records are written in the new format. A larger log record header (LRH) is required for the larger RBA and LRSN values, and there are changes to the format of the log data sets as well (collectively called *new format logs*). Members of a data sharing group can be converted one at a time and can run with a mix of converted and not converted members is supported.

Tips: The conversion only takes a few seconds. In addition, if the conversion fails, you can restart DB2 with the old BSDS while you resolve the problem of the failure. You can test the conversion with a copy of the BSDS before you really convert the actual BSDS.

3.1.6 Converting DB2 catalog and directory

Because catalog and directory are composed of table and index spaces, you have to change the RBA/LRSN formats for those also. You need to distinguish between subsystems, which are newly installed in DB2 11, and those that are migrated from DB2 10.

New installations

For new installations all catalog and directory table spaces and indexes are always created in the extended object format. The setting of **OBJECT_CREATE_FORMAT** system parameter is not honored for catalog and directory objects. As a result, if you query the catalog of a newly installed subsystem, the **RBA_FORMAT** column of **SYSIBM.SYSTABLEPART** and **SYSIBM.SYSINDEXPART** contains value 'E' for all catalog and directory objects.

Migrated subsystems

The tables that are newly created in CM are as follows:

```
NAME                 TSNAME
---------+---------+---------+
SYSOBD_AUX           SYSTSOBX
SYSQUERYPREDICATE    SYSTSQRE
SYSQUERYSEL          SYSTSQRS
SYSVARIABLES_DESC    SYSTSVAD
SYSVARIABLEAUTH      SYSTSVAU
SYSINDEXCLEANUP      SYSTSIXC
SYSSTATFEEDBACK      SYSTSSFB
SYSVARIABLES         SYSTSVAR
SYSVARIABLES_TEXT    SYSTSVAT
```

All the table spaces holding these tables are created using the **BASIC** format. You can verify this **RBA_FORMAT** browsing column in **SYSIBM.SYSTABLEPART**. The associated indexes are in basic format as well, because the **EXTENDED** format for page sets and for the BSDS is available only in NFM.

DSNTIJCV

DB2 11 introduces the **DSNTIJCV** new installation job for catalog conversion, which is customized during the ENFM installation/migration job generation.

You can use the **DSNTIJCV** installation job to convert the DB2 catalog and directory table spaces and their indexes to extended RBA/LRSN format. You need to be in NFM when you run this job. Do not rearrange the processing sequence and the table space grouping in this job. The COPY and REORG steps use **LISTDEF** filtering to exclude table spaces that are already in extended format, because they have been converted previously.

The main objective of the **DSNTIJCV** job is to migrate the catalog and directory to **EXTENDED** format. However, you can also it to change the catalog and directory back from **EXTENDED** format to **BASIC** format.

The following list shows a detailed description of the job steps as described in the job prolog:

```
JCVTRM00 STEP       TERMINATE PENDING UTILITIES FOR THIS JOB
```

```
JCVCVT01 STEP       CONVERT SYSUTILX DIRECTORY TABLE SPACE
JCVCPY01 STEP       IMAGE COPY THE CONVERTED SYSUTILX TABLE SPACE
JCVCPY02 STEP       IMAGE COPY THE SYSLGRNX TABLE SPACE
JCVCVT02 STEP       CONVERT THE FORMAT OF THE SYSLGRNX TABLE SPACE
JCVCPY03 STEP       COPY OTHER DIRECTORY TABLE SPACES TO BE CONVERTED
JCVCVT03 STEP       CONVERT THE OTHER DIRECTORY TABLE SPACES
JCVCPY04 STEP       COPY THE DIRECTORY LOB TABLE SPACES
JCVCVT04 STEP       CONVERT THE DIRECTORY LOB TABLE SPACES
JCVCPY05 STEP       COPY THE SYSTSCPY CATALOG TABLE SPACE
JCVCVT05 STEP       CONVERT THE SYSTSCPY CATALOG TABLE SPACE
JCVCPY06 STEP       COPY OTHER CATALOG TABLE SPACES TO BE CONVERTED (PART 1)
JCVCVT06 STEP       CONVERT OTHER CATALOG TABLE SPACES (PART 1)
JCVCPY07 STEP       COPY RELATED CATALOG LOB TABLE SPACES (PART 1)
JCVCVT07 STEP       CONVERT RELATED CATALOG LOB TABLE SPACES (PART 1)
JCVCPY08 STEP       COPY OTHER CATALOG TABLE SPACES TO BE CONVERTED (PART 2)
JCVCVT08 STEP       CONVERT OTHER CATALOG TABLE SPACES (PART 2)
JCVCPY09 STEP       COPY RELATED CATALOG LOB TABLE SPACES (PART 2)
JCVCVT09 STEP       CONVERT RELATED CATALOG LOB TABLE SPACES (PART 2)
JCVCPY10 STEP       COPY OTHER CATALOG TABLE SPACES TO BE CONVERTED (PART 3)
JCVCVT10 STEP       CONVERT OTHER CATALOG TABLE SPACES (PART 3)
JCVCPY11 STEP       COPY RELATED CATALOG LOB TABLE SPACES (PART 3)
JCVCVT11 STEP       CONVERT RELATED CATALOG LOB TABLE SPACES (PART 3)
JCVCPY12 STEP       COPY OTHER CATALOG TABLE SPACES TO BE CONVERTED (PART 4)
JCVCVT12 STEP       CONVERT OTHER CATALOG TABLE SPACES (PART 4)
JCVCPY13 STEP       COPY RELATED CATALOG LOB TABLE SPACES (PART 4)
JCVCVT13 STEP       CONVERT RELATED CATALOG LOB TABLE SPACES (PART 4)
```

Important: The catalog and directory are special DB2 objects. Carefully watch your subsystems RBAs and LRSNs. Definitely make sure that you do not hit the hard limit with the catalog and directory not being migrated to the **EXTENDED** format. The ability of utilities to convert, especially the **SYSUTILX**, **SYSTSCPY**, and **SYSLGRNX** utilities to the **EXTENDED** format after you hit the hard limit, is more restricted than for any other page set. It might occur that you cannot convert these objects using regular utilities, and you might be forced to follow a manual procedure to reset RBA and LRSN for your subsystem.

Tip: Make sure you closely watch for messages about potential soft limit hits for your DB2 subsystem. Consider a timely conversion of your subsystem's BSDSs before you run into major problems and convert the catalog directory early enough to avoid major problems with hitting the hard limit.

3.1.7 Converting data from 6 byte to 10 byte RBA/LRSN or vice versa

The most obvious way to convert the RBA/LRSN format of an existing page set from 6 bytes to 10 bytes are the **REORG TABLESPACE** and **REORG INDEX** utilities.

You can convert a partitioned table space index one part at a time with the following exceptions:

- PBG table spaces that are organized by hash must be converted at table space level, that is all parts in one REORG job.
- Tables with 8 byte XML version IDs must also be converted at the table level.
- Tables that are in an active clone relationship cannot be converted. You must drop the clone first.

The REORG INDEX and REORG TABLESPACE utilities

The **REORG TABLESPACE** and **REORG INDEX** utilities in **SHRLEVEL REFERENCE** or **NONE** have three new utility control statements that influence if and how an RBA/LRSN conversion is performed during the utility execution.

The **UTILITY_OBJECT_CONVERSION DSNZPARM** setting is used when the utility control statement does not specify the **RBALRSN_CONVERSION** keyword with the following options:

BASIC Existing table spaces and indexes that use extended 10 byte page format are converted to basic 6 byte page format. The **BASIC** option is allowed only if the **OBJECT CREATE FORMAT** field is also set to **BASIC**.

EXTENDED Existing table spaces and indexes that use 6 byte page format are converted to extended 10 byte page format. The **EXTENDED** option is allowed only if the **OBJECT CREATE FORMAT** field is also set to **EXTENDED**.

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

NONE No conversion is performed. This option is the default setting of this parameter. The **NONE** option is allowed regardless of the **OBJECT CREATE FORMAT** setting.

Table 3-1 shows what type of conversion is performed and the error messages, depending on the system parameter setting and the usage of the **RBALRSN_CONVERSION** keyword in the utility control statement. For **REORG TABLESPACE**, all the results listed in the four NFM columns also apply to the conversion of all indexes that are defined on the tables in the table space that is being reorganized.

Table 3-1 UTILITY_OBJECT_CONVERSION

Utility option	UTILITY_OBJECT_CONVERSION							
	NONE		BASIC		EXTENDED		NOBASIC	
	CM	NFM	CM	NFM	CM	NFM	CM	NFM
NONE	DSNU169I[a] DSNU123I[b]	NONE	DSNU169I DSNU123I	NONE	DSNU169I DSNU123I	NONE	DSNU169I DSNU123I	fails if object is BASIC
BASIC	DSNU169I DSNU123I	BASIC	DSNU169I DSNU123I	BASIC	DSNU169I DSNU123I	BASIC	DSNU169I DSNU123	fails
EXTENDED	DSNU169I DSNU123I	EXTENDED	DSNU169I DSNU123I	EXTENDED	DSNU169I DSNU123I	EXTENDED	DSNU169I DSNU123I	EXTENDED
omitted	ignored	NONE	ignored	BASIC	ignored	EXTENDED	ignored	EXTENDED

a. DSNU169I The OBJECT CONVERSION REQUESTED BY requestor-type requestor-name requestor-operand IS IGNORED
b. DSNU123I csect-name ATTEMPT TO USE NEW FUNCTION BEFORE NEW FUNCTION MODE

Even if the table space is not actually converted by the **REORG**, because it already exists in the requested format, the index is converted to the same format as the table space. See Figure 3-3.

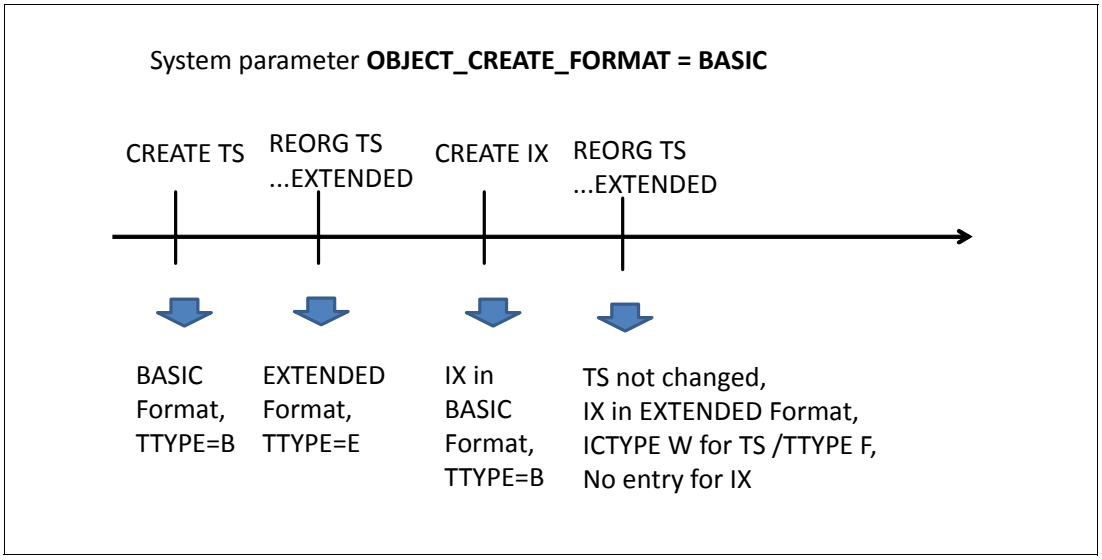


Figure 3-3 REORG TABLESPACE with Index conversion

If a conversion of the RBA/LRSN format occurs during the **REORG**, the job output provides the following message:

```
DSNU1169I -DB1A 220 18:59:09.53 DSNURFIJ - TABLESPACE SABIDB3.SEGMENT CONVERTED BY
KEYWORD TO EXTENDED RBA/LRSN FORMAT
```

Active CLONE relationship restriction: If there is an active **CLONE** relationship, page set conversion is not performed. If this request is based on a system parameter setting, the fact that a conversion is requested is ignored. If, however, you request this conversion through a **RBALRSN_CONVERSON** utility control statement, a DSNU1459 error message is issued.

A new **RBA_FORMAT** column is added to the **SYSIBM.SYSTABLEPART** and **SYSIBM.SYSINDEXPART** catalog tables. The values in this column are:

B	Basic, with 6 byte RBA/LRSN format
E	Extended, with 10 byte RBA/LRSN format
U	Undefined, where **DEFINE NO** was specified when creating the table space, and the table space is not an XML table space with XML versions.
blank	For migrated objects

In addition to this new column, which indicates the RBA/LRSN format for a given page set, the **TTYPE** column on the **SYSIBM.SYSCOPY** catalog table adds additional information when a utility execution leads to a format conversion. If the resulting format is **EXTENDED**, you see an **E** in this column and if it is **BASIC** the value is **B**. As a consequence, you might see the combinations as listed in Table 3-2 in **SYSIBM.SYSCOPY**.

Table 3-2 SYSCOPY values for ICTYPE and TTYPE

Utility	ICTYPE	TTYPE
REORG LOG YES	X	B or E
REORG LOG NO	W	B or E
RECOVER PIT	P	B or E
REBUILD INDEX	B	B or E
LOAD REPLACE LOG YES	R	B or E
LOAD REPLACE LOG NO	S	B or E

Note: For index spaces, the entries occur only if they are copy-enabled. In addition, they are also generated only through the **REORG INDEX**, **REBUILD INDEX**, and **RECOVER to PIT** utilities. Thus, if the RBA/LRSN format is changed, through a **REORG TABLESPACE**, no information about this change is kept in the **SYSIBM.SYSCOPY**.

LOAD REPLACE

In addition to the **REORG TABLESPACE** and **REORG INDEX** utilities, **LOAD** with option **REPLACE** is another way to convert a page set's RBA and LRSN length. The setting of **UTILITY_OBJECT_CONVERSION** system parameter applies, analogous to **REORG**, only past conversion mode. The **RBALRSN_CONVERSION** utility control statement can be set to **NONE**, **BASIC**, **EXTENDED**, and the utility behavior is basically the same as described in "The REORG INDEX and REORG TABLESPACE utilities" on page 37.

Keep in mind that specifying the **RBALRSN_CONVERSION** can only make sense on **LOAD REPLACE**. If you use **LOAD RESUME**, the nature of the utility is to load only a subset of the total number of rows into a table (space). Having a few rows use a 6 byte RBA and some using a 10 byte RBA really does not make sense. You receive the following error message when you try to request a conversion during a LOAD RESUME:

```
DSNU071 KEYWORD 'RBALRSN_CONVERSION REQUIRES KEYWORD 'REPLACE'.
```

REBUILD INDEX

In DB2 11 NFM, the same functionality and the same rules apply for the conversion of RBAs and LRSNs as described for **REORG INDEX** and **TABLESPACE** and for **LOAD REPLACE**.

DSN1COPY

DSN1COPY is a stand-alone utility. When you use this utility to overlay data of an existing table space with image copy data from an image copy that you created earlier, DB2 does not check or modify anything in the catalog.

Using the **DSN1COPY** utility can cause a mismatch between the information stored in the catalog and the actual table space format, as illustrated in Figure 3-4. This scenario assumes a TS is created in **EXTENDED** format (with the **OBJECT_CREATE_FORMAT=EXTENDED** system parameter). Take an image copy of the page set and subsequently reorganize it. Using the **REORG** control statement, specify the **RBALRSN_CONVERSION** option with **BASIC** value. This option changes the format of the table space from **EXTENDED** to **BASIC**. This change is also reflected in the **SYSIBM.SYSTABLESPACE** table **RBA_FORMAT** column, which now shows the **B** value. Next, run the **DSN1COPY** utility, and overlay the table space data with what is on image copy FC1, that is data in **EXTENDED** format. Now there is a mismatch between the information that is stored in the catalog and what is in the table space.

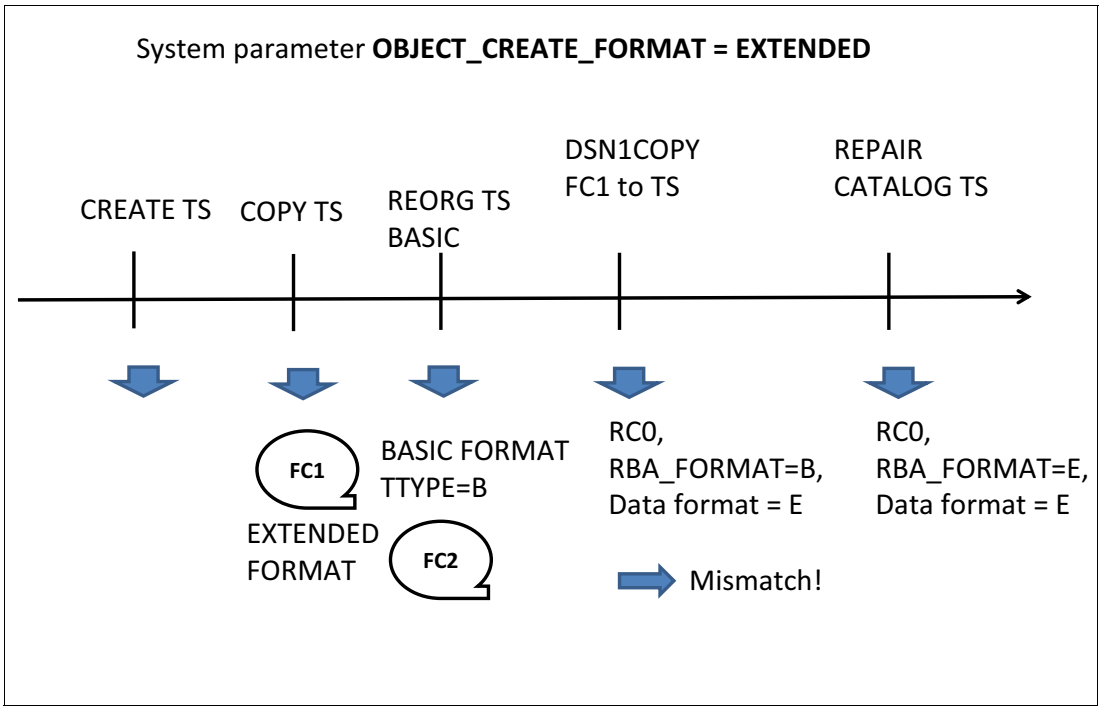


Figure 3-4 DSN1COPY: Catalog information mismatch

When you start working with the table stored in this table space, these is no message or, even worse, error message, but the mismatch is nevertheless not nice. You can use the **REPAIR** utility with the **CATALOG** option to fix this mismatch.

The **REPAIR CATALOG** option is a new option on the **REPAIR** utility. It indicates that the **REPAIR** utility is to validate information in the catalog for the specified table space. When you specify **REPAIR CATALOG**, the utility performs the following actions:

1. Compares the following information in the catalog with the data:
   - Row format (can be either reordered row format or basic row format)
   - RBA format (can be either 6 byte format or 10 byte format)
   - Data version information (same functionality that is performed with **REPAIR VERSIONS**)
   - Hash space value

   For these items, if the information in the catalog is different from the data, the **REPAIR** utility changes the values in the catalog to match the data.
2. Validates the following information:
   - DBID, PSID, and OBID
   - Table space type
   - **SEGSIZE**
   - **PAGESIZE**
   - Table definition

   For these items, if the information in the catalog is different from the data, the **REPAIR** utility does not correct the information in the catalog. Instead, the **REPAIR** utility fails and reports the mismatched information in a message. To correct the mismatched information, take the action that is documented for the message that you receive.

REPAIR CATALOG does not make any corrections for indexes. If you or the **REPAIR** utility made corrections to the data or catalog as a result of running **REPAIR CATALOG**, rebuild any indexes on the target tables.

The syntax for the **REPAIR** utility includes a **TEST** option that you can specify together with **REPAIR CATALOG**. This option indicates that the **REPAIR** utility is not to correct any mismatched information. Wih this option set, the utility checks all of the same information that it checks when you specify **REPAIR CATALOG**. However, any information differences between the data and catalog are reported only in messages. The utility does not take any corrective actions.

Example 3-9 shows the results before running **REPAIR CATALOG** for the situation described in Figure 3-4 on page 40, using the **TEST** option.

Example 3-9 Output of TEST option

```
DSNU674I  -DB1A 221 15:58:59.95 DSNUCBVR - RBA FORMAT FOR DBID=X'014F'
PSID=X'0005' IN THE DB2 CATALOG IS BASIC, BUT IN THE PAGE SET IS EXPANDED.
```

Then, subsequently running the **REPAIR** utility, the utility reports the messages shown in Example 3-10.

Example 3-10 Repair output

```
DSNU674I  -DB1A 221 16:41:23.70 DSNUCBVR - RBA FORMAT FOR DBID=X'014F'
PSID=X'0005' IN THE DB2 CATALOG IS BASIC, BUT IN THE PAGE SET IS EXPANDED.
DSNU695I  -DB1A 221 16:41:23.71 DSNUCBVR - INFORMATION IN THE CATALOG WAS UPDATED
                      TO MATCH THE PAGE SET
```

3.1.8 Additional considerations regarding utilities

This section discusses several aspects about how utilities are affected by extended RBA/LRSNs and how utilities support RBS/LRSNs.

BACKUP/RESTORE SYSTEM

When you run the **BACKUP SYSTEM** utility, a token is associated with the backup to identify it. In a system for which the BSDS is not converted so that it can hold the extended RBAs and LRSNs, the token is 18 bytes long. In this situation, the token is composed as explained in Table 3-3.

Table 3-3 Composition of BACKUP token

Field	Field description	Length
DB2 SSID	DB2 subsystem ID	4 bytes
TOD	Time of day of SYSPITR = LRSN	8 bytes
RBA	Checkpoint RBA of last checkpoint before BACKUP	6 bytes

Example 3-11 shows a token resulting from a **BACKUP SYSTEM** execution on a DB2 subsystem for which the BSDS is not converted to the extended RBA/LRSN.

Example 3-11 BACKUP token prior to BSDS conversion

```
TOKEN = X'C4C2F0C2CBC85E68889EE60800003E44A7B1'
```

After you have converted the BSDS, the information given in the job output is different. First, the length of the token is increased by 4 bytes, which are added to the RBA at the end of the token. No change occurs for the time of day, which you can expect, because the BSDS conversion is for both, 10 bytes by RBAs as well as 10 bytes by LRSNs. Second, the job

output is slightly changed. It shows both the tokenand the real value that is stored in the BSDS for the time of day.

Example 3-12 shows how the token changed and the additional information for the **DATA COMPLETE** LRSN, which in fact is 10 bytes long.

Example 3-12 BACKUP SYSTEM job output after BSDS conversion

```
DSNU1600I   220 17:06:57.72 DSNUVBBD - BACKUP SYSTEM UTILITY FOR DATA STARTING,
                      COPYPOOL = DSN$DB1A$DB
                      TOKEN = X'C4C2F1C1CBC8654135D8460400000000000014B43AB62'.
DSNU1614I   220 17:06:59.95 DSNUVBBD - BACKUP SYSTEM UTILITY FOR DATA COMPLETED
                      COPYPOOL = DSN$DB1A$DB
                      TOKEN = X'C4C2F1C1CBC8654135D8460400000000000014B43AB62'
                      DATA COMPLETE LRSN = X'0000000000014B43F6DA'
                      ELAPSED TIME = 00:00:02.
```

Also, if you print the BSDS afterwards and scroll to the section that lists all available **SYSTEM BACKUPS**, the long LRSN is also listed. Example 3-13 contains the information about one existing system level backup. Notice the entries showing up in the **EXTENDED** format.

Example 3-13 DSNJU004 after BACKUP SYSTEM for non-data sharing system

```
START STCK                                        DATA COMPLETE
     DATA               LOG                 RBLP                  LRSN
----------------  ----------------  ------------------  ------------------
CBC8654135D84604  0000000000000000  0000000000014B43AB62  0000000000014B43F6DA
                 TOKEN = C4C2F1C1CBC8654135D8460400000000000014B43AB62
                 Z/OS 1.13  CAT=YES
                 LOCATION NAME = DB1A
```

Things are a little different for data sharing systems. Example 3-14 shows almost the same data as Example 3-12, with the exception that the last 10 bytes of the token do not represent an RBA but an LRSN.

Example 3-14 BACK SYSTEM job output from data sharing system

```
DSNU1600I   220 16:05:48.87 DSNUVBBD - BACKUP SYSTEM UTILITY FOR DATA STARTING,
                      COPYPOOL = DSN$DB1B$DB
                      TOKEN = X'C4F1C2F1CBC8579652E78E0A00CBC85695DC17000000'.
DSNU1614I   220 16:05:49.80 DSNUVBBD - BACKUP SYSTEM UTILITY FOR DATA COMPLETED
                      COPYPOOL = DSN$DB1B$DB
                      TOKEN = X'C4F1C2F1CBC8579652E78E0A00CBC85695DC17000000'
                      DATA COMPLETE LRSN = X'00CBC8579651972A2200'
                      ELAPSED TIME = 00:00:00.
```

For completion, Example 3-15 shows the information listed in the BSDS for the data sharing system.

Example 3-15 DSNJU004 after BACKUP SYSTEM for data sharing

```
START STCK                                        DATA COMPLETE
     DATA               LOG                 RBLP                  LRSN
----------------  ----------------  ------------------  ------------------
CBC8579652E78E0A  0000000000000000  00CBC85695DC17000000  00CBC8579651972A2200
                 TOKEN = C4F1C2F1CBC8579652E78E0A00CBC85695DC17000000
                 Z/OS 1.13  CAT=YES
```

```
                    LOCATION NAME = DB1B
```

RECOVER

The **RECOVER** utility **TORBA**, **TOLOGPOINT**, and **RESTORBEFORE** keywords now accept 6 byte or 10 byte RBAs or LRSNs. Operands of 6 bytes or less are interpreted as being in **BASIC** format. Operands greater than 6 bytes are interpreted as being in **EXTENDED** format.

The **RECOVER** utility now can handle the different RBA/LRSN formats. Figure 3-5 illustrates what happens when you recover a page set to a point prior to changing the RBA/LRSN format.

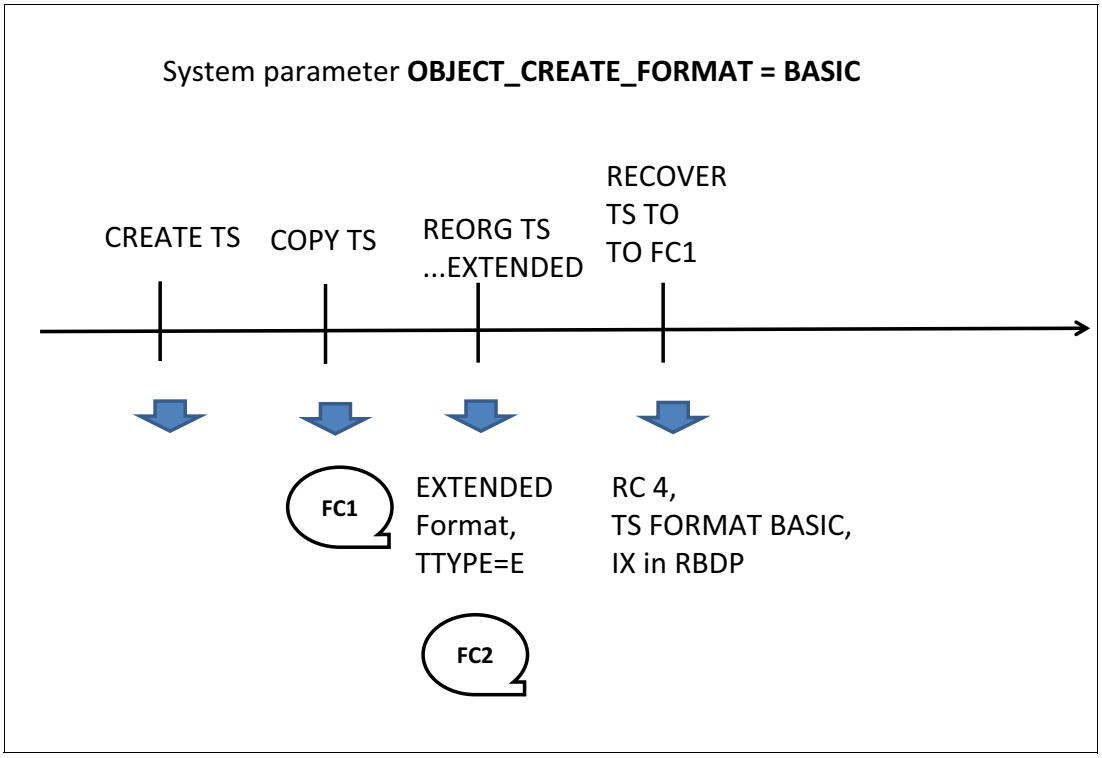


Figure 3-5 PIT RECOVERY

This example creates an object in **BASIC** format and take an image copy, which in this situation includes the data in **BASIC** format. Subsequently, the RBA/LRSN format changes using a **REORG TABLESPACE**. Because changing the RBA/LRSN format is possible only with a **SHRLEVEL REFERENCE** or **CHANGE REORG**, an image copy is mandatory. This image copy contains the data in **EXTENDED** format. If you had to perform a subsequent **RECOVER** to current, you use this image copy, and the object remains in **EXTENDED** format.

If you decide not to recover to **CURRENT**, but to image copy FC1, **RECOVER** also accepts this image copy as a recover basis. It uses the data as is on the image copy and recovers the data up to the point that you specify on your **RECOVER TABLESPACE** statement. If you specify any given RBA between FC1 and FC2, the result is a recovered object in **BASIC** format. The fact that the object is reverted to **BASIC** format is mentioned in the job output through the following message:

```
DSNU1169I -DB1A 220 18:59:09.53 DSNURFIJ - TABLESPACE SABIDB3.SEGMENT CONVERTED BY
KEYWORD TO BASIC RBA/LRSN FORMAT.
```

In addition, several catalog changes are also applied to reflect this conversion. **RBALRSN_FORMAT** in **SYSIBM.SYSTABLEPART** is set to **B** and also the entry for the PIT recovery in **SYSIBM.SYSCOPY** indicates this through the **TTYPE** column for the PIT recovery record.

REPORT RECOVERY

The **REPORT RECOVERY** utility output now accommodates the longer RBAs and LRSNs. The utility output is restricted to 120 bytes in length, which means that the existing layout is compressed. So if you rely on processing the output automatically, make sure that you review current processes and adjust accordingly.

3.2 NOT LOGGED for declared global temporary tables

A *declared global temporary table* (DGTT) is used by an application to store intermediate SQL results data while an application is still running. DB2 11 improves this implementation by allowing the option to avoid logging during insert, update, and delete activity to DGTTs. This option can improve the performance and usability of DGTTs by applications. It is also in line with DB2 family compatibility because for DB2 for Linux, UNIX, and Windows already supports not-logged DGTTs.

With the ability to choose to not log DGTT activity, it might be beneficial for you to use DGTTs instead of a *created global temporary table* (CGTT). Although CGTTs support not logging and can provide better performance because the schema is known prior to execution of the application program, CGTTs do not support certain key features, such as indexes.

3.2.1 Syntax extension

Figure 3-6 illustrates the **DECLARE GLOBAL TEMPORARY TABLE** syntax.

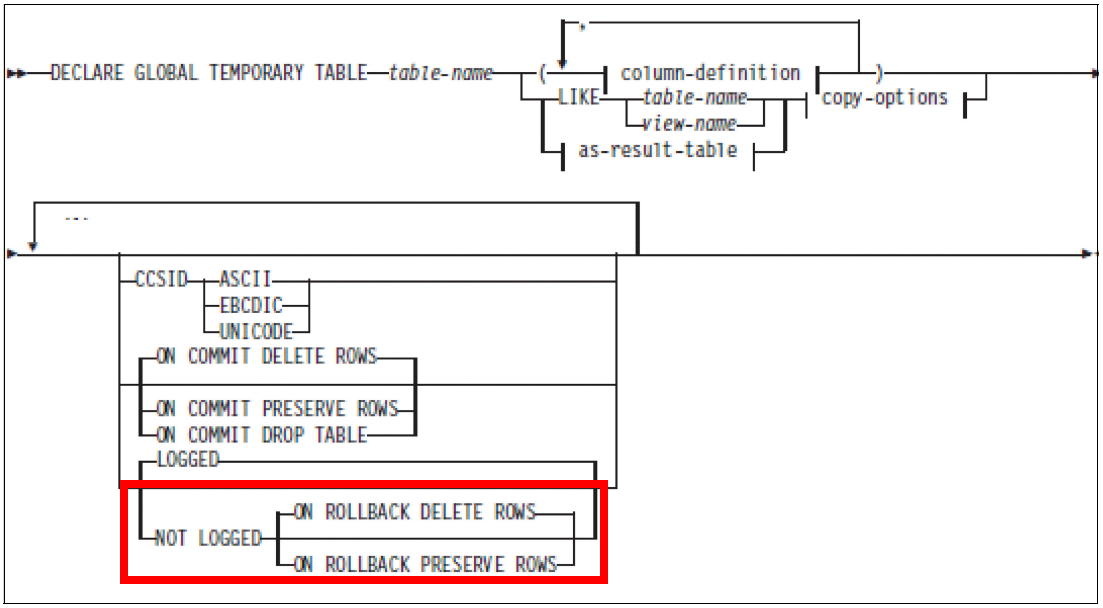


Figure 3-6 CREATE DGTT syntax with NOT LOGGED options

The syntax in Figure 3-6 shows that a DGTT now has the following different logging options:

- **LOGGED**

This option is the default and the current behavior. In this case, DB2 logs all changes, and during **ROLLBACK** or **ROLLBACK TO SAVEPOINT**, the changes to the DGTT are undone.

- **NOT LOGGED ON ROLLBACK DELETE ROWS**

  This option specifies that you do not want logging to occur for this table, and during **ROLLBACK** or **ROLLBACK TO SAVEPOINT**, all rows in the DGTT are deleted if any change was made in the duration.

- **NOT LOGGED ON ROLLBACK PRESERVE ROWS**

  This option specifies that you do not want logging to occur for this table, and during **ROLLBACK** or **ROLLBACK TO SAVEPOINT**, the rows in the DGTT are preserved as they are.

You usually think about rollbacks in an error situation when there is a negative **SQLCODE** or message returned to the application. This kind of rollback is not affected by the new behavior. In the case of an error situation during an SQL statement, where an **SQLCODE** or message is issued, if an update was made to a DGTT and **LOGGED** is specified, the changes to the DGTT are undone.

Also, in the case of an error situation during an SQL statement, where an **SQLCODE** or message is issued, if an update was made to a DGTT and **NOT LOGGED** is specified, all rows in that DGTT are deleted, regardless of the **DELETE/PRESERVE ROWS** qualification.

Important: Your decision on the **ROLLBACK** behavior for DGTTs affects only situations in which your application explicitly issues a **ROLLBACK** statement.

3.2.2 Undo processing for NOT LOGGED DGTTs

Undo processing operates by reading the information on the log in a backward direction and backs out the changes made by the current unit of recovery, as indicated by the log records encountered. Undo processing is entered for of a number of reasons:

- The application issues the **ROLLBACK SQL** statement.

  The undo processing depends on **ON ROLLBACK DELETE/PRESERVE** specification described previously. If you specified **NOT LOGGED ON ROLLBACK DELETE ROWS** on the **DECLARE GLOBAL TEMPORARY TABLE** statement and if the DGTT was updated since the last **COMMIT**, all rows in the DGTT are deleted. The DGTT itself remains available; however, all cursors open against the DGTT are closed.

  If you specified **NOT LOGGED ON ROLLBACK PRESERVE ROWS**, all rows in the DGTT are preserved as is. The DGTT and the rows are available.

- The application issues the **ROLLBACK TO SAVEPOINT** statement.

  In this case the undo processing depends on the **ON ROLLBACK DELETE/PRESERVE** specification described previously.

  If you specified **NOT LOGGED ON ROLLBACK DELETE ROWS** and if the DGTT was updated since the last **COMMIT**, all rows in the DGTT are deleted. The DGTT itself is available; however, all cursors open against the DGTT remain open but not positioned because the rows are deleted.

  If you used **NOT LOGGED ON ROLLBACK PRESERVE ROWS** on the **DECLARE GLOBAL TEMPORARY TABLE** statement, all rows in the DGTT are preserved as is. The DGTT and the rows are available.

- If there is an error while the application executes an **INSERT**, **UPDATE**, or **DELETE** statement, all rows are deleted from the DGTT. This type of error includes, for example, a duplicate

key violation. The DGTT itself is available; however, any cursors open against the DGTT remain open but no longer have position.

- If an application gets cancelled using a **-CANCEL THREAD** command, all rows are deleted from the DGTT, and the DGTT is dropped.
- For statements that insert multiple rows, the **ATOMIC** and **NOT ATOMIC CONTINUE ON SQLEXCEPTION** options of the **INSERT** statement determine the result of an error inserting any of the rows.
  - The **ATOMIC** option specifies that if the insert for any row fails, all changes made to the database by any of the inserts, including changes made by successful inserts, are undone. This option is the default.
  - The **NOT ATOMIC CONTINUE ON SQLEXCEPTION** option specifies that, regardless of the failure of any particular insert of a row, the **INSERT** statement will not undo any changes made to the database by the successful inserts of other rows, and inserting will be attempted for subsequent rows. However, the minimum level of atomicity is at least that of a single insert (that is, it is not possible for a partial insert to complete), including any triggers that might have been executed as a result of the INSERT statement.

  The **ATOMIC** or **NOT ATOMIC CONTINUE ON SQLEXCEPTION** clauses can be specified for a static multiple-row-insert. However, do not specify this clause for a dynamic **INSERT** statement. For a dynamic statement, use the **ATOMIC** or **NOT ATOMIC CONTINUE ON SQLEXCEPTION** clause as an attribute on the **PREPARE** statement.

3.2.3 Thread reuse

A thread is qualified for reuse if the table was defined with both the **ON COMMIT DELETE ROWS** attribute, which is the default, and the **NOT LOGGED ON ROLLBACK DELETE ROWS** attribute, which is not the default.

A thread is not qualified for reuse if the table is defined with **PRESERVE ROWS** specified either **ON COMMIT** or **NOT LOGGED ON ROLLBACK**.

3.2.4 Sample scenarios

The following examples summarize these types of scenarios.

Example 1

Assume that the application has the following sequence of SQL statements:

1. DECLARE GLOBAL TEMPORARY TABLE DT1 NOT LOGGED ON ROLLBACK DELETE ROWS
2. CREATE UNIQUE INDEX on DT1
3. INSERT INTO DT1 (successful)
4. INSERT INTO DT1 (duplicate key error)

In this error situation, because no **UNDO** log records are available due to **NOT LOGGED** specification, DB2 deletes all the rows in DT1. However, the DGTT, DT1, are available for **INSERT**, **UPDATE**, **DELETE**, and **FETCH**.

Example 2

Assume that the application has the following sequence of SQL statements:

1. DECLARE GLOBAL TEMPORARY TABLE DT1 NOT LOGGED ON ROLLBACK DELETE ROWS
2. CREATE UNIQUE INDEX on DT1
3. INSERT INTO DT1 (successful)

4. INSERT INTO DT1 (successful)
5. DECLARE GLOBAL TEMPORARY TABLE DT2 NOT LOGGED ON ROLLBACK DELETE ROWS (declaring a second DGTT) produces an internal error such as no space.

In this error situation, DB2 only undoes anything done for **DECLARE DT2**. DT1 remains as is, with 2 inserted rows, and available for further updates.

Example 3

Assume the following sequence of steps:

1. DECLARE GLOBAL TEMPORARY TABLE DT1 NOT LOGGED ON ROLLBACK DELETE ROWS
2. CREATE UNIQUE INDEX on DT1
3. INSERT INTO DT1 (successful)
4. INSERT INTO DT1 (successful)
5. DECLARE GLOBAL TEMPORARY TABLE DT2 NOT LOGGED ON ROLLBACK DELETE ROWS
6. INSERT INTO DT2 (successful)
7. INSERT INTO DT2 (duplicate key error)

In this situation, DB2 delete rows from DT2, because the error was on **INSERT** into DT2.

Example 4

Assumes the following sequence of SQL statements:

1. DECLARE GLOBAL TEMPORARY TABLE DT1 NOT LOGGED ON ROLLBACK DELETE ROWS
2. INSERT INTO DT1 with a subselect
3. COMMIT
4. DECLARE CURSOR C1 on DT1
5. OPEN C1
6. FETCH C1
7. UPDATE where current of C1
8. INSERT into DT1
9. ROLLBACK

This roll back causes DB2 to delete all the rows in DT1. Even the inserted rows that were committed. However, the DGTT, DT1, is available for **INSERT**, **UPDATE**, **DELETE**, and **FETCH**.

Example 5

Assume the application has the following sequence of SQL statements:

1. DECLARE GLOBAL TEMPORARY TABLE DT1 NOT LOGGED ON ROLLBACK DELETE ROWS
2. INSERT INTO DT1 with a subselect
3. COMMIT
4. DECLARE CURSOR C1 on DT1
5. OPEN C1
6. FETCH C1
7. ROLLBACK

This rollback does not delete the rows in DT1 because no updates were made to DT1 since the last commit.

Difference with this example in the roll back: Note the difference here. When **NOT LOGGED ON ROLLBACK DELETE ROWS** is specified, insert, update, and delete activity is not logged. During a **ROLLBACK** or **ROLLBACK TO SAVE POINT**, if there was any updates to the DGTT since the last **COMMIT** statement, all rows from the DGTT are deleted, and any open cursors against the DGTT have no position. If the declaration of the DGTT itself was not committed, the DGTT itself is rolled back.

Example 6

Assume the following sequence of statements:

1. DECLARE GLOBAL TEMPORARY TABLE DT1 NOT LOGGED ON ROLLBACK DELETE ROWS
2. INSERT INTO DT1 with a subselect
3. DECLARE CURSOR C1 on DT1
4. OPEN C1
5. FETCH C1
6. ROLLBACK

This rollback deallocates the DGTT because there was no **COMMIT** after it was declared. The **NOT LOGGED** attribute does not apply to log records written during the **DECLARE** and **DROP** of the DGTT.

Example 7

The following example is a sequence of steps for a DGTT, which is defined as **NOT LOGGED ON ROLLBACK PRESERVE ROWS**:

1. DECLARE GLOBAL TEMPORARY TABLE DT1 NOT LOGGED ON ROLLBACK PRESERVE ROWS
2. INSERT INTO DT1 with a subselect
3. COMMIT
4. DECLARE CURSOR C1 on DT1
5. OPEN C1
6. FETCH C1
7. UPDATE where current of C1
8. INSERT into DT1
9. DELETE FROM T2 (T2 is a different table unrelated to DGTT and is logged)
10. ROLLBACK

This roll back causes DB2 to rollback the **DELETE** from T2; however the DGTT, DT1, remains untouched because of the **PRESERVE ROWS** specification.

Example 8

Assume the following SQL statements are executed in the application:

1. DECLARE GLOBAL TEMPORARY TABLE DT1 NOT LOGGED ON ROLLBACK PRESERVE ROWS
2. INSERT INTO DT1 with a subselect
3. DECLARE CURSOR C1 on DT1
4. OPEN C1
5. FETCH C1
6. ROLLBACK

This rollback deallocates the DGTT because there was no **COMMIT** after it was declared. The **NOT LOGGED** attribute does not apply to log records written during the **DECLARE** and **DROP** of the DGTT.

Example 9

Assume that the application has the following sequence of SQL statements:

1. DECLARE GLOBAL TEMPORARY TABLE DT1 NOT LOGGED ON ROLLBACK DELETE ROWS
2. CREATE UNIQUE INDEX on DT1
3. INSERT INTO DT1 (successful)
4. SET SAVEPOINT SP1
5. INSERT INTO DT1 (successful)
6. DECLARE GLOBAL TEMPORARY TABLE DT2 NOT LOGGED ON ROLLBACK DELETE ROWS
7. INSERT INTO DT2 (successful)
8. ROLLBACK TO SP1

In this situation, DB2 deletes rows from DT1 and DT2. DB2 also undoes the declare of DT2. DT1 will still be available.

3.3 More open data sets (DSMAX)

The **DSMAX** subsystem parameter determines the maximum number of data sets that is to be allowed open at one time. DB2 11 increases the maximum number from 100,000 to 200,000. Although the maximum number of concurrently open data sets is 200,000, the practical limit might be significantly less on any given DB2 subsystem, depending on availability of virtual storage below the 2 GB bar. In most cases, a value of 50,000 to 75,000 open data sets is sufficient. The maximum number of 200, 000 open data sets is available in DB2 11 CM and has is retrofitted to DB2 10 through APAR PM88166.

3.4 PBG mapping tables to lift the 64 GB limit

When you reorganize a table space using **SHRLEVEL CHANGE**, you have to use a *mapping table*. For each row, the mapping table records the position of each record (RID) in the original table space and the one that it is to be found on the reorganized shadow copy.

DB2 11 includes the following enhancements related to the mapping tables:

- The mapping table can be created automatically by DB2 during REORG execution.
- The mapping table can grow up to 16 TB.

3.4.1 Autonomic creation of the mapping table

When processing a **REORG TABLESPACE SHRLEVEL CHANGE** request, the **REORG** utility has the option to create its own mapping table and mapping index, instead of relying on a user's input. Use the **REORG_MAPPING_DATABASE** system parameter to specify a valid database name, and do not specify anything for the mapping table on the utility control statement. This action directs **REORG** to allocate the mapping table in the database that is specified.

If the name is valid only in terms of following the naming rules for database names but if the name that you specify is not an existing database in the DB2 subsystem, the **REORG** fails while trying to allocate a table space for the mapping table in this database.

Example 3-16 shows the error message in the utility output in this case.

Example 3-16 Error message for not found map table space

```
DSNUGUTC -  REORG TABLESPACE SABIDB3.SEGMENT SHRLEVEL CHANGE
.35 DSNURMAP - MAPPING DATABASE yourdb IS INVALID
DSNUGBAC - UTILITY EXECUTION TERMINATED, HIGHEST RETURN CODE=8
```

The autonomic creation of mapping tables is described in more detail at 11.1.4, “Automated REORG mapping table management” on page 278.

3.4.2 Mapping tables up to 16 TB

Mapping tables must store information about each row’s source and target RID. Thus, a **REORG** job can hit the limit of 64 GB when very large table spaces are reorganized. Prior to DB2 11, this limit meant a mapping table had to be a segmented table space. With growing data needs, **REORG** jobs can hit the limit more often. DB2 11 allows you to use **PBG** table spaces for storing mapping tables, which practically eliminates the limit.

To determine how much space you need for a **REORG**, make the following calculation for the index:

```
1.1 * Number-of-rows-in-table-space * 31
```

4

Availability

DB2 11 for z/OS continues to bring changes that can improve availability. These new functions keep up with the demands of transaction processing and business intelligence that require on-demand actions and changes without interruption of service. DB2 11 delivers increased application and subsystem availability with more functions for schema evolution, autonomics, governance, and usability.

This chapter describes the following enhancements:

- Online schema changes and enhanced recovery options
- Automatic recovery of indexes from GRECP or LPL status
- Improved availability when altering limit keys
- Work file database enhancements
- Governing of parallel processing of utilities
- Compression dictionary availability for CDC tables
- DROP column support
- Defer define object enhancements
- Allow BIND, REBIND, and DDL to break-in persistent threads
- Idle thread break-in

Other enhancements that indirectly improve availability are described at 2.4, “Reduced need for REORG” on page 15 and 13.3, “Reduced need for REORG” on page 390.

4.1 Online schema changes and enhanced recovery options

With DB2 10, several online schema options allow you to make changes to database objects (indexes and table spaces) while maximizing availability of altered objects. **ALTER** statements (such as changing segment size, data set size, buffer pool with a different page size, the **MEMBER CLUSTER** attribute, and table space type) are among the changes that you can execute while applications are running. These changes are deferred and are not materialized immediately, leaving the altered objects available.

DB2 10 alters are called *pending changes*. Their materialization requires reorganization of the affected objects.

DB2 10 introduced the following supported pending definition changes:

- Alter segment size (**SEGSIZE**) on partition-by-growth universal table space (**PBG UTS**), partition-by-range universal table space (**PBR UTS**), and XML table space.
- Alter data set size (DSSIZE) on partition-by-growth universal table space (PBG UTS), partition-by-range universal table space (PBR UTS), XML table space, and large object (LOB) table space.
- Alter buffer pool with a different page size on partition-by-growth universal table space (PBG UTS), partition-by-range universal table space (PBR UTS), and LOB table space.
- Alter buffer pool with a different page size on index that is associated with universal table space (UTS).
- Alter MEMBER CLUSTER on partition-by-growth universal table space (PBG UTS), and partition-by-range universal table space (PBR UTS).
- Convert single-table classic partitioned table space to partition-by-range universal table space (PBR UTS).
- Convert single-table classic simple table space to partition-by-growth universal table space (PBG UTS).
- Convert single-table classic segmented table space to partition-by-growth universal table space (PBG UTS).

4.1.1 Scope of enhancements for online schema changes in DB2 11

DB2 10 does not allow point-in-time (PIT) recoveries prior to a materializing **REORG** job. DB2 11 lifts this restriction for some of the DB2 10 pending alters. Table 4-1 lists the online schema change types for which PIT recoveries are now possible after a materializing **REORG**.

Table 4-1 PIT recover allowed after materializing REORG

DB2 10 online schema change	LOB auxiliary table space	XML auxiliary table space	PBR table space
ALTER DSSIZE	YES	YES	YES
ALTER PAGESIZE	YES	NO	YES
ALTER SEGSIZE	NO	YES	YES
ALTER MEMBER CLUSTER	NO	NO	YES

Table 4-1, does not include PBG table spaces. For a list of restrictions that are currently in place, refer to 4.1.7, “Determine if a table space is eligible for PIT recovery prior to REORG” on page 59.

Prerequisite note: The prerequisite for being able to use the new functionality is that the materializing REORG job is executed in DB2 11 New Function Mode. Thus, this enhancement is available in New Function Mode (NFM). In addition, for table spaces that have gone through the supported schema changes in DB2 10 NFM or DB2 11 conversion mode (CM) and that have had the materializing **REORG**, even in DB2 11 NFM you cannot do a PIT to any point before that **REORG**.

4.1.2 How it works

Starting in DB2 11 NFM, in the cases listed in Table 4-1, you can recover an LOB table space, XML table space, or PBR to a point in time (PIT) that is prior to the materialization of supported DB2 10 table space attributes pending definition changes. After the **RECOVER** job is executed, the table space is placed in **REORP** restrictive state. You must run a subsequent **REORG** on the entire table space to finalize the point-in-time recovery process.

Figure 4-1 shows the details of a PIT recovery.

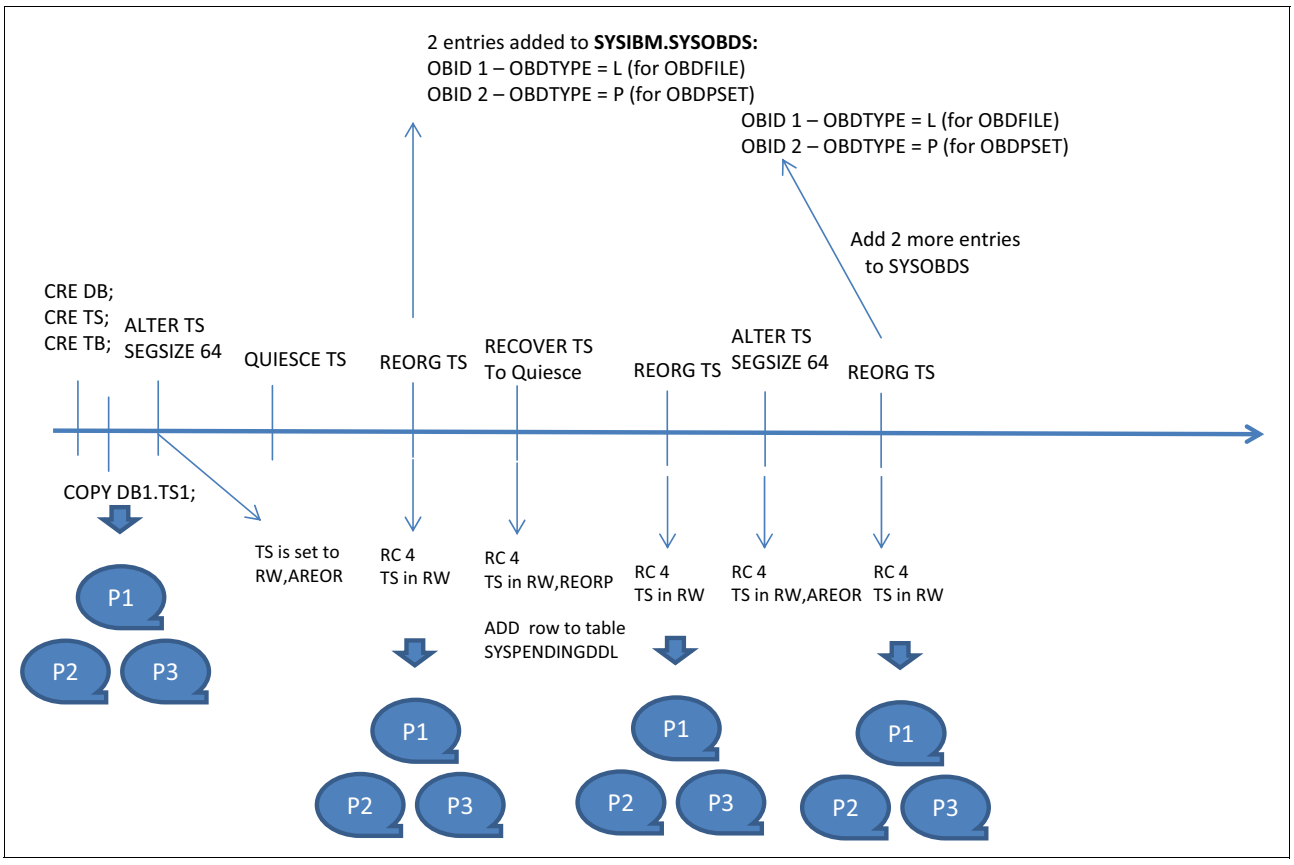


Figure 4-1 PIT Recovery after materializing REORG of DB2 10 change

This example starts with the creation of a database, a table space, and table. Then, it inserts some rows, as shown in Example 4-1 for the DDL.

Example 4-1 DDL for table creation

```
CREATE DATABASE SABIDB4;
CREATE TABLESPACE SABITS1 IN SABIDB4 NUMPARTS 3 ;
CREATE TABLE SABITB4 LIKE SYSIBM.SYSCOPY
```

```
PARTITION BY (DBNAME)
(PARTITION  1 ENDING AT ('AAAAAA'),
 PARTITION  2 ENDING AT ('EEEEEE'),
 PARTITION  3 ENDING AT ('ZZZZZZ'))
 IN SABIDB4.SABITS1                    ;
 COMMIT;
 INSERT INTO SABITB4 SELECT * FROM SYSIBM.SYSCOPY ;
```

Next, the example takes an image copy. If you have the FlashCopy technology available, take a FCIC.

Then, the example executes one of the DB2 10 **ALTER** statements listed in Table 4-1 as eligible for this enhancement. In this case the **SEGSIZE** of the table space is changed from 32 to 64. As a result of this change, the table space is placed in **AREOR** status. The **QUIESCE** utility that runs immediately after the **ALTER** facilitates an RBA for subsequent recovery.

To materialize the pending change and to get rid of the **AREOR** status, run the following command:

```
REORG TABLESPACE SABIDB4.SABITS1 SHRLEVEL REFERENCE
```

An inline image copy is mandatory for a **REORG SHRLEVEL REFERENCE**. Thus, **REORG** materializes the pending changes and also produces an image copy. In addition, starting with DB2 11, the materializing **REORG** leads to entries in the **SYSIBM.SYSOBDS** table. This table stores information about object definitions as they were at that time. The effect of this online schema change is that the following rows are added to the **SYSIBM.SYSOBDS** table:

- One row with OBDTYPE=L, which contains information about an **OBDFILE**. **OBDFILE** entries describe the file object descriptor (OBD).
- A second row with OBDTYPE=P, which contains information about an **OBDPSET**. **OBDPSET** records describe either a data page set or index page set OBD.

Tip: Refer to *DB2 11 for z/OS Diagnosis Guide and Reference,* LY37-3222 for details about the object descriptors and the layout of the **SYSIBM.SYSOBDS** table.

Note: One of the columns in the **SYSIBM.SYSOBDS** table is **VERSION**. This column is the version of the original object for an OBD image that was captured during the **ALTER** that creates a new version. This value is -1 if this row is inserted because of a materializing **REORG**.

The next step is to actually do the recovery to the RBA of the QUIESCE point. The recover works fine, but leaves the table space in **REORP** status because the contents of the image copy do not match the structure of the table space as it is currently described in the catalog. **REORP** is a restrictive state. Thus, this state does not allow any interaction with the underlying table space. In addition to setting the **REORP** status, a row is added to the **SYSIBM.SYSPENDINGDDL** table, which indicates that there are necessary actions to take for this table space. The row that is inserted into the **SYSIBM.SYSPENDINGDDL** table contains the information listed in Table 4-2.

Table 4-2 SYSPENDINGDDL entry after RECOVER to PIT before materializing REORG

Column name	Value
DBNAME	Database Name

Column name	Value
TSNAME	Table Space Name
DBID	DBID
PSID	PSID of table space
OBJSCHEMA	Database name
OBJNAME	Table space name
OBJOBID	Table space file OBID
OBJTYPE	‘S’
STATEMENT_TYPE	‘R’
OPTION_ENVID	Default to 0
OPTION_KEYWORD	‘**TOLOGPOINT**’ or ‘**TORBA**’
OPTION_VALUE	RBA/LRSN value to be recovered to as a string of characters
OPTION_SEQNO	1
CREATEDTS	Time stamp of when **RECOVER** was executed
RELCREATED	Current release of DB2
IBMREQD	‘N’
ROWID	Generated by DB2
STATEMENT_TEXT	Empty string
COLNAME	Empty string
PARTITION	Default to 0
PARTITION_KEYWORD	Empty string

Note: The only way to remove this entry from the **SYSIBM.SYSPENDINGDDL** table is to run a **REORG SHRLEVEL CHANGE** or **REFERENCE**.

ALTER TABLESPACE ... DROP PENDING CHANGES does not help for entries that are added as a result of **RECOVER TABLESPACE** to a PIT before a **REORG** that materialized some of the online schema changes.

In addition to the entries in the **SYSIBM.SYSPENDINGDDL** and **SYSIBM.SYSOBDS** tables, these steps also generate entries in the **SYSIBM.SYSCOPY** tables. Selecting the rows that are produced shows the information in Example 4-2.

Example 4-2 Selecting from SYSCOPY

```
SELECT ICTYPE, STYPE, TTYPE, OLDEST_VERSION, DSNUM
FROM SYSIBM.SYSCOPY
ORDER BY TIMESTAMP DESC

---------+---------+---------+---------+---------+-------
ICTYPE  STYPE  TTYPE     OLDEST_VERSION        DSNUM
```

```
---------+---------+---------+---------+---------+-------
P        C                            -1            0
F        W                             0            0
W                 F                    0            0
A        S        00000032             0            0
Q        W                            -1            0
F        T        C                    0            0
F        T        C                    0            3
F        T        C                    0            2
F        T        C                    0            1
C        L        E                    0            0
```

This information includes a couple of new entry types. **TTYPE** for the **ALTER** (row 4) shows the **SEGSIZE** setting that was in place at the time the **ALTER** was executed. The **OLDEST_VERSION** column contains a -1 for the QUIESCE and the PIT.

Note: The value -1 for **OLDEST_VERSION** for the QUIESCE and the PIT is set for entries for which the version information really does not matter.

To make the table space fully available again, now run a **REORG** with **SHRLEVEL REFERENCE** or **CHANGE**.

REORG SHRLEVEL NONE is not allowed in this situation, which is different behavior from the situation in which the table space also has one or more entries in **SYSIBM.SYSPENDINGDDL** and is currently placed in **AREOR** status. When you try to run **REORG SHRLEVEL NONE** in one of those situations, **REORG** runs but does not materialize the pending change. In this situation, with **REORP** due to **RECOVER**, you receive the following message:

```
DSNU2921I -DB1A 212 13:37:42.18 DSNURFIT - OPTION SHRLEVEL REFERENCE IS REQUIRED
ON TABLESPACE SABIDB4.SABITS1 TO COMPLETE THE POINT-IN-TIME RECOVERY PROCESS
```

If you specify **SHRLEVEL CHANGE**, instead of **REFERENCE**, DB2 accepts it, but under the covers it executes a **SHRLEVEL REFERENCE** and lets you know about this change through the following message:

```
DSNU124I  -DB1A 212 14:11:42.09 DSNURFIT - SHRLEVEL CHANGE SPECIFICATION IS
IGNORED AND SHRLEVEL REFERENCE IS IN EFFECT FOR CURRENT UTILITY EXECUTION
```

You have to make sure that you reorganize the entire table space to finalize the PIT recovery process. A partition level recovery is not an option here, and you are notified about it the a **DSNU256I** message.

Next in this scenario shown in Figure 4-1 on page 53, **ALTER** the **SEGSIZE** of the table space again, which again leads to **AREOR** advisory state. To remove the **AREOR** status and to materialize this change, run **REORG SHRLEVEL REFERENCE/CHANGE** again. This **REORG** now leads to two additional rows in the **SYSIBM.SYSOBDS** table. Because the structure of the table space has changed again, DB2 now has to save the table space layout as it was before the materializing **REORG** to enable you to run subsequent PIT recoveries at a later point in time.

4.1.3 Effect of MODIFY RECOVERY

As described previously and as shown in Figure 4-1 on page 53, pending definition changes now also lead to new entries in the **SYSIBM.SYSOBDS** catalog table. Because this table grows large over time, you can remove entries using the **MODIFY RECOVERY** utility. This utility removes entries that correspond to the rows that are being removed from the **SYSIBM.SYSCOPY** catalog table also, as shown in Figure 4-2.

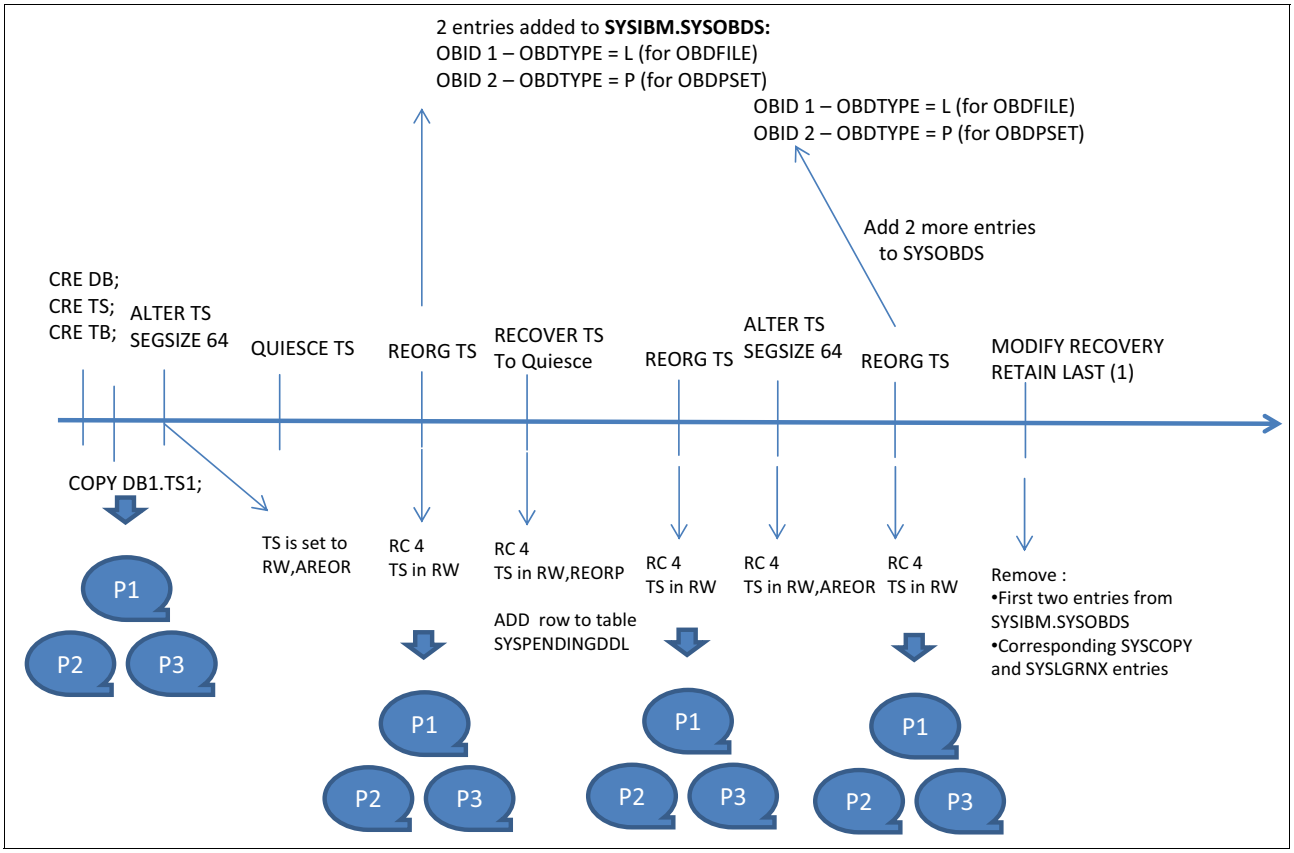


Figure 4-2 MODIFY RECOVERY scenario

Using this utility with a table space in REORP status: The **MODIFY RECOVERY** utility is not allowed if a table space is in **REORP** restrictive status after a **RECOVER** job was run to recover the data to a PIT before the materialization of pending definition changes on the table space. You must run **REORG** before you can use the **MODIFY RECOVERY** utility on this table space again.

4.1.4 Considerations for LOBs

Although Table 4-1 on page 52 lists some of the online schema changes for LOBs that qualify for PIT recoveries after materializing **REORG**, this section describes the specialties that come along with LOBs and XML table spaces.

If you have recovered a table space that contains LOB columns and if the LOB table spaces have undergone materialized online schema changes, to finalize the PIT recovery process you must reorganize the LOB table spaces first after RECOVER. If your base table space did not have any materialized online schema changes then there is no need to also reorganize it

after the **RECOVER**. If after the **RECOVER** you check the table space status for both the LOB and base table space or spaces, you can see that only the ones with materialized online schema changes are set to **REORP** status. However, if your base table space is in **REORP** status after the PIT recovery, you must reorganize the base table space. This reorganization then pulls the LOB table spaces again and reorganize them again. The reason for that is that AUX YES is set on default due to the **REORP** restrictive status on the base table space.

If by accident you do not follow this described order, that is with LOB table spaces first, then the reorganization of the base table space fails with a return code 8 and a DSNU1159I message. If the base table space is not in **REORP** but the LOB table space is and if you run a **REORG** on the base table space, which includes the reorganization of the LOB table space or spaces, the **REORG** runs and completes with RC 4. In addition, if you look at the job output, you do not find any indication that the **REORP** status has not been reset. Instead the **REORG** utility issues the following message:

```
AUXILIARY TABLESPACE DSN00051.LI8ZSE08 WILL BE REORGANIZED IN PARALLEL WITH THE
BASE TABLEPACE
```

However, if you check the status of the LOB table space after the reorganization of the base table space is complete, notice that the LOB table space is still in **REORP** status.

4.1.5 Restrictions for the window between PIT recovery and REORG

PIT recovery includes the following restrictions:

- You cannot execute any **CREATE**, **ALTER**, **RENAME**, and **DROP TABLE** statements on the table space objects, objects contained in the table space, and any auxiliary objects associated with the table space if the subsequent **REORG** has not been executed yet.
  - That is, pending definition changes are not allowed on the table space objects, objects contained in the table space, indexes on tables in the table space, and any auxiliary objects associated with the table space during the window between the PITR and the subsequent **REORG**.
  - The following actions are not allowed:
    - **CREATE AUXILIARY TABLE**
    - **CREATE INDEX**
    - **CREATE TABLE**
    - **DROP TABLE** (of base TS, associated LOB TS, associated XML TS)
    - **RENAME INDEX** (of index on base table, auxiliary index, XML index, and so on)
    - **RENAME TABLE**
    - **ALTER INDEX** (of index on base table, auxiliary index, XML index, and so on)
    - **ALTER TABLE** (of base TS, associated LOB TS, associated XML TS)
    - **ALTER TABLESPACE** (of base TS, associated LOB TS, associated XML TS)

  Existing `SQLCODE -20385` with new reason code 28 is issued.
- The only utilities that are allowed when the table space is in **REORG**-pending (**REORP**) restrictive status during the window between the PITR and the subsequent **REORG** are:
  - **REORG**
  - **RECOVER** to the same point in time (PIT), that means that you could to another PIT to the same RBA or **LOGPOINT** or a **RECOVER** to **CURRENT**
  - **REPORT RECOVERY**
  - **REPAIR DBD**

All the other utility jobs (**COPY**, **COPYTOCOPY**, **MERGECOPY**, **MODIFY RECOVERY**, **LOAD**, and **UNLOAD**) fail and **DSNU933I** new message returns code 8 (RC=8) to indicate that the object needs to be reorganized to make the object descriptor consistent.

4.1.6 More restrictions for PIT recovery after materializing REORG

Consider the following **RECOVER** restrictions for recovering a table space to a PIT prior to a materializing **REORG** job:

- You cannot recover an index to a PIT prior to the materialization of pending definition changes.
- If allowed, the PIT recovery must always be done for the entire table space.
- PIT recovery prior to a materializing **REORG** is not allowed if there are outstanding unmaterialized pending definition changes on the object. If you need to do the PIT recovery, you can do an **ALTER TABLESPACE .... DROP PENDING CHANGES**.
- If you use **VERIFYSET NO**[1] an such a PIT recovery, DB2 overrides it and executes **VERIFYSET YES** instead.
- The table space that you want to recover to a PIT before the materialization must not contain a **CLONE** table.
  - If a **CLONE** exists, you must drop the relationship first. When you try to recover nevertheless without dropping you see the message:

    ```
    DSNU1319I -DB1A 212 18:17:19.78 DSNUCAIN - POINT-IN-TIME RECOVERY IS NOT
    ALLOWED BECAUSE TABLESPACE DSN00051.LI8ZSE08 CONTAINS A CLONE TABLE
    ```

  - If you executed **EXCHANGE DATA BETWEEN TABLE** base and clone past the point to which you want to recover, **RECOVER** is not allowed. You get the message:

    ```
    DSNUCASA - RECOVER CANNOT PROCEED FOR TABLESPACE DSN00051.SABITB6
    BECAUSE A SYSIBM.SYSCOPY RECORD HAS BEEN ENCOUNTERED WHICH HAS
    DBNAME=DSN00051 TSNAME=SABITB6 DSNUM=0 ICTYPE=A STYPE=E
    STARTRBA=X'00000000000014797BB88' LOWDSNUM=0 HIGHDSNUM=0
    ```

4.1.7 Determine if a table space is eligible for PIT recovery prior to REORG

There are many restrictions for a PIT recovery to a PIT prior to the materialization of a pending change. If you do not want to run into surprises when you attempt a PIT recovery, you need to analyze the situation. Right now there is no easy way to determine whether a PIT recovery will work for a given table space. The **REPORT RECOVERY** utility does not provide help in terms of available issues. The only thing that might help is the eye-catcher ##, which surrounds the **ICTYPE** information for all table spaces for which a materializing **REORG** was run previously.

Thus, the best way to check whether a PIT is feasible is based on the entries in the **SYSIBM.SYSCOPY** catalog table, which includes the necessary information for the **RECOVER** utility.

[1] Specifies whether the **RECOVER** utility verifies that all related objects that are required for a PIT recovery are included in the **RECOVER** control statement. This option applies to point-in-time recoveries of base objects and related objects. VERIFYSET NO behavior is always in effect for PIT recoveries of catalog and directory objects.

4.2 Automatic recovery of indexes from GRECP or LPL status

This improvement applies only to a special situation in which DB2 has to mark an index as rebuild pending (**RBDP**). A **RBDP** restrictive state can greatly affect the system and application availability. Thus, although it is a specific issue and might not happen often, this function is an improvement in DB2 11. In a small timing window, if there is an index tree structural modification in progress in a situation in which the index is put into LPL or **GRECP** status, the LPL or **GRECP** recovery might fail, as illustrated in Figure 4-3.

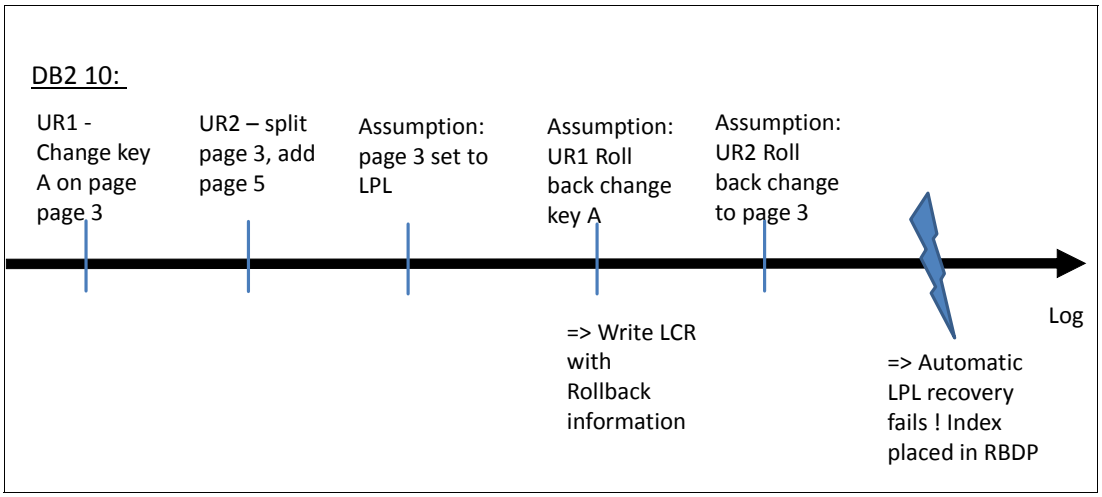


Figure 4-3 RBDP after failing LPL or GRECP recovery in DB2 10

This situation can lead to the **RBDP** restrictive state of the index space, which then will require a potentially long **REBUILD INDEX** utility execution.

In the example, assume that UR1 changes a key A on page 3 of the index space. UR2 does some work that leads to a needed split of page 3. Now, for some reason page 3 is put in LPL state. While in LPL, furthermore assume that the change of key A on page 3 is rolled back. The rollback is noted on the log as compensation log record (CLR). In addition, UR2 also rolls back the page split. If automatic LPL recovery occurs, DB2 first applies the rollback of the change for key A, and then the rollback of the split to page 3, which then leads to the loss of the information regarding the rollback of key A. Because this behavior is not correct, DB2 sets RBDP on the index space.

This described behavior is an availability exposure addressed by DB2 11 with a two-pass LPL/GRECP recovery. The behavior works as follows:

- The CLR is skipped in the first pass. The **UNDO** of the page split is applied first.
- Then in a second pass the CLR is applied and the **UNDO** is skipped. As a result, the LPL recovery can complete correctly and the **RBDP** status can be avoided.

If DB2 uses this two-pass LPL/GRECP recovery, you are notified by a **DSNI051I** message, which indicates the starting point of the second pass **LOG APPLY** during LPL or GRECP recovery.

Attention: After completion of the two-pass LPL/GRECP recovery, DB2 sets **ICOPY** status for **COPY YES** indexes. This setting keeps DB2 from applying the logs that are written before the image copy taken time and, therefore, avoids marking an index into **RBDP** also. Thus, to avoid running into a situation where copy-enabled indexes are set to **ICOPY** and you cannot use the **RECOVER INDEX** utility, monitor the system log for a **DSNI051I** message.

4.2.1 RESTORE SYSTEM after two-pass LPL/GRECP recovery has occurred

With the start of a two-pass LPL/GRECP recovery, DB2 writes a new log record type to indicate that this index has gone through the two-pass LPL or GRECP recovery.

If there are indexes that have gone through the two-pass LPL or **GRECP** recovery earlier or if the indexes are still in LPL or **GRECP** state and the CLRs are written before physical undo logs, these indexes are left in rebuild pending state after the **RESTORE SYSTEM** utility.

4.2.2 RECOVER INDEX after two-pass LPL/GRECP recovery has occurred

If you recover a **COPY YES** index that has gone through the two-pass LPL or GRECP recovery and if the **RECOVER** utility needs to apply the logs processed by the two-pass LPL or GRECP recovery, or if the index is still in LPL or GRECP state, but the LCLRs are written before physical undo logs, this index is left in rebuild pending state after the **RECOVER** utility. Thus, you still need to rebuild this index after the **RECOVER** utility. However, the occurrence of this type of situation is rare.

4.3 Improved availability when altering limit keys

For range-partitioned tables, data rows are stored in data partitions based on the user-specified partition limit key values. After using the table space for a certain time, the partition boundaries that you originally decided to use might not fit well anymore due to data skew or data growth. As a consequence, you might have need to redistribute data across adjacent partitions by using one of the following methods:

- Run the **REORG TABLEPACE** command with the **REBALANCE** option, which was introduced with DB2 8.
- Use the **ALTER** option to change the limit key values with the **ALTER TABLE DDL** statement.

In DB2 10, when you alter the limit keys for a range-partitioned table space, all affected partitions are immediately placed in a **REORG** pending restrictive state. When table space partitions are in **REORP** status, the partitions either need to be loaded with **LOAD REPLACE** or reorganized immediately.

To improve data availability, DB2 11 handles the alter of limit keys for range-partitioned table as pending changes. Thus, similar to many other pending changes introduced starting with DB2 10 and continuing in DB2 11, the limit key values for affected partitions are not applied immediately but are recorded in the **SYSIBM.SYSPENDINGDDL** DB2 catalog table.

Index-controlled partitioning: The information in this section applies to universal range-partitioned (PBR) and classic table-controlled range-partitioned table spaces. If you still use index-controlled partitioning, you must switch to table-controlled partitioning first. The easiest way to accomplish this step is to **ALTER** the clustering index to **NOT CLUSTER** and then **ALTER** it back to **CLUSTER**.

To illustrate the needed step, the following statements create a table with three partitions:

```
CREATE TABLESPACE SABITS1 IN SABIDB3 NUMPARTS 3 ;
CREATE TABLE SABITB3 LIKE DSN81110.EMP
 PARTITION BY (EMPNO)
 (PARTITION  1 ENDING AT ('000020'),
  PARTITION  2 ENDING AT ('000040'),
```

```
  PARTITION  3 ENDING AT ('999999'))
  IN SABIDB3.SABITS1                    ;
```

Issue the following command:

```
ALTER TABLE SABITB3 PARTITION 1 ENDING AT (‘000020)
```

DB2 responds with the following message:

```
DSNT404I SQLCODE = 610, WARNING:  A CREATE/ALTER ON OBJECT SABINE.SABITB3 HAS
         PLACED OBJECT IN ADVISORY REORG PENDING
```

Checking the **SYSIBM.SYSPENDINGDDL** table, a row is added to it, but the table space is not placed in **AREOR**. In fact there is nothing that needs to be changed. The row that is placed in the **SYSIBM.SYSPENDINGDDL** table shows the following characteristics:

- OBJTYPE column = 'T'
- PARTITION column = n (physical partition number of the affected partition)
- PARTITION_KEYWORD = “ALTER”
- OPTION_KEYWORD = “ENDING AT”
- OPTION_VALUE = “xxxx”, which is the limit key

Because the previous alteration did not really change the limit key for the table, you can change partition 1 from ‘000020’ to ‘000040’ in the next approach. This change results in the following misleading error message:

```
DSNT408I SQLCODE = -636, ERROR:  RANGES SPECIFIED FOR PARTITION 2 ARE NOT VALID
```

This table definition attempts to alter partition 1 to the high key that is currently specified for partition 2. Thus, to set the limit key for partition 1 to ‘000040’, the limit key for partition 2 must be increased first. You can accomplish this increase and the increase for partition 1 in one UR or in separate URs. After successful execution of the mentioned two alters, the entire table space is set to **AREOR** advisory state, as shown in Example 4-3.

Example 4-3 AREOR for all three partitions of the PBR

```
DSNT360I  -DB1A *********************************
 DSNT361I  -DB1A *  DISPLAY DATABASE SUMMARY
                 *    GLOBAL
 DSNT360I  -DB1A *********************************
 DSNT362I  -DB1A     DATABASE = SABIDB3  STATUS = RW
                     DBD LENGTH = 4028
 DSNT397I  -DB1A
 NAME     TYPE PART  STATUS            PHYERRLO PHYERRHI CATALOG  PIECE
 -------- ---- ----- ----------------- -------- -------- -------- -----
 SABITS1  TS    0001 RW,AREOR
     -THRU      0003
 SABITS1  TS
 ******* DISPLAY OF DATABASE SABIDB3  ENDED      **********************
 DSN9022I  -DB1A DSNTDDIS 'DISPLAY DATABASE' NORMAL COMPLETION
 ***
```

At this point nothing has really changed. To materialize the changed limit keys, run **REORG** with either **SHRLEVEL CHANGE** or **SHRLEVEL REFERENCE**. Within the **REORG** utility output, notice the following messages, which indicate that in fact the **REORG** materializes the pending changes:

```
DSNU2916I - PENDING ALTER LIMIT KEY VALUES ARE BEING MATERIALIZED
DSNU1163I - APPLYING PENDING DEFINITION CHANGES COMPLETE FOR SABIDB3.SABITS1
```

The reorganization ends with return code 4 if it completes successfully. A **SYSIBM.SYSCOPY** record with ICTYPE=A and STYPE=K is inserted for each affected data partition.

Note: If you decided to change the limit keys yourself, rather than letting DB2 do it through **REORG ... REBALANCE**, with the **REBALANCE** keyword, is not allowed if pending limit key changes are waiting for its materialization. However, if you change your mind after you issued the **ALTER TABLE** statements and you prefer to let DB2 do the rebalancing, you still have to option to use **ALTER TABLESPACE .... DROP PENDING CHANGES**. The **DROP PENDING CHANGES** option removes all entries for this table space from **SYSIBM.SYSPENDINGDDL** and removes the **AREOR** status.

Packages of the underlying table are invalidated by the **REORG** utility.

This function is available in NFM and it only applies to UTS PBR and table controlled partitioning. An **ALTER LIMIT KEY** on index controlled partitioned table spaces would set them in **REORGP**.

APAR PM89655 adds the following new **DSNZPARM** values to help in this situation:

- The **PREVENT_ALTERTB_LIMITKEY** system parameter is used to disable altering of limit key values through an **ALTER TABLE** statement for index-controlled partitioned table spaces. The default value is **NO**, the existing behavior. This system parameter takes effect in DB2 11 for z/OS NFM (and is ignored in DB2 11 for z/OS CM). Such altering would cause the table space to be placed in reorganization pending (**REORP**) status.
- The **PREVENT_NEW_IXCTRL_PART** system parameter is used to prevent creation of new index-controlled partitioned tables. The default value is NO, the existing behavior. This system parameter takes effect in DB2 10 for z/OS NFM (and is ignored in DB2 10 for z/OS CM). Table controlled partitioning should be used.

4.3.1 Considerations for tables containing LOBs

Starting with DB2 10, if you request FlashCopy image copies as inline copies, the image copies are created for both, the base table space and the LOB or AUX table spaces.

FlashCopy image copies are used if:

- You set the subsystem parameter FLASHCOPY_REORG_TS to YES and omit the FLASHCOPY keyword on the REORG table space control statement
- You use whatever setting for subsystem parameter FLASHCOPY_REORG_TS and specify FLASHCOPY YES on the utility control statement.

When requesting a FlashCopy inline image copy, you must specify a **TEMPLATE** prior to the **REORG** statement.

Sample utility control statements are:

```
TEMPLATE SCOPY1 UNIT(SYSDA) DISP(MOD,CATLG,CATLG)
SPACE=(10,10) TRK
DSN(DB2R8.&SN..D&JDATE..T&TIME..P&PART.)
REORG TABLESPACE DSN00063.PARTTB
SHRLEVEL REFERENCE
AUX YES COPYDDN(SCOPY1)
```

4.3.2 LOAD REPLACE

Prior to DB2 11 NFM, all **ALTER** limit key executions are immediate in a sense that the new limit keys are stored in the catalog immediately, and the affected table space partitions are placed into a **REORP** restrictive state. The **REORG** or **LOAD REPLACE** can be used to materialize alter limit key changes by reformatting the data sets and removing the **REORP** status on the affected data partitions.

However, with DB2 11 NFM you get the new behavior with pending alter limit keys. The only way to materialize pending changes is using the **REORG** utility. So a **LOAD REPLACE** can no longer be used for the materialization of new limit keys after you are in NFM. If you run a **LOAD REPLACE** while you still have unmaterialized pending changes for your limit keys, the load should run successful but does load new data records based on the existing limit key values prior to the pending alter limit key changes.

4.3.3 RECOVER

A PIT recovery across an online **REORG** that materialized the pending alter limit key changes is now supported. However, be aware that the data partitions for which you applied limit key changes before the **REORG** are placed in **REORP** restrictive state. In the **RESTORE** phase, all rows are processed on a page-per-page basis. Thus, all image copied pages of partition 1 are going back to partition 1, all copied pages of partition 2 are written to the VSAM cluster of partition 2, and so on, as shown in Figure 4-4.

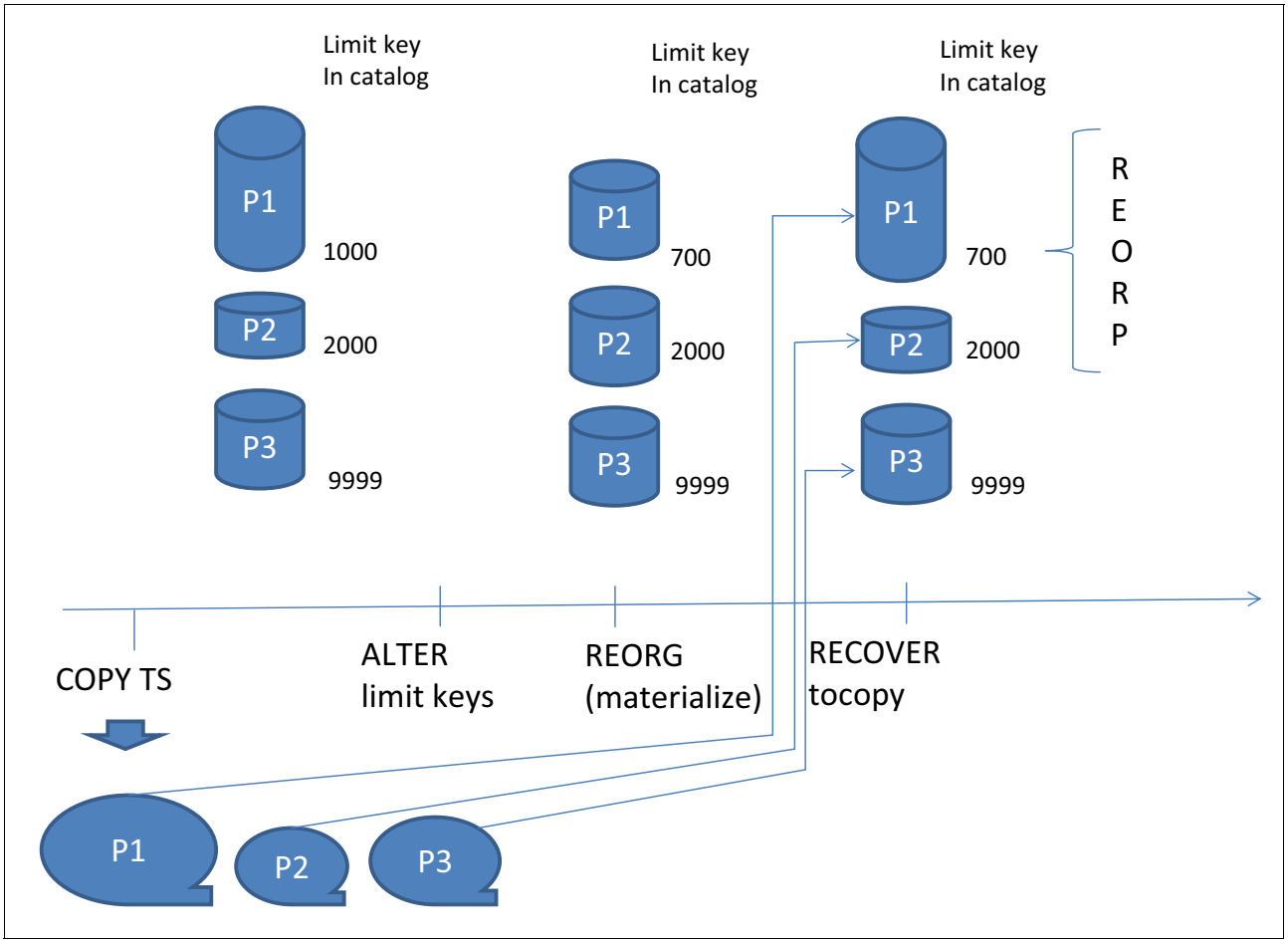


Figure 4-4 RECOVER with ALTER LIMIT option

In addition, the limit keys that are stored in the catalog are still the limit keys that you set earlier using **ALTER TABLE**. With the recover you just recover the data to a prior PIT and not the schema. So after the **RECOVER**, there is a mismatch between which rows are expected in each of the partitions and the data that is really there. You can resolve this mismatch using a **REORG** on the affected table space partition.

Important: Only partitions 1 and 2 are placed in **REORP** status, because the **ALTER** of the limit key for partition 1 affected only these two partitions. As a consequence, you reorganize only these two partitions to make the table space available.

4.4 Work file database enhancements

Each DB2 subsystem has a **WORKFILE** database. In a data sharing system, each data sharing member has its own **WORKFILE** database.

DB2 uses the **WORKFILE** database to keep the declared global temp tables (DGTTs) and other non-DGTT temporary data, such as created global temp tables (CGTTs) and DB2 work data, to support queries by SQL applications that need to perform SORTs that are held in 'work files. DGTTs are used for DGTTs declared by external applications, DB2-internal implementation of static scrollable cursors, DB2-internal implementation instead of triggers, and so on.

A **WORKFILE** database can have up to 500 table spaces for all kinds of temporary data together. A table space in the **WORKFILE** database can be created as a PBG table space that is partitioned and segmented or as a classic segmented table space that is non-partitioned and segmented. PBG tables spaces are DB2-managed. Non-PBG table spaces can be either DB2-managed or user-managed or both. Each table space in the **WORKFILE** database can hold multiple DGTT tables or work files (depending on their size). A single work file's (non-DGTT) data can reside in multiple **WORKFILE** database table spaces (that is one work file can span up to 255 table spaces) if necessary.

However, a single DGTT cannot span multiple table spaces. That is, a DGTT can use storage in its initially-assigned table space only. If a DGTT is assigned a table space that is also shared by work files, it is possible that the DGTT can run out of necessary space. To alleviate storage shortage for DGTTs, DB2 currently implements a preference logic in the selection of **WORKFILE** database table spaces for DGTTs versus work files, as follows:

For DGTTs, DB2 attempts to allocate storage from a list of DB2-defined table spaces with non-zero secondary quantity (`SECQTY = -1` or `SECQTY > 0`) first, before looking for storage from table spaces with a zero secondary quantity. The selection criteria for work files is in the opposite direction. DB2-defined table spaces with `SECQTY = 0` or user-defined table spaces are first used for work files.

This preference logic works effectively only if the **WORKFILE** database contains a mixture of DB2-managed table spaces with zero and non-zero secondary quantities and user-defined table spaces.

DB2 10 includes the **WFDBSEP** system parameter, which isolates certain type of work file table spaces for DGTT or non-DGTT use only.

If **WFDBSEP** is set to **YES**, then:

- PBG table spaces and DB2-managed non-PBG (classic segmented) table spaces with a non-zero secondary quantity (that is, SECQTY = -1 or > 0) are reserved for *DGTT use only* (as DGTT-Preferred).
- DB2-managed non-PBG (classic segmented) table spaces with a zero secondary quantity (SECQTY = 0) and user-managed non-PBG table spaces are reserved for *work files use only*.

If the preferred table spaces are not available or they do not have sufficient storage, they get a SQLCODE -904 resource unavailable error. Even though DB2 10 keeps track of the **WORKFILE** database space usage through some statistic trace records, there are no alerts issued by DB2 when critical storage shortage conditions are approached either at agent level or at system level.

DB2 11 is enhanced to allow you to tell DB2 using two new system parameters which we are going to describe on the following few pages, to issue warning messages when **WORKFILE** database space-usage approaches a critical level. These system parameters allow you to monitor space used by DGTTs and “regular” work files separately.

4.4.1 WFSTGUSE_AGENT_THRESHOLD subsystem parameter

DB2 11 introduces a new subsystem parameter **WFSTGUSE_AGENT_THRESHOLD** to define the agent-level space-usage alert threshold. The system parameter is online-changeable.

The **WFSTGUSE_AGENT_THRESHOLD** subsystem parameter determines the percentage of available space in the work file database on a DB2 subsystem or data sharing member that can be consumed by a single agent before DB2 issues a warning message.

The percentage can range from 0 to 100, with a default value of 0, which basically means that this function is not used, that is DB2 does not issue agent-level space usage warnings for the work file database. For a value greater than 0, refer to Table 4-3 to determine how this **DSNZPARM** influences DB2 behavior.

Table 4-3 WFSTGUSE_AGENT_THRESHOLD

WFDBSEP setting	DB2 behavior
YES	Issue a warning message in the following situations: - When the percentage of total *temporary work file* space for one agent reaches or exceeds the percentage of **WFSTGUSE_AGENT_THRESHOLD**. - The percentage of total configured work file-storage in the **WORKFILE** database consumed by one agent reaches or exceeds the percentage set in **WFSTGUSE_AGENT_THRESHOLD**.
NO	When an agent’s combined total used storage for DGTTs (temporary work files) and sort work files reaches or exceeds the percentage set in **WFSTGUSE_AGENT_THRESHOLD**

Table 4-4 shows a few scenarios to demonstrate DB2 behavior in detail.

Table 4-4 WFSTGUSE_AGENT_THRESHOLD sample

WFDBSEP	Total Sort work file defined	Total Temp work file defined	WFSTGUSE_AGENT _THRESHOLD	WFDB storage used by agent	Warning issued?
YES	1 GB	1 GB	10%	20% Sortwork	YES
YES	1 GB	1 GB	10%	20% Temp	YES
YES	1 GB	1 GB	10%	7% Sortwork 7% Temp	NO
YES	1 GB	1 GB	10%	10% Sortwork 5% Temp	YES
YES	1 GB	1 GB	10%	4% Sortwork 7% Temp	NO
NO	1 GB	1 GB	10%	20% Sortwork	YES
NO	1 GB	1 GB	10%	20% Temp	YES
NO	1 GB	1 GB	10%	7% Sortwork 7% Temp	YES
NO	1 GB	1 GB	10%	10% Sortwork 5% Temp	YES
NO	1 GB	1 GB	10%	4% Sortwork 7% Temp	YES

Example 4-4 shows message **DSNI052I**, which was issued as a result of reaching the specified threshold for one agent. In this case, the setting for **WFDBSEP** is **NO**. As a result, the message **FOR DECLARED GLOBAL TEMP TABLES AND WORK FILES** displays in Example 4-4.

Example 4-4 WFSTGUSE per agent message

```
DSNI052I  -DB1A DSNISGNS AN AGENT HAS EXCEEDED THE  519
THRESHOLD FOR STORAGE USE
        IN WORK FILE DATABASE DSNDB07
        FOR DECLARED GLOBAL TEMP TABLES AND WORK FILES.
        THRESHOLD = 1 PERCENT.
        TOTAL STORAGE CONFIGURED = 336 KB
        CONNECTION ID = DB1A
        CORRELATION ID = SABINE
        LUWID = USIBMSC.SCPDB1A.CBC5C3757ECE=21
        PACKAGE NAME = DSNESM68
        PLAN NAME = DSNESPCS
```

Note: DB2 issues this message only once per commit scope. So do not wait for repeated messages before you take action.

The total space configured for a table space in the **WORKFILE** database is determined during restart based on the **PRIQTY** and **SECQTY** specified on **CREATE TABLESPACE**. For the calculation DB2 assumes that the user has made the defined storage available for all the table spaces using a storage group. One exception occurs for user-defined table spaces. DB2 can only calculate the space for those objects when it opens the page set.

Note: If DB2 has not opened all user-defined table space objects for the **WORKFILE** database yet, DB2 does not know about the sizes for these objects. Therefore, it might happen that an agent has used up a high percentage of the available space in **DSNDB07** but that a warning is not issued, because one or two small user-defined work files are not opened yet.

4.4.2 WFSTGUSE_SYSTEM_THRESHOLD subsystem parameter

The **WFSTGUSE_SYSTEM_THRESHOLD** subsystem parameter determines the percentage of available space in the work file database on a DB2 subsystem or data sharing member that can be consumed by all agents before a warning message is issued.

Acceptable values are in the range between 0 to 100. The default is set to 90%. Example 4-5 sets the threshold to a low value of 2%. The message that you get in case the system-wide threshold is reached is slightly different from that in Example 4-4. Example 4-5 shows the message for the system-level space warning.

Example 4-5 WFSTGUSE per system message

```
DSNI053I  -DB1A DSNISGNS THE DB2 SUBSYSTEM HAS  521
EXCEEDED THE THRESHOLD FOR
        STORAGE USE IN WORK FILE DATABASE DSNDB07
        FOR DECLARED GLOBAL TEMP TABLES AND WORK FILES.
        THRESHOLD = 2 PERCENT.
        TOTAL STORAGE CONFIGURED = 336 KB.
```

When this value is 0, DB2 does not issue system-level space usage warnings for the work file database.

When the value is not equal zero, DB2 behaves as listed in table Table 4-5.

Table 4-5 DB2 behavior for WSTGUSE based on WFDBSEP setting

WFDBSEP setting	DB2 behavior
YES	Issue a warning message in the following situations: ► When the total system level storage used for DGTTs reaches or exceeds the **WFSTGUSE_SYSTEM_THRESHOLD** percentage of the total configured DGTT-storage in the **WORKFILE** database. ► The total system-level storage used for work files reaches or exceeds the **WFSTGUSE_SYSTEM_THRESHOLD** percentage of the total configured work files-storage in the **WORKFILE** database.
NO	Issue a warning message when the total system level storage used for DGTTs and work files together reaches or exceeds the **WFSTGUSE_SYSTEM_THRESHOLD** percentage of the total configured storage in the **WORKFILE** database.

Note: The **DSNI053** message is issued at 5 minutes interval, if the criteria for issuing the message continues to exist. You cannot influence this interval.You can adjust this interval if you get too many messages in the MSTR address space using the following methods:

- Increase the percentage for the **WFSTGUSE_SYSTEM_THRESHOLD** system parameter, but keep in mind the implications.
- Increase the number or size of the **WORKFILE** database table spaces.

The total space configured for a table space in the **WORKFILE** database is determined during restart based on the **PRIQTY** and **SECQTY** specified on **CREATE TABLESPACE**. For the calculation DB2 assumes that the user made the defined storage available for all the table spaces using a storage group. One exception occurs for user-defined table spaces. DB2 can calculate only the space for those objects when it opens the page set. Refer to Table 4-6 for more details.

Important: If DB2 has not opened all user-defined table space objects for the **WORKFILE** database, DB2 does not know about the sizes for these objects. Therefore, it might happen that an agent has used up a high percentage of the available space in **DSNDB07** but that a warning is not issued, because one or two small user-defined work files are not opened.

Table 4-6 Maximum WORKFILE storage configured

Table space	PRIQTY	SECQTY	Maximum storage configured for the table space
PBG		-1	**DSSIZE * MAXPARTITIONS GB**
PBG		> 0	**DSSIZE * MAXPARTITIONS GB**
PBG		0	**MIN(PRIQTY, DSSIZE) GB**
DB2-managed Non-PBG	< 2 GB	-1	64 GB
DB2-managed Non-PBG	< 2 GB	> 0	64 GB
DB2-managed Non-PBG	< 2 GB	0	**PRIQTY GB**
DB2-managed Non-PBG	>= 2 GB		64 GB
User-managed Non PBG with number of data sets > 1			Number of data sets * 2 GB
User-managed Non PBG with number of data sets > 1		0	**PRIQTY GB**
User-managed Non PBG with number of data sets > 1		> 0	2 GB

4.4.3 Systems programmer response to DSNI052I/DSNI053I

Determine whether the warning threshold that is specified by subsystem parameter **WFSTGUSE_SYSTEM_THRESHOLD** or **WFSTGUSE_AGENT_THRESHOLD** are too low.

If this is not the case, take one or both of the following actions:

- Create additional table spaces in the work file database of the object types that are specified in this message.

  Use the **DSNTWFG** exec in job step **DSNTIST** of installation job **DSNTIJTM** to create additional table spaces.
- Change the value of subsystem parameter **WFDBSEP** from **NO** to **YES**.

4.5 Governing of parallel processing of utilities

DB2 processes various tasks for various utilities in parallel. Sometimes you have to ask for it, and sometimes DB2 determines the possible degree of parallelism by itself. The following utilities can have some work run in parallel:

- **REORG TABLESPACE**
- **REBUILD INDEX**
- **CHECK INDEX**
- **UNLOAD**
- **LOAD**

This utilities include a new **PARAMDEG_UTIL** subsystem parameter that allows you to limit the number of parallel tasks for utilities for a data sharing member.

REORG TABLESPACE

REORG TABLESPACE uses parallel index build if more than one index needs to be built (including the mapping index for **SHRLEVEL CHANGE**). You can either let the utility dynamically allocate the data sets that **SORT** needs for this parallel index build or provide the necessary data sets yourself. The number of subtasks must be less than or equal to the number that is specified by the **PARALLEL** option. If you do not specify the **PARALLEL** option, the **PARAMDEG_UTIL** subsystem parameter determines the maximum degree of parallelism for the utility.

Neither the **PARALLEL** nor **PARAMDEG_UTIL** affect the degree of parallelism used by the unload phase of **REORG**.

REBUILD INDEX

For **REBUILD INDEX** the **PARALLEL(num-subtasks)** keyword specifies the maximum number of subtasks that are to be started in parallel to rebuild indexes. If the **PARALLEL** keyword is omitted, the maximum number of subtasks is limited by either the number of partitions being unloaded or the number of indexes built.

REBUILD INDEX typically allocates subtasks in groups of two or three, so the actual number of subtasks that are started might be less than the number specified on **PARALLEL**.

The specified number of subtasks for **PARALLEL** always overrides the specification of the **PARAMDEG_UTIL** subsystem parameter. Thus, **PARALLEL** can be smaller or larger than the value of **PARAMDEG_UTIL**.

The **num-subtasks** value specifies the maximum number of subtasks and must be an integer between 0 and 32767, inclusive. If the specified value for num-subtasks is greater than 32767, the **REBUILD INDEX** statement fails. If 0 or no value is specified for num-subtasks, the **REBUILD INDEX** utility uses the optimal number of parallel subtasks. If the specified value for num-subtasks is greater than the calculated optimal number, the **REBUILD INDEX** utility limits the number of parallel subtasks to the optimal number with applied constraints.

Incompatibility: In DB2 11 conversion mode, the degree of parallelism can increase for the **REBUILD INDEX** utility.

The **REBUILD INDEX** utility previously limited the degree of parallelism to 18 subtasks. Now, because of the **PARALLEL** option value or the **PARAMDEG_UTIL** subsystem parameter value, the amount of parallelism might increase.

CHECK INDEX

If you specify more than one index, **CHECK INDEX** checks the indexes in parallel unless they are constrained by available memory, sort work files, or the **PARALLEL** option. Sorting the index keys and checking multiple indexes in parallel, rather than sequentially, reduces the elapsed time for a **CHECK INDEX** job.

If you do not specify the **PARALLEL** option, the **PARAMDEG_UTIL** subsystem parameter determines the maximum degree of parallelism for the utility.

UNLOAD

The unload utility also has a new **PARALLEL** keyword. This keyword specifies the maximum number of subtasks to be used in parallel to process the unloading of a partitioned table space. If the **PARALLEL** keyword is omitted, the maximum number of subtasks possible is determined by the number of partitions being unloaded.

The **(num-subtasks)** value specifies the maximum number of subtasks that are to be processed in parallel. The value must be an integer between 0 and 32767, inclusive. If the specified value for num-subtasks is greater than 32767, the **UNLOAD** statement fails. If 0 or no value is specified for num-subtasks, the **UNLOAD** utility uses the optimal number of parallel subtasks after applying constraints. If the specified value for num-subtasks is greater than the calculated optimal number, the **UNLOAD** utility limits the number of parallel subtasks to the optimal number. The specified number of subtasks for **PARALLEL** always overrides the specification of **PARAMDEG_UTIL**, so it can be smaller or larger than the **DSNZPARM** value.

LOAD

For a single input data set, the **PARALLEL** option specifies the maximum number of subtasks that are to be used in parallel when loading a table space from a single input data set and building the indexes. By using parallel subtasks, the utility can potentially reduce the elapsed time of the load operation.

For multiple input data sets, where there is one data set for each partition, the **PARALLEL** option specifies the maximum number of subtasks to be used with loading the data partitions, building the indexes, and gathering statistics. This option applies to classic and range-partitioned table spaces. If the **PARALLEL** option is omitted, the load operation uses the optimal number of subtasks with applied constraints.

The PARALLEL option overrides PARAMDEG_UTIL: The specified number of subtasks for the **PARALLEL** option always overrides the specification of the **PARAMDEG_UTIL** subsystem parameter. Thus, the **PARALLEL** value can be smaller or larger than the value of **PARAMDEG_UTIL**.

Restriction: The **PARALLEL** option is not valid in the following situations:

- For a single input data set, the **LOAD** statement includes any of the following options:
  - **SPANNED YES**
  - **INCURSOR**
  - **PRESORTED**
  - **FORMAT INTERNAL**
  - **COLGROUP**
- The table space to be loaded is a PBG table space.
- The table to be loaded has XML columns and is in a simple or segmented table space, and the **LOAD** statement includes the **SHRLEVEL CHANGE** option.
- The table to be loaded has LOB or XML columns, and the **LOAD** statement includes the **SHRLEVEL NONE** option.

The **LOAD** utility calculates an optimal number of subtasks to process in parallel based on memory constraints, virtual storage constraints, and the number of available processors. If 0 or no value is specified for num-subtasks, the **LOAD** utility uses the optimal number of parallel subtasks. If the specified value for num-subtasks is greater than the calculated optimal number, the **LOAD** utility limits the number of parallel subtasks to the optimal number. If the specified value for num-subtasks is less than the calculated optimal number, the **LOAD** utility uses the specified value. If the specified value for num-subtasks is greater than 32767, the **LOAD** statement fails.

4.6 Compression dictionary availability for CDC tables

The DB2 instrumentation facility interface (IFI) provides the ability to read log records for data replication products to process the changes to a table from insert, update, and delete operations. If the table is compressed, IFI needs a dictionary to decompress the log record data. Compression dictionaries are created during **LOAD** or **REORG** or by using **COMPRESS** on **INSERT**. DB2 uses these compression dictionaries when compressed record information must be read from the log for replication purposes.

However, if you execute a subsequent **REORG** or **LOAD** operation and if you do not specify the **KEEPDICTIONARY** option, DB2 builds a new compression dictionary during the utility execution. This action is OK for all subsequent DML on the table data. However, a problem can occur if a data replication product needs to read log records that require information included in the old compression dictionary. The old compression dictionary is kept in memory as long as DB2 is up and running but is discarded if you have to shut down the DB2 system. In addition, in a data sharing group, the old compression dictionary is kept only in the memory of the member on which you run the **LOAD** or **REORG**. The other members or the data sharing group are not aware of the old compression dictionary, and the information about its existence cannot be shared with different data sharing members.

DB2 11 provides relief to this situation. The change applies only to tables that are created with **CHANGE DATA CAPTURE** and **COMPRESS YES**. If you run **LOAD** or **REORG** for those objects, DB2 now saves the old compression dictionary. DB2 externalizes the old compression dictionary to the log and adds a record with **ICTYPE 'J'** to the **SYSIBM.SYSCOPY** table. The **START_RBA** column of this new records points to the RBA of the data sharing member's log to which the compression dictionary is externalized.

In a data sharing environment, DB2 merges log records if the value of the IFI **READS** qualification **WQALFLTR** is X'00'. If **WQALFLTR** is X'01', log records are not merged. The iinstrumentation facility component identifier (IFCID) can retrieve log records from the archive data sets.

This improvement is available in CM.

4.7 DROP column support

With DB2 you had the ability to complete the following tasks:

- Add a column to an existing table (V1)
- Alter a column on an existing table (V8)
- Rename a column on an existing table (V9)

The ability to drop a column from an existing table has been a requirement because an abandoned column produces the following types of costs:

- Space in every row stored in the table
- Space in every image copy of the table space
- Space taken up in the log records written for the table
- Additional CPU and elapsed time in all aspects of accessing and maintaining the data
- DBA time “remembering” that the column is redundant

Without the ability to drop a column from an existing table, the procedure of getting rid of the redundant column is to schedule an outage, unload the data, drop the table, re-create the table without that column, load the data, and potentially redo grants.

DB2 11 lets you drop existing columns from a table through an **ALTER TABLE ... DROP COLUMN.. RESTRICT** SQL statement if there are no dependent objects such as indexes, triggers, unique or check constraints, row permissions, column masks and so on, defined on this column. Views dependent on the table are implicitly regenerated.

DB2 10 introduced pending changes for online schema changes. An **ALTER TABLE** to drop one or more columns of a table is a pending change. The drop is not materialized immediately after the **ALTER TABLE** statement completes, instead one row for each dropped column is inserted into the **SYSIBM.SYSPENDINGDDL** catalog table and the table space holding the affected table is placed in AREOR advisory pending state.

For example, if the **sc1.table1** table resides in the **DB1.TS1** table space, and the table has 10 columns, with the names **column1...column10**. To drop column10, execute the following **ALTER TABLE** statement:

```
ALTER TABLE sc1.table1 DROP COLUMN column10 RESTRICT
```

Upon execution of the this **ALTER** statement, the entries listed in Table 4-7 are inserted into **SYSPENDINGDDL**.

Table 4-7 Entry values for SYSPENDINGDDL

SYSPENDINGDDL catalog field	Value
DBNAME '	DB1
TSNAME	TS1
DBID	DBIS of DB1

SYSPENDINGDDL catalog field	Value
PSID	PSID of TS1
OBJSCHEMA	sc1
OBJNAME	table1
COLNAME	colun10
OBJOBID	OBID of sc1.table1
OBJTPYE	T
STATEMENT_TYPE	A
COLUMN_KEYWORD	DROP
OPTION_KEYWORD	RESTRICT
OPTION_VALUE	empty string
STATEMENT_TEXT	ALTER TABLE sc1.table1 DROP COLUM column10 RESTRICT

Because it is only an advisory state, the object is fully available for any DML statements. In order to actually materialize the change, you have to run a **REORG TABLESPACE** utility with **SHRLEVEL REFERENCE** or **CHANGE**.

Important: You must execute the **REORG TABLESPACE** utility to materialize the pending **DROP COLUMN** on the entire table space. If you only **REORG** a subset of partitions of a partitioned table space, the **REORG** runs, but the pending changes continue to be pending changes and the table space remains in **AREOR** status.

You can run a **REORG** with **SHRLEVEL NONE** as well while the table space is in **AREOR** status, but **SHRLEVEL NONE** would not materialize the pending change or changes.

The **REORG** with **SHRLEVEL REFERENCE** or **CHANGE** performs the following actions:

- Generate a new table version.
- Update the catalog definition to remove any references to the dropped column.
- Revoke any associated column level privileges.
- Update the data to remove any data for the dropped column.
- Collect statistics for the table space and associated indexes with the default options (**TABLE ALL INDEX ALL UPDATE ALL HISTORY ALL**) if the **STATISTICS** keyword was not specified with the utility.
- Invalidate any packages and dynamic cached statements that are dependent on the table.
- Create a **SYSCOPY** record for the dropped column.
- Remove the pending drop column entry from **SYSPENDINGDDL**.
- Issue a **DSNU1166I** warning message with `RC=4` to indicate that some partition statistics might 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.

Tip: A new option is available on the **RUNSTATS** utility control statement, **RESET ACCESSPATH**. You might want to run **RUNSTATS** with this statement prior to running **REORG TABLESPACE** to materialize the pending **DROP COLUMN**. **RUNSTATS TABLESPACE ... RESET ACCESSPATH** does not gather any statistics. Instead it only resets any access path related statistics in the catalog tables.

- The user should execute the **RUNSTATS** utility to collect the partition statistics again.
- Clear the **AREOR** state from the table space.

4.7.1 Changes to the catalog as a result of dropping a column

As a result of dropping one or more columns from a table, several updates need to be applied to the DB2 catalog. You see the following subset of changes after the successful materializing **REORG**:

- A new table version is generated:
  - **SYSTABLESPACE.CURRENT_VERSION** is updated
  - **SYSTABLES.VERSION** is updated.

  Note: If you drop multiple columns and run **REORG TABLESPACE** only once to materialize this change, only one version is generated for it.

- **SYSTABLES.COLCOUNT** is decreased.
- **SYSTABLES.RECLENGTH** is decreased.
- **SYSCOLUMNS.COLNO** is changed for all subsequent columns.
- If the dropped column is a LOB column, the related auxiliary objects for all partitions are dropped.

4.7.2 Undo a DROP COLUMN

A **DROP COLUMN** cannot be undone after you materialized it using **REORG TABLESPACE**. However, after you execute a pending **ALTER TABLE DROP COLUMN**, and before the drop is materialized by a **REORG** utility, you can remove the pending alter using **ALTER TABLESPACE DROP PENDING CHANGES**.

Important: The removal of the pending alter that you initiated using the **ALTER TABLE** command does not work through another **ALTER TABLE** command but needs to be an **ALTER TABLESPACE** command instead!

In addition, removing pending alters for a given table space is not granular. If you use the **DROP PENDING CHANGES** option on the **ALTER TABLESPACE** command, *all* pending changes for the specified table space are removed.

4.7.3 Impact of DROP COLUMN on utilities

This section describes the impact of **DROP COLUMN** on utilities.

RECOVER

Recovery to a PIT prior to a materializing **REORG** is not allowed. Because the materializing **REORG** must be a **SHRLEVEL REFERENCE** or **SHRLEVEL CHANGE REORG**, the **REORG** produces an image copy. As shown in Figure 4-5, two image copies FC1 and FC2 that existed prior to the materializing **REORG** are no longer valid image copies.

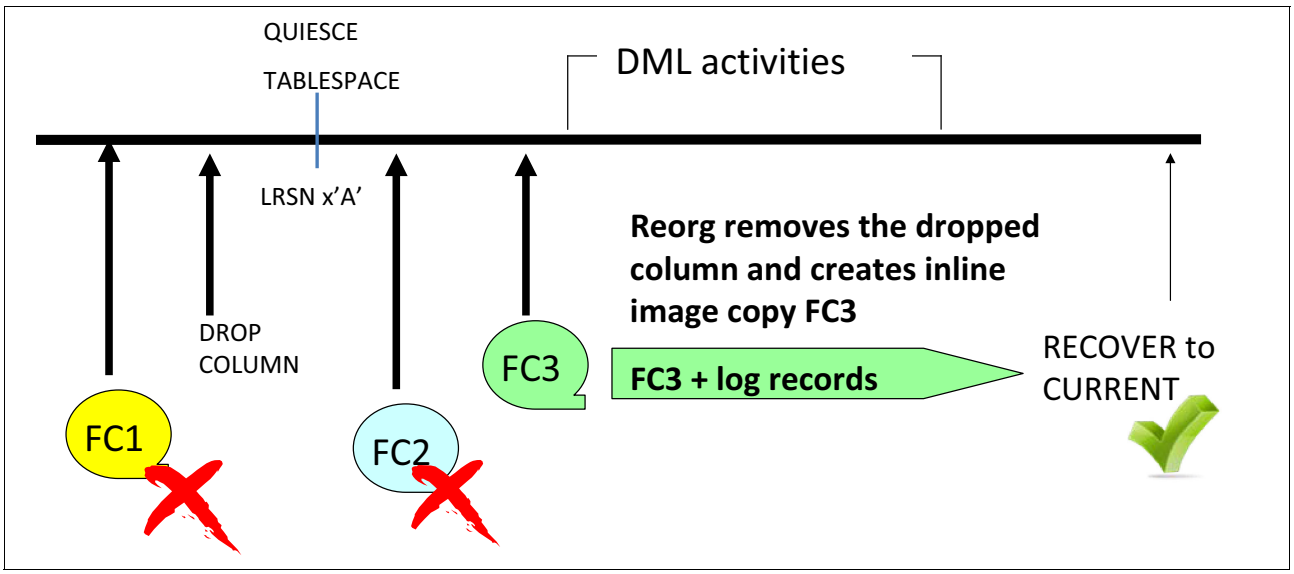


Figure 4-5 RECOVER to CURRENT after DROP COLUMN

In case you need to a recover after the reorganization, this process works properly if you need to recover to **CURRENT**. However, if your want to recover to any point prior to the image copy that was taken with the materializing **REORG**, as shown in Figure 4-6, the process fails with a **DSNU5561I** message and RC8.

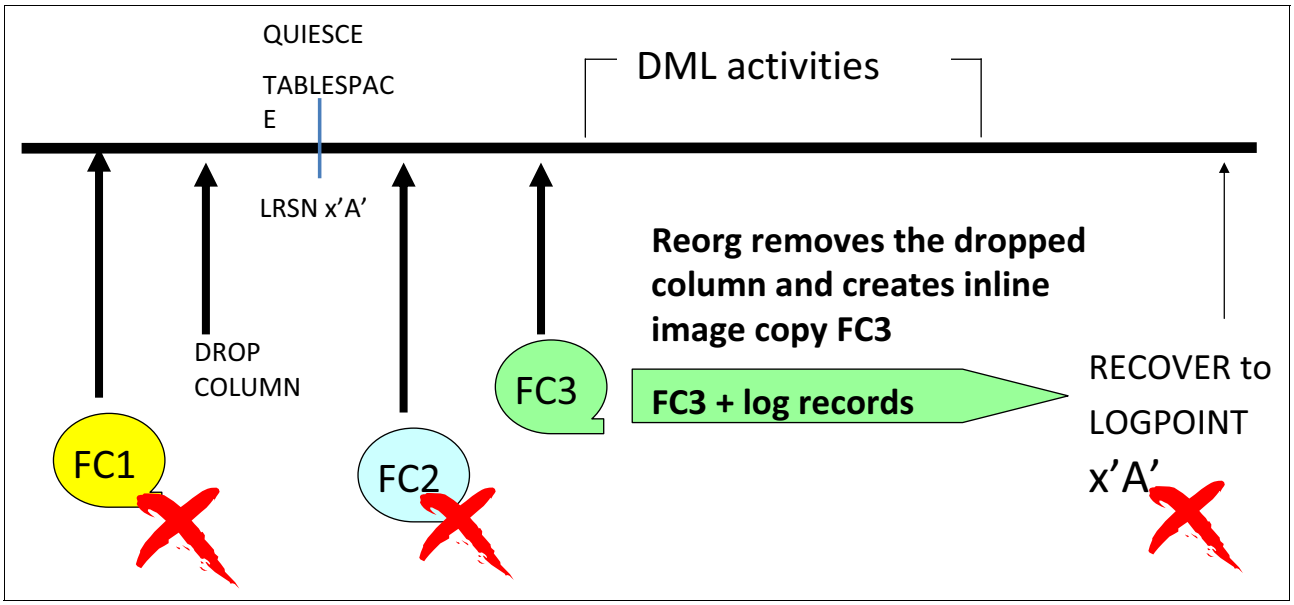


Figure 4-6 RECOVER to LOGPOINT after DROP COLUMN

Tip: Because the image copy that is created during the **REORG** is the only good image copy that you have at this point, create at least two image copies during the **REORG**.

UNLOAD

Unless you do something to the physical data sets of image copies FC1 and FC2 in Figure 4-7 on page 78, they continue to exist even if they are not usable by **RECOVER** any more. As a consequence, you might think that unloading data might work even after the **DROP COLUMN** is materialized. It does not. If the image copy contains data for dropped columns, the attempt to unload ends with **DSNU1227I** and RC8.

DSN1COPY

The **DSN1COPY** utility is another method to copy data from still existing image copies to the table space cluster. If you run the **DSN1COPY** utility and use, for example, image copy FC2 as shown in Figure 4-7, you are using an image copy on which the structure of the data rows is different than the one that is known to the DB2 catalog. The **DSN1COPY** utility is a stand-alone utility. When you run it, DB2 does not check the contents of the input data set and also does not compare it to whatever is defined in the DB2 catalog. As a consequence, a **DSN1COPY** utility execution that copies over the contents of FC2 to the table space cluster, completes with RC 0 if done correctly.

If you subsequently try to select from the table you run into errors if the expected data type does not fit with the encountered data type on the data page.

Figure 4-7 illustrates how the layout of the **EMPLOYEE** table changes after **DROP COLUMN** and the materializing **REORG**. If later on you use **DSN1COPY** to copy the contents of FC1 to the table space cluster of DB1.TS1, no problems occur, because **DSN1COPY** does not check the column layout.

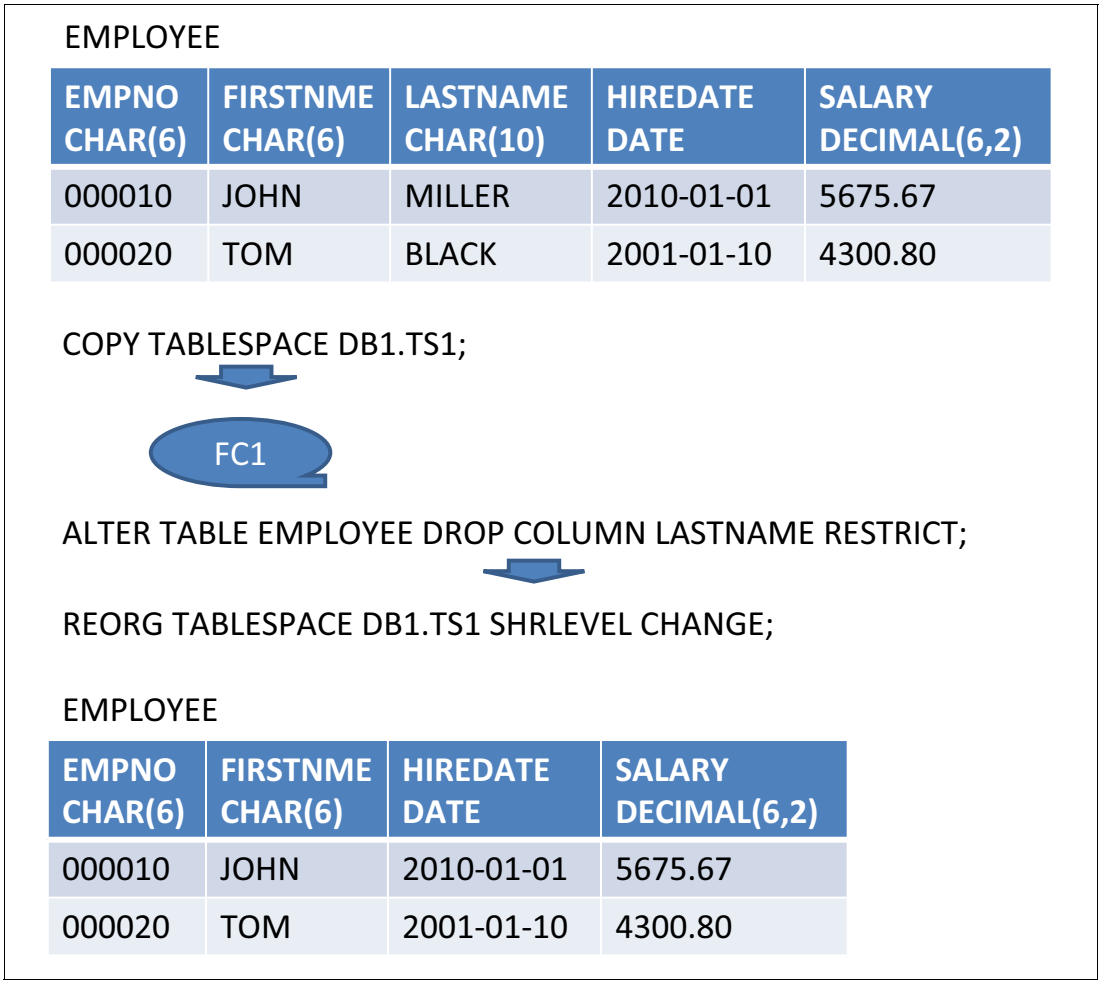


Figure 4-7 TABLE layout after dropping a column

As shown in Figure 4-8, a subsequent usage of the **EMPLOYEE** table might or might not be successful. If you just selected from columns that are before the dropped column, you get the correct results back. If you also select from column, which were repositioned in the table order, there is a high potential that the attempt to read the data will fail. Even if it does not fail, because the data types fit with the column definition, you will definitely get the wrong data returned, that is the information that was previously stored in the dropped column.

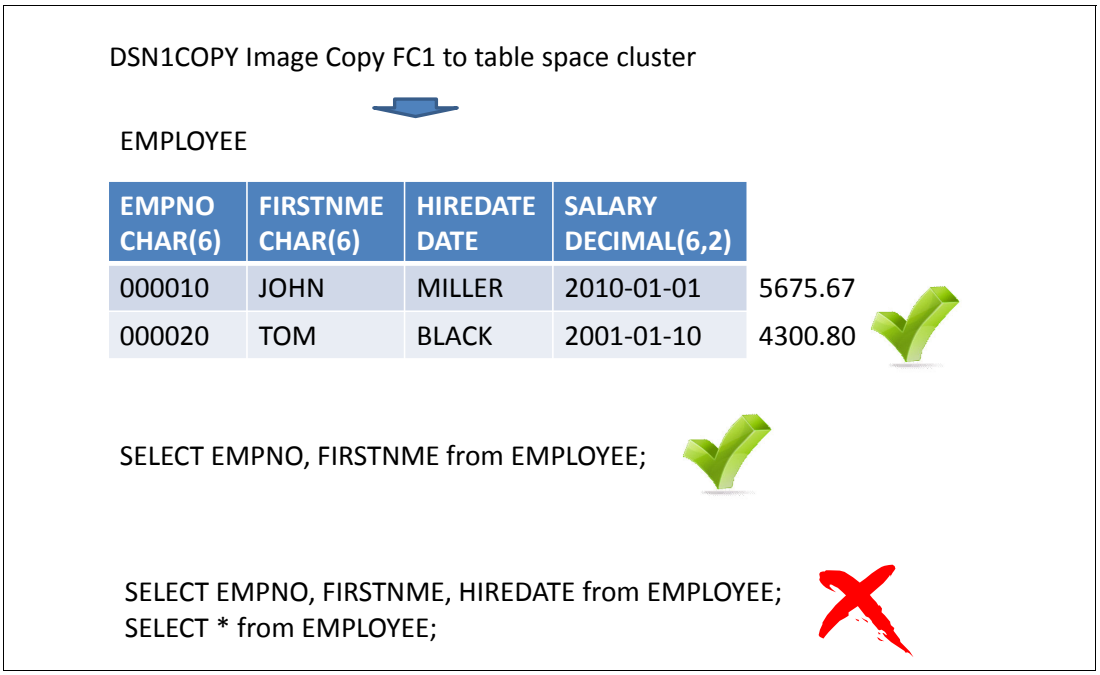


Figure 4-8 Effect of DSN1COPY for a table with dropped column

4.7.4 Impact of DROP COLUMN on applications

Dropping a column is not just a matter of executing the new **ALTER** statement introduced by DB2 11. It also needs planning and careful analysis and communication.

Any packages and statements in the dynamic statement cache that are dependent on the table holding the column that you are planning to drop are invalidated after you run the materializing **REORG** statement. If system parameter **ABIND** is set to **YES** or **COEXIST**, an automatic rebuild occurs the next time the application is called. The rebind succeeds if the application does not have any reference to the column that you dropped. In case you missed any packages that are in fact still referencing the column or columns, the automatic rebuild fails and the package is set to inoperative status. Thus, you have to correct the failing SQL statement in the package and execute a **BIND** package to reactivate the package in question.

The analogous is true for dynamic SQL statements, which are invalidated in the dynamic statement cache.

4.7.5 Restrictions for DROP COLUMN

Keep in mind the following restrictions to the **DROP COLUMN** function:

- Only allowed for tables residing in a UTS.
- Not allowed if the table is an MQT or for tables which are referenced by an MQT
- Not allowed for columns belonging to a system-period temporal table or a history table
- Not allowed for tables with **EDITPROC** or **VALIDPROC** defined on it.

- Not allowed for CGTTs.
- The partitioning key column cannot be dropped.
- The hash key column for a table space defined as organized by hash cannot be dropped.
- **DOCID** columns cannot be dropped.
- **ROWID** columns with **GENERATED BY DEFAULT** or with a dependent LOB cannot be dropped.
- Security label columns (row permissions or column masks) cannot be dropped.

4.8 Defer define object enhancements

The enhancements described in this section are mostly helpful for applications whose database design has the following characteristics:

- Objects are created with the **DEFER YES** option
- Many objects that share one database

For these objects, a problem can occur when users insert or load data into the logically existing tables that are not physically defined. When these tables are first used, DB2 needs to take an exclusive lock for the database descriptor (DBD). This lock is held until the unit of recovery that completes the data successfully creates the cluster and then completes all the inserts or load action. If there are other tables that are also used for the first time, they also require a DBD lock so that there is a high potential that these tables will run into a time out condition.

Figure 4-9 illustrates this situation. UR1 issues an **INSERT** statement, which leads to a DBD lock and a **VSAM DEFINE**. This lock then starts the inserts. The DBD lock is not released until UR1 commits.

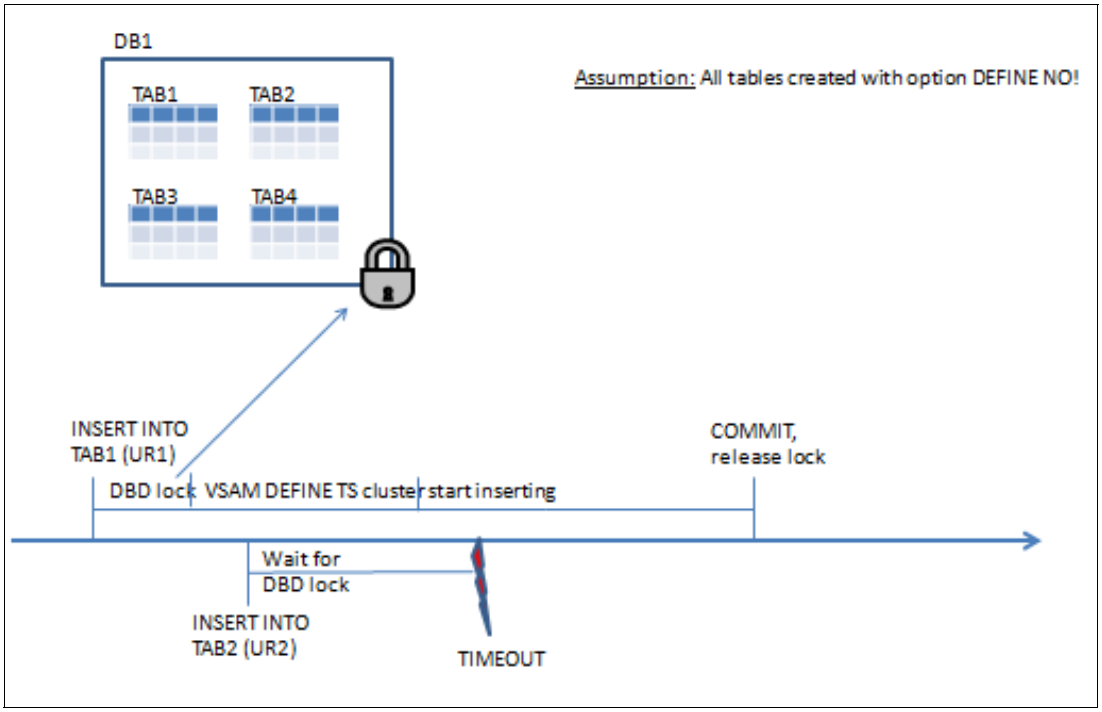


Figure 4-9 DBD lock on first insert

DB2 11 allows the lock to be releases on the database descriptor earlier. Thus, instead of waiting for the inserts to complete the unit of recovery per commit, DB2 11 now releases the lock as soon as the table or index space is physically defined. Refer to Figure 4-10. Because

UR1 releases the DBD lock immediately after the **DEFINE CLUSTER**, UR2 does not time out, but UR2 can itself take the DBD lock, **DEFINE** the cluster for table TAB2, release the lock again, and start with the inserts.

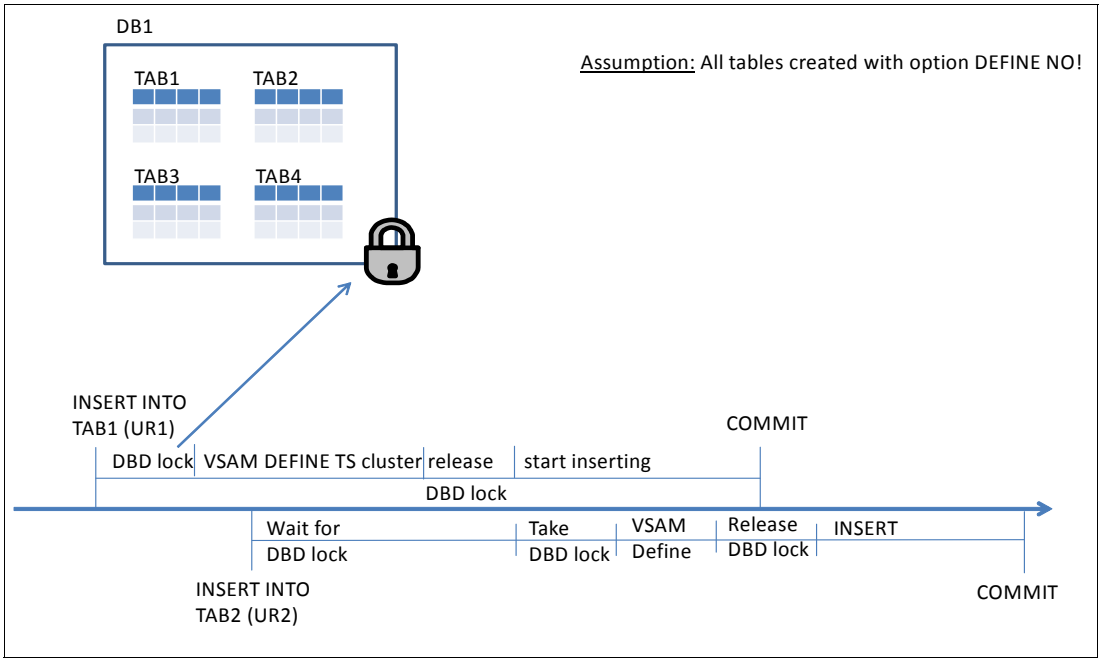


Figure 4-10 DEFER DEFINE enhancement

4.9 Allow BIND, REBIND, and DDL to break-in persistent threads

You need to find windows of opportunity when you can make changes to applications, which sometimes requires that you bind packages and plans or make changes to the database by submitting DDL. Historically, it is difficult to execute a **BIND** command or a DDL statement at times when applications are running, especially applications bound with **RELEASE(DEALLOCATE)**.

One of the **BIND/REBIND** options for **BIND PLAN** and **BIND PACKAGE** is **RELEASE**. The **RELEASE** option determines when to release resources that a program uses, either at each commit point or when the program terminates.

You can set **RELEASE** to **COMMIT**, **DEALLOCATE**, or **INHERITFROMPLAN**. **COMMIT** releases resources at each commit point, unless cursors are held. If the application accesses the object again, it must acquire the lock again. **RELEASE(DEALLOCATE)** releases resources only when the program terminates. **RELEASE(DEALLOCATE)** has the advantage that it requires less CPU (up to 20%) than **RELEASE(COMMIT)**. Alternatively, **RELEASE(COMMIT)** has the advantage of allowing other processes, such as **BIND/REBIND** and the execution of DDL, to break in more easily and use needed resources to complete successfully rather than time out.

DB2 10 introduced the **-MODIFY DDF PKGREL(COMMIT/BNDOPT)** command. When you issue the **-MODIFY DDF PKGREL(COMMIT)** command, DB2 temporarily changes the behavior of DDF threads bound with **RELEASE(DEALLOCATE)** so that they act as though they were bound with **RELEASE(COMMIT)**. The idea behind it allows **BIND**, **REBIND**, and DDL and utilities to break in to persistent threads on **COMMIT** rather than waiting for the application to allocate and therefore undergo the risk of running into time out situations.

DB2 11 introduces a **PKGREL_COMMIT** system parameter that can be set to **YES** or **NO**. **YES** is the default. If set to **YES** it allows a persistent DB2 thread, at **COMMIT** or **ROLLBACK**, to implicitly release a package that is bound with **RELEASE(DEALLOCATE)** and active on that thread if there is a **BIND REPLACE/REBIND PACKAGE**, online schema change operation (for example, DDL), or online **REORG** with deferred **ALTER** operation that needs to quiesce or invalidate the package. So think of it as though you are tapping the executing holder of the thread on the shoulder, and when it comes to their next commit, they change to **RELEASE(COMMIT)** for that thread.

Important: This behavior is available only for **ACTIVE** persistent threads. If the thread is waiting for some other things to complete and is not currently **ACTIVE** in DB2, the BIND/REBIND/DDL/Utility is not able to break in.

This described behavior is not available for the following circumstances:

- Packages that have OPEN and HELD cursors at the time of the **COMMIT** or **ROLLBACK**
- Packages that are bound with **KEEPDYNAMIC(YES)**
- **COMMIT** or **ROLLBACK** options that occur within a DB2 stored procedure

The **PKGREL_COMMIT** system parameter is online changeable.

This option is available only in NFM.

4.10 Idle thread break-in

RELEASE is a **BIND** option that tells DB2 how to handle the caching of package locks and structures. The default is setting for this **BIND** option is **COMMIT**. Thus, DB2 frees package structures upon commit. The alternative option is **DEALLOCATE**. With **RELEASE(DEALLOCATE)** package structures persist until full thread deallocation.

RELEASE(DEALLOCATE) potentially can improve performance in the following circumstance:

- Long running batch jobs that **COMMIT** frequently
- Thread reuse for example for CICS protected threads or JCC Type 2 applications.

RELEASE(DEALLOCATE) is not used pervasively for the following reasons:

- Your system might have virtual storage constraints, which was true primarily for DB2 9 and earlier.
- **RELEASE DEALLOCATE** was not allowed for DDF workload from DB2 V6 and DB2 10.
- If application do not release resources upon commit, this means that BINDs or REORGS for example have little chance to break in during long running batch applications.

DB2 10 introduced high performance **DBATS**, the re-enablement of the **RELEASE(DEALLOCATE)** bind option for DDF threads. For the high performance DBATs you have a chance to change the originally picked bind option of **RELEASE(DEALLOCATE)** on the fly. The way to do this is by using the **-MODIFY DDF PKGREL(COMMIT)** command. After you issue this command, new DDF DBAT threads bound with **RELEASE(DEALLOCATE)** behave as though they were bound with **RELEASE(COMMIT)** and, therefore, increase the chances for **BIND**s and **REORG**s to break in at **COMMIT**. The existing DBATs remains as **RELEASE(DEALLOCATE)** until 200 commits.

After you are finished with your planned actions, you can reverse the behavior to **RELEASE(DEALLOCATE)** using **-MODIFY DDF PKGREL(BNDOPT)** command. If you do not issue this command, behavior is reversed back automatically after 200 commits.

DB2 11 continues to improve this situation for processes that have a need to break in. **RELEASE(COMMIT)** mode takes effect on the next **COMMIT** if there is a waiter and not just for new DBATs.

4.10.1 Improvements for DDF threads

As with DB2 10, you enable the improvements for DDF thread using the **-MODIFY DDF PKGREL(COMMIT)** command.

Disabling has changed a bit. You can still use **-MODIFY DDF PKGREL(COMMIT)** command, but in addition, it is reversed to **RELEASE(DEALLOCATE)** much earlier than before. It is automatically done on the next **COMMIT** if there is a waiter on a package lock.

4.10.2 Improvements for non-DDF threads

DB2 11 now also focuses on non-DDF threads.

Active threads

DB2 11 introduces a the **PKRGEL_COMMIT** system parameter. This subsystem parameter specifies whether at **COMMIT** or **ROLLBACK** a persistent DB2 thread releases a package that is active on that thread if certain DB2 operations are waiting for exclusive access to that package.

Idle threads

In DB2 11 with APAR PM95929 applied, if a package lock appears to be at risk of a timeout, a broadcast is sent to all members of a data sharing group to recycle any idle threads from local attaches, that is for example CICS, IBM IMS™, RRSAF and so on. The risk of timeout is indicated at 1/2 of the internal resource lock manager (IRLM) timeout limit (the **IRLMRWT** subsystem parameter).

A thread is eligible for recycling if all of the following statements are true:

- It is at a transaction boundary.
- It is not running in DB2.
- It has not committed or rolled back in an interval that is larger than half of **IRLMRWT**.

The recycle processing fences the API and issues a “dummy” **COMMIT** for the idle thread, which allows package locks to be freed. If an attempt to use the idle thread is made during the recycle processing, it is delayed until the recycle is complete. The recycle is rapid and delays are minimal.

Important: This process does not address threads with held cursor across commits and does not address long running transactions holding package locks.

Maintenance in this area includes APARs PM95929, PM96001, and PM96004.

5

Data sharing

DB2 data sharing can provide the following advantages over 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 11 for z/OS Data Sharing: Planning and Administration*, SC19-4055.

DB2 11 for z/OS provides a number of enhancements to data sharing. These enhancements provide improved availability, scalability, and performance. All of these enhancements are available in DB2 11 conversion mode (CM) unless otherwise noted.

This chapter describes the following enhancements to data sharing:

- Group buffer pool write-around protocol
- Improved castout processing
- Improved DELETE_NAME performance
- Restart light with CASTOUT option
- Locking enhancements
- Index availability and performance
- Group buffer pool write performance
- Automatic LPL recovery at end of restart
- Log record sequence number spin avoidance

5.1 Group buffer pool write-around protocol

DB2 environments that have data sharing enabled can have multiple applications concurrently accessing data from any member of a data sharing group, with many members potentially reading and writing the same data. When multiple members of a data sharing group open the same table space, index space, or partition, and at least one of them opens it for writing, the data is said to be of inter-DB2 read/write interest to the members.

To control access to data that is of inter-DB2 read/write interest, whenever the data is changed, DB2 caches it in a storage area that is called a group buffer pool (GBP) in the coupling facility. When there is inter-DB2 read/write interest in a particular table space, index, or partition, it is dependent on the group buffer pool, or GBP-dependent (group buffer pool-dependent).

In DB2 data sharing, when batch jobs or utilities run against GBP-dependent objects, it can result in heavy, sustained GBP page write activity. When this happens, the GBP can begin to fill up with changed pages which can result in application slowdowns or, in severe cases, pages being written to the logical page list (LPL), which cause DB2 data outages.

Over the years, this has consistently been one of the top data sharing customer complaints, and many customers, after having suffered through these slowdowns or outages, have over-allocated the GBP structures, which introduces its own set of problems. An over-allocated GBP drives up cost and can result in performance or scalability concerns because of the overly large GBP structure sizes. As a large GBP becomes polluted with lots of changed pages, the resulting flood of castout related CF commands, and in some cases the CF, becomes unresponsive because of this flood of write and castout commands.

DB2 11 addresses these issues by providing a capability to bypass writing pages to the GBP in certain situations and write the pages directly to DASD instead, while using the GBP to send buffer invalidate signals. This feature is referred to as *GBP write-around*.

Two thresholds are used to determine whether GBP write-around is invoked for all objects in a GBP or for a page set/partition: the GBP castout threshold; and the class castout threshold. When the GBP castout threshold hits 50%, meaning that 50% of the GBP is occupied by changed pages, then write-around is invoked for writing pages for all objects. When the class castout threshold hits 20%, meaning that 20% of a class castout queue is occupied by changed pages, then DB2 employs the write-around protocol for the page set/partition. The write-around process at the GBP level continues until the GBP castout threshold drops to 40%. The write-around process at the page set/partition level continues until the class castout threshold drops to 10%. These threshold values are fixed; you cannot change them.

If either threshold is reached, the write-around protocol at the appropriate level is invoked. The processing occurs as follows:

- Changed pages are conditionally written to the GBP. Conditional write means that if a page is already cached in the GBP then the write is allowed to proceed, and if the page is not already cached then the write fails. A **DISPLAY GROUPBUFFERPOOL** command with the **MDETAIL** option shows the number of pages written through a write-around processing in the **DSNB777I** informational message that is displayed as part of the output.
- If the write failed, the page is written to DASD, and the GBP is used to send buffer invalidate signals to the other members.

Example 5-1 shows a **DISPLAY GROUPBUFFERPOOL** command for a buffer pool that includes an object that was updated from one member and selected from another member, which forces the object to become GBP-dependent. Following the command is partial output from the command, including the **DSNB777I** message.

Example 5-1 DISPLAY GROUPBUFFERPOOL output with write-around statistics

```
                              DB2 COMMANDS                    SSID: D1B1
===>

Position cursor on the command line you want to execute and press ENTER

Cmd 1 ===> -DISPLAY GROUPBUFFERPOOL(GBP0) MDETAIL


DSNB750I  -D1B1 DISPLAY FOR GROUP BUFFER POOL GBP0 FOLLOWS
DSNB755I  -D1B1 DB2 GROUP BUFFER POOL STATUS
             CONNECTED                                 = YES
             CURRENT DIRECTORY TO DATA RATIO           = 5
             PENDING DIRECTORY TO DATA RATIO           = 5
             CURRENT GBPCACHE ATTRIBUTE                = YES
             PENDING GBPCACHE ATTRIBUTE                = YES

DSNB777I  -D1B1   ASYNCHRONOUS WRITES
                CHANGED PAGES               = 0
                CLEAN PAGES                 = 0
              FAILED DUE TO LACK OF STORAGE  = 0
              WRITE-AROUND PAGES            = 0
```

The benefit of GBP write-around is that DB2 automatically detects the flooding of writes to the GBP, and automatically responds by dynamically switching to the GBP write-around protocol for those objects that are causing the heaviest write activity. Only the deferred writes are affected. Commits continues to write GBP-dependent page sets to the GBP. After the GBP storage shortage is relieved, DB2 resorts back to normal GBP write activity for all GBP-dependent objects.

If a page is already in the GBP, the deferred writes will update the page in the GBP rather than writing the page to DASD (conditional write).

GBP write-around does not solve the underlying I/O subsystem issues that contributed to the GBP being flooded. The I/O subsystem problem will remain, however, it doesn't matter that the batch updates slow down a little bit as long as the COMMITs perform better. But, it is possible that eventually the COMMITs themselves will flood the GBP if the batch updates continue to flood the I/O subsystem.

This support is provided in z/OS 1.12 and above, with Coupling Facility Control Code (CFCC) Level 17 and 18 on z196 and later hardware.

5.2 Improved castout processing

In prior versions of DB2, data sharing environments with heavy write activity cause pages to be written to the group buffer pools faster than the castout engines can process them. As a result, the group buffer pools become congested with changed pages and, in extreme cases, group buffer pool full conditions might occur. This inefficient castout processing often results

in application response time issues. DB2 11 provides the following enhancements to make castout processing more efficient.

- Reduced wait time for I/O completion
- Reduced notify message size sent to castout owners
- More granular class castout threshold

5.2.1 Reduced wait time for I/O completion

The read of the GBP for castout processing now overlaps the write I/O operation to DASD. In prior versions, DB2 waited until a page read from the GBP was written to DASD before another page was read from the GBP. This wait time is reduced by overlapping the read for castout with the write to DASD.

5.2.2 Reduced notify message size sent to castout owners

The size of the message indicating the status of the castout processing is reduced. Previously, the notification message sent to castout owners was a list of pages, which can be large if many pages are cast out. Now the message includes a list of page sets or partitions, instead of a list of pages, which considerably reduces the size of the message.

5.2.3 More granular class castout threshold

Castout processing now provides more granularity for the class castout threshold. Previously, the class castout threshold was specified as a percentage of the number of data entries. The smallest allowable number was 1%. For a really large GBP, a value of 1% still results in thousands of pages being cast out at a time, which can stress the castout engines and the coupling facility. The capability to allow for a value smaller than 1% was needed.

The syntax for the **ALTER GROUPBUFFERPOOL** command is changed to allow the class castout threshold to be specified as either a percentage of the number of data entries or an absolute number of pages. The new syntax for the **CLASST** option is as follows:

CLASST(*class-threshold1*,*class-threshold2*)

You can use the *class-threshold1* variable to represent the class castout threshold in terms of a percentage of data entries. It can be specified as an integer between 0 and 90, representing 0% to 90%. The default value is 5%.

You can use the *class-threshold2* variable to represent the class castout threshold in terms of an absolute number of pages. It can be specified as an integer between 0 and 32767. The default value is 0.

Do not specify a value for both variables. If you do, the value of *class-threshold2* is ignored. If you want to specify the threshold in terms of a percentage of data entries, which was the only behavior prior to DB2 11, specify a non-zero value for the first variable and a zero value for the second variable. If you need to specify a value smaller than 1% for a large GBP, specify a zero value for the first variable and the desired number of pages at which castout should occur for the second variable.

In the data sharing environment for this example, the GBP0 group buffer pool is defined with CLASST values of 5 and 0. Example 5-2 shows the output of a **DISPLAY GBPOOL** command, which is an abbreviation of the **DISPLAY GROUPBUFFERPOOL** command, showing the class castout threshold (**CLASST**) values of 5 and 0.

Example 5-2 DISPLAY GBPOOL command output for percentage based CLASST threshold

```
DSNB750I  -D1B1 DISPLAY FOR GROUP BUFFER POOL GBP0 FOLLOWS
DSNB755I  -D1B1 DB2 GROUP BUFFER POOL STATUS
             CONNECTED                              = YES
             CURRENT DIRECTORY TO DATA RATIO        = 5
             PENDING DIRECTORY TO DATA RATIO        = 5
             CURRENT GBPCACHE ATTRIBUTE             = YES
             PENDING GBPCACHE ATTRIBUTE             = YES
DSNB756I  -D1B1   CLASS CASTOUT THRESHOLD                = 5, 0
             GROUP BUFFER POOL CASTOUT THRESHOLD    = 30%
             GROUP BUFFER POOL CHECKPOINT INTERVAL  = 4 MINUTES
             RECOVERY STATUS                        = NORMAL
```

Example 5-3 shows an example of an **ALTER GROUPBUFFERPOOL** command to specify a class castout threshold such that pages are cast out when 500 changed pages are in the GBP.

Example 5-3 ALTER GBPOOL command to express CLASST in number of pages

```
                            DB2 COMMANDS                    SSID: D1B1
===>
DSNE294I SYSTEM RETCODE=000      USER OR DSN RETCODE=0
Position cursor on the command line you want to execute and press ENTER

Cmd 3 ===> -ALTER GBPOOL(GBP0) CLASST(0,500)
```

Example 5-4 shows the output of the **ALTER GBPOOL** command.

Example 5-4 ALTER GBPOOL command output showing CLASST in number of pages

```
DSNB804I  -D1B1 CLASS CASTOUT THRESHOLD SET TO 0,500 FOR GBP0
DSN9022I  -D1B1 DSNB1CMD '-ALTER GBPOOL' NORMAL COMPLETION
***
```

You can only expect to see a benefit to using the second threshold value for a GBP that is large enough that it is causing a delay at castout time.

5.3 Improved DELETE_NAME performance

After all pages for a page set are cast out, the page set becomes non-GBP-dependent. DB2 then uses a cache **DELETE_NAME** request to delete both data and directory entries from the GBP. At the same time, cross-invalidation signals are sent to each member that has interest in the page set to indicate that there is no longer GBP-dependency for that page set.

Normally this cross-invalidation process completes without issue. In rare cases, there might be a high number of **DELETE_NAME** requests due to time-outs when sending cross-invalidation signals to data sharing members when there is a long distance between the members and the coupling facility.

In prior versions, DB2 delivered some maintenance that deleted only the data entries to avoid timeouts caused by cross invalidation. However, these enhancements did not provide the desired effect.

DB2 11 resolves this issue by suppressing the cross-invalidation signals during the processing of **DELETE_NAME** requests.

All the necessary cross-invalidation signals have already been sent when the pages were previously written to the GBP. The cross-invalidations for **DELETE_NAME** are not needed after casting out the pages; therefore, the cross-invalidation signals can be suppressed.

New features in cross-system extended services for z/OS (XES) and CFCC are required to use the DELETE_NAME performance enhancement. The specific requirements are as follows:

- The GBP must be allocated in a coupling facility of CFLEVEL=17 or higher. The “suppress cross-invalidation” functionality is supported on the following combinations of hardware and CFCC levels:
  - z114 (2818) DR93G CFCC EC N48162 CFCC Release 17 at the requisite microcode load (MCL) level
  - z196 (2817) DR93G CFCC EC N48162 CFCC Release 17 at the requisite microcode load (MCL) level
  - zEC12 (2827) CFCC Release 18
- The DB2 member that performs castout must be running on z114 or z196 that supports the “suppress cross-invalidation” functionality or a zEC12. The following z/OS releases support the “suppress cross-invalidation” functionality:
  - z/OS V1R12 and later with APAR OA38419 installed

This feature has also been retrofitted to DB2 9 and DB2 10 through APAR PM67544.

5.4 Restart light with CASTOUT option

If a failed DB2 subsystem was running on a z/OS image that is no longer available, that subsystem can hold locks on behalf of transactions that were executing at the time of the failure. If those locks were global locks, which are locks on resources that are actively shared, then the update type locks in the lock structure are changed to retained locks. In this scenario, it is critical to restart the failed DB2 in another z/OS image in the same Parallel Sysplex (where another member might be active) to release the retained locks. Otherwise, no other access is allowed to the resources protected by those locks until the underlying changes are either committed or backed out.

When restarting a failed DB2 subsystem on another z/OS image, the other z/OS image might not have the resources to handle the workload of an additional DB2 subsystem. The option to restart DB2 in **LIGHT** mode enables DB2 to restart with a minimal storage footprint to quickly release retained locks and then terminate normally.

Prior to DB2 11, restart light released most, but not all, retained locks. Restart light was designed to restart quickly with a minimal storage footprint. Therefore, restart light did not go through castout processing and, as a result, retained page set P-locks in IX or SIX mode were not released. Utilities can be blocked from running by these retained page set P-locks, therefore impacting overall DB2 availability.

DB2 11 introduces an enhancement to the restart light process to also include castout processing. The syntax of the **START** command is enhanced to include the **LIGHT(CASTOUT)** option. When this new option is used, transaction retained locks are released as usual, but the restart also kicks off castout processing. After castout is completed, the page sets become non-GBP-dependent, and the page set P-locks in **IX** or **SIX** mode are released.

The default for the **LIGHT** restart option is **NO**, which means that DB2 performs a complete restart and not a light restart. Of the remaining three **LIGHT** values, **CASTOUT**, **YES** and **NOINDOUBTS**, the value of **CASTOUT** involves the most steps, because it also includes castout processing. The **YES** and **NOINDOUBTS** options do not provide castout processing. The values for the **LIGHT** option are mutually exclusive.

Restart might take longer to run with **LIGHT(CASTOUT)** than with **LIGHT(YES)**, but the benefit is that utilities can now be run without disruption, therefore increasing availability. If you are dependent upon running utilities as soon as possible after the restart, then you might want to investigate the **LIGHT(CASTOUT)** option. If you are more focused on getting the DB2 subsystem up as quickly as possible and will deal with retained page set P-locks on your own, you might want to use **LIGHT(YES)** or **LIGHT(NOINDOUBTS)** instead.

5.5 Locking enhancements

DB2 11 includes a number of locking enhancements that provide improved reliability, availability, and scalability. Many of these locking enhancements also provide benefits in a data sharing environment. This section describes the following locking enhancements:

- Conditional propagation of child Update locks to the coupling facility
- Improved performance in handling lock waiters
- Increase in maximum number of CF lock table entries
- Throttle batched unlock requests
- Improved IRLM resource hash table algorithm

5.5.1 Conditional propagation of child Update locks to the coupling facility

Prior to internal resource lock manager (IRLM) 2.3, IRLM propagates U state child locks for S state parent page set P-Locks in all cases. Propagation of U state child locks are not necessary in cases where only a single member is doing updates to a table in a data sharing environment. Propagating a large number of U state child locks to the CF incurs unnecessary overhead and should be avoided until it is necessary.

In DB2 11, IRLM 2.3 propagates shared S state parent page set P-locks to the CF as XES exclusive requests and suppresses any update U state child lock propagations until there is global contention on the parent page set P-lock.

This enhancement improves the performance of **SELECT FOR UPDATE** statements in data sharing environments.

5.5.2 Improved performance in handling lock waiters

In prior versions of DB2, in a large data sharing group with a large number of processes waiting on locks, there can be a performance cost for managing the lock waiters. DB2 11, with IRLM 2.3, introduces an improved deadlock and contention algorithm. This improved algorithm results in reduced CPU time for processes with many lock waiters and also reduces the number of lock suspensions.

5.5.3 Increase in maximum number of CF lock table entries

IRLM previously limited the number of CF lock structure table entries (LTEs) to a maximum value of 1 GB. Because the maximum LTEs supported by XES is 2 GB, there is no reason for IRLM to put its own limit for the LTEs. Therefore, in DB2 11, with IRLM 2.3, you can specify a lock table size as big as 2 GB as supported by XES.

This enhancement reduces contention when accessing the lock structure in the CF. It also reduces the possibility of false contention, which can occur when the number of LTEs is too small compared to the number of different resource names that can acquire locks.

You can use the **MODIFY irlmproc,SET** z/OS command to change the number of lock table entries. Example 5-5 shows the syntax for this command.

Example 5-5 Syntax of MODIFY irlmproc,SET command

```
>>-MODIFY--irlmproc,SET-+-,DEADLOCK=nnnn---------------+------><
                        +-,LTE=nnnn--------------------+
                        +-,MLT=nnnnnU------------------+
                        +-,PVT=nnnn--------------------+
                        +-,TIMEOUT=nnnn,subsystem-name-+
                        |         .-10--.              |
                        '-,TRACE=-+-nnn-+--------------'
```

You can increase the number of lock table entries by changing the value of the LTE option. You must set it to an exact power of 2. Each increment in value represents 1,048,576 LTE entries. To set it to the maximum number of LTEs, 2 GB, specify a value of 2048.

Because the command is a z/OS IRLM command, it can be issued only from a z/OS console. You also need to rebuild the CF lock structure to enable the new LTE size.

5.5.4 Throttle batched unlock requests

If a thread in a data sharing environment holds million of locks and is going through de-allocation, IRLM sends unlock requests for all of the locks in a batch unlock request to XES. If this process is done under non-preemptible SRB mode in a uni-processor environment, the unlock request process occupies the processor until all of the unlock requests are processed. As a result, this flood of unlock requests can cause XES to receive an abend178 because real storage manager (RSM) cannot find any available real storage frame.

IRLM 2.2 and 2.3 are enhanced through APAR PM60449 to change the way that IRLM handles batch unlock processing when a thread is holding more than 128,000 locks. The unlock request runs as a queued request under an IRLM SRB and does a status STOP SRB after running for some time. The batch unlock processing continues when this SRB is resumed again.

This change reduces storage constraints on XES and also reduces secondary latch contention in IRLM. This change allows other higher priority work from RSM and XES to run, which can improve conditions that result in a hang.

5.5.5 Improved IRLM resource hash table algorithm

IRLM 2.2 introduced through APAR PK50095 the capability to have an expanded resource hash table to handle anticipated higher locking volumes. IRLM provided support for a 64 KB hash table in DB2 10.

IRLM internally goes through the resource hash table serially every deadlock cycle while holding the main latch, preventing any other work from processing in IRLM. With higher volumes of threads and more and more locks being held, this hash table processing drives up the CPU time with every deadlock cycle.

In DB2 11, with IRLM 2.3, the resource hash table algorithm is improved to perform the deadlock cycle processing more efficiently. This improvement results in less CPU time associated with this process and reduced contention on lock structure access.

5.6 Index availability and performance

DB2 11 includes a number of index enhancements that provide improved availability and performance in data sharing environments. This section describes the following index enhancements:

- Avoid placing indexes in RBDP state during group restart
- Reduce synchronous log writes during index structure modifications

5.6.1 Avoid placing indexes in RBDP state during group restart

There are certain recovery scenarios where objects are placed in the logical page list (LPL) or in group buffer pool recovery pending (GRECP) state. A small timing window exists where, if there is an index tree structural modification (Index SMOD) in progress when the index is put into LPL or GRECP state, the index manager can write logical compensation log records (LCLRs) before writing the physical **NOT APPLY** undo log records for the unfinished index SMODs. Later, the LPL or GRECP recovery fails and the index is left in rebuild pending (RBDP) state when the LCLRs are processed before the physical undo logs are processed. After the LPL or GRECP recovery completes in this scenario, you still need to spend time to rebuild the index, which can take hours if the affected index is large. Ideally, you want to be able to run LPL and GRECP recovery and recover the indexes at the same time.

In DB2 11, if there is an index tree structural modification in progress when the index is put into LPL or GRECP state, then DB2 goes through a two-pass LPL or GRECP log apply process to recover the index. The second pass makes indexes available after the LPL or GRECP recovery process is completed. A **DSNI051I** message is issued at the start of the second pass. The LPL or GRECP recovery might take longer to finish when the second pass is needed, but you will not need to spend a longer amount of time to rebuild the index. This enhancement reduces DB2 outage time and increases index availability.

Also refer to 4.2, “Automatic recovery of indexes from GRECP or LPL status” on page 60 for more information.

This enhancement is available in DB2 11 New Function Mode (NFM) only.

5.6.2 Reduce synchronous log writes during index structure modifications

As rows are inserted in data sharing environments, index pages often need to be split as new keys fill the existing pages. The index split logic for GBP-dependent indexes causes two synchronous log writes, which can have a significant impact on transaction or batch performance. A similar situation exists for deletes where empty index pages get pruned from the index tree. There are five synchronous log writes in the delete case.

A related issue occurs when there are massive deletes from an index while the index is not GBP-dependent, and then the index becomes GBP-dependent as the backout starts. This situation has caused some backouts to take 10 to 20 times longer than the unit of work, because of the need to force several log write I/Os when adding deleted pages back into the index tree as part of the undo of the deletes.

DB2 11 reduces the synchronous log writes for index split and index page delete operations. Rather than do the log force write I/Os after processing begins and then again at the end of the process, the log force write I/Os occur only once at the end of the process. This enhancement improves performance for index splits and for pseudo deletes.

DB2 11 also reduces backout time by reducing the number of log force write I/Os during a rollback of deleted pages. DB2 11 can tell whether an index split operation completed successfully and will not roll back a successfully completed index split operation.

5.7 Group buffer pool write performance

The GBP batch write processing in DB2 11 has been enhanced to avoid pagefix operations by allocating fixed storage for GBP batch write. This enhancement provides a reduction in the path length for the **COMMIT** processing.

5.8 Automatic LPL recovery at end of restart

There are occasionally times when there are not enough pages available in the GBP to accept a write from the local buffer pool. The write is attempted a few times while DB2 goes through the castout process in an attempt to free more pages. However, after a few unsuccessful attempts, DB2 gives up and inserts an entry for the page in the LPL, where it is unavailable for any access until the LPL condition is removed. The LPL exception condition is set if pages cannot be read from or written to the GBP.

Prior to DB2 11, in a data sharing environment, when pages were added to the LPL by an active member while one of the members was down and holding retain locks, there was no automatic LPL recovery performed when the failed member restarted. This process resulted in an extended outage for application programs, which impacted overall system availability.

To recover these LPL pages, you had to manually resolve the LPL objects by issuing a **-START DB(xx) SPACE(yy)** command for every object with an LPL exception condition. This manual process can be time consuming if there are many objects with pages in the LPL, therefore extending the time that some applications are unavailable. This manual process can also be error prone, because some of the objects can be missed when issuing the **-START** commands, especially when there is a long list of objects to be recovered.

DB2 11 improves upon the LPL recovery process by initiating automatic LPL recovery of objects at the end of both normal (non-PIT recovery) restart and restart light. If the restart

involves any indoubt or postponed abort (PA) units of recovery (URs) then the LPL recovery associated with those URs is not automatically triggered at the end of restart. These objects cannot be automatically recovered because DB2 does not know the entire LPL log range for indoubt and PA URs until they are resolved.

The auto-LPL recovery process is not triggered for any of the following circumstances:

- If the DB2 member is started in access maintenance mode
- If the DB2 member is started in point-in-time (PIT) Recovery mode
- If the DB2 member is started at a tracker site
- If the DB2 member is involved in any type of conditional restart
- If **DEFER ALL** was specified on installation panel DSNTIPS
- For objects explicitly listed in a **DEFER** object list on installation panel DSNTIPS
- For table spaces defined as **NOT LOGGED**

The processing for automatic LPL recovery is similar to the auto-GRECP recovery processing that is done at the end of restart in that the auto-LPL recovery uses the existing messages to report LPL recovery progress, errors and successful completion. It makes only one attempt to automatically recover LPL objects. If the auto-LPL recovery fails at restart time, then either the DBA can manually recover the LPL objects by issuing **-START DB** commands or mainline auto-LPL recovery can perform the recovery at the appropriate time.

In the following conditions, an LPL object can remain in LPL after the end of restart auto-LPL recovery is completed:

- If any error is encountered during log apply
- If an active member continues to add new pages to the LPL or extends the LPL range while the restarting member is performing the auto-LPL recovery

The mainline auto-LPL recovery processing has always had retry logic to drive the LPL recovery one more time if the LPL page range or log range is extended. Auto-LPL continues to honor the same retry logic except during restart light. Because auto-LPL recovery is initiated during restart light, the auto-LPL recovery task serializes with DB2 shutdown; the end of restart auto-LPL recovery will complete before DB2 is terminated at the end of restart light.

At the end of auto-LPL recovery, each member issues a **DSNI049I** message on the console to indicate that LPL recovery of all objects is completed. This is the same message that has been issued in prior versions of DB2 when the **-START DB** command completes, and is still issued when the **-START DB** command completes if you need to issue the command because auto-LPL recovery cannot complete due to one of the reasons listed above. The message is always issued to make it easier for you to automate the recovery process using whatever tool or manual procedure you have implemented.

5.9 Log record sequence number spin avoidance

Enhancements were made in both DB2 9 and DB2 10 in the area of log record sequence number (LRSN) spin avoidance. DB2 9 allowed for duplicate LRSN values for consecutive log records on a given member. DB2 10 further extended LRSN spin avoidance by allowing for duplicate LRSN values for consecutive log records for inserts to the same data page.

Each of these enhancements meant that a DB2 member did not need to “spin” consuming CPU resources under the log latch to wait for the next LRSN increment. This function can avoid significant CPU overhead and log latch contention (LC19) in data sharing environments with heavy logging.

The DB2 9 and DB2 10 enhancements avoided the need to “spin” in the Log Manager to avoid duplicate LRSNs for most cases. However, some cases still exist where CPU spinning is necessary, which adds overhead. For example, consecutive **DELETE** or **UPDATE** operations to the same page require LRSN spin.

DB2 11 extends the LRSN to use more of the TOD clock precision. RBA and LRSN values are expanded from 6 bytes to 10 bytes so that it can take hundreds of years to exhaust a DB2 subsystem’s or data sharing group’s logging capacity, based on current and projected logging rates. Details about the expanded RBA and LRSN vales are provided in 3.1, “Extended RBA and LRSN” on page 24.

Data sharing environments can take advantage of the larger LRSN values and avoid LRSN spin altogether.

Part 2

Application functions

This part describes functions that help to enable the application development with DB2 11 for z/OS.

DB2 11 delivers several SQL enhancements that can help applications to ease development and porting. DB2 11 also provides several enhancements to the support of pureXML, stored procedures, and connectivity.

Application-enabling infrastructure changes allow for intersection with big data. XML and e-business have each a dedicated chapter.

This part includes the following chapters:

6

SQL

This chapter describes the SQL features and enhancements that are delivered with DB2 11 for z/OS. The audience for this chapter is the application developers and database administrators.

This chapter includes the following topics:

- Introduction
- Global variables
- Array data type
- Aliases and public aliases for SEQUENCES
- New built-in functions
- SET CURRENT APPLICATION COMPATIBILITY
- Temporal special registers
- Temporal support on VIEWs
- DGTT
- CUBE, ROLLUP and GROUPING SETS
- LIKE_BLANK_INSIGNIFICANT DSNZPARM

Good news for application managers and developers:

DB2 11 has mechanisms in place to limit potential SQL (and XML) incompatibilities on application DML statements by allowing you to complete the following tasks:

- Identify applications affected by incompatible SQL (and XML) changes through trace records
- Control the compatibility level to DB2 10 at an application (package) level

See 6.6, “SET CURRENT APPLICATION COMPATIBILITY” on page 114 and 12.6, “Release incompatibilities” on page 357.

6.1 Introduction

This section provides a summary of SQL features and enhancements to existing SQL statements.

Table 6-1 lists the key new SQL features which are discussed in other parts of this chapter. This table contains DML, DCL, and DDL statements that are new to DB2 11 for z/OS.

Table 6-1 Summary of SQL statements/features in DB2 11 for z/OS

SQL statement/feature	Description
CREATE VARIABLE	Defines Global variables. Read 6.2, “Global variables” on page 102.
CREATE TYPE (array) (**DROP TYPE** clause can be used to drop the array data type)	Defines an array data type. Read 6.3, “Array data type” on page 104 and 6.5, “New built-in functions” on page 112 for the associated built-in functions.
SET CURRENT APPLICATION COMPATIBILITY	Sets application compatibility level for dynamic/distributed applications. Works similar to the static equivalent APPLCOMPAT bind parameter. Read 6.6, “SET CURRENT APPLICATION COMPATIBILITY” on page 114.
SET CURRENT TEMPORAL BUSINESS_TIME	Sets the business time for use with temporal tables.
SET CURRENT TEMPORAL SYSTEM_TIME	Sets the system time for use with temporal tables.
SET assignment-statement (this is not truly new)	Sets the assignment-statement (DB2 SQL) The **SET** assignment-statement is a reclassification of the documentation of the **SET** host-variable and **SET** transition-variable statements into a single statement.
ALTER FUNCTION (SQL scalar) and CREATE FUNCTION (SQL scalar)	New clauses: ► **BUSINESS_TIME SENSITIVE** ► **SYSTEM_TIME SENSITIVE** ► **ARCHIVE SENSITIVE** ► **APPLCOMPAT** Changed clauses: ► data-type ► data-type2, which can include array-type-name
ALTER PROCEDURE (SQL native) and **CREATE PROCEDURE** (SQL native)	New clauses: ► **BUSINESS_TIME SENSITIVE** ► **SYSTEM_TIME SENSITIVE** ► **ARCHIVE SENSITIVE** ► **APPLCOMPAT** Changed clause: ► data-type, which can include array-type-name

SQL statement/feature	Description
CREATE PROCEDURE (external)	Changed clause: ► data-type, which can include array-type-name
ALTER TABLE DROP COLUMN	New clause
ALTER TABLE ENABLE ARCHIVE	New clauses: ► **ENABLE ARCHIVE** ► **DISABLE ARCHIVE**
ALTER TABLESPACE and **CREATE TABLESPACE**	Changed clause: ► PCTFREE, which can now include **FOR UPDATE** smallint
COMMENT	Changed clause: ► data-type, which can include array-type-name
CREATE INDEX	New clauses: ► **INCLUDE NULL KEYS** ► **EXCLUDE NULL KEYS**
DECLARE GLOBAL TEMPORARY TABLE	New clause: ► **LOGGED** ► **NOT LOGGED**
EXECUTE	Changed clauses: The object of the **USING** clause can be an SQL variable, SQL parameter, global variable, or host variable.
FETCH	Changed clauses: The object of the **INTO** clause can be a host variable, a global variable, an SQL parameter, an SQL variable, a transition variable, or an array element.
GRANT (function or procedure privileges) and corresponding **REVOKE**	Changed clauses: ► data-type, which can include array-type-name
GRANT (type or JAR privileges) and corresponding **REVOKE**	Changed clauses: The object of the **TYPE** clause can be a distinct type or an array type
SELECT INTO	Changed clauses: The object of the **INTO** clause can be a host variable, a global variable, an SQL parameter, an SQL variable, a transition variable, or an array element.
SET PATH	Changed clauses: The **SYSTEM PATH** now includes the following schemas: ► **SYSIBM** ► **SYSFUN** ► **SYSPROC** ► **SYSIBMADM**

SQL statement/feature	Description
SQL statement with subselect	Changed clauses: The collection-derived-table clause is added to table-reference in the **FROM** clause of a subselect.
VALUES INTO	Changed clauses: The object of the **INTO** clause can be a host variable, a global variable, an SQL parameter, an SQL variable, a transition variable, or an array element.

Refer to *DB2 11 for z/OS What's New?*, GC19-4068 for an alphabetical listing of the summary of changes to existing and new SQL statements.

6.2 Global variables

Traditionally within a relational database system, most interactions between an application and the DBMS are in the form of SQL statements within a connection. To share information between SQL statements within the same application context, the application that issued the SQL statements has to do this work by copying the values from the output arguments, such as host variables, of one statement to the input host variables of another. Similarly, when applications issue host-language calls to another application, host variables need to be passed among applications as input or output parameters for the applications to share common variable. Furthermore, SQL statements that are defined and contained within the DBMS, such as the SQL statements in the trigger bodies, cannot access this shared information.

These restrictions limit the flexibility of relational database systems and, thus, the ability of users of such systems to implement complex, interactive models within the database itself. Users of such systems are forced to put supporting logic inside their applications to access and transfer user application information and internal database information within a relational database system. Ensuring the security of the information that is transferred and accessed is also left to the user to enforce in their application logic.

To overcome this restriction and to maximize the flexibility of a DBMS, *global variables* are introduced in DB2 11 for z/OS. A global variable can be created, instantiated, accessed, and modified by the applications. Global variables are named memory variables that you can access and modify through SQL statements. Global variables enable you to share relational data between SQL statements without the need for application logic to support this data transfer. You can control access to global variables through the **GRANT** (global variable privileges) and **REVOKE** (global variable privileges) statements.

A global variable is associated with a specific application context, and contains a value that is unique to that application scope. A created global variable is available to any active SQL statement running against the database on which the variable was defined. A global variable can be associated with more than one application scope, but its value will be specific to each application scope.

SQL statements sharing the same connection (that is, under the same application scope) can create, access, and modify the same global variables. This enhancement includes the following functional additions to DB2.

6.2.1 DDL and catalog information

A new DDL statement allowing the application to create global variable to be shared among SQL statements using the same connection. A sample **CREATE** statement is depicted in Example 6-1.

Example 6-1 Sample create global variable statement

```
CREATE VARIABLE BATCH_START_TS TIMESTAMP
DEFAULT CURRENT TIMESTAMP;
```

The new **SYSIBM.SYSVARIABLES** table includes one row for each global variable that is created.

The new **SYSIBM.SYSVARIABLEAUTH** table includes one row for each privilege of each authorization ID that has privileges on a global variable.

The **SYSIBM.SYSVARIABLES_TEXT** table is an auxiliary table for the **DEFAULTTEXT** column of the **SYSIBM.SYSVARIABLES** table.

6.2.2 Qualifying global variables

Global variable names are qualified two-part names. For unqualified global variables, the implicit qualifier facilitates the naming resolution of global variables. DB2 determines the implicit qualifier for global variables as follows:

The schemas in the SQL **PATH** are searched in order from left to right for a matching global variable. If a global variable matches the global variable name in reference, resolution is complete. If no matching global variable is found after completing this step, an error is returned.

6.2.3 Global variable's scope

The scope of global variable's definition is similar to that of DB2 special register's, in that, when created, the definitions of global variables are shared across different DB2 connections. However, each connection maintains its own instance of the global variable, such that the variable's content is only shared among SQL statements within the same connection.

For example, if you use a Global variable, which was created using the DDL in Example 6-1 in a program, the first invocation of this Global variable has the same value as the current time stamp. The subsequent use of this global variable retains the initial instantiated value for the duration of the connection.

Example 6-2 shows how a Global variable's value of `2013-08-02-14.59.46.423414` remains the same in different SQL statements (when referenced) within the same DB2 connection.

Example 6-2 Scope of global variable: Different SQL statements on the same DB2 connection

```
-- Initial execution of the SQL
SELECT          BATCH_START_TS, CURRENT TIMESTAMP
FROM SYSIBM.SYSDUMMY1
;
-- Result set from the initial execution

BATCH_START_TS CURRENT TIMESTAMP
2013-08-02-14.59.46.423414  2013-08-02-14.59.46.423414
```

```
-- Second execution of the same SQL statement in the same SPUFI session

SELECT          BATCH_START_TS, CURRENT TIMESTAMP
FROM SYSIBM.SYSDUMMY1
;
-- Result set from the second execution
BATCH_START_TS CURRENT TIMESTAMP

2013-08-02-14.59.46.423414  2013-08-02-14.59.46.424678

-- Third execution of the same SQL statement in the same SPUFI session

SELECT          BATCH_START_TS, CURRENT TIMESTAMP
FROM SYSIBM.SYSDUMMY1
;
-- Result set from the third execution
BATCH_START_TS CURRENT TIMESTAMP

2013-08-02-14.59.46.423414  2013-08-02-14.59.46.425282
```

If you rerun this set of SQL statements at a different point in time (for example, in a different SPUFI session another time), it results in a different instantiated value for the global variable. That value remains in effect until the end of that connection.

6.2.4 Global variable's naming resolution

DB2 naming resolution precedence rule is modified to include global variable references. If at the time of naming resolution, the definition of the referenced global variable does not exist, message DSNX200I or DSNX100I will be issued during **BIND** if **VALIDATE(BIND)** or **VALIDATE(RUN)** were specified, respectively, on the **BIND** command.

Note: It is also possible that during static bind time, the checked objects preceding global variables in naming resolution might not exist yet, resulting in the object name being resolved to global variables, provided that the variable definitions exist. In this case, the resolved name remains as global variables, even if the object becomes available before the execution of this statement, because the naming resolution was already done at static bind time.

6.3 Array data type

An array type is a user-defined data type that is an ordinary array or an associative array. The elements of an array type are based on one of the existing built-in data types.

Note: The array data type can only be used as one of the following data types:

- An SQL variable
- A parameter or **RETURNS** data-type of an SQL scalar function
- A parameter of a native SQL procedure
- The target data type for a **CAST** specification

Currently, the array data type is not supported in other contexts, such as columns of tables and views, triggers, and client interfaces that are not essential for migrating applications.

The **CREATE TYPE** (array) statement defines an array data type. The **SYSIBM.SYSDATATYPES** table contains one row for each array data type defined.

DROP TYPE array_name statement drops an array data type created using the **CREATE TYPE** array_name statement.

6.3.1 Ordinary arrays

An array with a user-defined upper bound on the number of elements, which are referenced by their ordinal position in the array.

After the execution of the assignment statement in Example 6-3, the cardinality of `mySimpleA` is set to 100. The elements with array indexes with values 1 to 99 are implicitly initialized to **NULL**.

Example 6-3 Sample - Ordinary Array definition

```
CREATE TYPE simple AS INTEGER ARRAY[];
BEGIN
   SET mySimpleA[100] = 123;
END
```

6.3.2 Associative arrays

An array with no user-defined upper bound on the number of elements, which are ordered by and can be referenced by an array index value. Array index values are unique and do not have to be contiguous.

After the execution of the assignment statement in Example 6-4, the cardinality of the array is set to 1.

Example 6-4 Associative array data type - sample CREATE, DECLARE, and SET statements

```
CREATE TYPE assoc AS INTEGER ARRAY[INTEGER];
BEGIN
   SET myAssocA[100] = 123;
END
```

6.3.3 ARRAY_EXISTS predicate

The **ARRAY_EXISTS** predicate tests for the existence of an array element with the specified index in an array. Example 6-5 shows the syntax of this new predicate.

Example 6-5 ARRAY_EXISTS predicate syntax

```
ARRAY_EXISTS (array-expression,array-index)
```

The **ARRAY_EXISTS** predicate produces the following results:

- *True* if array-variable includes an array index that is equal to the result of casting array-index to the data type of the array index of array-variable.
- *False* under either of the following conditions:
  - The array-variable does not include an array index that is equal to the result of casting array-index to the data type of the array index of array-variable.
  - Either argument is null.
- Never unknown.

6.4 Aliases and public aliases for SEQUENCES

DB2 10 for z/OS supports aliases for tables, views, and aliases. The definition of these aliases is recorded in the **SYSIBM.SYSTABLES** catalog table, with a value of **A** for the **TYPE** column. IBM DB2 for Linux, UNIX, and Windows supports aliases on aliases, tables, views, nicknames (federated related), module names (related to competitive database product with PL/SQL), and sequence objects. Competitive database products also allow synonyms to be created for sequence objects.

DB2 11 extends the support for **SEQUENCE** objects so that you can now create:

- A private **ALIAS** for a **SEQUENCE**
- A public **ALIAS** for a **SEQUENCE**

This section describes how to create and use these aliases for **SEQUENCES** and provide related considerations.

6.4.1 Private ALIAS for a SEQUENCE

The **CREATE ALIAS** syntax is extended as shown in Figure 6-1.

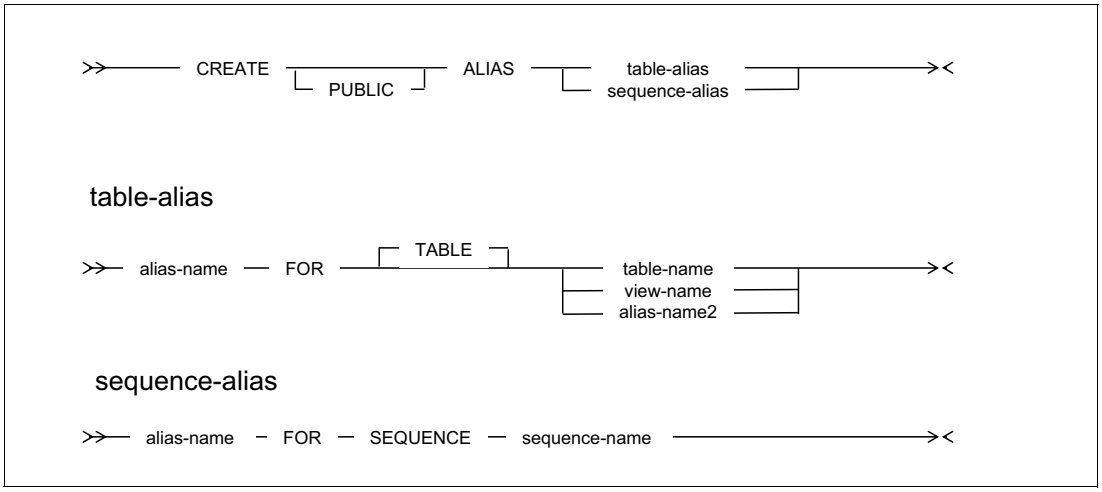


Figure 6-1 CREATE ALIAS statement syntax

Assuming that your **CURRENT SQLID** is user2, to create a private **ALIAS** `user2.SEQ1` for `user1.SEQ1` sequence, you can use either of the following DDL statements:

```
CREATE ALIAS user2.SEQ1 FOR user1.SEQ1
```

```
CREATE ALIAS SEQ1 FOR user1.SQ1
```

Both statements result in the creation of a private ALIAS user2.SEQ1. However, you cannot issue both statements. The second **CREATE ALIAS** fails with -601.

The information for table aliases was recorded in the **SYSIBM.SYSTABLES** table, and the information about the existence of the alias for user1.SEQ1 is recorded in the **SYSIBM.SYSSEQUENCES** table.

For this example, the **SYSIBM.SYSSEQUENCES** table include the following information:

```
SCHEMA           user2
OWNER            user2
NAME             SEQ1
SEQTYPE          A
SEQID            an ID, for example 453
CREATEDBY        user2
INCREMENT        0
START            0
MAXVALUE         0
….
```

Other columns include characteristics of the **SEQUENCE** 0 or N, as follows:

```
…..
SEQSCHEMA   user1 (schema of the sequence the alias depends on)
SEQNAME     SEQ1 (name of the sequence, the alias depends on)
….
```

You can use this alias with any of the **SEQUENCE** related statements, such as **NEXT VALUE** or **PREVIOUS VALUE**. Thus, for example, user2 can successfully execute the following statement, assuming that the user1.TAB1 table is an existing table:

```
INSERT INTO  user1.TAB1 VALUES(NEXT VALUE FOR SEQ1, 'AAA');
```

This **INSERT** command inserts one row into user1.TAB1 table and inserts the next available value for the user1.SEQ1 sequence, using characteristics such as **INCREMENTS**, **MAXVALUE**, and so on, of the user1.SEQ1 sequence.

6.4.2 Public ALIAS for a SEQUENCE

As mentioned previously, in addition to the ability of creating private aliases for sequences, DB2 11 allows you to create public aliases. All public aliases are created in the new **SYSPUBLIC** schema. This creation happens implicitly or explicitly, meaning that DB2 automatically assigns schema **SYSPUBLIC** to the **ALIAS** if you omit the schema name. As a consequence, the following statements create a **SYSPUBLIC SEQ2** public alias for a user1.SEQ1 schema:

```
CREATE PUBLIC ALIAS SEQ2 FOR SEQUENCE user1.SEQ1
CREATE PUBLIC ALIAS SYSPUBLIC.SEQ2 for SEQUENCE user1.SEQ1
```

Both statements lead to the same information inserted into the **SYSIBM.SYSSEQUENCES** table, as follows:

```
SCHEMA              SYSPUBLIC
OWNER               user2
NAME                SEQ1
SEQTYPE             A
SEQID               a ID, for example 453
```

```
CREATEDBY           user2
INCREMENT           0
START               0
MAXVALUE            0
….
```

Other columns include characteristics of the **SEQUENCE** 0 or N, as follows:

```
…..
SEQSCHEMA   user1 (schema of the sequence the alias depends on)
SEQNAME     SEQ1 (name of the sequence, the alias depends on)
….
```

A public alias can always be referenced without qualifying its name with a schema name. The implicit qualifier of a public alias is **SYSPUBLIC**, which can also be specified explicitly.

When DB2 resolves an unqualified name, private aliases are considered before public aliases.

When there is a reference to a sequence, DB2 must resolve the reference to one of the following sequences:

- A private alias for a sequence
- A public alias for a sequence
- A sequence

For example, assume that a sequence named *orders_seq exists*, defined as follows, and an alias is named *orders_seq_A* is defined for this sequence. The orders_seq sequence generates odd values starting with 1, as follows:

```
CREATE SEQUENCE orders_seq AS INT
START WITH 1
INCREMENT BY 2
MINVALUE 1
NO MAXVALUE
NO CYCLE
NO CACHE
ORDER ;

CREATE ALIAS orders_seq_A FOR SEQUENCE orders_seq;
```

Another sequence named *orders_seq2* exists as follows, and a public alias named *orders_seq_A* is defined for this sequence. The orders_seq2 sequence generates even values starting with 2, as follows:

```
CREATE SEQUENCE orders_seq2 AS INT
START WITH 2
INCREMENT BY 2
MINVALUE 2
NO MAXVALUE
NO CYCLE
NO CACHE
ORDER ;

CREATE PUBLIC ALIAS orders_seq_A FOR SEQUENCE orders_seq2;
```

Note that both of these aliases have the same name, but different schemas. The first alias defined is a private alias, and it is qualified with the default schema. The second alias was defined as a public alias, which means that it is qualified by **SYSPUBLIC**.

The following customer_orders_t table demonstrates the use of sequence aliases:

```
CREATE TABLE customer_orders_t
( order_id INT NOT NULL ,
order_date DATE NOT NULL)
```

A **NEXT VALUE** sequence reference provides the value for the **ORDER_ID** column of the table in the following **INSERT** statement. The sequence reference specifies **ORDERS_SEQ_A** for the sequence. This name can represent a sequence itself, or it can be a reference to a private alias for a sequence or a public alias for a sequence. DB2 goes through a process of name resolution to determine the sequence to be used.

```
INSERT INTO customer_orders_t
VALUES (NEXT VALUE FOR orders_seq_A, CURRENT DATE) ;
```

Issuing a select statement shows the value that was generated for the sequence (the value of the **ORDER_ID** column) and that determined which sequence alias was used.

```
SELECT * FROM customer_orders_t;
```

Returns:

```
ORDER_ID            ORDER_DATE

-----------         ---------- -----------

1                   07/11/2012
```

The value of 1 for the **ORDER_ID** column indicates that the **ORDERS_SEQ** sequence generated the value for the column. DB2 used the **ORDERS_SEQ** sequence, because the unqualified reference to **ORDERS_SEQ_A** resolved to the private alias **ORDERS_SEQ_A**, which is defined for the sequence **ORDERS_SEQ**.

6.4.3 Dropping an alias for sequence

Dropping a sequence alias (private or public) is restricted if any of the following dependencies exist:

- A trigger that uses the sequence in a **NEXT VALUE** or **PREVIOUS VALUE** expression exists.
- An inline SQL function1 that uses the sequences in a **NEXT VALUE** or **PREVIOUS VALUE** expression exists.

When an alias for a sequence is dropped, all packages that refer to the sequence alias are invalidated.

6.4.4 Security considerations

To create an alias for a sequence, the privilege set must include at least one of the listed authorities or privileges:

- The **CREATEIN** privilege on the schema
- **SYSADM** or **SYSCTRL** authority
- System **DBADM**

6.4.5 Considerations regarding application compatibility setting

Public aliases or private aliases can only be created and used in New Function Mode (NFM). In addition to that, to make use of private or public aliases, the value of special register **CURRENT APPLICATION COMPATIBILITY** must implicitly or explicitly be set to V11R1.

Assume that you have multiple sequences defined maybe to generate only odd or even numbers as discussed earlier. If these sequences, or public or private aliases, all have the same name, DB2 resolves the names as also discussed before. However, the results differ, depending on the value that is currently set for the **CURRENT APPLICATION COMPATIBILITY** special register.

Refer to the next two figures to explore the differences. Figure 6-2 represents V11R1 compatibility.

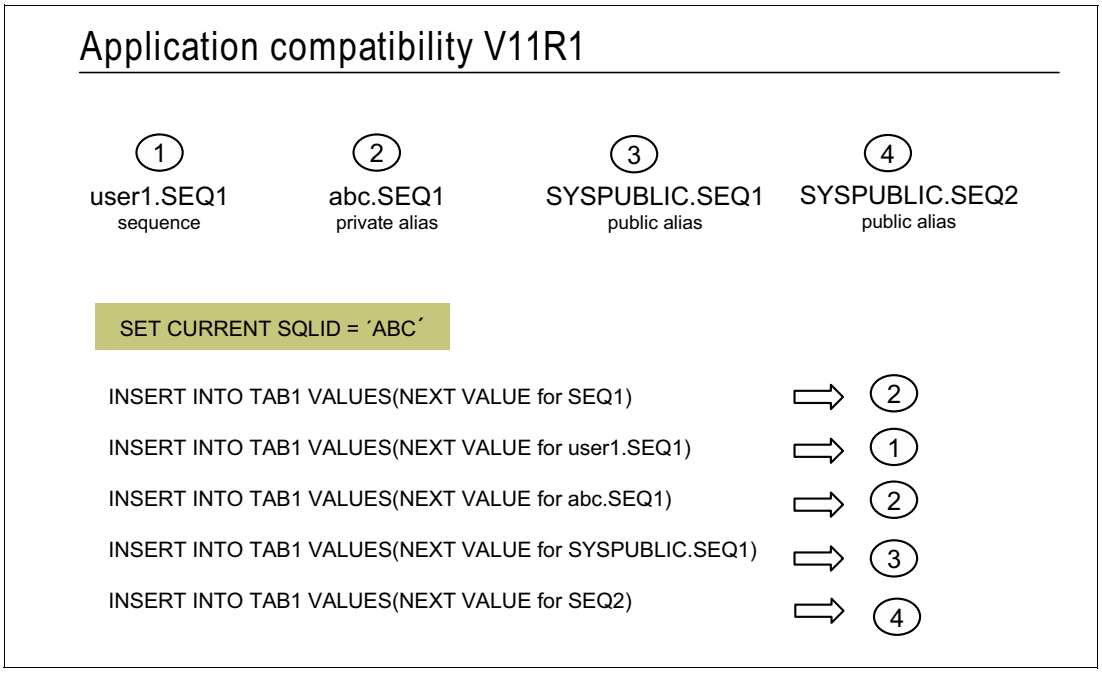


Figure 6-2 Application compatibility V11R1

The behavior for **APPLICATION COMPATIBILITY** set to V11R1 is as described before, but if you set it to V10R1, as shown in Figure 6-3, you receive negative SQL codes in all cases in which the new aliases for sequences are used.

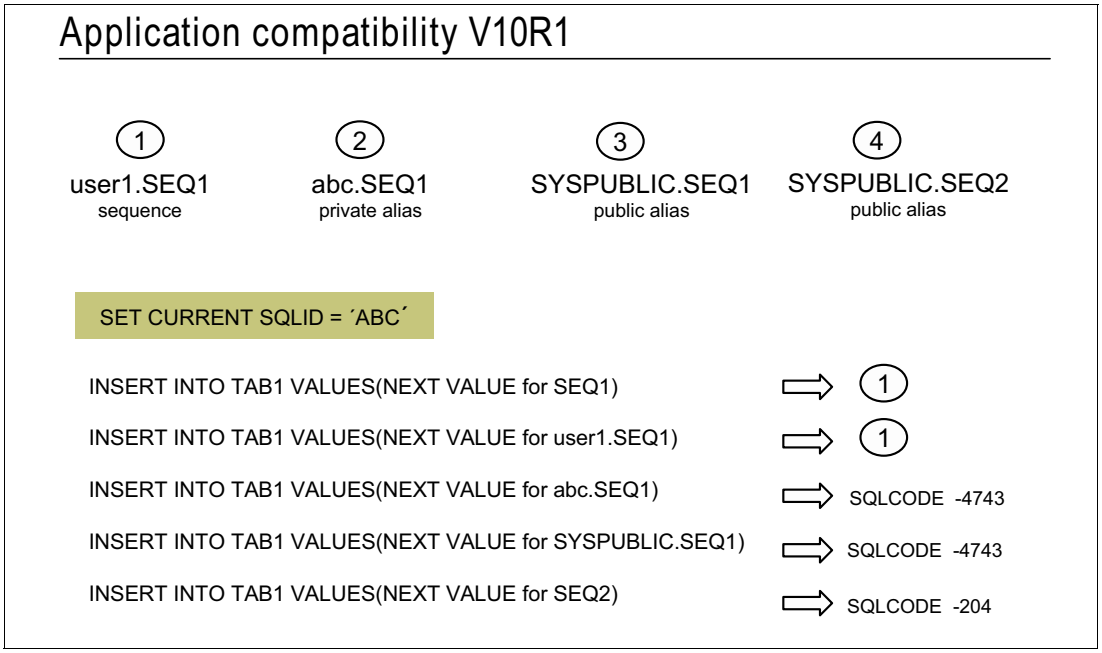


Figure 6-3 Application compatibility V10R1

In two of these cases you receive the following message because you try to explicitly request the use of aliases for sequences:

```
DSNT408I SQLCODE = -4743, ERROR:  ATTEMPT TO USE A FUNCTION WHEN THE
         APPLICATION COMPATIBILITY SETTING IS SET FOR A PREVIOUS LEVEL
```

The -204 code received in the example where you implicitly try to use the public alias for the sequence occurs because DB2 does not even consider the existence of the sequence definition 4.

For the sequence, if `APPLCOMPAT='V10R1'`, and the sequence is not qualified, there is no attempt to resolve it in the **SYSPUBLIC** schema. If it cannot be resolved at the first try through the private schema, DB2 issues `-204`.

If `APPLCOMPAT='V11R1'`, and the sequence is not qualified, DB2 tries to resolve it in the **SYSPUBLIC** schema if it cannot be resolved at the first try through the private schema.

If the sequence is qualified, DB2 only tries to resolve in that specified qualifier.

After a sequence is resolved, if it is resolved to a public alias, it must be in V11R1. Otherwise, a `-4743` error is issued.

CURRENT PATH does not include **SYSPUBLIC**; however, the public aliases that exist in this schema are found.

This behavior needs to be thoroughly checked when you start using aliases for sequences in NFM, while still running application with an application compatibility setting other than V11R1.

6.5 New built-in functions

DB2 11 for z/OS includes new built-in functions that improve the power of the SQL language. The schema is SYSIBM. Refer to *DB2 11 for z/OS SQL Reference*, SC19-4066 for syntax alternatives and additional examples.

6.5.1 ARRAY_AGG

The **ARRAY_AGG** function returns an array in which each value of the input set is assigned to an element of the array. **ARRAY_AGG** can be invoked in the following situations:

- Select list of a **SELECT INTO** statement
- Select list of a fullselect in the definition of a cursor that is not scrollable
- Select list of a scalar fullselect as a source data item for a **SET** assignment-statement (or SQL PL assignment-statement)
- A **RETURN** statement in an SQL scalar function

Example 6-6 shows a sample **CREATE** array data type followed by a sample UDF where the array data type is used in a **RETURN** statement of an SQL scalar function. This sample UDF uses the **ARRAY_AGG** (aggregate) function and returns an array data type to the caller.

Example 6-6 Array data type create statement and sample use case in a scalar function

```
CREATE TYPE PHONELIST AS CHAR(4) ARRAY[];

CREATE FUNCTION PHONELIST_UDF (LOWSAL DECIMAL(9,2))
      RETURNS PHONELIST
      LANGUAGE SQL
      CONTAINS SQL
      NO EXTERNAL ACTION
RETURN
(SELECT ARRAY_AGG(PHONENO ORDER BY SALARY)
     FROM DSN8111O.EMP WHERE SALARY > LOWSAL)
```

6.5.2 ARRAY_DELETE

The **ARRAY_DELETE** function deletes elements from an array. This function can be specified only in the following specific contexts:

- As a source value for a **SET** assignment-statement (or SQL PL assignment-statement) or **VALUES INTO** statement.
- As the value to be returned in a **RETURN** statement in an SQL scalar function.

6.5.3 ARRAY_FIRST

The **ARRAY_FIRST** function returns the minimum array index value of an array. The data type of the result is the data type of the array index, which is **INTEGER** for an ordinary array. If array expression is not null and the array is not empty, the value of the result is the minimum array index value, which is 1 for an ordinary array.

6.5.4 ARRAY_LAST

The **ARRAY_LAST** function returns the maximum array index value of an array. The data type of the result is the data type of the array index, which is **INTEGER** for an ordinary array. If the array expression is not null and the array is not empty, the value of the result is the maximum array index value, which is the cardinality of the array for an ordinary array.

6.5.5 ARRAY_NEXT

The **ARRAY_NEXT** function returns the next larger array index value, relative to a specified array index value.

6.5.6 ARRAY_PRIOR

The **ARRAY_PRIOR** function returns the next smaller array index value, relative to a specified array index value.

6.5.7 CARDINALITY

The **CARDINALITY** function returns the number of elements in an array. The data type of the result is BIGINT.

The result of the **CARDINALITY** function is as follows:

- For an ordinary array, the result is the highest array index for which the array has an assigned element. Elements that have been assigned the null value are considered to be assigned elements.
- For an associative array, the result is the actual number of unique array index values that are defined in array-expression.
- For an empty array, the result is 0.

6.5.8 MAX_CARDINALITY

The **MAX_CARDINALITY** function returns the maximum number of elements that an array can contain. This value is the cardinality that was specified in the **CREATE TYPE** statement for an ordinary array type.

The result of the **MAX_CARDINALITY** function is as follows:

- For an ordinary array, the result is the maximum number of elements that an array can contain.
- For an associative array, the result is the null value.

6.5.9 TRIM_ARRAY

The **TRIM_ARRAY** function deletes elements from the end of an ordinary array. It can be invoked only in the following contexts:

- A source value for **SET** assignment-statement or SQL PL assignment-statement, or a **VALUES INTO** statement
- The value that is returned in a **RETURN** statement in an SQL scalar function

6.5.10 UNNEST (table function)

Treat an array like a table to fetch data (that is, rows) from the array. You can use the **UNNEST** construct (collection-derived table), which returns a result table that contains a row for each element of an array. For example, using the **UNNEST** operation, you can retrieve a list of the phone numbers from the array returned by the **PHONELIST_UDF** as shown in Example 6-7.

Example 6-7 Sample invocation of UNNEST table function

```
SELECT * FROM dsn81110.emp WHERE phoneno = ANY (SELECT T.PHONE FROM
unnest(phonelist_udf(30000)) AS T(PHONE))
```

The **WITH ORDINALITY** clause in Example 6-8 indicates that the result table is to include an additional column that reflects the ordinal position of each array element within the array. This additional column is the last column of the result table from the **UNNEST** operation.

Example 6-8 Sample invocation of UNNEST table function with ORDINALITY clause

```
SELECT T.ARRAY_IX_SEQ, T.PHONE
FROM UNNEST(PHONELIST_UDF(20000)) WITH ORDINALITY AS T(PHONE, ARRAY_IX_SEQ);
```

In Example 6-8, the correlation clause following the **WITH ORDINALITY** clause, specifies that the additional column is named **ARRAY_IX_SEQ**, and the array element column is named **PHONE**. These column names can be explicitly referenced in the select list of the query.

6.5.11 Arrays in MERGE statement

With the introduction of limited support for arrays, an array value (that is, “whole array”) can be specified in **MERGE** statements in a context that allows for an array value. For example, an array can be referenced in a predicate of a merge statement.

Note that a value to be assigned to a column with a **MERGE** statement must not be an array value, because a column cannot be defined as an array. However, an array value can be referenced in an expression that provides the source value to be assigned, as long as the result of the expression is assignable to the target column.

6.6 SET CURRENT APPLICATION COMPATIBILITY

This special register is applicable only to dynamic SQL. **CURRENT APPLICATION COMPATIBILITY** specifies the DB2 release level that the dynamic SQL is compatible with. The data type is **VARCHAR(10)**. A routine environment cannot inherit this special register value from the caller's environment, even if the routine was created with the **INHERIT SPECIAL REGISTER** option.

You can change the value of this special register by executing the **SET CURRENT APPLICATION COMPATIBILITY** statement as shown in Example 6-9.

Example 6-9 APPLICATION COMPATIBILITY - Setting the special register values

```
SET CURRENT APPLICATION COMPATIBILITY = 'V11R1'
SET CURRENT APPLICATION COMPATIBILITY = 'V10R1'
```

As shown in Example 6-9, possible values for the **APPLICATION COMPATIBILITY** special register are V10R1 and V11R1.

- V10R1: Dynamic SQL statements are executed as they were in V10R1.
- V11R1: Dynamic SQL statements are executed with the new functionality of V11R1, which is not necessarily compatible with V10R1.

The value of V11R1 cannot be specified until DB2 11 for z/OS is in New Function Mode (NFM). By the same token, new options offered in V11 can be used in dynamic SQL statements only when this special register value has the V11R1 value (or implicitly inherited it from the **DSNZPARM** value default). For example, the **SYSTIMESENSITIVE**, **BUSTIMESENSITIVE**, and **ARCHIVESENSITIVE** options cannot be explicitly specified with the value of YES in NFM if this special register is set to V10R1. However, the IFCID 376 trace record can be used to identify those applications that can observe V11 incompatible changes when this special register is set to V11R1.

Additional details about application compatibility feature is discussed in 12.7, “Controlling application compatibility” on page 373.

See also *DB2 11 for z/OS Installation and Migration*, SC19-4056.

6.7 Temporal special registers

The **SET** statement in Example 6-10 sets the **CURRENT TEMPORAL BUSINESS_TIME** special register to last month. Assume that temporal table is an application-period temporal table with a **BUSINESS_TIME** period. The setting of the special register **CURRENT TEMPORAL BUSINESS_TIME** affects the update of temporal table that follows.

Example 6-10 shows sample set statements.

Example 6-10 Sample SET CURRENT TEMPORAL BUSINESS_TIME statement

```
SET CURRENT APPLICATION COMPATIBILITY = 'V11R1'
SET CURRENT TEMPORAL BUSINESS_TIME = CURRENT TIMESTAMP - 1 MONTH;
SET CURRENT TEMPORAL SYSTEM_TIME = CURRENT TIMESTAMP + 5 DAYS;
```

Example 6-11 shows a setting of compatibility.

Example 6-11 SET CURRENT TEMPORAL SYSTEM_TIME to past time period

```
SELECT EMPNO, BONUS FROM DB2R4.EMP_TEMPORAL
WHERE EMPNO = 10;
---------+---------+---------+---------+-----
EMPNO         BONUS
---------+---------+---------+---------+-----
000010      1000.00


SELECT EMPNO, BONUS FROM DB2R4.EMP_TEMPORAL
FOR  SYSTEM_TIME AS OF CURRENT TIMESTAMP - 2 DAYS
WHERE EMPNO = 10;
---------+---------+---------+---------+-----
EMPNO         BONUS
---------+---------+---------+---------+-----
000010      1000.00
```

```
SET CURRENT APPLICATION COMPATIBILITY = 'V11R1';
SET CURRENT TEMPORAL SYSTEM_TIME = CURRENT TIMESTAMP - 2 DAYS;
UPDATE DB2R4.EMP_TEMPORAL SET BONUS = 777 WHERE EMPNO = 10;
--DSNE615I NUMBER OF ROWS AFFECTED IS 1

SELECT EMPNO, BONUS FROM DB2R4.EMP_TEMPORAL
FOR  SYSTEM_TIME AS OF CURRENT TIMESTAMP
WHERE EMPNO = 10;
---------+---------+---------+---------+---------
EMPNO         BONUS
---------+---------+---------+---------+---------
000010       777.00

SELECT EMPNO, BONUS FROM DB2R4.EMP_TEMPORAL
FOR  SYSTEM_TIME AS OF CURRENT TIMESTAMP - 5 DAYS
WHERE EMPNO = 10;
---------+---------+---------+---------+---------+
EMPNO         BONUS
---------+---------+---------+---------+---------+
000010      1000.00

SELECT EMPNO, BONUS FROM DB2R4.EMP_TEMPORAL
FOR  SYSTEM_TIME AS OF CURRENT TIMESTAMP - 2 DAYS
WHERE EMPNO = 10;
---------+---------+---------+---------+--------
EMPNO         BONUS
---------+---------+---------+---------+--------
000010      1000.00

SET CURRENT APPLICATION COMPATIBILITY = 'V11R1';
SET CURRENT TEMPORAL SYSTEM_TIME = CURRENT TIMESTAMP - 2 DAYS;
SELECT EMPNO, BONUS FROM DB2R4.EMP_TEMPORAL
WHERE EMPNO = 10;
---------+---------+
EMPNO         BONUS
---------+---------+
000010       777.00
```

Example 6-12 is not exactly working like a time machine.

Example 6-12 SET CURRENT TEMPORAL SYSTEM_TIME to future time period

```
SET CURRENT APPLICATION COMPATIBILITY = 'V11R1';
SET CURRENT TEMPORAL SYSTEM_TIME = CURRENT TIMESTAMP + 2 MONTHS;
UPDATE DB2R4.EMP_TEMPORAL SET BONUS = 2222 WHERE EMPNO = 10;
--DSNE615I NUMBER OF ROWS AFFECTED IS 1

SELECT EMPNO, BONUS FROM DB2R4.EMP_TEMPORAL
FOR  SYSTEM_TIME AS OF CURRENT TIMESTAMP + 1 MONTH
WHERE EMPNO = 10
---------+---------+---------+---------+---------+
EMPNO         BONUS
---------+---------+---------+---------+---------+
000010      2222.00
```

```
SELECT EMPNO, BONUS FROM DB2R4.EMP_TEMPORAL
FOR  SYSTEM_TIME AS OF CURRENT TIMESTAMP
WHERE EMPNO = 10
---------+---------+---------+---------+----
EMPNO         BONUS
---------+---------+---------+---------+----
000010      2222.00

SET CURRENT APPLICATION COMPATIBILITY = 'V11R1';
SET CURRENT TEMPORAL SYSTEM_TIME = CURRENT TIMESTAMP - 2 DAYS;
SELECT EMPNO, BONUS FROM DB2R4.EMP_TEMPORAL
WHERE EMPNO = 10
---------+---------+---------+---------+----
EMPNO         BONUS
---------+---------+---------+---------+----
000010      2222.00
```

This behavior is not expected and can be explained by browsing the underlying tables directly, as shown in Table 6-2 and Table 6-3.

Table 6-2 EMP_TEMPORAL_HIST table contents

EMPNO	BONUS	START_TS	END_TS
10	1000	2013-08-26-00.21.00.753406376000	2013-09-09-00.59.10.635906699000
10	777	2013-09-09-00.59.10.635906699000	2013-09-09-01.36.34.073591227000

Table 6-3 EMP_TEMPORAL table contents

EMPNO	BONUS	START_TS	END_TS
10	2222	2013-09-09-01.36.34.073591227000	9999-12-30-00.00.00.000000000000

6.7.1 Scope of session-level special registers

For temporal query, it implicitly adds “FOR SYSTEM_TIME AS OF Current Temporal System_Time” for system-period temporal table, and “FOR BUSNESS_TIME AS OF Current Temporal Business_Time” for application-period temporal table.

6.7.2 SYSTIMESENSITIVE and BUSTIMESENSITIVE

Two new bind options are available in DB2 11 for z/OS so that applications can choose whether to be sensitive to the Current Temporal System_Time and Current Temporal Business_Time special registers.

Default is **YES** for these two **BIND** parameters.

6.8 Temporal support on VIEWs

In DB2 11 for z/OS, the **PERIOD** specification is extended to **CREATE VIEW** statements to provide the temporal support to users of views. The period specification is ignored if the view does not reference relevant type of temporal tables.

Example 6-13 contains the DDL used to define the temporal tables referenced in this section.

Example 6-13 Sample temporal table DDL statements

```
CREATE TABLE DB2R4.EMP_TEMPORAL
   (EMPNO                CHAR(6) FOR SBCS DATA NOT NULL,
    FIRSTNME             VARCHAR(12) FOR SBCS DATA NOT NULL,
    MIDINIT              CHAR(1) FOR SBCS DATA NOT NULL,
    LASTNAME             VARCHAR(15) FOR SBCS DATA NOT NULL,
    WORKDEPT             CHAR(3) FOR SBCS DATA WITH DEFAULT NULL,
    PHONENO              CHAR(4) FOR SBCS DATA WITH DEFAULT NULL,
    HIREDATE             DATE WITH DEFAULT NULL,
    JOB                  CHAR(8) FOR SBCS DATA WITH DEFAULT NULL,
    EDLEVEL              SMALLINT WITH DEFAULT NULL,
    SEX                  CHAR(1) FOR SBCS DATA WITH DEFAULT NULL,
    BIRTHDATE            DATE WITH DEFAULT NULL,
    SALARY               DECIMAL(9, 2) WITH DEFAULT NULL,
    BONUS                DECIMAL(9, 2) WITH DEFAULT NULL,
    COMM                 DECIMAL(9, 2) WITH DEFAULT NULL,
    START_TS             TIMESTAMP (12) WITHOUT TIME ZONE NOT NULL
      GENERATED ALWAYS AS ROW BEGIN,
    END_TS               TIMESTAMP (12) WITHOUT TIME ZONE NOT NULL
      GENERATED ALWAYS AS ROW END,
    TRANS_ID             TIMESTAMP (12) WITHOUT TIME ZONE
      GENERATED ALWAYS AS TRANSACTION START ID,
    PERIOD SYSTEM_TIME (START_TS, END_TS),
    CONSTRAINT EMPNO
    PRIMARY KEY (EMPNO))
  PARTITION BY (EMPNO ASC)
   (PARTITION 1 ENDING AT ('099999'),
    PARTITION 2 ENDING AT ('199999'),
    PARTITION 3 ENDING AT ('299999'),
    PARTITION 4 ENDING AT ('499999'),
    PARTITION 5 ENDING AT ('999999'))
  AUDIT NONE
  DATA CAPTURE NONE
  CCSID      EBCDIC
  NOT VOLATILE
  APPEND NO  ;
COMMIT;


CREATE TABLE DB2R4.EMP_TEMPORAL_HIST
   (EMPNO                CHAR(6) FOR SBCS DATA NOT NULL,
    FIRSTNME             VARCHAR(12) FOR SBCS DATA NOT NULL,
    MIDINIT              CHAR(1) FOR SBCS DATA NOT NULL,
    LASTNAME             VARCHAR(15) FOR SBCS DATA NOT NULL,
    WORKDEPT             CHAR(3) FOR SBCS DATA WITH DEFAULT NULL,
    PHONENO              CHAR(4) FOR SBCS DATA WITH DEFAULT NULL,
    HIREDATE             DATE WITH DEFAULT NULL,
    JOB                  CHAR(8) FOR SBCS DATA WITH DEFAULT NULL,
    EDLEVEL              SMALLINT WITH DEFAULT NULL,
    SEX                  CHAR(1) FOR SBCS DATA WITH DEFAULT NULL,
    BIRTHDATE            DATE WITH DEFAULT NULL,
    SALARY               DECIMAL(9, 2) WITH DEFAULT NULL,
    BONUS                DECIMAL(9, 2) WITH DEFAULT NULL,
```

```
    COMM                 DECIMAL(9, 2) WITH DEFAULT NULL,
    START_TS             TIMESTAMP (12) WITHOUT TIME ZONE NOT NULL,
    END_TS               TIMESTAMP (12) WITHOUT TIME ZONE NOT NULL,
    TRANS_ID             TIMESTAMP (12) WITHOUT TIME ZONE
      WITH DEFAULT NULL )
  PARTITION BY (EMPNO ASC)
   (PARTITION 1 ENDING AT ('099999'),
    PARTITION 2 ENDING AT ('199999'),
    PARTITION 3 ENDING AT ('299999'),
    PARTITION 4 ENDING AT ('499999'),
    PARTITION 5 ENDING AT ('999999'))
  AUDIT NONE
  DATA CAPTURE NONE
  CCSID      EBCDIC
  NOT VOLATILE
  APPEND NO  ;
COMMIT;

ALTER TABLE
  EMP_TEMPORAL
  ADD VERSIONING USE HISTORY TABLE
  EMP_TEMPORAL_HIST ;
```

Example 6-14 shows a sample **CREATE VIEW** statement defined on a temporal table and a sample **SELECT** statement that can query the view as of certain point in time.

Example 6-14 Sample VIEW statement on a temporal table along with a temporal Query

```
CREATE VIEW v0 (EMPNO, SALARY, COMM)
AS SELECT EMPNO, SALARY, COMM
FROM EMP_TEMPORAL ;

-- The following is a sample temporal query on the above view
SET CURRENT APPLICATION COMPATIBILITY = 'V11R1';
SELECT * FROM V0
FOR SYSTEM_TIME AS OF TIMESTAMP  '2013-08-25 23:55:00' ;
```

For illustration purposes, if the bi-temporal table contains rows with the **START_TS** and **END_TS** column values shown in Table 6-4, none of the rows are picked up by the query coded in the Example 6-14. However, the result set includes all the rows if the same query is run without the **FOR SYSTEM_TIME AS OF TIMESTAMP** clause.

Table 6-4 Sample time stamp values

START_TS	END_TS
2013-08-26-00.21.00.753406376000	9999-12-30-00.00.00.000000000000

The second sample **SELECT** statement as shown in Example 6-15 returns all the rows for which the timestamp value specified in the **AS OF** clause (that is, 2013-10-10 00:22:00) lies between the **START_TS** and **END_TS** values on the rows involved. For example, if all the rows pertaining to the view had the same set of values tabulated in Table 6-4, all the rows are returned by the query coded in Example 6-15.

Example 6-15 Selecting with AS OF

```
SET CURRENT APPLICATION COMPATIBILITY = 'V11R1';
SELECT * FROM VO
FOR SYSTEM_TIME AS OF TIMESTAMP  '2013-10-10 00:22:00' ;
```

Both the sample temporal queries are coded with the SET CURRENT APPLICATION COMPATIBILITY = 'V11R1 statement to emphasize the fact that this is a new function and it will not work if your Application Compatibility level is set to V10R1.

Note: DB2 11 for z/OS removes DB2 10 restrictions that period specification and period clause can only be specified with base table references and appropriate type of temporal tables.

6.9 DGTT

The **DECLARED GLOBAL TEMPORARY TABLE** statement now includes a clause to specify logging behavior. The logging attribute for the DGTT is at the table level as opposed to the logging attribute that is at the table space level for base tables.

A DGTT has the following different logging options:

- **LOGGED**, which is the default and the current behavior. In this case, DB2 logs all changes and during **ROLLBACK** or **ROLLBACK TO SAVEPOINT**, the changes to the DGTT are undone.
- **NOT LOGGED ON ROLLBACK DELETE ROWS**, which specifies no logging and during **ROLLBACK** or **ROLLBACK TO SAVEPOINT**, all rows in the DGTT are deleted if any change was made in the duration.
- **NOT LOGGED ON ROLLBACK PRESERVE ROWS**, which specifies no logging and during **ROLLBACK** or **ROLLBACK TO SAVEPOINT**, the rows in the DGTT will be preserved as they are.

In the case of an error situation during an SQL statement, where an SQLCODE or message is issued, if an update was made to a DGTT and **LOGGED** is specified, the changes to the DGTT are undone.

In the case of an error situation during an SQL statement, where an SQLCODE or message is issued, if an update was made to a DGTT and **NOT LOGGED** is specified, all rows in that DGTT are deleted, regardless of the **DELETE/PRESERVE ROWS** qualification.

DB2 can provide full incremental bind avoidance when used in a loop by switching to short prepare with **RELEASE(DEALLOCATE)**.

6.10 CUBE, ROLLUP and GROUPING SETS

Grouping-sets and super-groups are two new options under **GROUP BY** clause (of the **SELECT** statement). A super-group stands for **ROLLUP**, **CUBE** or grand-total clause. **ROLLUP** is helpful in

providing subtotaling along a hierarchical dimension such as time or geography. **CUBE** is helpful in queries that aggregate based on columns from multiple dimensions.

With support for rollup, cube, and grouping-sets specifications, the SQL coding complexity can be reduced greatly and the SQL performance can be improved dramatically.

6.10.1 GROUPING SETS

The **GROUPING SETS** option can be thought of as the union of two or more groups of rows into a single result set. It is logically equivalent to the union of multiple subselects with the group by clause in each subselect corresponding to one grouping set. This is similar to the DB2 for Linux, UNIX, and Windows and DB2 for IBM System i® support for grouping-sets and super-group specifications.

Example 6-16 shows a sample SQL statement by using the **GROUP BY** clause with the **GROUPING SETS** option. Figure 6-4 shows the result set of this SQL statement.

Example 6-16 Sample SQL statement utilizing GROUP BY GROUPING SETS

```
SELECT WORKDEPT, EDLEVEL, SEX, SUM(SALARY) as SUM_SALARY, AVG(SALARY) as
AVG_SALARY, COUNT(*) as COUNT
 FROM DSN8111O.EMP WHERE SALARY > 20000
  GROUP BY GROUPING SETS (WORKDEPT, EDLEVEL, SEX)
```

The result set is logically equivalent to the union all of three subselects with the group by clause in each subselect corresponding to one column each from the three columns on the grouping sets specification (while the other two column values are shown as **NULL**s).

WORKDEPT	EDLEVEL	SEX	SUM_SALARY	AVG_SALARY	COUNT
A00	NULL	NULL	204250.00	40850.00000000	5
B01	NULL	NULL	41250.00	41250.00000000	1
C01	NULL	NULL	118890.00	29722.50000000	4
D11	NULL	NULL	258350.00	25835.00000000	10
D21	NULL	NULL	143250.00	28650.00000000	5
E01	NULL	NULL	40175.00	40175.00000000	1
E11	NULL	NULL	82250.00	27416.66666666	3
E21	NULL	NULL	124570.00	24914.00000000	5
NULL	14	NULL	157570.00	26261.66666666	6
NULL	15	NULL	27380.00	27380.00000000	1
NULL	16	NULL	332655.00	27721.25000000	12
NULL	17	NULL	153610.00	25601.66666666	6
NULL	18	NULL	257020.00	36717.14285714	7
NULL	19	NULL	46500.00	46500.00000000	1
NULL	20	NULL	38250.00	38250.00000000	1
NULL	NULL	F	492580.00	30786.25000000	16
NULL	NULL	M	520405.00	28911.38888888	18

Figure 6-4 Result of sample query using GROUPING SETS (WORKDEPT, EDLEVEL, SEX)

6.10.2 ROLLUP

A **ROLLUP** grouping is an extension to the **GROUP BY** clause that produces a result set containing sub-total rows in addition to the “regular” grouped rows. Subtotal rows are “super-aggregate” rows that contain further aggregates whose values are derived by applying the same column functions that were used to obtain the grouped rows. These rows are called sub-total rows, because that is their most common use; however, any column function can be used for the aggregation.

A **ROLLUP** grouping is a series of grouping-sets. The general specification of a **ROLLUP** with *n* elements:

```
GROUP BY ROLLUP(C1,C2,...,Cn-1,Cn)
```

is equivalent to

```
GROUP BY GROUPING SETS((C1,C2,...,Cn-1,Cn),
(C1,C2,...,Cn-1),
...
(C1,C2),
(C1),
() )
```

For example, SUM and AVG are used in Example 6-17 is similar to Example 6-16 on page 121.

Example 6-17 Sample ROLLUP construct

```
SELECT WORKDEPT, EDLEVEL, SEX, SUM(SALARY) as SUM_SALARY,
AVG(SALARY) as AVG_SALARY, COUNT(*) as COUNT
 FROM DSN8111O.EMP WHERE SALARY > 20000
  GROUP BY ROLLUP (WORKDEPT, EDLEVEL, SEX)
```

Example 6-18 shows the result set for the query in Example 6-17.

Example 6-18 Sample ROLLUP result set

```
WORKDEPT EDLEVEL SEX  SUM_SALARY AVG_SALARY     COUNT
 -------- ------- ---- ---------- ------------- -----
 A00           14 M      29250.00 29250.00000000     1
 A00           18 F      52750.00 52750.00000000     1
 A00           19 M      46500.00 46500.00000000     1
 B01           18 M      41250.00 41250.00000000     1
 C01           16 F      23800.00 23800.00000000     1
 C01           18 F      28420.00 28420.00000000     1
 C01           20 F      38250.00 38250.00000000     1
 D11           16 M     130400.00 26080.00000000     5
 D11           17 F      43590.00 21795.00000000     2
 D11           18 F      29840.00 29840.00000000     1
 D21           14 M      22180.00 22180.00000000     1
 D21           15 F      27380.00 27380.00000000     1
 D21           16 F      36170.00 36170.00000000     1
 D21           17 M      28760.00 28760.00000000     1
 E01           16 M      40175.00 40175.00000000     1
 E11           16 F      29750.00 29750.00000000     1
 E11           17 F      26250.00 26250.00000000     1
 E21           14 M      51520.00 25760.00000000     2
 E21           16 M      23840.00 23840.00000000     1
```

```
A00           14 NULL   29250.00 29250.00000000     1
A00           18 NULL   52750.00 52750.00000000     1
A00           19 NULL   46500.00 46500.00000000     1
B01           18 NULL   41250.00 41250.00000000     1
C01           16 NULL   23800.00 23800.00000000     1
C01           18 NULL   28420.00 28420.00000000     1
C01           20 NULL   38250.00 38250.00000000     1
D11           16 NULL  130400.00 26080.00000000     5
D11           17 NULL   43590.00 21795.00000000     2
D11           18 NULL   29840.00 29840.00000000     1
D21           14 NULL   22180.00 22180.00000000     1
D21           15 NULL   27380.00 27380.00000000     1
D21           16 NULL   36170.00 36170.00000000     1
D21           17 NULL   28760.00 28760.00000000     1
E01           16 NULL   40175.00 40175.00000000     1
E11           16 NULL   29750.00 29750.00000000     1
E11           17 NULL   26250.00 26250.00000000     1
E21           14 NULL   51520.00 25760.00000000     2
E21           16 NULL   23840.00 23840.00000000     1
A00         NULL NULL  128500.00 42833.33333333     3
B01         NULL NULL   41250.00 41250.00000000     1
C01         NULL NULL   90470.00 30156.66666666     3
D11         NULL NULL  203830.00 25478.75000000     8
D21         NULL NULL  114490.00 28622.50000000     4
E01         NULL NULL   40175.00 40175.00000000     1
E11         NULL NULL   56000.00 28000.00000000     2
E21         NULL NULL   75360.00 25120.00000000     3
NULL        NULL NULL  750075.00 30003.00000000    25
```

Note that the n elements of the **ROLLUP** translate to *n*+1 grouping sets. Note also that the order in which the grouping-expressions is specified is significant for ROLLUP. For example:

```
GROUP BY ROLLUP(a,b)
```

Is equivalent to

```
GROUP BY GROUPING SETS((a,b),
(a),
() )
```

While

```
GROUP BY ROLLUP(b,a)
```

Is the same as

```
GROUP BY GROUPING SETS((b,a),
(b),
() )
```

The SQL code in Example 6-19 is the equivalent of Example 6-17 on page 122.

Example 6-19 Selecting with grouping sets

```
SELECT WORKDEPT, EDLEVEL, SEX, SUM(SALARY) as SUM_SALARY, AVG(SALARY) AS
AVG_SALARY, count(*) as COUNT
 FROM DSN8111O.EMP WHERE SALARY > 20000
```

```
 GROUP BY GROUPING SETS ((WORKDEPT, EDLEVEL, SEX),(WORKDEPT, EDLEVEL),
(WORKDEPT),())
```

Grand total for the result of this query is the last row on the query result set shown in Example 6-18 on page 122, which can also be identified by the row containing null values for all the three columns on the **ROLLUP** clause (that is, WORKDEPT, EDLEVEL, SEX).

6.10.3 CUBE

A **CUBE** grouping is an extension to the **GROUP BY** clause that produces a result set that contains all the rows of a **ROLLUP** aggregation and, in addition, contains *crosstabulation* rows. Cross-tabulation rows are additional *super-aggregate* rows that are not part of an aggregation with sub-totals.

Like a **ROLLUP**, a **CUBE** grouping can also be thought of as a series of grouping-sets. In the case of a **CUBE**, all permutations of the cubed grouping-expression-list are computed along with the grand total. Therefore, the n elements of a **CUBE** translate to 2**n (2 to the power n) grouping-sets. For example, a specification of

```
GROUP BY CUBE(a,b,c)
```

is equivalent to

```
GROUP BY GROUPING SETS((a,b,c),
(a,b),
(a,c),
(b,c),
(a),
(b),
(c),
() )
```

Notice that the three elements of the **CUBE** translate to eight grouping sets. The order of specification of elements does not matter for **CUBE**.

Example 6-20 shows a sample SQL statement using CUBE in the GROUP BY clause with similar code used in the ROLLUP description in Example 6-17 on page 122.

Example 6-20 Sample SQL statement using CUBE construct in a GROUP BY clause

```
SELECT WORKDEPT, EDLEVEL, SEX, SUM(SALARY) as SUM_SALARY, AVG(SALARY) AS
AVG_SALARY, count(*) as COUNT
 FROM DSN81110.EMP WHERE SALARY > 20000
 GROUP BY CUBE (WORKDEPT, EDLEVEL, SEX);
```

Example 6-21 shows the result set for the query in Example 6-20.

Example 6-21 Result set from the sample CUBE construct

```
WORKDEPT EDLEVEL SEX  SUM_SALARY AVG_SALARY     COUNT
 -------- ------- ---- ---------- -------------- -----
 A00           14 M      29250.00 29250.00000000     1
 A00           18 F      52750.00 52750.00000000     1
 A00           19 M      46500.00 46500.00000000     1
 B01           18 M      41250.00 41250.00000000     1
 C01           16 F      23800.00 23800.00000000     1
 C01           18 F      28420.00 28420.00000000     1
```

```
C01          20 F      38250.00 38250.00000000     1
D11          16 M     130400.00 26080.00000000     5
D11          17 F      43590.00 21795.00000000     2
D11          18 F      29840.00 29840.00000000     1
D21          14 M      22180.00 22180.00000000     1
D21          15 F      27380.00 27380.00000000     1
D21          16 F      36170.00 36170.00000000     1
D21          17 M      28760.00 28760.00000000     1
E01          16 M      40175.00 40175.00000000     1
E11          16 F      29750.00 29750.00000000     1
E11          17 F      26250.00 26250.00000000     1
E21          14 M      51520.00 25760.00000000     2
E21          16 M      23840.00 23840.00000000     1
NULL         14 M     102950.00 25737.50000000     4
NULL         15 F      27380.00 27380.00000000     1
NULL         16 F      89720.00 29906.66666666     3
NULL         16 M     194415.00 27773.57142857     7
NULL         17 F      69840.00 23280.00000000     3
NULL         17 M      28760.00 28760.00000000     1
NULL         18 F     111010.00 37003.33333333     3
NULL         18 M      41250.00 41250.00000000     1
NULL         19 M      46500.00 46500.00000000     1
NULL         20 F      38250.00 38250.00000000     1
A00        NULL F      52750.00 52750.00000000     1
A00        NULL M      75750.00 37875.00000000     2
B01        NULL M      41250.00 41250.00000000     1
C01        NULL F      90470.00 30156.66666666     3
D11        NULL F      73430.00 24476.66666666     3
D11        NULL M     130400.00 26080.00000000     5
D21        NULL F      63550.00 31775.00000000     2
D21        NULL M      50940.00 25470.00000000     2
E01        NULL M      40175.00 40175.00000000     1
E11        NULL F      56000.00 28000.00000000     2
E21        NULL M      75360.00 25120.00000000     3
A00          14 NULL   29250.00 29250.00000000     1
A00          18 NULL   52750.00 52750.00000000     1
A00          19 NULL   46500.00 46500.00000000     1
B01          18 NULL   41250.00 41250.00000000     1
C01          16 NULL   23800.00 23800.00000000     1
C01          18 NULL   28420.00 28420.00000000     1
C01          20 NULL   38250.00 38250.00000000     1
D11          16 NULL  130400.00 26080.00000000     5
D11          17 NULL   43590.00 21795.00000000     2
D11          18 NULL   29840.00 29840.00000000     1
D21          14 NULL   22180.00 22180.00000000     1
D21          15 NULL   27380.00 27380.00000000     1
D21          16 NULL   36170.00 36170.00000000     1
D21          17 NULL   28760.00 28760.00000000     1
E01          16 NULL   40175.00 40175.00000000     1
E11          16 NULL   29750.00 29750.00000000     1
E11          17 NULL   26250.00 26250.00000000     1
E21          14 NULL   51520.00 25760.00000000     2
E21          16 NULL   23840.00 23840.00000000     1
NULL       NULL F     336200.00 30563.63636363    11
NULL       NULL M     413875.00 29562.50000000    14
```

```
NULL          14 NULL  102950.00 25737.50000000     4
NULL          15 NULL   27380.00 27380.00000000     1
NULL          16 NULL  284135.00 28413.50000000    10
NULL          17 NULL   98600.00 24650.00000000     4
NULL          18 NULL  152260.00 38065.00000000     4
NULL          19 NULL   46500.00 46500.00000000     1
NULL          20 NULL   38250.00 38250.00000000     1
A00         NULL NULL  128500.00 42833.33333333     3
B01         NULL NULL   41250.00 41250.00000000     1
C01         NULL NULL   90470.00 30156.66666666     3
D11         NULL NULL  203830.00 25478.75000000     8
D21         NULL NULL  114490.00 28622.50000000     4
E01         NULL NULL   40175.00 40175.00000000     1
E11         NULL NULL   56000.00 28000.00000000     2
E21         NULL NULL   75360.00 25120.00000000     3
NULL        NULL NULL  750075.00 30003.00000000    25
```

In Example 6-20, **CUBE (WORKDEPT, EDLEVEL, SEX)** and **CUBE (EDLEVEL, WORKDEPT, SEX)** yield the same result sets. The use of the word *same* applies to content of the result set, not to its order.

6.10.4 Grand total

Both **CUBE** and **ROLLUP** return a row that is the overall (grand total) aggregation, which can be separately specified with empty parentheses within the **GROUPING SET** clause. It can also be specified directly in the **GROUP BY** clause, although there is no effect on the result of the query.

6.10.5 Grouping expression

When used in conjunction with grouping-sets and super-groups, the **GROUPING** function returns a value that indicates whether a row returned in a **GROUP BY** result is a row that is generated by a grouping set that excludes the column represented by expression. The result of the function is a small integer value, such as 1 or 0.

For details, see:

http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/index.jsp?topic=%2Fcom.ibm.db2z11.doc.updates%2Fsrc%2Ftpc%2Fdb2z_bif_grouping.htm

6.11 ALTER TABLE DROP COLUMN

This function drops the identified column from the table. Any privileges that are associated with the column are revoked.

A column cannot be dropped if any of the following conditions are true:

- The containing table space is not a universal table space.
- The table is a created global temporary table.
- The table is a system-period temporal table.
- The table is a history table.
- The table is an archive-enabled table.

- The table is an archive table.
- The table has an edit procedure or a validation exit procedure.
- The table contains check constraints.
- The table is a materialized query table.
- The table is referenced in a materialized query table definition.
- The column is defined as a security label column.
- The column is an XML column.
- The column is a **DOCID** column.
- The column is a hidden **ROWID** column.
- The column is defined as **ROWID GENERATED BY DEFAULT**, and the table contains a hidden ROWID column.
- The column is a **ROWID** column on which there is a dependent LOB column.
- The column is part of the table partitioning key.
- The column is part of the hash key.
- All of the remaining columns in the table are hidden.
- A view that is dependent on the table has **INSTEAD OF** triggers.
- A trigger is defined on the table.
- Any of the following objects are dependent on the table:
  - Extended indexes
  - Row permissions
  - Column masks
  - Inline SQL table functions

ALTER TABLE DROP COLUMN is considered a pending definition change, at the time that the **ALTER** statement is executed, semantic validation and authorization checking are performed as usual. However, the drop is not applied to the current definition or data at the time of the **ALTER** (that is, catalog and data are untouched). An entry is recorded in the **SYSIBM.SYSPENDINGDDL** catalog table for the pending drop column, and the table space is placed in an advisory **REORG**-pending (**AREOR**) state.

6.12 LIKE_BLANK_INSIGNIFICANT DSNZPARM

The **LIKE_BLANK_INSIGNIFICANT DSNZPARM** value specifies whether blanks are significant when applying the **LIKE** predicate to a string. If set, the blank insignificant behavior applies.

This **DSNZPARM** value provides a new behavior for the **LIKE** predicate that treats trailing blanks within fixed length character strings as insignificant. This behavior is “more compatible” with the results for variable length strings.

This system parameter is off after a fresh install of DB2 for z/OS. If the DB2 system is migrated to DB2 11, the **LIKE_BLANK_INSIGNIFICANT** behavior is disabled by default. The system parameter can be enabled in conversion mode (CM).

This option can significantly impact the behavior of SQL statements if you have a **LIKE** predicate in your SQL statement and if the column referred in the **LIKE** predicate includes undesirable trailing blanks.

Before the **LIKE** predicate is applied, any trailing blanks in a **CHARACTER** or **GRAPHIC** column are stripped to the last non-blank character. If the column contains all blanks, the blank in character position 1 is not stripped. After stripping occurs, the **LIKE** predicate is applied against the stripped column data. Example 6-22 illustrates this behavior.

Example 6-22 LIKE BLANK INSIGNIFICANT DSNZPARM behavior with trailing blanks

```
CREATE TABLE BINSIGNIFICANT (C1 CHAR(10));
INSERT INTO BINSIGNIFICANT VALUES('   AA   ');
INSERT INTO BINSIGNIFICANT VALUES('A  AA  A');
INSERT INTO BINSIGNIFICANT VALUES('AA   ');
INSERT INTO BINSIGNIFICANT VALUES('AAA  A     ');

SELECT * FROM BINSIGNIFICANT
WHERE C1 LIKE '%AA'

---------+---------+---------+---------
C1
---------+---------+---------+---------
   AA
AA
```

Trailing blanks note: Although trailing blanks in the column data are insignificant, trailing blanks in the **LIKE** predicate are significant.

Example 6-23 illustrates the situation when the **LIKE** predicate contains one or more of the "match any character" (usually underscore) in the last position. The "blank significant" (pre-V11 behavior), might have resulted in a match if the last character in the column contained the blank character. The "blank insignificant" behavior no longer results in a match when the column data contains trailing blanks (because the trailing blanks are being stripped during predicate evaluation).

Example 6-23 Sample LIKE predicate to illustrate the stripping of trailing blanks

```
SELECT C1
FROM BINSIGNIFICANT
WHERE C1 LIKE '%AA_';
```

The **LIKE** predicate in Example 6-23 does not even match the two fixed length strings it matched in Example 6-22, although there is a trailing blank in those two rows immediately after the string AA.

7

Application enablement

This chapter describes DB2 11 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 includes the following topics:

- Ensuring application compatibility
- Transparent archiving of temporal data
- Providing support for big data
- Using the scoring adapter to add predictive analytics to OLTP applications
- Using JavaScript Object Notation with IBM DB2
- Suppressing null indexes

7.1 Ensuring application compatibility

DB2 11 for z/OS has mechanisms in place to limit potential SQL and XML incompatibilities on application DML statements. It allows you to ensure application compatibility by using the following functions:

- Identify applications that are affected by incompatible SQL and XML changes through trace records

  This function provides a mechanism to discover which applications will be affected.

  - The **DSNTIJPM** migration job is updated to warn of Static SQL packages affected.
  - IFCID 366 is updated to report on Dynamic SQL.

- Control the compatibility level to DB2 10 at application level

  The following methods apply to transitioning to new behavior:

  - **APPLCOMPAT(VnnR1) BIND/REBIND** option for static SQL
  - **APPLCOMPAT DSNZPARM** for static SQL indicates default value
  - **CURRENT APPLICATION COMPATIBILITY** special register for dynamic SQL
  - **DSN_PROFILE_ATTRIBUTES** for IBM DRDA® applications

For dynamic SQL statements, the **APPLICATION COMPATIBILITY** special register value must be set to appropriate value, and for static SQL statements, the **APPLCOMPAT** bind option must contain the desired value.

Details about the **APPLICATION COMPATIBILITY** special register are discussed in 6.6, “SET CURRENT APPLICATION COMPATIBILITY” on page 114.

Details about the **APPLCOMPAT BIND/REBIND** option are discussed at 12.6.1, “Application and SQL release incompatibilities” on page 357.

The **APPLCOMPAT** bind option applies only to DML (or DDL that contains DML, such as **CREATE VIEW**, **CREATE MASK**, MQTs, and so on) not DDL or DCL. For example, DB2 allows the user to **CREATE** objects, but application compatibility is not checked until the object is referenced:

- The **CREATE TYPE** (array) is allowed, but **SET array-variable = ...** is subject to application compatibility rules (that is, array-variable only available in V11R1 mode).
- The **BIND** option applies to packages: stored procedures, user-defined functions (UDFs), triggers, applications, and so on.

Note: In a distributed environment, if DB2 is configured with **APPLCOMPAT**(V11R1), the value of the accounting string that is returned has a maximum length of 255 bytes. If DB2 is configured with **APPLCOMPAT**(V10R1), the value of the accounting string that is returned has a maximum length of the value returned is 200 bytes.

7.2 Transparent archiving of temporal data

DB2 11 for z/OS provides basic archive and retrieval functions using SQL through a two table approach. The table that contains the current data is called an *archive-enabled* table, and the table that holds the pre-existing rows is called an *archive table*. An application can design its own way to archive data, or DB2 can automatically move rows deleted from an archive-enabled table to the associated archive table. The retrieval of data from a base table or a base table plus its associated archive table is controlled by the setting of a built-in system defined global variable without changing SQL in the applications.

Important: For archive-enabled tables, you do not need to change the application. No DBA intervention to recall data is required.

Whether DB2 automatically moves deleted rows to the archive table depends on the setting of a new global variable. Additionally, you can use the **LOAD** utility with resume behavior to archive data.

You can define a table as an archive-enabled table with an associated archive table for historical rows. The **ALTER TABLE** statement is extended with an **ENABLE ARCHIVE** clause to change an existing table into an archive-enabled table with an associated archive table. You can use the table as the archive table is specified in the **USE** clause. Defining a table as an archive-enabled table results in package invalidation of existing applications that reference the table.

After a table is defined as an archive-enabled table:

- **ALTER TABLE** with the **ADD COLUMN** clause also implicitly adds the new column to the associated archive table.
- If the **SYSIBMADM.GET_ARCHIVE** global variable is set to **Y**, data is retrieved from the archive table when an archive-enabled table is referenced in a table-reference. The access of historical data in the archive table is “transparent” to the application. All subsequent SQL statements including those from invoked function, stored procedure, and trigger. This allows the application to see both active and archive data without modifying the SQL statements in multiple packages. DB2 rewrites the query with **UNION ALL** operator.
- If the **SYSIBMADM.MOVE_TO_ARCHIVE** global variable is set to **Y**, historical data is stored in the associated archive table when a row is deleted in an archive-enabled table. The storing of a row of historical data in the archive table is “transparent” to the application. When the global variable is set to **Y**, an update operation will return an error.
- Any reference to an archive-enabled table for existing values in an **INSERT**, **UPDATE**, **DELETE**, or **MERGE** will not include rows of the associated archive table.
- A system-period temporal table or application-period temporal table cannot be referenced in a data manipulation statement when the archive-enabled table is also referenced when both tables are considered for transparent archive transformations.

Tip: A table cannot be defined as both an archive-enabled table and a system-period temporal table.

Two new updatable global variables are introduced to give control over whether archived rows for rows deleted from an archive-enabled table are automatically written to an associated archive table and whether rows in the archive table are included when an archive-enabled table is referenced in a table-reference.

If the majority of applications retrieve data from base table, there should be no performance degradation by **UNION ALL** to its associated archive table.

You can use the **DISABLE ARCHIVE** clause on the **ALTER TABLE** statement to remove the relationship between the archive-enabled table and the associated archive table. After the **ALTER** statement is successfully processed, both tables are considered ordinary tables. See 7.2.7, “Static application scenario” on page 135 for additional details.

7.2.1 Controls of archive transparency

The following bind/routine options are added to control the sensitivity to the settings of the **SYSIBMADM.GET_ARCHIVE** global variable:

- **ARCHIVESENSITIVE** (default **YES**)
  - **BIND PACKAGE**
  - **REBIND PACKAGE**
  - **REBIND TRIGGER PACKAGE**
  - **CREATE TRIGGER** (implicit trigger package)
- **ARCHIVE SENSITIVE** (default **YES**)
  - **CREATE FUNCTION** (SQL scalar)
  - **ALTER FUNCTION** (SQL scalar)
  - **CREATE PROCEDURE** (SQL native)
  - **ALTER PROCEDURE** (SQL native)
- **ARCHIVE SENSITIVE** (DB2I panels)
  - DB2I Panel **DSNEBP10**
  - DB2I Panel **DSNEBP11**
  - DB2I Panel **DSNEBP19**

Important: The **CREATE TRIGGER** and **REBIND TRIGGER PACKAGE** options fail if the trigger has an archive-enabled table reference in the **WHEN** clause and the trigger package is generated with **ARCHIVESENSITIVE YES**.

7.2.2 Sample code for enabling archive transparency

To use archive transparency, you need two tables properly defined. Then, issue an **ALTER TABLE** statement to define the relationship between the two tables.

Example 7-1 shows sample DDL statements for an archive-enabled table that stores active data, the archive table that stores archive data, and the **ALTER** table to enable archive transparency feature.

Example 7-1 DDL for ARCHIVE ENABLE

```
-- Main table which will be archive enabled
CREATE TABLE POLICY_INFO_AET
(POLICY_ID CHAR(10) NOT NULL,
COVERAGE INT NOT NULL);

-- Archive table to store archive data
CREATE TABLE POLICY_INFO_ARC
(POLICY_ID CHAR(10) NOT NULL,
COVERAGE INT NOT NULL);

-- If the APPLCOMPAT is not set to V11R1 then use the following statement
SET CURRENT APPLICATION COMPATIBILITY = 'V11R1';

-- ARCHIVE ENABLE -- ALTER table to enable archive transparency
ALTER TABLE POLICY_INFO_AET ENABLE ARCHIVE USE POLICY_INFO_ARC;
```

A single **ALTER** statement to add a column or multiple columns on an archive-enabled table also adds the same column or columns to an archive table. Example 7-2 shows a sample **ALTER TABLE** statement.

Example 7-2 ALTER TABLE ADD COLUMN on an archive enabled table

```
ALTER TABLE POLICY_INFO_AET ADD COLUMN UPDATE_TS TIMESTAMP(12);
```

After executing the **ALTER TABLE ADD COLUMN** statement in Example 7-2, DB2 also implicitly adds an **UPDATE_TS** column to the archive table automatically (that is, `policy_info_arc` table).

7.2.3 Inserting rows into archive enabled table

The **INSERT**, **UPDATE**, and **MERGE** statements are all blocked in archive mode. These statements use the following options:

- If `SYSIBMADM.MOVE_TO_ARCHIVE = 'Y'`, the **INSERT**, **UPDATE**, and **MERGE** statements fail based on the assumption that in archive mode, you can only **DELETE**.
- If `SYSIBMADM.MOVE_TO_ARCHIVE = 'N'`, no archive mode failure occurs (that is, it is business as usual).
- If `SYSIBMADM.MOVE_TO_ARCHIVE = 'E'`, the behavior is similar to `Y`, but adds flexibility. The `E` setting does not restrict the use of the data change statements. Users that favor the restriction can set the global variable to `Y`, and users that do not want the restriction can set the global variable to `E`.

Example 7-3 shows a sample SQL **INSERT** statement, along with valid values for the **APPLICATION COMPATIBILITY** special register and the **MOVE_TO_ARCHIVE** global variable.

Example 7-3 Sample INSERT statement with MOVE_TO_ARCHIVE set to N

```
SET CURRENT APPLICATION COMPATIBILITY = 'V11R1';
SET SYSIBMADM.MOVE_TO_ARCHIVE = 'N';
INSERT INTO POLICY_INFO_AET
(SELECT J_POLICY, COVERAGE_NBR, CHG_TIMESTAMP
 FROM DSN81110.TEMP)
```

Example 7-4 shows the negative SQLCODE and the accompanying error message when an INSERT statement was attempted on an archive enabled table with `SET SYSIBMADM.MOVE_TO_ARCHIVE = 'Y'`.

Example 7-4 Error message on an INSERT with MOVE_TO_ARCHIVE set to ' N'

```
DSNT408I SQLCODE = -20555, ERROR:  AN ARCHIVE-ENABLED TABLE IS NOT ALLOWED IN THE
SPECIFIED CONTEXT. REASON CODE  2
```

7.2.4 Deleting rows from an archive enabled table

A single **DELETE** statement can trigger transparent archive when **MOVE_TO_ARCHIVE** global variable is in effect. No additional privilege is required on an archive table. Only the privileges on the delete of an archive-enabled table are needed.

Given an SQL **DELETE** from an archive enabled table, regardless of whether it is dynamic or static SQL and whether the package is bound with the **ARCHIVESENSITIVE** option **YES** or **NO**:

- If the built-in **SYSIBMADM.MOVE_TO_ARCHIVE** global variable contains the Y value, for each row deleted from the archive-enabled table, DB2 inserts it into the corresponding archive table.
- If the built-in **SYSIBMADM.MOVE_TO_ARCHIVE** global variable contains the **N** value (the default), DB2 does no data propagation to the archive table. It basically works similar to a regular delete statement.

This feature is basically an SQL performance improvement to help archiving data with a single **DELETE**. For the application, there is no change on the data-change SQL statements (that is, transparent to the application).

Example 7-5 shows a sample **DELETE** statement along with valid values for the **APPLICATION COMPATIBILITY** special register and the **MOVE_TO_ARCHIVE** global variable.

Example 7-5 Sample DELETE from an Archive Enabled Table

```
SET CURRENT APPLICATION COMPATIBILITY = 'V11R1';
SET SYSIBMADM.MOVE_TO_ARCHIVE = 'Y';
DELETE FROM POLICY_INFO_AET
WHERE UPDATE_TS < CURRENT TIMESTAMP - 7 YEARS;
```

7.2.5 Querying archive enabled table

There is no need to change SQL statements in an existing application to get current data only or to get both current and archive data. The setting of the **SYSIBMADM.GET_ARCHIVE** built-in global variable offers transparent access to archive data for queries.

If the application wants to get the result from both base and archive table, use the following option:

```
SET SYSIBMADM.GET_ARCHIVE = 'Y' ;
```

Note: If the CID 65 field QW0065ER DS CL2 EXPANSION REASON shows the A value, the implicit query transformation driven by the built-in **SYSIBMADM.GET_ARCHIVE** global variable or the **ARCHIVESENSITIVE** did happen.

If the application just wants to access the active data, use the following default option:

```
SET SYSIBMADM.GET_ARCHIVE = 'N' ;
```

Note: If the CID 65 field **QW0065ER DS CL2 EXPANSION REASON** shows a *blank* value, the implicit query transformation driven by the built-in **SYSIBMADM.GET_ARCHIVE** global variable and **ARCHIVESENSITIVE** bind option did *not* happen.

Example 7-6 shows a sample query along with valid values for the **APPLICATION COMPATIBILITY** special register and the **MOVE_TO_ARCHIVE** global variable. This **SELECT** statement is internally converted to include an implicit **UNION ALL** operation with the corresponding archive table.

Example 7-6 Sample SELECT statement on an archive enabled table

```
SET CURRENT APPLICATION COMPATIBILITY = 'V11R1';
SET SQSIBMADM.GET_ARCHIVE = ‘Y’ ;
SELECT * FROM POLICY_INFO_AET
```

7.2.6 Using a dynamic transaction with archive transparency

The steps to use a dynamic transaction with archive transparency include:

- Define the archive-enabled table
- Define the archive table
- Build the link using **ALTER TABLE ENABLE ARCHIVE**
- Archive data, before **DELETE** by executing `SET SYSIBMADM.MOVE_TO_ARCHIVE = 'Y' ;`

When **DELETE** statements are executed on an archive enabled table, DB2 deletes the record or records as usual. In addition, the deleted rows are inserted into the corresponding archive table.

To access both base and archive data, before **PREPARE** or **EXECUTE IMMEDIATE**, use the following option:

```
SET SYSIBMADM.GET_ARCHIVE = 'Y' ;
```

Note: Except the target table of **INSERT/UPDATE/DELETE/MERGE**, DB2 expands all archive-enabled table references to the table expression with **UNION ALL** of base and archive table.

7.2.7 Static application scenario

The cursor on an archive enabled table is defined as usual as for a regular table as shown in Example 7-7 in an application program.

Example 7-7 Sample cursor statement in a static application

```
DECLARE CUR1 CURSOR FOR
SELECT * FROM POLICY_INFO_AET
WHERE POLICY_ID = :H1 ;
```

Bind the package that contains the cursor with **ARCHIVESENSITIVE YES**. Note the default is YES, which can be omitted.

DB2 generates the following internal sections:

- The normal section with blank value of **EXPANSION_REASON** in **SYSIBM.SYSPACKSTMT**:

  ```
  SELECT * FROM POLICY_INFO_AET WHERE POLICY_ID = :h1 ;
  ```

- The extended section with `A` value of **EXPANSION_REASON** in **SYSIBM.SYSPACKSTMT**:

  ```
  SELECT * FROM (SELECT * FROM POLICY_INFO_AET UNION ALL SELECT * FROM
  POLICY_INFO_ARC) WHERE POLICY_ID = :h1 ;
  ```

When using **EXPLAIN** facilities (either through **BIND** with **EXPLAIN(YES)** or through the **EXPLAIN** statement), **PLAN_TABLE** includes two plans for each query referencing archive-enabled table or tables (that is, one without archive table access and one with archive table access). The `expansion_reason` includes blank and “A” respectively.

7.2.8 DISABLE ARCHIVE

The **ALTER TABLE** statement can also be used to specify that the table is no longer an archive-enabled table using the **DISABLE ARCHIVE** clause. The table name must identify an archive-enabled table The definition of the columns is not changed, but the table is no longer treated as an archive-enabled table. The data in both tables are unaffected by the **ALTER** statement. The relationship between the archive-enabled table and the associated archive table is removed. The archive table is not dropped. Only the relationship between the two tables is removed. Subsequent queries that reference the table will not consider rows in the archive table regardless of the setting of the **SYSIBMADM.GET_ARCHIVE** global variable, and deleted rows will not be moved to the archive table regardless of the setting of **SYSIBMADM.MOVE_TO_ARCHIVE** global variable.

Example 7-8 shows a sample **ALTER TABLE** statement with the **DISABLE ARCHIVE** clause on the sample archive-enabled table that is used in this chapter.

Example 7-8 DDL for DISABLE ARCHIVE statement

```
-- If the APPLCOMPAT is not set to V11R1 then use the following statement
SET CURRENT APPLICATION COMPATIBILITY = 'V11R1';

-- ALTER table to disable archive transparency
ALTER TABLE POLICY_INFO_AET DISABLE ARCHIVE;
```

After successful execution of the **ALTER TABLE ... DISABLE ARCHIVE** statement, the packages and statements in the dynamic statement cache that reference this table are invalidated.

Archiving cannot be disabled if there are any views, materialized query table definitions, or inline SQL table functions that reference the table.

7.2.9 Analytics Accelerator - HPSS considerations

The IBM DB2 Analytics Accelerator V3 (and later) has an archive transparency function with High Performance Storage feature. This feature of the Accelerator is supported by DB2 11 for z/OS, which allows applications to deal with local archive tables compared to completely offloaded archive tables with Analytics Accelerators.

7.3 Providing support for big data

Big data systems are used for analytics. They accept “queries” and return results, usually asynchronously. With DB2 11 for z/OS, applications can connect to both big data systems and DB2 to provide integration. However, SQL connection to big data systems provides convenience with simplified interface and better productivity.

Opportunities exist to lower risk and cost and to create up-sell and cross-sell opportunities by looking at information about social networking (such as Facebook, Twitter, and LinkedIn), and studying data from telemetry devices (machine to machine), detecting customer sentiment in emails, audio, video. However, the challenge remains as to how to integrate this “noise.”

Slowly over time, the circle of trust widened, as shown in Figure 7-1, to include other forms of “differently structured” data.

For example, email is generally structured “From, to, date, and time stamp, subject, attachments, main body with containing sentences, verbs, nouns, adjectives, propositions and closing remarks.” It is structured differently from non-relational data. This is to enhance and augment our knowledge about entities relevant to our business. That way, we can gain deeper insights that help lower business risks and costs, and increase revenue and profit through innovative business models.

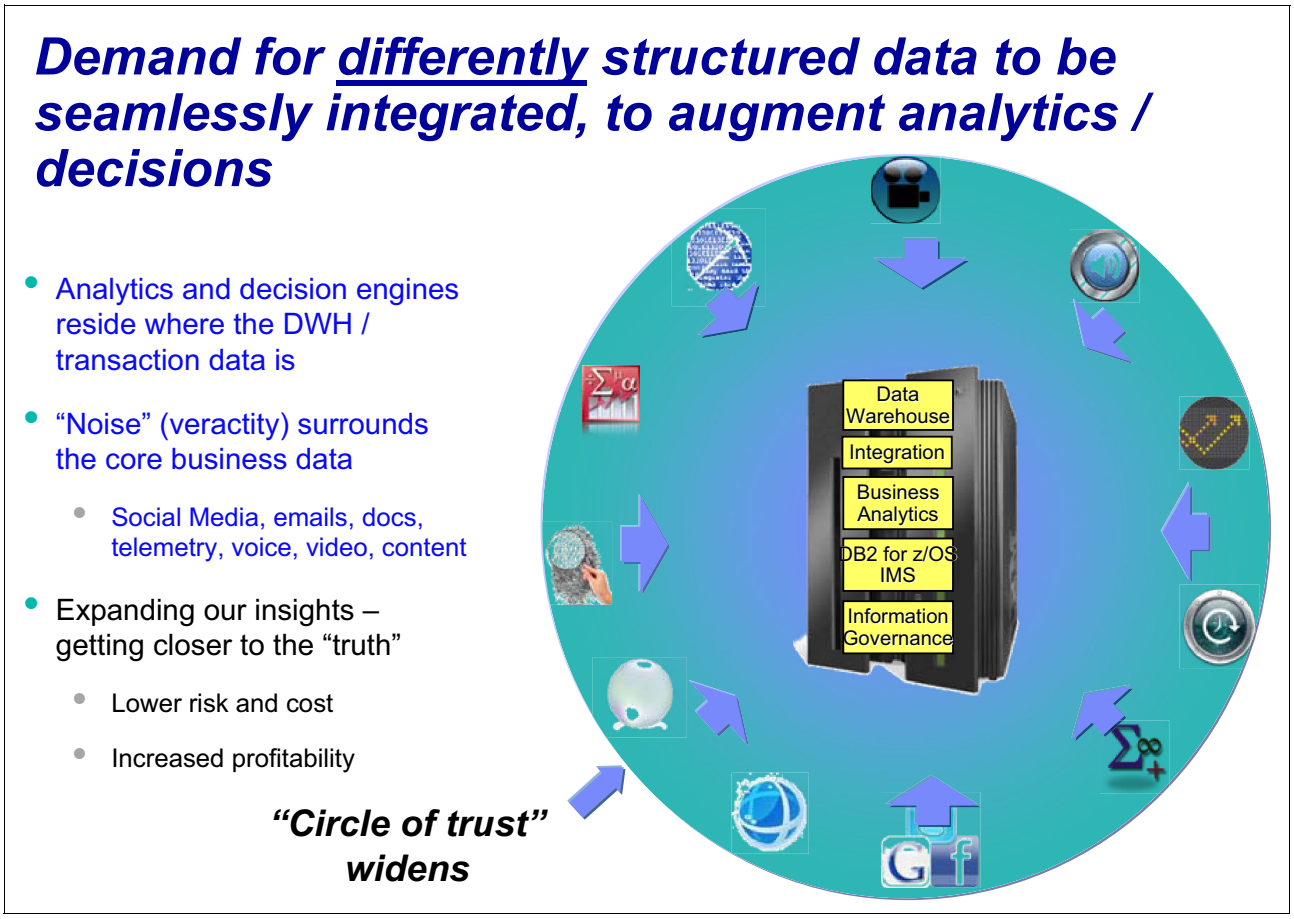


Figure 7-1 Need for differently structured data to gain business insights

The amount and types of data being captured for business analysis is growing. A classic example of this large superset of data is web logs, which contain unstructured raw data.

7.3.1 Enhancing big data analytics with Apache Hadoop

In an increasing trend unstructured data is being stored on new frameworks. These infrastructures encompass hardware and software support such as new file systems, query languages, and appliances. A prime example of DB2 use of Apache Hadoop is shown in DB2 11 for z/OS enhancing Analytics on z platform with big dataDB2 11 for z/OS enhancing Analytics on z platform with big dataFigure 7-2.

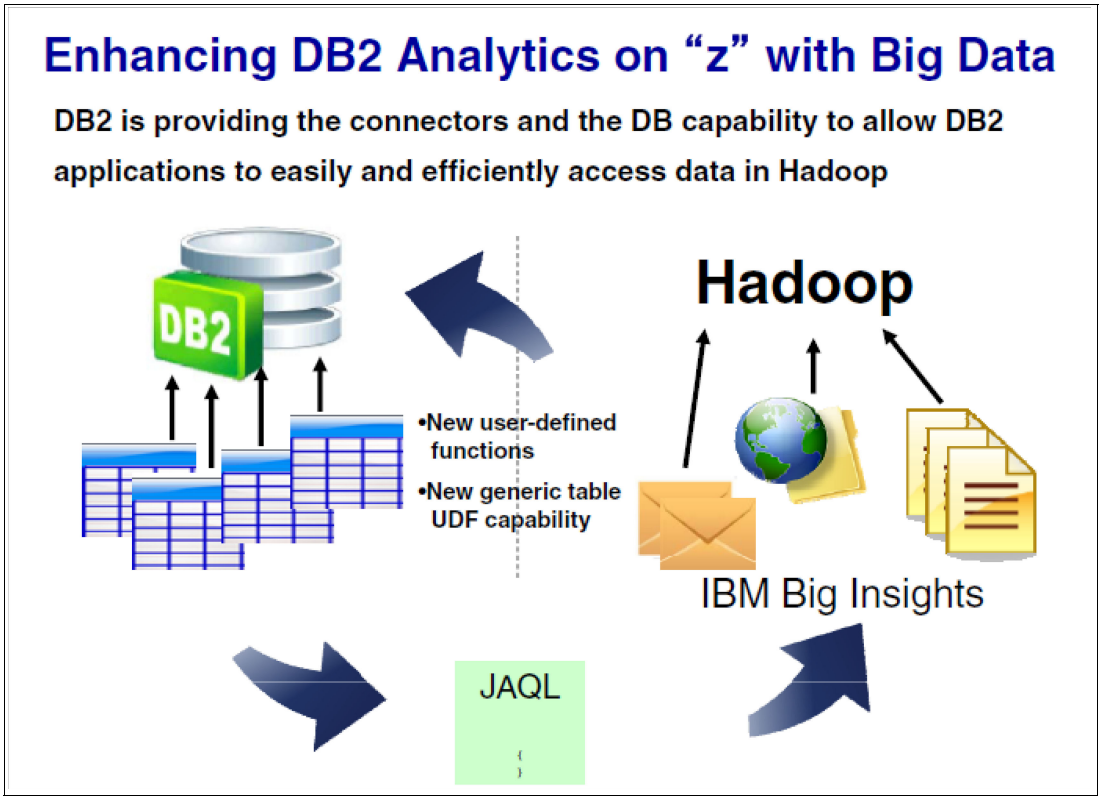


Figure 7-2 DB2 11 for z/OS enhancing Analytics on z platform with big data

Apache Hadoop is a Java-based framework that supports data intensive distributed applications and allows applications to work with thousands of nodes and petabytes of data.

As shown in Figure 7-3, Hadoop framework is ideal for distributed processing of large data sets. It is designed to run on large clusters of commodity hardware.

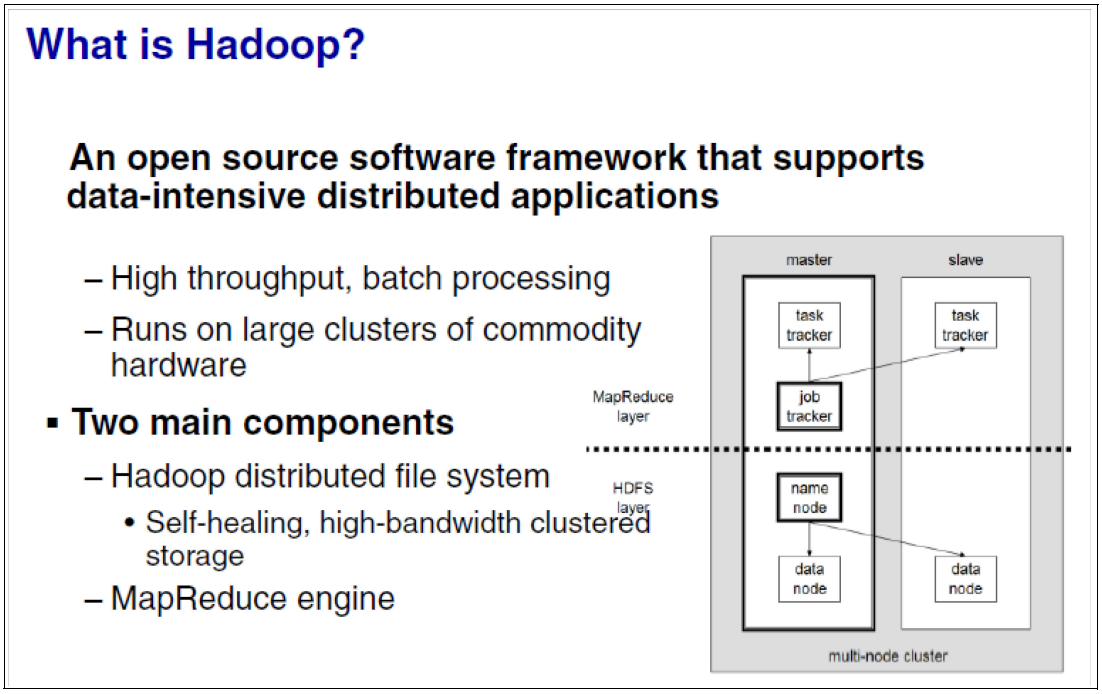


Figure 7-3 Hadoop key components

The Hadoop framework includes the following main components:

- The file systems (HDFS)

  HDFS is a distributed, scalable, and portable file system written in Java for the Hadoop framework, that provides high-throughput access to application data.

- The MapReduce engine

  The MapReduce engine consists of one JobTracker, to which client applications submit MapReduce jobs. The JobTracker pushes work to available TaskTracker nodes in the cluster, striving to keep the work as close to the data as possible.

Hadoop distributed file system

The Hadoop distributed file system (HDFS) is designed to be highly fault tolerant, as illustrated in Figure 7-4.

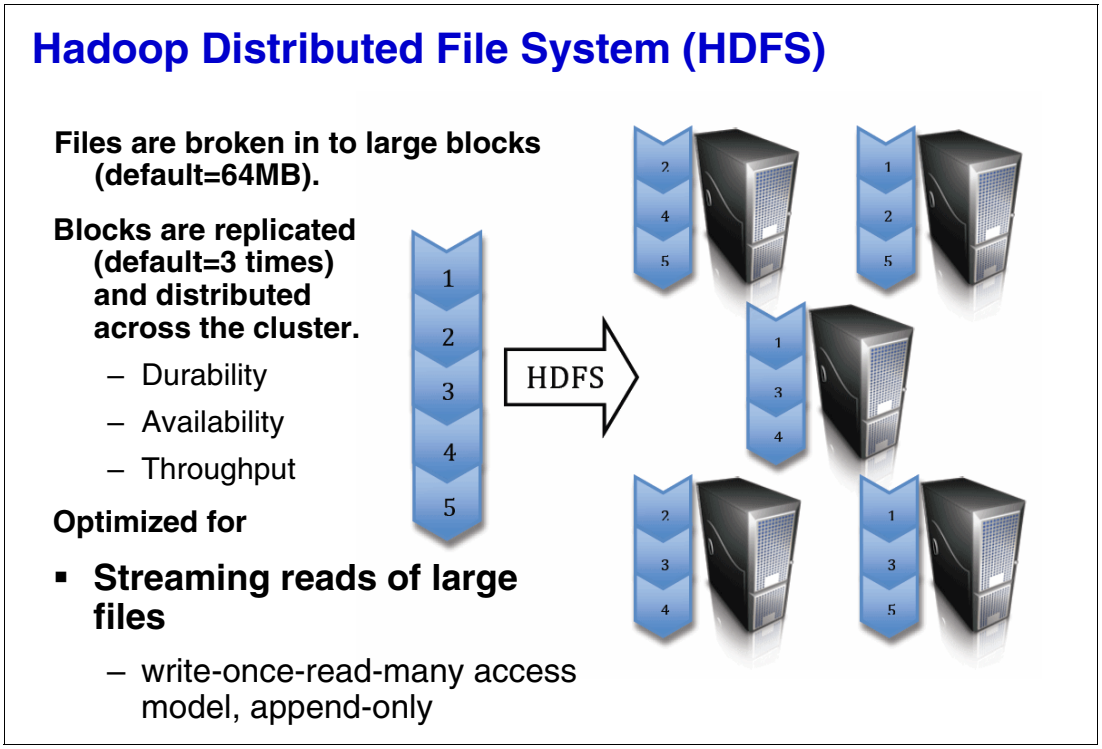


Figure 7-4 HDFS overview

Each node in a Hadoop instance typically has a single name-node; a cluster of data-nodes form the HDFS cluster. Each data-node serves up blocks of data over the network using a block protocol specific to HDFS.

Large files are broken into blocks of fixed size (default = 64 MB), and distributed across multiple machines. Blocks are replicated. Block Replicas are distributed across servers and racks. It achieves reliability by replicating the data across multiple nodes, and hence does not require RAID storage. With the default replication value, 3, data is stored on three nodes. Two are on the same rack, and one node is on a different rack. This policy cuts the inter-rack write traffic which generally improves write performance. The chance of rack failure is far less than that of node failure. Data nodes can talk to each other to rebalance data, to move copies around, and to keep the replication of data high.

HDFS was designed to handle very large files. HDFS was designed for mostly immutable files and might not be suitable for systems requiring concurrent write operations. HDFS applications need a write-once-read-many access model for files. After a file is created, written, and closed, it need not be changed. This assumption simplifies data coherency issues and enables high throughput data access.

MapReduce

Multiple Mappers send data to the multiple Shuffles, which then pass data to the Reducers for “Divide and Conquer” plus mass parallel processing, as shown in Figure 7-5.

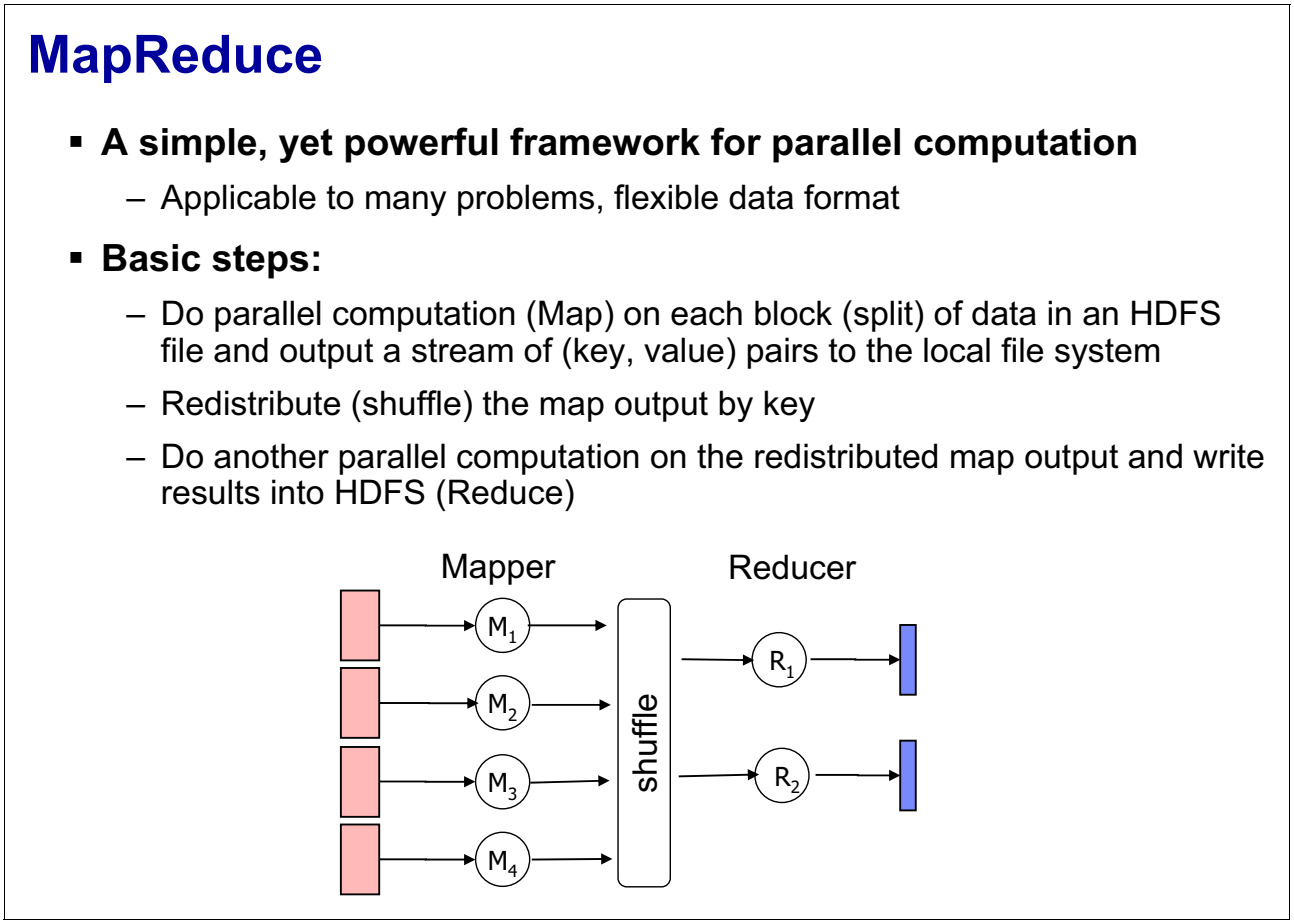


Figure 7-5 MapReduce overview

For example, for an input file that is split into four blocks, with each block containing a list of receipts, there are four mappers. Each mapper reads one block in parallel. The mapper reads each receipt and generates a pair (seller, amount). All the pairs for seller1 are sent to reducer R1, and all seller2 pairs are sent to R2. The reducer then calculates the total amount for each seller.

The Map function only cares about the current key and value. The Reduce function only cares about the current key and its values. A Mapper can invoke Map on an arbitrary number of input keys and values or just some fraction of the input data set. A Reducer can invoke Reduce on an arbitrary number of the unique keys but all the values for that key.

Jaql, the JSON query language

Java MapReduce provides most flexibility and performance, but tedious development cycle (it is similar to the assembly language of Hadoop). Jaql is a functional, declarative query language that is designed by IBM to process large data sets. For parallelism, Jaql rewrites high-level queries, when appropriate, into “low-level” queries consisting of MapReduce jobs.

Read more about how DB2 and JSON work together in 7.5, “Using JavaScript Object Notation with IBM DB2” on page 149.

SQL within Jaql

Jaql integrates an SQL expression that should make it easier for users with an SQL background to write MapReduce scripts in Jaql for the BigInsights environment. SQL within Jaql also makes it easier to integrate existing SQL applications and tooling with Jaql.

Analyze big data with JAQL

Jaql query has three components, as shown in Figure 7-6. You can think of a Jaql query as a pipeline. A Jaql query reads input data from a source. A source is anything from which data can be read, such as a file. A source is the only mandatory part of a Jaql query. Next, the data is manipulated according to the operators or functions that were specified in the query. Finally, the data is output to a sink. A sink is anything to which data can be written.

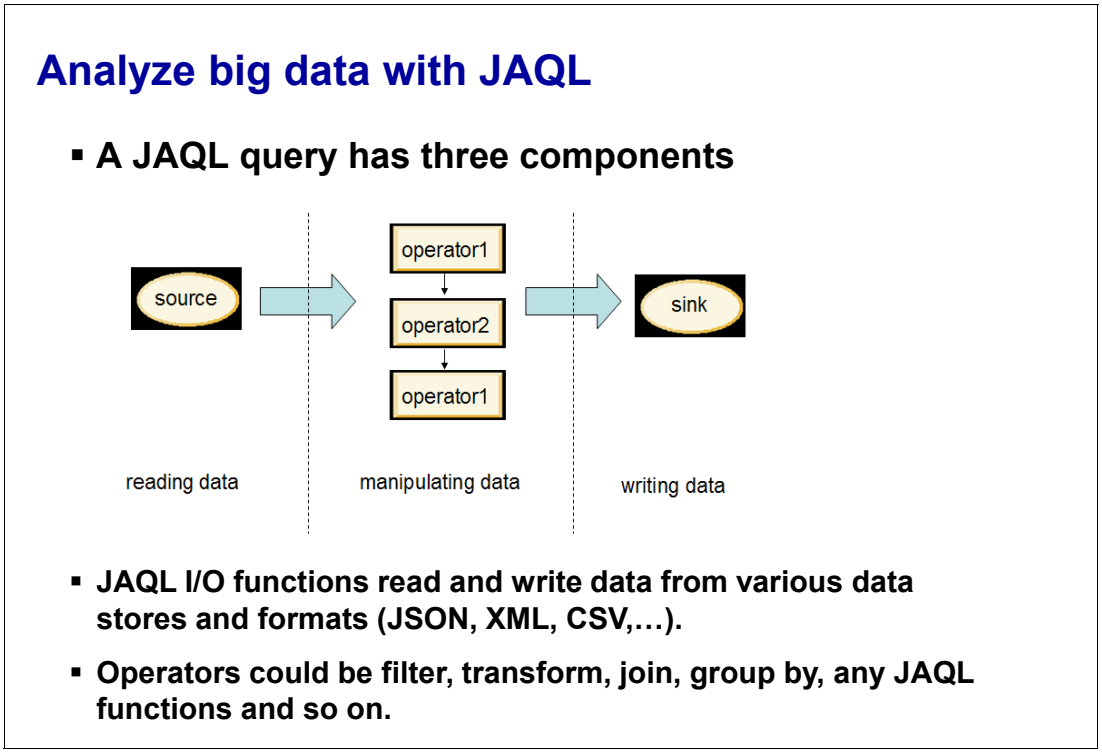


Figure 7-6 JAQL query components

Typical results from big data are in a table form.So, DB2 11 for z/OS provides a single table function to interface with such systems, which requires generic table UDF that returns a table of any shape. Existing infrastructure require one UDF per result type/shape.

- Phase 1: specify the return in FROM clause after the table UDF
- Phase 2: use DESCRIBE interface to get result table shape dynamically

Integration with Hadoop-based IBM BigInsights big data platform

The goal here is to integrate DB2 11 for z/OS with Hadoop based BigInsights Bigdata platform thereby enabling traditional applications on DB2 for z/OS to access Big Data analytics.

The following UDFs are provided to access BigInsight from DB2 for z/OS:

- **`HDFS_Read`** is a user-defined table function to read a file in Hadoop file system. The output schema is determined at query time.
- **`JAQL_Submit`** is a user-defined scalar function to submit a JAQL script to BigInsight.

Analytics jobs can be specified using JSON Query Language (JAQL) submitted to BigInsights, and the results stored in Hadoop Distributed File System (HDFS). The table UDF (**HDFS_READ**) reads the Bigdata analytic result from HDFS, for subsequent use in an SQL query. The **HDFS_READ** UDF output table can have variable shapes.

DB2 11 supports generic table UDF by enabling this function. It also supports the security model of BigInsight, though the BigInsights console and Representational State Transfer (REST).

REST with SOA: REST is the preferred way of communicating in service-oriented architecture (SOA) environments.

Consider the following DB2-BigInsights integration use case:

- BigInsights ingests data that usually is not ingested by established structured data analysis systems such as DB2, for example email from all clients sent to an insurance company.
- DB2 kicks off a Hadoop job on BigInsights that analyzes emails and identifies customers who have expressed dissatisfaction with the service. It looks for the words *cancel*, *terminate*, *switch*, or synonyms thereof and the names of the company's competitors.
- The BigInsights job runs successfully, creates a file of results (names and email addresses of customers at risk), and terminates.
- DB2 reads the BigInsights result file using user-defined table function (**HDFS_READ**).
- DB2 joins the result with the Agent table and alerts the agents of the at-risk customers. The agents act upon the at-risk customer and offer a promotion to stave off defection.

Other use cases are shown in Figure 7-7. These five use cases are the sweet spots for the scenarios discussed in this chapter.

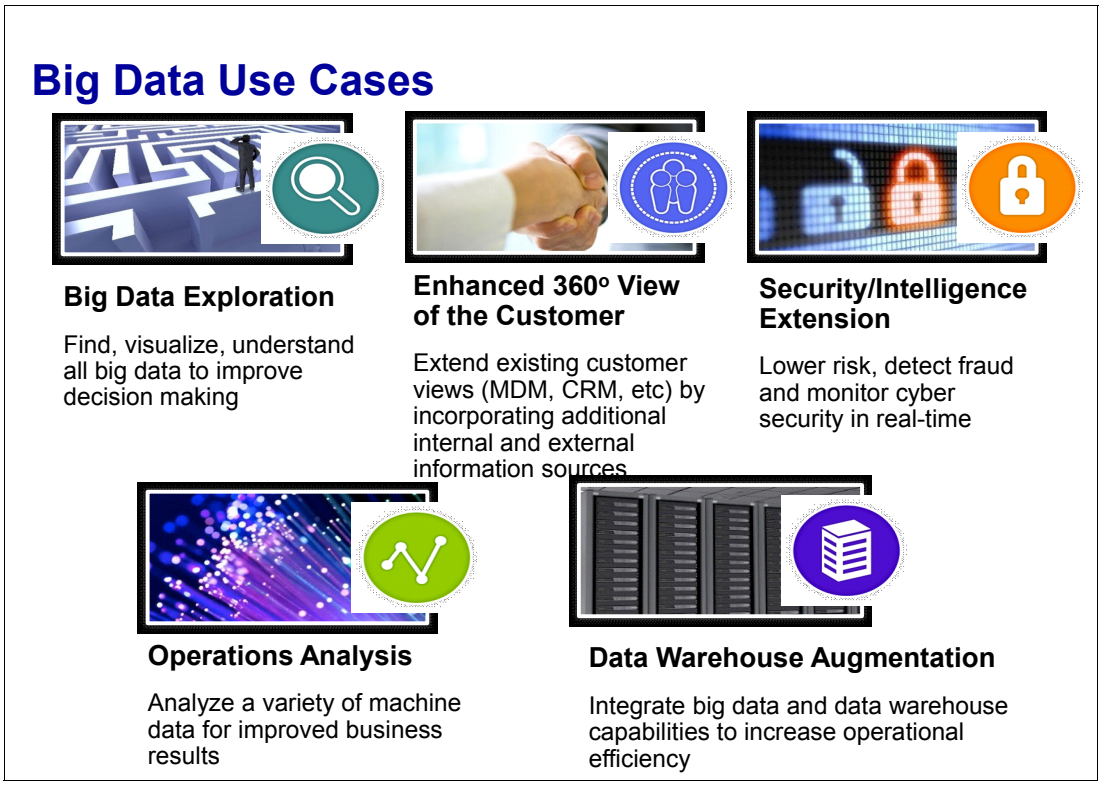


Figure 7-7 Big data use cases

7.3.2 Example HDFS_READ with a generic table UDF

The **HDFS_READ** UDF is a generic table UDF that access HDFS and returns a table of variable shape. Generic TUDF does not specify output table schema at create time but at reference time. The file in HDFS must be in delimited format. It connects BigInsights through REST API.

The **HDFS_READ** UDF takes the following arguments:

- URL, that sets the server address and path of the file in the HDFS
- The option-string that specifies **DELIMITER**, **USER**, and **PASSWORD**

Example 7-9 shows the sample table UDF code along with how it can be invoked from a **SELECT** statement.

Example 7-9 Sample Generic Table UDF code

```
CREATE FUNCTION hdfsRead (handle VARCHAR (xxxx) )
RETURNS GENERIC TABLE
LANGUAGE JAVA PARAMETER STYLE JAVA
EXTERNAL NAME ‘jar:com.ibm.XAP.hdfsRead' ;

SELECT TX.*
FROM TABLE(hdfsRead(‘http://172.16.134.134:8080/data/sample.csv’
) AS TX (station VARCHAR(35),
year VARCHAR(4),
month VARCHAR(2),
loc VARCHAR(10),
```

```
count VARCHAR(5),
minCO2 VARCHAR(7),
maxCO2 VARCHAR(7),
avgCO2 VARCHAR(20));
```

The **HDFS_READ** table function returns one row for each record (or line) in the file. If the number of the result columns m is less than the number of fields in each record, the first m fields of each record is returned. If a field has no value (two adjacent comma), a null value is returned for the corresponding column.

Example 7-10 shows a sample CSV file stored in HDFS.

Example 7-10 Sample HDFS_READ from a CSV file

```
-- Sample CSV file content stored in HDFS
1997,Ford, E350,"ac, abs, moon",3000.00
1999,Chevy, "Venture ""Extended Edition""", ,4900.00
1996,Jeep,Grand Cherokee,"MUST SELL! air, moon roof, loaded",4799.00

-- Sample SELECT statement
Select * From table (HDFS_Read('http://BI.foo.com/data/controller/dfs/file.csv',
                                                    'user=scott
password=tiger'))
                      as X (year integer, make varchar(10), model varchar(30),
                              description varchar(40), price decimal(8,2));

--Results
```

YEAR	MAKE	MODEL	DESCRIPTION	PRICE
1997	Ford	E350	ac, abs, moon roof, fully loaded	3000
1999	Chevy	Venture "Extended Edition"	(null)	4900

7.3.3 Example JAQL_SUBMIT

The first query in Example 7-10, submits to BigInsight a Jaql query which read receipts data in JSON format, select the interested fields: seller's ID, the subtotal amount of the receipt, which is total minus the tax, then group by seller and calculate the total revenue of each seller, and write it to an output file `test1.csv` in comma delimited format.

Assume that T2 is the table contained the tax filing information of each company. The second query in Example 7-11 read the Jaql result file and join it with T2 to find those companies whose reported revenue in their tax form is not equal to the total revenue collected from receipts.

Example 7-11 Sample JAQL_SUBMIT

```
Select Jaql_submit('read(hdfs("receipts.json"))
 ->transform{seller:$.issuer.id,
subtotal: $.subtotal }
  -> group by $seller =$.seller
     into {$seller, revenue: sum($.subtotal))
  -> write (del(location=''/test1.csv''))', ‘’
   'http://kea.svl.ibm.com', ‘TIMEOUT=60 USER=scott PASSWORD=tiger'))
```

```
From SYSIBM.SYSDUMMY1;

Select * From TABLE(HDFS_READ(
 'http://kea.svl.ibm.com/data/controller/dfs/test1.csv',
 'USER=scott PASSWORD=tiger‘ ))
  AS X (seller  varchar(10), revenue decimal(10,2)), T2
Where X.seller = T2.id and X.revenue <> T2.revenue
```

Example 7-12 shows another example of **JAQL_SUBMIT** UDF with nested construct.

Example 7-12 Nested UDF calls

```
Select *
From Table(hdfs_read(Jaql_submit('read(hdfs("receipts.dat"))
                     -> transform { seller: $.Emisor.rfc,
                                    amount: $.Conceptos }
                     -> group by $seller =$.seller
                        into {$seller, total:sum($.amount))
                     -> write(del(location=''/tmp/test1.csv''))',
  'http://kea.svl.ibm.com:8080/data/controller/dfs/tmp/test1.csv',
                     'http://kea.svl.ibm.com:8080',
                     ‘USER=SCOTT PASSWORD=TIGER'),
          ‘USER=SCOTT PASSWORD=TIGER’))
        AS X (seller  varchar(10),
              total   decimal(10,2) );
```

7.4 Using the scoring adapter to add predictive analytics to OLTP applications

You can use IBM SPSS® Modeler Server 15, together with SPSS Modeler Server Scoring Adapter 15 for DB2 on z/OS, to add predictive analytics to OLTP applications that are running on z/OS. You use SPSS Modeler Server to create and train the models and publish them into DB2 on z/OS.

The scoring adapter for DB2 on z/OS provides a scoring engine that runs the UDF run time. The adapter defines a UDF that applications can start by using SQL to run the scoring models synchronously, in-line within their transactions, by using live transaction data as the input for scoring to maximize the effectiveness of scoring results.

Figure 7-8 shows a sample SPSS Modeler stream. You must publish a model nugget, for example, `NN_Is_Fraud`, to the scoring adapter.

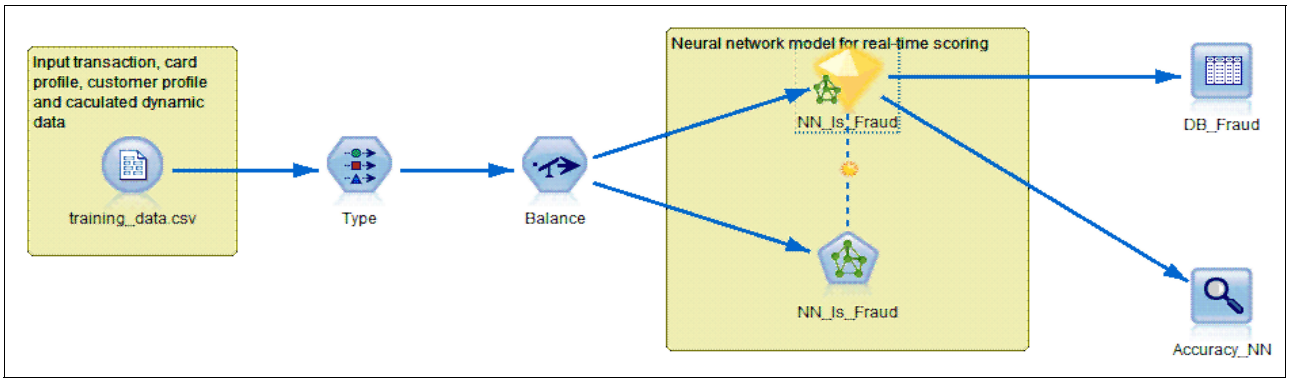


Figure 7-8 SPSS Modeler stream

You must establish a connection to the database before you publish the model nugget. After you set up the connection, publish the nugget to the scoring adapter by clicking **File** → **Publish for server scoring adapter**, as shown in Figure 7-9.

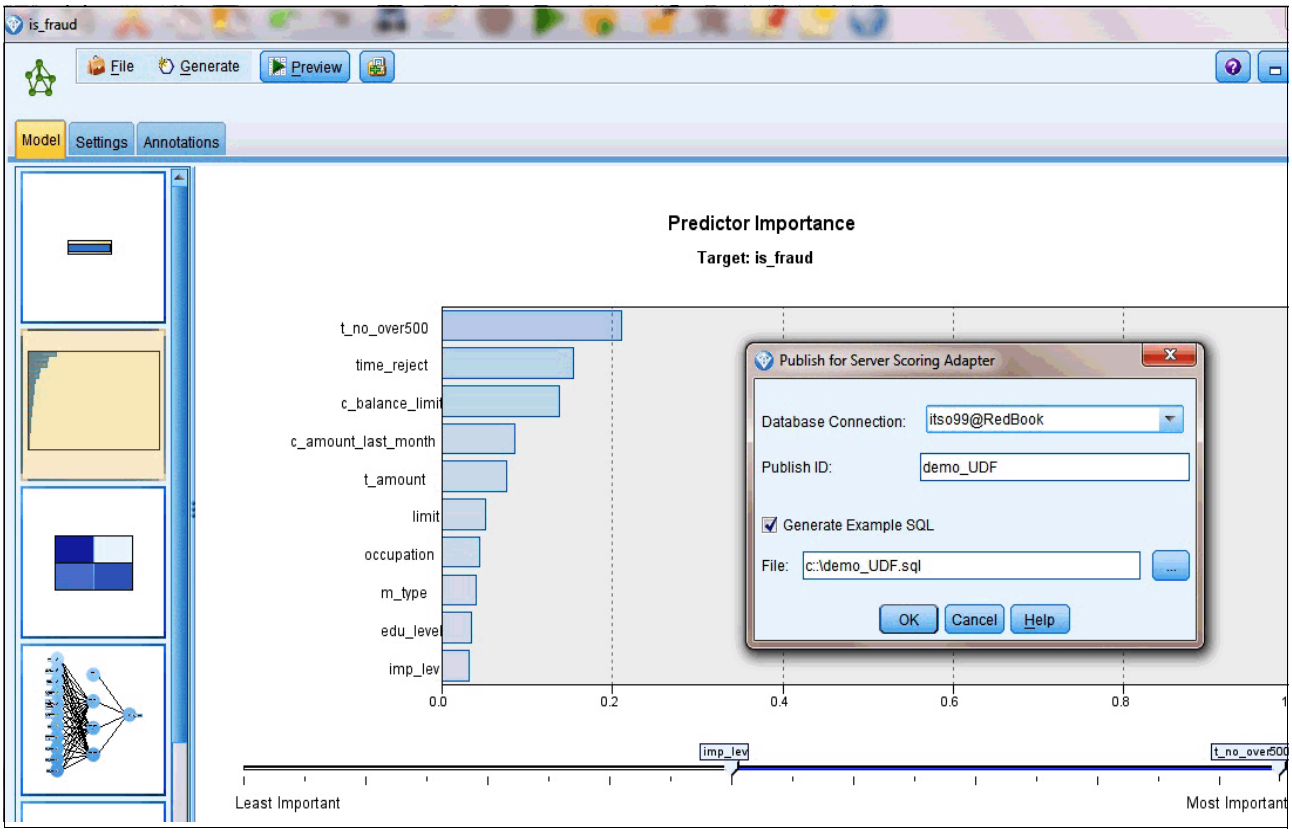


Figure 7-9 Publish for server scoring adapter option

When a model is published to a server scoring adapter, it generates a sample SQL statement. This SQL statement uses UDFs to start the SPSS model that was built earlier and generates a predictive score that can be used by a decision management system.

Example 7-13 shows a sample SQL statement for a scoring adapter.

Example 7-13 Sample SQL statement for a scoring adapter for DB2 on z/OS

```
SELECT
   UNPACK
      (HUMSPSS.SCORE_COMPONENT('P',
               'demo_UDF',
               PACK(CCSID 1208,  T0.C0,T0.C1,T0.C2,T0.C3,T0.C4,T0.C5,T0.C6,T0.C7,
                                 T0.C8,T0.C9,T0.C10,T0.C11,T0.C12,T0.C13,T0.C14,
                                 T0.C15,T0.C16,T0.C17,T0.C18,T0.C19,T0.C20,T0.C21,
                                 T0.C22,T0.C23
                   )
                )
      ).* AS
      (
      C24 BIGINT,C25 DOUBLE,C26 DOUBLE
      )

   FROM (
      SELECT
         T0."CARD_ID" AS C0,
         T0."T_AMOUNT" AS C1,
         T0."T_NO_OVER500" AS C2,
         T0."C_BALANCE_LIMIT" AS C3,
         T0."C_AMOUNT_LAST_MONTH" AS C4,
         T0."TIME_NO_3HOUR" AS C5,
         T0."TIME_AMOUNT_3HOUR" AS C6,
         T0."M_HISTORY" AS C7,
         T0."M_TYPE" AS C8,
         T0."T_TIME" AS C9,
         T0."E_TIME_LAG" AS C10,
         T0."TIME_REJECT" AS C11,
         T0."E_REJECT" AS C12,
         T0."LIMIT" AS C13,
         T0."GENDER" AS C14,
         T0."EDU_LEVEL" AS C15,
         T0."MAR_STAT" AS C16,
         T0."IMP_LEV" AS C17,
         T0."OCCUPATION" AS C18,
         T0."ECO_CAT" AS C19,
         T0."ANNUAL_SALARY" AS C20,
         T0."OWN_HOU_FLAG" AS C21,
         T0."VENDORS_IN_30_MINUTES" AS C22,
         T0."MERCHANT_COUNTRY" AS C23
         FROM ${TABLE0} T0
      ) AS T0
```

In the example, the SQL query returns Score(C24), Confidence(C25), and Normalized Propensity(C26) as output predicted scores that can be used by a decision management system for making runtime decisions. Running the scoring adapter SQL within the DB2 environment provides scalability and performance similar to DB2 for z/OS. This situation makes it possible to handle large transaction volumes and heavy workloads and also meet stringent response time requirements and SLAs.

7.5 Using JavaScript Object Notation with IBM DB2

JavaScript Object Notation (JSON) is a lightweight, text-based, human-readable format. It is a language-independent data interchange format that is becoming increasingly popular as an alternative to XML. JSON is currently supported by many different programming language APIs, which makes it a simple to use document format.

JSON is based on a subset of the JavaScript Programming Language, Standard ECMA-262 Third Edition, December 1999. It is a text format that is language independent but uses conventions that are familiar to programmers of the C-family of languages.

Many applications might use it just for data interchange. Thus, they rarely save the JSON files to disk as the interchanges occur between Internet-connected computers. However, for databases, sending and retrieving data from the IBM DB2 platform to the Internet and thus creating the requirement to be able to use DB2 as a repository of JSON documents is critical.

The IBM DB2 Accessories Suite for DB2 11 feature is enhanced with necessary components to enable DB2 for z/OS to be used as a JSON document store. Included are the following JSON capabilities:

- A programming interface for Java applications to store, update and intelligently query JSON documents
- A command line processor for performing administration tasks and data access operations on JSON data
- A Wire Listener service that extends the support to other languages through an open source wire protocol for accessing JSON data

The JSON capability is a driver-based solution that embraces the flexibility of the JSON data representation within the context of an RDBMS that includes well-known enterprise features and quality of service.

With this offering, applications can manage JSON documents in DB2 for z/OS using a new application programming interface (API) which is designed after the MongoDB data model and query language. This API uses available DB2 for z/OS capabilities to store, modify, and retrieve JSON documents. This allows existing DB2 administration skills, resources, and processes to be utilized for managing this new type of data in DB2 for z/OS.

Using JSON capability, users can interact with JSON data in the following ways:

- They can administer and interactively query JSON data using a command line shell.
- They can programmatically store and query data from Java programs using a driver for JSON supplied by IBM that enables them to connect to their JSON data through the same JDBC driver used for SQL access.
- They can use any driver that implements the MongoDB protocol. This function enables them to access JSON stored from a variety of modern languages, including node.js, PHP, Python, and Ruby, and more traditional languages such as C, C++, and Perl.

7.6 Suppressing null indexes

Having to index every data row affects performance and the size of the index. When creating an index, it is useful to exclude one or more values from being indexed, such as values that will never be used in a query, for example NULL, blank, and 0. DB2 11 NFM can improve insert performance of NULL entries by the option of excluding NULL rows from indexes.

The **CREATE INDEX** statement is changed to state **EXCLUDE NULL KEYS**, and the **RUNSTATS** utility collect statistics only on non-**NULL** value.

All table statistics derived from an index are adjusted by the number of excluded **NULL** values. Therefore the table statistics will be the same whether they were derived from a table scan, an **EXCLUDE NULL KEYS** index, or a non-**EXCLUDE NULL KEYS** index (or **INCLUDE NULL KEYS** index). After converting existing indexes to **EXCLUDE NULL** indexes, monitor application performance. Insert performance should improve and query performance difference should be minimal.

8

XML

Extensible Markup Language (XML) is a markup language that 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 on February 10, 1998.

DB2 9 for z/OS introduced support for the XML data type through the use of its pureXML capabilities and 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 *DB2 10 for z/OS Technical Overview*, SG24-7892 and *Extremely pureXML in DB2 10 for z/OS*, SG24-7915.

DB2 11 provides many additional enhancements to XML functionality. Some of these enhancements are also retrofitted to DB2 10.

This chapter describes the following XML enhancements in DB2:

- XQuery support
- XML performance enhancements in DB2 10 and DB2 11
- XQuery FLWOR expressions performance enhancements
- XMLTABLE performance enhancements in DB2 11

8.1 XQuery support

The capability to store and access XML documents in DB2 for z/OS using an XML data type was first introduced in DB2 9. The initial implementation of XML in DB2 made use of the XPath language, which is a language for navigating XML documents and addressing parts of XML documents. XPath is a subset of XQuery, which is a richer language for accessing XML documents.

Although you can code SQL statements to access XML documents in DB2 9, the XPath language make it difficult to write meaningful queries against XML data. In addition, the XPath language support in DB2 for z/OS is a small subset of the XQuery language that is supported on DB2 for Linux, UNIX, and Windows, which makes porting applications from DB2 for Linux, UNIX, and Windows to DB2 for z/OS difficult. As a result, you likely faced lost productivity due to the following issues:

- Rewriting XQuery queries to use syntax that DB2 supported
- Using a mixture of XPath and SQL/XML, which can be difficult to express
- Some query semantics not supported by XPath on DB2
- Challenges porting applications between different DB2 family members

The XQuery language support provided in DB2 11 and retrofitted to DB2 10 using APARs PM47617 and PM47618 allows application programmers to express such semantics and avoid unnecessary query rewrites. You can now spend less time switching back and forth between XQuery and SQL/XML and can express queries purely using XQuery instead. Differences between XQuery language support in DB2 for z/OS and DB2 for Linux, UNIX, and Windows still exist, but now in DB2 for z/OS, you can use commonly used XQuery language features, such as `for`, `let`, constructors, and `if-then-else`.

You still use the same XML functions available in prior versions of DB2, such as **`XMLQUERY`**, **`XMLEXISTS`**, and **`XMLTABLE`**. The **`XMLQUERY`** function is used in the **`SELECT`** clause of an SQL query to specify which XML data to retrieve. The **`XMLTABLE`** function is used in the **`FROM`** clause to extract XML data in a table format. The **`XMLEXISTS`** function is used in the **`WHERE`** clause to specify under which conditions to retrieve the data.

The difference is that you now have a much richer set of XML expressions that you can supply to the XML functions. You can use the following types of expressions alone or in combination:

- **`FLWOR`** expressions

  A **`FLWOR`** expression is a loop construct for manipulating XML documents in an application-like manner. The name (pronounced *flower*) is an acronym for the keywords used in the expression (FLWOR = For-Let-Where-Order By-Return).

- XQuery constructors

  Instead of using publishing functions for creating XML elements, documents, and other XML constructs, you can now write them as literals anywhere that you can write an expression of the same type.

- Conditional expressions

  You can use IF-THEN-ELSE logic anywhere you can use an expression within an XQuery expression.

- Built-in functions

  New built-in function for XQuery to return the average of the values in a sequence.

- XQuery prolog

  The prolog has new declarations that define the processing environment for a query for XQuery.

In the following sections we provide some details and examples of using each of these types of expressions.

8.1.1 FLWOR expressions

Table 8-1 lists a description and example of usage for each keyword for **FLWOR**.

Table 8-1 XQuery FLWOR expression keywords

Keyword	Description
for	Allows a variable to loop through a sequence of values. These can be literals, XPath expressions, and so on. The looping variable is prefixed with $, as with other variables in XML expressions. An example is as follows: `for $i in /$po//item`
let	Assigns a variable a single value. This can be a literal, an XPath expression, and so on. The variable is prefixed with $. An example is as follows: `let $p := $po/ipo:purchaseOrder/items/item/USPrice`
where	Defines the criteria for which values are to be returned, as with an SQL query. An example is as follows: `where $j/name=$i/billTo/name and $j/status="premier"`
order by	Orders the output specified in the return clause, as with an SQL query. An example is as follows: `order by xs:decimal($i/USPrice) descending`
return	Specifies what is to be returned from each iteration of the FLWOR expression. The final result is their concatenation. `return $i`

The **FLWOR** expression keyword syntax provides similar capabilities for XML data as SQL keywords do for relational data. The following example queries show how these keywords are used.

The first example creates two tables with XML data:

- One table to contain purchase order information
- One table to contain customer status information

Each table includes two columns:

- An **INTEGER** column that is defined as an **IDENTITY** column
- An XML column to contain the customer data

Example 8-1 shows the DDL for the purchaseOrdersXML table.

Example 8-1 DDL for purchaseOrdersXML table

```
CREATE TABLESPACE DB2R3XTS IN DSNDB04 BUFFERPOOL BP4
USING STOGROUP SYSDEFLT
PRIQTY 1000
SECQTY 1000
;
CREATE TABLE purchaseOrdersXML
```

```
(id_col INTEGER GENERATED BY DEFAULT
         AS IDENTITY(START WITH 1
                     INCREMENT BY 1,
                     CACHE 20) NOT NULL,
 po XML
)
IN DSNDB04.DB2R3XTS
;
```

Example 8-2 shows the DDL for the statusXML table.

Example 8-2 DDL for statusXML table

```
CREATE TABLESPACE DB2R3XT2 IN DSNDB04 BUFFERPOOL BP4
USING STOGROUP SYSDEFLT
PRIQTY 1000
SECQTY 1000
;
CREATE TABLE statusXML
(id_col INTEGER GENERATED BY DEFAULT
         AS IDENTITY(START WITH 1
                     INCREMENT BY 1,
                     CACHE 20) NOT NULL,
 status XML
)
IN DSNDB04.DB2R3XT2
;
```

Important: Unlike SQL statements on a relational data, XQuery statements are case sensitive. If you intend to replicate these tests, you need to make sure that you set *CAPS OFF* if running these examples in SPUFI.

Example 8-3 shows the SQL statements to insert two rows into the purchaseOrdersXML table. Note a value for the **ID_COL** column is not provided, because that column is defined as an **IDENTITY** column and a value is generated by default.

Example 8-3 INSERT statements for purchaseOrdersXML table

```
INSERT INTO purchaseOrdersXML
(po)
VALUES(XMLPARSE(DOCUMENT
'<ipo:purchaseOrder
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:ipo="http://www.example.com/IPO"
  orderDate="1999-12-01">
  <shipTo exportCode="1" xsi:type="ipo:UKAddress">
    <name>Helen Zoe</name>
    <street>55 Eden Street</street>
    <city>San Jose</city>
    <state>CA</state>
    <postcode>CB1 1JR</postcode>
  </shipTo>
  <shipTo exportCode="1" xsi:type="ipo:UKAddress">
    <name>Joe Lee</name>
    <street>66 University Avenue</street>
```

```
    <city>Palo Alto</city>
    <state>CA</state>
    <postcode>CB1 1JR</postcode>
  </shipTo>
  <billTo xsi:type="ipo:USAddress">
    <name>Robert Smith</name>
    <street>8 Oak Avenue</street>
    <city>Old Town</city>
    <state>PA</state>
    <zip>95819</zip>
  </billTo>
  <items>
    <item partNum="833-AA">
      <productName>Lapis necklace</productName>
      <quantity>1</quantity>
      <USPrice>99.95</USPrice>
      <ipo:comment>Want this for the holidays!</ipo:comment>
      <shipDate>1999-12-05</shipDate>
    </item>
    <item partNum="945-ZG">
      <productName>Sapphire Bracelet</productName>
      <quantity>2</quantity>
      <USPrice>178.99</USPrice>
      <shipDate>2000-01-03</shipDate>
    </item>
  </items>
 </ipo:purchaseOrder>
'))
;
INSERT INTO purchaseOrdersXML
(po)
VALUES(XMLPARSE(DOCUMENT
'<ipo:purchaseOrder
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:ipo="http://www.example.com/IPO"
  orderDate="1999-12-01">
  <shipTo exportCode="1" xsi:type="ipo:UKAddress">
    <name>James Doe</name>
    <street>77 Eden Street</street>
    <city>San Jose</city>
    <state>CA</state>
    <postcode>CB1 1JR</postcode>
  </shipTo>
  <shipTo exportCode="1" xsi:type="ipo:UKAddress">
    <name>Hal Yee</name>
    <street>99 University Avenue</street>
    <city>Palo Alto</city>
    <state>CA</state>
    <postcode>CB1 1JR</postcode>
  </shipTo>
  <billTo xsi:type="ipo:USAddress">
    <name>Albert James</name>
    <street>5 Oak Avenue</street>
    <city>Old Town</city>
    <state>PA</state>
```

```
    <zip>95819</zip>
  </billTo>
  <items>
    <item partNum="844-AA">
      <productName>Lapis bracelet</productName>
      <quantity>1</quantity>
      <USPrice>89.95</USPrice>
      <ipo:comment>Want this for the holidays!</ipo:comment>
      <shipDate>1999-12-08</shipDate>
    </item>
    <item partNum="947-ZG">
      <productName>Sapphire Earring</productName>
      <quantity>2</quantity>
      <USPrice>187.99</USPrice>
      <shipDate>2000-01-05</shipDate>
    </item>
  </items>
 </ipo:purchaseOrder>
'))
;
```

Example 8-4 shows the SQL statements to insert two rows into the statusXML table. Again, only the contents of the XML column are shown. DB2 generates the value for the **IDENTITY** column.

Example 8-4 INSERT statements for statusXML table

```
INSERT INTO statusXML
(status) VALUES(XMLPARSE(DOCUMENT
'<status>
  <statusItem>
    <name>Robert Smith</name>
    <status>premier</status>
    <comment>Orders a lot of jewelry</comment>
    <comment>Has friends in the Silicon Valley</comment>
  </statusItem>
  <statusItem>
    <name>Jason C</name>
    <status>blacklist</status>
    <comment>This guy doesn''t pay his bills!</comment>
  </statusItem>
</status>'))
;
INSERT INTO statusXML
(status) VALUES(XMLPARSE(DOCUMENT
'<status>
  <statusItem>
    <name>Albert James</name>
    <status>regular</status>
    <comment>Occasionally orders jewelry</comment>
    <comment>Has friends in San Francisco city</comment>
  </statusItem>
  <statusItem>
    <name>James B</name>
    <status>blacklist</status>
    <comment>This guy doesn''t pay his bills!</comment>
```

```
  </statusItem>
</status>'))
;
```

Now that there is XML data in the two tables, the next examples show the **FLWOR** code expressions to retrieve XML data and format the results.

Simple FLWOR use case

The **for** keyword allows a variable to loop through a sequence of values, similar to a cursor on relational data. Example 8-5 shows how to use the **for** keyword to read through all the rows in the purchaseOrdersXML table.

Example 8-5 Use of FLWOR "for" keyword to loop through a sequence of values

```
SELECT XMLQUERY(
  'declare namespace ipo="http://www.example.com/IPO";
  for $i in $po/ipo:purchaseOrder
  return
    <itemsShipped xmlns:ipo="http://www.example.com/IPO">
      <to> {$i/shipTo/name/text()} </to>
      <items> {$i/items/item} </items>
    </itemsShipped>'
  PASSING PO as "po")
FROM purchaseOrdersXML;
```

Example 8-6 shows the results of the query. When you run this example query in SPUFI, the results for each row is shown on one line. This example is formatted to make it easier for you to read.

Example 8-6 Results of sample XQuery using FLWOR keyword "for"

```
<?xml version="1.0" encoding="IBM037"?>
<itemsShipped xmlns:ipo="http://www.example.com/IPO">
  <to>Helen ZoeJoe Lee</to>
  <items>
    <item xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" partNum="833-AA">
      <productName>Lapis necklace</productName>
      <quantity>1</quantity>
      <USPrice>99.95</USPrice>
      <ipo:comment>Want this for the holidays!</ipo:comment>
      <shipDate>1999-12-05</shipDate>
    </item>
    <item xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" partNum="945-ZG">
      <productName>Sapphire Bracelet</productName>
      <quantity>2</quantity>
      <USPrice>178.99</USPrice>
      <shipDate>2000-01-03</shipDate>
    </item>
  </items>
</itemsShipped>
<?xml version="1.0" encoding="IBM037"?>
<itemsShipped xmlns:ipo="http://www.example.com/IPO">
  <to>James DoeHal Yee</to>
  <items>
    <item xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" partNum="844-AA">
```

```
      <productName>Lapis bracelet</productName>
      <quantity>1</quantity>
      <USPrice>89.95</USPrice>
      <ipo:comment>Want this for the holidays!</ipo:comment>
      <shipDate>1999-12-08</shipDate>
    </item>
    <item xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" partNum="947-ZG">
      <productName>Sapphire Earring</productName>
      <quantity>2</quantity>
      <USPrice>187.99</USPrice>
      <shipDate>2000-01-05</shipDate>
    </item>
  </items>
</itemsShipped>
DSNE610I NUMBER OF ROWS DISPLAYED IS 2
```

Notice that before each row returned there is a string of text that looks as follows:

```
<?xml version="1.0" encoding="IBM037"?>
```

This information is called the *XML declaration*, which is kind of like a header for an XML document. If the data has an XML declaration before it is sent to the database server, the XML declaration is not preserved. However, for DB2 ODBC and embedded SQL applications, implicit serialization is used by default. With implicit serialization, the DB2 database server adds an XML declaration, with the appropriate encoding specification, to the data. For Java and .NET applications, the DB2 database server does not add an XML declaration, but if you retrieve the data into a DB2 XML object and use certain methods to retrieve the data from that object, the IBM Data Server Driver for JDBC and SQLJ adds an XML declaration.

If you do not want to display the XML declaration as part of the output and just want to display the data, you can use the **XMLSERIALIZE** function with the **EXCLUDING XMLDECLARATION** option on the results of the **XMLQUERY** function to exclude the XML declaration.

Example 8-7 shows a sample XQuery statement using the **XMLSERIALIZE** function.

Example 8-7 Sample XQuery using FLWOR keyword “for” and XMLSERIALIZE

```
SELECT XMLSERIALIZE(XMLQUERY(
  'declare namespace ipo="http://www.example.com/IPO";
  for $i in $po/ipo:purchaseOrder
  return
    <itemsShipped xmlns:ipo="http://www.example.com/IPO">
      <to> {$i/shipTo/name/text()} </to>
      <items> {$i/items/item} </items>
    </itemsShipped>'
  PASSING PO as "po")
AS CLOB
VERSION '1.0'
EXCLUDING XMLDECLARATION)
FROM purchaseOrdersXML;
```

The query results are exactly the same but without the XML declaration information before each row. This scenario excludes the **XMLSERIALIZE** function from all subsequent examples in this chapter to make the SQL statements easier to read. It also removes the XML declaration from all subsequent results, for the same reasons. The inclusion or exclusion of XML

declaration information in your query results depends on the source from which you execute your query and whether you use the **XMLSERIALIZE** function. <<STOP>>

Use of all FLWOR keywords

Now that we have seen a simple example of a FLWOR expression in XQuery, let's build an example that uses all the keywords. Let us read the statusXML table to return all customers who have a status of "blacklist". Example 8-8 shows the XQuery expression that we wrote to accomplish the desired result.

Example 8-8 Sample XQuery using all FLWOR keywords

```
SELECT XMLQUERY (
'for $i1 in $st/status/statusItem
let $sts := $i1/status
where $sts = "blacklist"
order by $i1/name
return
$i1/name'
PASSING STATUS AS "st")
FROM statusXML;
```

Example 8-9 shows the results of this query.

Example 8-9 Results of sample XQuery using all FLWOR keywords

```
<name>Jason C</name>
<name>James B</name>
```

Note that only the names for Jason C and James B are returned, because they are the only names with a status of "blacklist". They show up on different rows of the result because they reside in different documents within the database.

Be careful when using the **let** keyword in combination with the **for** keyword. Example 8-8 uses the **for** keyword to set up a loop through all the statusItem elements. This, it uses the **let** keyword to assign the variable `$sts` to each single status element within the loop that is set up with the **for** keyword. This variable definition refers to `$i1` instead of `$st/status/statusItem`. As a result, you have to read through the only documents once.

If you had used the **let** keyword to assign the variable `$sts` to each single status element by referring to the element directly, instead of through the **for** loop, you code the **let** expression as follows:

```
let $sts := $st/status/statusItem/status
```

This code produces an incorrect result because the **where** keyword operates on the `$sts` variable, and the `$sts` variable is based on the entire element name. It does not refer to the **for** loop. Therefore, the criteria specified in the **where** clause is not applied to the data returned by the **for** loop, and you get data that you do not expect to be returned.

Use of FLWOR expression to join two tables

The next example takes the **FLWOR** expression capability one step further. Using the two tables created earlier, purchaseOrdersXML and statusXML, this example writes an XQuery **FLWOR** expression to find those purchase orders that were ordered by a “premier” customer. A join of the two tables is required to produce this result. Example 8-10 shows the **FLWOR** expression.

Example 8-10 XQuery FLWOR expression to express a join

```
SELECT XMLSERIALIZE(XMLQUERY(
'declare namespace ipo="http://www.example.com/IPO";
 for $i in $po/ipo:purchaseOrder,
      $j in $status/status/statusItem
 where $j/name=$i/billTo/name and $j/status="premier"
 return $i'
PASSING T1.PO as "po", T2.status as "status")
AS CLOB VERSION '1.0'  EXCLUDING XMLDECLARATION)
FROM purchaseOrdersXML T1, statusXML T2
WHERE XMLEXISTS('declare namespace ipo="http://www.example.com/IPO";
    $status/status/statusItem[status="premier"
and
 name =$po/ipo:purchaseOrder/billTo/name]'
PASSING T1.PO as "po", T2.status as "status");
```

Note that three of the **FLWOR** keywords are present in this example. The **for** keyword is used to allow us to loop through all of the purchase orders (using variable `$i`) and through all of the customer statuses (using variable `$j`). The **where** keyword is used to join the two tables on name and to only show rows with a status of “premier.” The **return** keyword is used to return the purchase order data specified in the variable `$i`.

Example 8-11 shows the results.

Example 8-11 Results of XQuery FLWOR expression to express a join

```
<ipo:purchaseOrder
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ipo="http://www.example.com/IPO"
orderDate="1999-12-01">
<shipTo exportCode="1" xsi:type="ipo:UKAddress">
  <name>Helen Zoe</name>
  <street>55 Eden Street</street>
  <city>San Jose</city>
  <state>CA</state>
  <postcode>CB1 1JR</postcode>
</shipTo>
<shipTo exportCode="1" xsi:type="ipo:UKAddress">
  <name>Joe Lee</name>
  <street>66 University Avenue</street>
  <city>Palo Alto</city>
  <state>CA</state>
  <postcode>CB1 1JR</postcode>
</shipTo>
<billTo xsi:type="ipo:USAddress">
  <name>Robert Smith</name>
  <street>8 Oak Avenue</street>
  <city>Old Town</city>
  <state>PA</state>
```

```
  <zip>95819</zip>
</billTo>
<items>
  <item partNum="833-AA">
    <productName>Lapis necklace</productName>
    <quantity>1</quantity>
    <USPrice>99.95</USPrice>
    <ipo:comment>Want this for the holidays!</ipo:comment>
    <shipDate>1999-12-05</shipDate>
  </item>
    <item partNum="945-ZG">
    <productName>Sapphire Bracelet</productName>
    <quantity>2</quantity>
    <USPrice>178.99</USPrice>
    <shipDate>2000-01-03</shipDate>
  </item>
</items>
</ipo:purchaseOrder>
DSNE610I NUMBER OF ROWS DISPLAYED IS 1
```

If you look at the data that was inserted into the **statusXML** table, Robert Smith is the only premier customer. Therefore, the results of the query show only the purchase order that was billed to Robert Smith.

Note that an XQuery “join” is different from an SQL join. An XQuery join is used to join nodes between two documents. It is not intended to join multiple rows of a DB2 table. The **XMLEXISTS** and **XMLTABLE** functions can be used for that purpose. Use an XQuery join as an extension to the SQL syntax, not in place of the SQL join. That is why the example shows an XQuery join on the name nodes as an **XMLEXISTS** function. The XQuery join produces the intended results, but it does not eliminate those rows that do not qualify. The result is returned as empty rows. The **XMLEXISTS** function eliminates the non-qualifying rows.

It is best to ensure that an XML index exists for this type of query. An XML index can be used only with the **XMLEXISTS** and **XMLTABLE** functions; it cannot be used for the **XMLQUERY** function because of the non-qualifying empty rows.

8.1.2 XQuery constructors

Constructors create XML structures within a query. DB2 9 supported only SQL/XML constructors, such as **XMLELEMENT** and **XMLATTRIBUTES**. The XQuery language provides the following kinds of constructors:

- Direct constructors
- Computed constructors

DB2 11 supports both direct element constructors and document node constructors. *Direct constructors* use an XML-like notation to create XML structures within a query. XQuery provides direct constructors for creating element nodes (which might include attribute nodes, text nodes, and nested element nodes), processing instruction nodes, and comment nodes. For example, the constructor shown in Example 8-12 creates a book element that contains an attribute and some nested elements.

Example 8-12 Example of an XQuery constructor

```
<book isbn="isbn-0060229357">
  <title>Harold and the Purple Crayon</title>
```

```
  <author>
    <first>Crockett</first>
    <last>Johnson</last>
  </author>
</book>
```

A *document node constructor* constructs the root node of an XML document. It is equivalent to the **XMLDOCUMENT** function but can be used in an XQuery expression.

Enclosed expressions are used in constructors to provide computed values for element and attribute content. These expressions are evaluated and replaced by their value when the constructor is processed. Enclosed expressions are enclosed in curly braces ({}) to distinguish them from literal text. Enclosed expressions can be used in the following constructors to provide computed values:

- Direct element constructors:
  - An attribute value in the start tag of a direct element constructor can include an enclosed expression.
  - The content of a direct element constructor can include an enclosed expression that computes both the content and the attributes of the constructed node.
- Document node constructor:
  - An enclosed expression can be used to generate the content of the node.

The **FLWOR** example shown in Example 8-5 on page 157 includes two examples of using braces to build an XQuery constructor based on a computed element or attribute value:

```
<to> {$i/shipTo/name/text()} </to>
<items> {$i/items/item} </items>
```

These two cases constructed the elements <to> and <items>. Example 8-6 on page 157 shows the results from this query, which includes the following lines:

```
<to>Helen ZoeJoe Lee</to>
  <items>
```

These two lines represent the XQuery elements constructed in the example.

8.1.3 Conditional expressions

Conditional expressions use the keywords **if**, **then**, and **else** to evaluate one of two expressions based on whether the value of a test expression is true or false. DB2 11 supports conditional expressions within an XQuery expression.

This example shows how conditional expressions work in XQuery. Example 8-13 shows a query that produces a shipping cost for items in purchase orders. If the price of the item is less than US $100, the shipping cost is US $5.00. Otherwise, the shipping cost is US $10.00.

Example 8-13 Sample XQuery using conditional expression

```
SELECT XMLQUERY(
 'declare namespace ipo="http://www.example.com/IPO";
  for $i in $po/ipo:purchaseOrder/items/item
  return (
    if (xs:decimal($i/USPrice) < 100)
      then fn:concat($i/productName, " : shipping=US$", 5)
    else fn:concat($i/productName, " : shipping=US$", 10))'
```

```
  PASSING po as "po")
FROM purchaseordersXML;
```

Example 8-14 shows the results of the query.

Example 8-14 Results of sample XQuery using conditional expression

```
Lapis necklace : shipping=US$5 Sapphire Bracelet : shipping=US$10
Lapis bracelet : shipping=US$5 Sapphire Earring : shipping=US$10
```

8.1.4 Built-in functions

DB2 9 provided some built-in functions that you could use with your XPath queries. With the implementation of XQuery, you can now take advantage of a `fn:avg` built-in function to return the average of the values in a sequence.

This example uses the `fn:avg` function to show the average US price for items in a purchase order. Example 8-15 shows a sample XQuery statement to calculate this average.

Example 8-15 Sample XQuery using fn:avg built-in function

```
SELECT XMLQUERY(
 'declare namespace ipo="http://www.example.com/IPO";
  for $i in $po/ipo:purchaseOrder/items
  return (fn:avg($i/item/USPrice))'
  PASSING po as "po")
FROM purchaseordersXML;
```

Because there are two purchase orders in the purchaseordersXML table, with two items in each purchase order, when the query is run, it produces two rows with one value in each row.

```
139.47
138.97
```

The US prices for the two items in the first purchase order are $99.95 and $178.99, and the US prices for the two items in the second purchase order are $89.95 and $187.99. If you do the math, you can see that the function is producing the average value for each purchase order.

8.1.5 XQuery prolog

The prolog is series of declarations that define the processing environment for a query. Each declaration in the prolog is followed by a semicolon (;). The prolog is an optional part of the query; a valid query can consist of a query body with no prolog.

The prolog can contain the following different types of declarations:

- Boundary space declaration
- Copy namespaces declaration
- Namespace declarations
- Default namespace declaration

The namespace declarations and default namespace declaration are available in the XPath query language. The boundary space declaration and copy namespaces declaration are added with the XQuery support.

Boundary space declaration

The boundary space declaration controls whether whitespace between the tags is preserved. Example 8-16 shows the syntax for the declaration.

Example 8-16 Syntax for boundary-space declaration

```
>>----declare--boundary-space--+--strip-----+--;----><
                               '--preserve--'
```

The boundary-space declaration can have the following values:

strip	Specifies that boundary whitespace is removed when elements are constructed.
preserve	Specifies that boundary whitespace is preserved when elements are constructed.

The default behavior is to strip the boundary whitespace.

Copy namespaces declaration

XML namespaces are used for providing uniquely named elements and attributes in an XML document. They are defined in a W3C recommendation. An XML instance can contain element or attribute names from more than one XML vocabulary, which is a collection of element and attribute names with definitions of their meanings and their structural relationships and constraints.

Because there can potentially be some ambiguity between identically named elements or attributes, each vocabulary can be given a namespace, and the namespace can be referenced in the XML expression to resolve any ambiguity. A namespace name is a uniform resource identifier (URI).

XML lets you create your own vocabulary or tags that are meaningful. After you have created the vocabulary using XSD, you can associate it with an XML instance using an xsi namespace. Consider the following example of a namespace in XML:

```
<item xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" partNum="833-AA">
```

Although a namespace is associated with an XML document, you might or might not want to display the namespace information when you display data from the document.

The copy namespaces declaration controls how the namespace bindings are assigned when an existing element node is copied by an element constructor or document node constructor. Example 8-17 shows the syntax for a copy namespaces declaration.

Example 8-17 Syntax for copy namespaces declaration

```
>>----declare--copy-namespaces--+--preserve-----+-,--inherit--;----><
                                '--no-preserve--'
```

The copy namespaces declaration can have the following values:

preserve	Specifies that all in-scope namespaces of the original element are retained in the new copy.
no-preserve	Specifies that unreferenced in-scope namespaces of the original element are not retained in the new copy.

The **inherit** option specifies that the copied node inherits in-scope namespaces from the constructed node.

Use of boundary space and copy namespaces declarations

These two new declarations introduced with the XQuery support can make a considerable difference in how the result of your queries are displayed, as shown in the following examples.

The query in Example 8-18 specifies that you want to reserve and whitespace between the tags and that all in-scope namespaces of the original element are retained in the new copy that is created by the XQuery expression. This query shows that some whitespace in the definition of the <product> element.

Example 8-18 Declaration example preserving boundary space and copy namespaces

```
SELECT XMLSERIALIZE(
       XMLQUERY (
'declare namespace ipo="http://www.example.com/IPO";
declare boundary-space preserve;
declare namespace ipo2="http://www.example.com/IPO2";
declare copy-namespaces preserve, inherit;
<order_list> {
for $i in $po/ipo:purchaseOrder/items/item
return
<product> {$i/productName, $i/USPrice} </product>}
</order_list>'
PASSING po as "po")
AS CLOB
VERSION '1.0'
EXCLUDING XMLDECLARATION)
FROM purchaseordersXML
;
```

Example 8-19 shows the results of this query. Note the space between the elements <order_list> and <product> and between <product> and <productName>.

Example 8-19 Results of query to preserve boundary space and copy namespaces

```
<order_list> <product> <productName
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ipo="http://www.example.com/IPO">Lapis necklace</productName><USPrice
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ipo="http://www.example.com/IPO">99.95</USPrice> </product><product>
<productName xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ipo="http://www.example.com/IPO">Sapphire Bracelet</productName><USPrice
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ipo="http://www.example.com/IPO">178.99</USPrice> </product>
</order_list>
<order_list> <product> <productName
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ipo="http://www.example.com/IPO">Lapis bracelet</productName><USPrice
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ipo="http://www.example.com/IPO">89.95</USPrice> </product><product>
<productName xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ipo="http://www.example.com/IPO">Sapphire Earring</productName><USPrice
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ipo="http://www.example.com/IPO">187.99</USPrice> </product>
</order_list>
DSNE610I NUMBER OF ROWS DISPLAYED IS 2
```

The next example runs the same query with the declarations changed. It strips out any whitespace and does not preserve any copied namespaces. Example 8-20 shows this query.

Example 8-20 Declaration example not preserving boundary space and copy namespaces

```
SELECT XMLSERIALIZE(
       XMLQUERY (
'declare namespace ipo="http://www.example.com/IPO";
declare boundary-space strip;
declare namespace ipo2="http://www.example.com/IPO2";
declare copy-namespaces no-preserve, inherit;
<order_list> {
for $i in $po/ipo:purchaseOrder/items/item
return
<product> {$i/productName, $i/USPrice} </product>}
</order_list>'
PASSING po as "po")
AS CLOB
VERSION '1.0'
EXCLUDING XMLDECLARATION)
FROM purchaseordersXML
;
```

Example 8-21 shows the results.

Example 8-21 Results of query to not preserve boundary space and copy namespaces

```
<order_list><product><productName>Lapis
necklace</productName><USPrice>99.95</USPrice></product><product><productName>Sapp
hire Bracelet</productName><USPrice>178.99</USPrice></product></order_list>
<order_list><product><productName>Lapis
bracelet</productName><USPrice>89.95</USPrice></product><product><productName>Sapp
hire Earring</productName><USPrice>187.99</USPrice></product></order_list>
DSNE610I NUMBER OF ROWS DISPLAYED IS 2
```

Notice that there is no space between the element names and that the namespaces are not listed. As a result, the output is much smaller than in the case where the boundary spaces and the copy namespaces are preserved.

8.2 XML performance enhancements in DB2 10 and DB2 11

There are a number of performance enhancements to XML processing in DB2 11 that were also retrofitted to DB2 10. This section describes the following XML performance enhancements:

- Eliminate hotspots during XML insert
- Validate binary XML
- Avoid revalidation during LOAD
- Partial revalidation
- XMLTABLE performance improvements

8.2.1 Eliminate hotspots during XML insert

A performance issue with XML documents in DB2 10 might occur when a document is inserted. Because DB2 10 generates **DOCID** values in sequential order through an implicitly created sequence object, and the indexes on the XML **DOCID** and **NODEID** are non-partitioned indexes (NPIs), concurrent inserts of XML documents into the same table causes hotspots in these NPIs. As the number of threads increases, the time spent waiting for a page latch increases.

DB2 11 allows randomization of the DOCIDs, which eliminates the hotspots in both indexes. To enable randomization of the DOCIDs, set the **RANDOMIZE XML DOCID** system parameter on installation panel **DSNTIP8** to **YES**. This change affects only newly created XML columns; it has no impact on existing XML columns.

There is a slight regression in the performance of sequential prefetch due to the loss of index look-aside capabilities.

This feature is also retrofitted to DB2 10 through APARs PM31486, PM31487, and PM44216.

8.2.2 Validate binary XML

When DB2 10 validates binary XML, DB2 needs to serialize the binary XML into string XML, which defeats the purpose of using binary XML. DB2 11 now validates binary XML directly, without the need to serialize it.

DB2 11 can now perform an **INSERT** of an XML column with type modifier using binary XML. This results in a 30-40% CPU reduction compared to DB2 10, and a 15-18% CPU reduction versus the **INSERT** performance of string XML.

The **LOAD** utility also benefits from this enhancement, with a 41% CPU reduction compared to DB2 10, and an 18% improvement versus string XML.

This performance enhancement requires z/OS V1R13 with PTF UA63422 or z/OS V1R12 with PTF UA65591.

8.2.3 Avoid revalidation during LOAD

XML schema validation is the process of determining whether the structure, content, and data types of an XML document are valid according to an XML schema. In addition, XML schema validation strips ignorable whitespace from the input document.

Consider the case where you have two tables that contain the same XML column with the same schema. You want to unload the data from one table and load it into another table. Example 8-22 shows what the steps to perform this task might look like.

Example 8-22 Example of avoiding XML schema revalidation

```
1. CREATE TABLE T1
   (ID         INTEGER,
   XMLCOL      XML(XMLSCHEMA ID SYSXSR.PO1));

2. CREATE TABLE T2
   (ID         INTEGER,
   XMLCOL      XML(XMLSCHEMA ID SYSXSR.PO1));

3. Populate table T1
```

```
4. UNLOAD from T1 using binary XML

5. LOAD into T2 using binary XML
```

In the example case, DB2 11 provides savings during step 5 by avoiding XML validation because the data was already validated according to the same schema when it was initially loaded or inserted into table T1. Tests to avoid this revalidation have shown elapsed time savings between 29 and 43% and CPU savings between 61 and 76%.

8.2.4 Partial revalidation

The initial implementation of XML required that, when an XML document was modified, the whole document had to be revalidated. This could be costly if it is a large document.

DB2 11 provides the capability to revalidate only the changed part of a document. Consider the sequence of steps shown in Example 8-23.

Example 8-23 UPDATE of an XML document with partial revalidation

```
1. CREATE TABLE T1 (ORDER XML(XMLSCHEMA ID sysxsr.PO1));

2. INSERT INTO T1 (ORDER) VALUES (...);

3. UPDATE T1 SET ORDER =
      XMLModify('replace value of node //item[1]/shipDate with "2012-05-25"');
```

In this example, only <shipDate> is revalidated. The performance benefit depends on the size of the document and the size of the updated portion. For documents that are 10 MB in size, we have seen up to a 92% CPU reduction. For smaller documents that are only 10 KB in size, we have seen up to a 60% CPU reduction.

8.2.5 XMLTABLE performance improvements

The following **XMLTABLE** performance improvements are available in both DB2 10 and DB2 11:

- Remove unreferenced column definitions, which resulted in a 47% CPU reduction on an **XMLTABLE** expression with 30 columns
- Merge common column path expressions, which resulted in up to a 74% reduction on XPath storage consumption
- Storage reuse for output XML columns, which resolves an SQLCODE -904 issue and is available with APAR PM69176.

8.3 XQuery FLWOR expressions performance enhancements

Although XQuery support is also provided in DB2 10 with some maintenance applied, there are additional XQuery performance enhancements that are available only in DB2 that are related to **FLWOR** expression. These enhancements are described in this section.

Just as there is the concept of predicate push down when evaluating SQL predicates, there is the same concept with the XQuery language. In the case of XQuery, the predicates are pushed down from an XQuery expression to an XPath expression.

Consider the following **FLWOR** expression:

```
for $i in /order/items/item
where $i/price > 100
return $i/desc
```

DB2 pushes the **WHERE** predicate into XPath as follows:

```
for $i in /order/items/item[price > 100]
return $i/desc
```

Another performance improvement is that DB2 can translate a simple **FLWOR** expression to XPath. For example, the following **FLWOR** expression:

```
for $i in /order/shipTo return $i
```

Can be rewritten into an XPath expression as follows:

```
/order/shipTo
```

These enhancements can provide up to a 60% CPU reduction for qualified queries.

8.4 XMLTABLE performance enhancements in DB2 11

The following **XMLTABLE** enhancements are available only in DB2 11:

- Date/Time predicate pushdown
- Optimize index key range for varchar predicates
- Pushdown of column casting into XPath

8.4.1 Date/Time predicate pushdown

You can code some XML queries that require you to evaluate date and time predicates. An example of such a query using the **XMLTABLE** function is as follows:

```
SELECT X.* FROM T1,
     XMLTABLE('//item' passing T1.order
       COLUMN partNO varchar(8) path '@partNo',
           shipDate Date  path 'shipDate') X
  Where x.shipDate = '2011-05-23';
```

In this particular example, the date predicate is outside of the **XMLTABLE** function. Thus, even if there is an XML index, it is not used.

DB2 11 pushes the date predicate into XPath, enabling the use of an existing XML index. The translated query looks as follows:

```
SELECT * FROM T1,
    XMLTABLE('//item[shipDate=$d]' passing T1.order,
                   cast('2011-05-23' as Date) as "d"
    COLUMN partNO varchar(8) path '@partNo',
           shipDate Date  path 'shipDate') X;
```

Note: The date predicate is now within the **XMLTABLE** function. An index can now be used, which provides magnitudes of performance improvement.

8.4.2 Optimize index key range for varchar predicates

When you use the **XMLTABLE** function to return the results of an XQuery expression as a table, and you define a column in that **XMLTABLE** function as **VARCHAR**, prior to DB2 11 there was no way to specify an upper limit to the values that could qualify for the query. Consider the following example query:

```
SELECT X.* FROM T1,
     XMLTABLE('//item' passing T1.order
       COLUMN partNO varchar(6) path '@partNo',
           shipDate Date  path 'shipDate') X
  Where x.partNo = '872-AA';
```

Evaluation of this query was challenging because there was no upper end to the index values, because the results were **VARCHAR**.

DB2 11 improves the performance of this query by adding an extra upper bound predicate to reduce the index key range. The query is rewritten as follows:

```
SELECT * FROM T1,
    XMLTABLE('//item[left(@partNo,6)=$d]'
     passing T1.order, '872-AA' as "d"
    COLUMN partNO char(8) path '@partNo',
           shipDate Date  path 'shipDate') X
  where XMLEXISTS ('//item[@partNo>=$d and
                        @partNo <= $u]' passing
   T1.order, '872-AA' as "d", '872-AA'||X'FF' as "u");
```

Tests resulted in orders of magnitude improvement in CPU time and Getpage counts for affected queries.

8.4.3 Pushdown of column casting into XPath

When the **XMLTABLE** function's output columns are of any SQL type, DB2 used to evaluate the column XPath first and generate an intermediate XML node sequence, then cast the intermediate result to the SQL data type. DB2 11 now combines these two steps into one.

This enhancement provides up to a 40% CPU reduction and a 20% savings in storage for some **XMLTABLE** queries with many output columns.

9

Connectivity and administration routines

DB2 11 sees further improvements of universal drivers for accessing data on any local or remote server. In addition, DB2 11 for z/OS provides a number of enhancements to improve the availability and performance of distributed applications.

This chapter describes these topics:

- Client information enhancements
- Cancel thread and cancel SQL statement improvements
- Continuous block fetching
- Support for global variables
- Local stored procedure execution improvement
- Multi-threaded Java stored procedure environment
- ADMIN_COMMAND_MVS stored procedure
- Drivers, clients, and connectivity requirements

9.1 Client information enhancements

The DB2 client information fields are available on each distributed connection to a DB2 for z/OS database server. These fields are also available in other members of the DB2 family of databases. They enable a distributed application to provide additional information to DB2 that can be used to classify the workload within WLM, or to filter accounting reports, for example.

Up to DB2 10 for z/OS, client information lengths are arbitrary and are restrictive compared to other platforms of the DB2 product family.

Changes introduced in DB2 11 help to address the following needs:

- To be able to use longer fields to store business meaningful information in the client application registers
- To be able to override the current DRDA correlation token with a business value to correlate application work across the enterprise

At a glance, the improvements introduced by DB2 11 for z/OS in this area can be summarized as follows:

- Expansion of the length of some Client information fields
- Introduction of a new Client information field: Client Correlation Token
- Introduction of a new built-in session global variable: SYSIBM.CLIENT_IPADDR

These changes, how to take advantage of them, and their practical considerations are described in the following sections.

9.1.1 Expansion of the length of some Client information fields

DB2 11 enhances Client information in DB2 for z/OS by expanding the lengths of these fields:

- Client User ID
- Client Application Name
- Client Workstation Name
- Client Accounting Information

Table 9-1 list the Client information field length DB2 11 for z/OS in contrast with the lengths of these fields in DB2 10.

Table 9-1 Client information fields length changes with DB2 11 for z/OS

Field name	Max length in DB2 10	Max length in DB2 11
Client User ID	16 bytes	128 bytes
Client Application Name	32 bytes	255 bytes
Client Accounting Information	200 bytes	255 bytes
Client Workstation Name	18 bytes	255 bytes
Client Correlation Token	N/A	255 bytes

The longer client information length provide better granularity in the exploitation of this information, and compatibility with other databases member of the DB2 family of products. The longer client info strings are exploited in:

- WLM enclave classification
- DB2 supplied **SYSPROC.WLM_SET_CLIENT_INFO** stored procedure

- **RRSAF DSNRLI SET_CLIENT_ID** function
- Rollup accounting
- Profile Monitoring for remote threads and connections
- Resource Limit Facility (RLF)
- Client information special registers
- **DISPLAY THREAD** command output
- Various trace records, such as IFCIDs 172, 196, 313, and 316
- Various messages that present thread-info

The 9.1.4, “Using the client information fields” on page 178 provides examples and guidelines to exploit the longer client information fields.

9.1.2 Introduction the new client information field Client Correlation Token

DB2 11 introduces a the *client correlation token* client information field, which is an unique token that allows the application to correlate application work across the distributed transaction. The default value of the client correlation token is the DRDA correlation token, The data type is **VARCHAR(255)**.

Example 9-1 shows the results of the **-DIS THD(*) DETAIL** command. Note the **DSNV442I** message.

Example 9-1 -DIS THD(*) DETAIL

```
DSNV401I  -DB1A DISPLAY THREAD REPORT FOLLOWS -
DSNV402I  -DB1A ACTIVE THREADS -
NAME     ST A   REQ ID           AUTHID   PLAN     ASID TOKEN
SERVER   RA *     6 db2jcc_appli DB2R1    DISTSERV 0131   395
 V437-WORKSTATION=9.55.137.33
      USERID=DB2R1
      APPLICATION NAME=db2jcc_application
 V441-ACCOUNTING=JCC0366O9.55.137.33
 V442-CRTKN=::9.55.137.33.52646.CBBF468B1B73
 V482-WLM-INFO=DDFBAT:1:3:1
 V445-G9378921.CDA6.CBBF468B1B73=395 ACCESSING DATA FOR
  (  1)::9.55.137.33
 V447--INDEX SESSID           A ST TIME
 V448--(  1) 38420:52646      W R2 1321311013868
```

The DB2 **DSNV442I** message contains detail output from the **DISPLAY THREAD** command. It provides details about the correlation token after the **V442-CRTKN** keyword.

The CURRENT CLIENT_CORR_TOKEN special register

The **CURRENT CLIENT_CORR_TOKEN** special register contains the value of the client correlation token from the client information that is specified for the connection. The data type is **VARCHAR(255)**.

Example 9-2 shows a simple query example that can be used to retrieve the value of this special register using SQL.

Example 9-2 Retrieve the CURRENT CLIENT_CORR_TOKEN value using SQL

```
SELECT CURRENT CLIENT_CORR_TOKEN
  FROM SYSIBM.SYSDUMMY1;
```

Example 9-3 shows the output of the execution of this query in this environment.

Example 9-3 Value of CURRENT CLIENT_CORR_TOKEN

```
::9.55.137.33.54132.CBBF794F9C68
```

The correlation token is made up of three components separated by periods. Its structure is shown in Example 9-4.

Example 9-4 Client correlation token components

```
ip-address.port-address.unique-id
```

The the correlation token includes the following components:

`ip-address`	The IP address of the originating requester, which is 3 to 39 characters in length
`port-address`	The port address, which is 1 to 8 characters in length
`unique-id`	An unique logical unit of work identifier, which is 12 characters in length

You can change the value of this special register to a more meaningful value, for example by using one of the following application programming interfaces:

- **`SQLE_CLIENT_INFO_PROGRAMID`** (`sqleseti`)
- `java.sql.Connection.setClientInfo` (JDBC)
- The **`RRS DSNRLI SIGNON`**, **`AUTH SIGNON`**, **`CONTEXT SIGNON`**, or **`SET_CLIENT_ID`** function

Example 9-5 shows how to extract and how to override this special register in a Java program.

Example 9-5 Java and CURRENT CLIENT_CORR_TOKEN

```
import com.ibm.db2.jcc.DB2Connection;
import java.sql.*;

public class DB211NewDriverCorrToken {

    public static Connection con = null;
    public static CallableStatement cstmt;
    public static ResultSet results;
    public static boolean debug = true;

    public static void main(String args[]) throws Exception {
        Statement stmt; ResultSet rs; String corr_token;
         String url = "jdbc:db2://redbook8:38420/DB1A" +
                 ":user=db2r1;password=******;";
        try {
            Class.forName("com.ibm.db2.jcc.DB2Driver");
        } catch (java.lang.ClassNotFoundException e) {
            System.err.print("ClassNotFoundException: ");
            System.err.println(e.getMessage());
        }
        con = DriverManager.getConnection(url);
        DB2Connection db2con = (DB2Connection) con;
        con.setAutoCommit(false);
        // Extract original Correlation Token
        stmt = con.createStatement();
        rs = stmt.executeQuery("SELECT CURRENT CLIENT_CORR_TOKEN FROM SYSIBM.SYSDUMMY1;");
        rs.next(); corr_token = rs.getString(1);
```

```
        System.out.println("CORR TOKEN = " + corr_token);
        rs.close(); stmt.close();
        // Override original Correlation Token
        db2con.setClientInfo("ClientCorrelationToken","BRXLS_APPCRIS");
        stmt = con.createStatement();
        rs = stmt.executeQuery("SELECT CURRENT CLIENT_CORR_TOKEN FROM SYSIBM.SYSDUMMY1;");
        rs.next();
        corr_token = rs.getString(1);
        System.out.println("CORR TOKEN = " + corr_token);
        rs.close(); stmt.close();
    }
}
```

This program connect to DB2, selects the **CURRENT CLIENT_CORR_TOKEN** value in a string variable, overrides this information, and prints the new value. Example 9-6 shows the execution results.

Example 9-6 Java program output, overriding the correlation token

```
CORR TOKEN = ::9.55.137.134.62461.CBC82DBC97F5
CORR TOKEN = BRXLS_APPCRIS
```

Example 9-7 shows how the **-DIS THD(*) DETAIL** command provides the correlation token information in **DSNV442I** message.

Example 9-7 -DIS THD(*) DETAIL and the client correlation token value

```
NAME     ST A   REQ ID           AUTHID   PLAN     ASID TOKEN
SERVER   RA *     8 db2jcc_appli DB2R1    DISTSERV 0133   139
 V437-WORKSTATION=192.168.150.1
      USERID=db2r1
      APPLICATION NAME=db2jcc_application
 V441-ACCOUNTING=JCC04170192.168.150.1
 V442-CRTKN=BRXLS_APPCRIS
 V482-WLM-INFO=DDFBAT:1:3:1
 V445-G9378986.F843.CBC840370D33=139 ACCESSING DATA FOR
  (  1)::9.55.137.134
 V447--INDEX SESSID           A ST TIME
 V448--(  1) 38420:63555      W R2 1322014211550
```

In DB2 11, you can override the client correlation token. This option allows you to use this register for application purposes. You can, for example, feedback a program with different data depending on the value of this register.

As an example, consider the simple table created and populated with the information shown in Example 9-8.

Example 9-8 DDL and Insert for example table

```
CREATE TABLE COD_TAB (BUS_COD INTEGER, CORR_ID VARCHAR(255));
INSERT INTO COD_TAB  (BUS_COD, CORR_ID) VALUES (1000,'BRXLS_APPCRIS');
INSERT INTO COD_TAB  (BUS_COD, CORR_ID) VALUES (2000,'ROME_APPCRIS');
INSERT INTO COD_TAB  (BUS_COD, CORR_ID) VALUES (3000,'MILANO_APPCRIS');
INSERT INTO COD_TAB  (BUS_COD, CORR_ID) VALUES (4000,'PARIS_APPCRIS');
INSERT INTO COD_TAB  (BUS_COD, CORR_ID) VALUES (5000,'SJOSE_APPCRIS');
INSERT INTO COD_TAB  (BUS_COD, CORR_ID) VALUES (5000,'MADRID_APPCRIS');
```

Example 9-9 shows the contents of the table after the execution of the SQL.

Example 9-9 Contents of example table

```
---------+---------+---------+---------+---------+---------+---------+
SELECT * FROM COD_TAB;
---------+---------+---------+---------+---------+---------+---------+
    BUS_COD  CORR_ID
---------+---------+---------+---------+---------+---------+---------+
       1000  BRXLS_APPCRIS
       2000  ROME_APPCRIS
       3000  MILANO_APPCRIS
       4000  PARIS_APPCRIS
       5000  SJOSE_APPCRIS
       5000  MADRID_APPCRIS
DSNE610I NUMBER OF ROWS DISPLAYED IS 6
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
---------+---------+---------+---------+---------+---------+---------+
```

In this table, the **BUS_COD** column represents any kind of business information, such as a business code. The **CORR_ID** column contains the information to be matched with the value of the correlation token. Example 9-10 shows a SQL example where the **BUS_COD** value is returned based on the information in the special register **CURRENT CLIENT_CORR_TOKEN**.

Example 9-10 Using the CURRENT CLIENT_CORR_TOKEN in SQL

```
SELECT BUS_COD
    FROM COD_TAB
    WHERE CORR_ID = CURRENT CLIENT_CORR_TOKEN
```

Example 9-11 is a Java implementation of this technique. When invoked, this program receives a certain value as input parameter. This value is used in the program to set the client correlation token information. The value of the **CURRENT CLIENT_CORR_TOKEN** in the exploited in the embedded SQL.

Example 9-11 Java and SQL exploiting CURRENT CLIENT_CORR_TOKEN

```
import com.ibm.db2.jcc.DB2Connection;
import java.sql.*;

public class DB211NewDriverCorrToken {

    public static Connection con = null;
    public static CallableStatement cstmt;
    public static ResultSet results;
    public static boolean debug = true;

    public static void main(String args[]) throws Exception {
        System.out.println("Input correlation token = " + args[0]);
        Statement stmt;
        ResultSet rs;
        String corr_token;
        String url = "jdbc:db2://redbook8:3842O/DB1A"
                + ":user=db2r1;password=******;";
        try {
            Class.forName("com.ibm.db2.jcc.DB2Driver");
        } catch (java.lang.ClassNotFoundException e) {
            System.err.print("ClassNotFoundException: ");
            System.err.println(e.getMessage());
        }
```

```
        con = DriverManager.getConnection(url);
        DB2Connection db2con = (DB2Connection) con;
        con.setAutoCommit(false);
        // Override original Correlation Token
        db2con.setClientInfo("ClientCorrelationToken", args[0]);
        stmt = con.createStatement();
        rs = stmt.executeQuery("SELECT CURRENT CLIENT_CORR_TOKEN FROM SYSIBM.SYSDUMMY1;");
        rs.next();
        corr_token = rs.getString(1);
        System.out.println("CORR TOKEN = " + corr_token);
        rs.close();
        stmt.close();
        // Use new Correlation Token value
        db2con.setClientInfo("ClientCorrelationToken", args[0]);
        stmt = con.createStatement();
        rs = stmt.executeQuery("SELECT BUS_COD FROM COD_TAB\n"
                + "    WHERE CORR_ID = CURRENT CLIENT_CORR_TOKEN;");
        rs.next();
        corr_token = rs.getString(1);
        System.out.println("BUS CODE = " + corr_token);
        rs.close();
        stmt.close();

    }
}
```

These tests executed the program by passing the **BRXLS_APPCRIS** value as the parameter. Example 9-12 shows the execution output when running this program.

Example 9-12 Java application execution output

```
Input correlation token = BRXLS_APPCRIS
CORR TOKEN = BRXLS_APPCRIS
BUS CODE = 1000
```

This client information field cannot be used for classifying DDF work within WLM.

9.1.3 Introduction of a new built-in session global variable

DB2 11 provides a new built-in session global variable named **SYSIBM.CLIENT_IPADDR**. This global variable contains the value of the client IP address for the connection, as follows:

- For remote client connections, the value is the host IP address the application that is used to establish the connection.
- For local host applications, the value is NULL.
- For remote host applications, the value is the IP address that is associated with the DB2 subsystem used to establish the connection.

The data type is CHAR(39). SYSIBM.CLIENT_IPADDR displays the TCP/IP IPv6 colon hexadecimal format. For example:

```
IPv6 : 1111:2222:3333:4444:5555:6666:7777:8888
IPv4 : (9.30.115.135) mapped as IPv6 : 0000:0000:0000:FFFF:9:30:115:135
```

DB2 obtains TCP/IP IPv6 address value from network, the client does not provide it or set it. DB2 sets this value only if client is using TCP/IP or SSL protocol.

Note: The value of **SYSIBM.CLIENT_IPADDR** is NULL if the client did not connect to TCP/IP or SSL protocol.

SYSIBM.CLIENT_IPADDR can be used for classifying DDF work with WLM using the Client IP Address (CIP) WLM workload qualifier. For DDF workload type, the CIP is the source client IPv6 address associated with the DDF server thread. The maximum length is 39 bytes.

Example 9-13 shows how you can query the **DB2 SYSIBM.SYSVARIABLES** table to get details about **CLIENT_IPADDR**.

Example 9-13 Query on SYSIBM.SYSVARIABLES

```
---------+---------+---------+---------+---------+---------+---------+
SELECT
CAST(SCHEMA AS CHAR(10)) AS SCHEMA,
CAST(NAME AS CHAR(20)) AS NAME,
CAST(TYPENAME AS CHAR(10)) AS TYPE , LENGTH , DEFAULT
FROM SYSIBM.SYSVARIABLES
WHERE NAME = 'CLIENT_IPADDR' WITH UR;
---------+---------+---------+---------+---------+---------+---------+
SCHEMA      NAME                 TYPE             LENGTH  DEFAULT
---------+---------+---------+---------+---------+---------+---------+
SYSIBM      CLIENT_IPADDR        CHAR                 39  N
DSNE610I NUMBER OF ROWS DISPLAYED IS 1
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
---------+---------+---------+---------+---------+---------+---------+
```

9.1.4 Using the client information fields

This section describes how the changes in client information fields are used.

DB2 messages

The following DB2 messages are updated in DB2 11 to take advantage of longer field lengths:

DSNB260I A long-running reader has reached the maximum permitted time without issuing either a **COMMIT** or **ROLLBACK** statement.

DSNI031I Lock escalation has occurred for the specified object.

DSNR048I This message is produced periodically during the backout process of an in-abort unit of recovery.

DSNR035I This message indicates that during checkpoint processing, DB2 encountered an uncommitted unit of recovery (UR) that has an inflight or indoubt status.

DSNJ031I A UR has reached the threshold number of log records that were written without a commit or rollback operation.

DSNT318I A plan cannot get an internal resource lock manager (IRLM) lock because the resource is held by a P-lock in the data sharing group, and the maximum amount of time to wait for the locked resource was exceeded.

DSNT375I A plan has been denied an IRLM lock because of a detected deadlock.

DSNT376I A plan has been denied an IRLM lock because of a timeout.

DSNT377I A plan cannot gain an IRLM lock because a required resource is currently undergoing recovery.

DSNT378I A plan cannot get an IRLM lock because the resource is held by a retained lock on behalf of another member in the data sharing group, and the amount of time to wait for the locked resource was exceeded.

Accounting with OMPE[1]

The **START TRACE** command starts DB2 traces. You can limit the collection of trace data to particular applications or users and to limit the data collected to particular traces and trace events. You can use trace filters to exclude the collection of trace data from specific contexts and to exclude the collection of specific traces and trace events. The following types of trace filters are available:

- **USERID** or **XUSERID**

  Specifies the user ID. Use **USERID** to constrain the trace to the specified user IDs or **XUSERID** to exclude the specified user IDs. You can specify multiple values and wildcard values. The value can be up to 16 characters.

- **APPNAME** or **XAPPNAME**

  Specifies the application name. Use **APPNAME** to constrain the trace to the specified applications or **XAPPNAME** to exclude the specified applications. You can specify multiple values and wildcard values.

- **WRKSTN** or **XWRKSTN**

  Specifies the workstation name. Use **WRKSTN** to constrain the trace to the specified workstations or **XWRKSTN** to exclude the specified workstations. You can specify multiple values and wildcard values.

Note: The **START TRACE** command filtering parameters, **USERID** and **XUSERID**, **APPNAME** and **XAPPNAME**, and **WRKSTN** and **XWRKSTN** are not be enhanced to support the new longer lengths for the client information fields.

Example 9-14 shows an OMPE JCL example as used in this environment for the creation of the reports exposed in this section.

Example 9-14 OMPE command JCL example

```
//PE       EXEC PGM=FPECMAIN
//STEPLIB  DD DISP=SHR,DSN=OMPE.V520.D130306.V11DRP5.TKANMOD
//INPUTDD  DD DISP=SHR,DSN=SMFDATA.DB2RECS.G4829V00
//JOBSUMDD DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//ACRPTDD  DD SYSOUT=*
//UTTRCDD1 DD SYSOUT=*
//SYSIN    DD *
  ACCOUNTING
      REPORT
        LAYOUT(LONG)
        INCLUDE(SUBSYSTEM(DB1A))
      TRACE
        LAYOUT(LONG)
        INCLUDE(SUBSYSTEM(DB1A))
EXEC
/*
```

[1] IBM Tivoli® OMEGAMON® XE for DB2 Performance Expert on z/OS

Example 9-15 shows part of a OMPE Accounting Trace Long report. The report was created using records produced by an application using a supported version of JDBC driver. Being the driver 10.1 fix pack 0 if it is supported by DB2 11. However, this driver cannot use the longer client information fields.

Example 9-15 OMPE Accounting Trace Long - JDBC driver 10.1 fix pack 0

```
1   LOCATION: DB1A                    OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V5R2M0)                    PAGE: 1-159
       GROUP: N/P                                ACCOUNTING TRACE - LONG                    REQUESTED FROM: NOT SPECIFIED
      MEMBER: N/P                                                                                          TO: NOT SPECIFIED
   SUBSYSTEM: DB1A                                                                                    ACTUAL FROM: 08/01/13 20:56:51.31
 DB2 VERSION: V11

 ---- IDENTIFICATION -------------------------------------------------------------------------------------------------------------
 ACCT TSTAMP: 08/01/13 22:38:35.36    PLANNAME: DB211Jav        WLM SCL: DDFBAT          CICS NET: N/A
 BEGIN TIME : 08/01/13 22:38:13.08    PROD TYP: JDBC DRIVER                              CICS LUN: N/A
 END TIME   : 08/01/13 22:38:35.36    PROD VER: V4 R13M0        LUW NET: G9378921        CICS INS: N/A
 REQUESTER  : ::9.55.137.33           CORRNAME: db2jcc_a        LUW LUN: D8DE
 MAINPACK   : DB211Jav                CORRNMBR: ppli            LUW INS: CBBFAC8093E3    ENDUSER : ClientUser_01234
 PRIMAUTH   : DB2R1                   CONNTYPE: DRDA            LUW SEQ:            2    TRANSACT: DB211JavaNewDriver_0123456789012
 ORIGAUTH   : DB2R1                   CONNECT : SERVER                                   WSNAME  : WorkstationName_01
```

Example 9-16 show a report that belongs to the same application but running with the DB2 10.5 fix pack 2 of the driver. In this case, the application is able to exploit the longer fields. In these reports, note the **ENDUSER**, **TRANSACT**, and **WSNAME** fields differences.

Example 9-16 OMPE Accounting Trace Long - JDBC driver 10.5 fix pack 2

```
1   LOCATION: DB1A                    OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V5R2M0)                    PAGE: 1-148
       GROUP: N/P                                ACCOUNTING TRACE - LONG                    REQUESTED FROM: NOT SPECIFIED
      MEMBER: N/P                                                                                          TO: NOT SPECIFIED
   SUBSYSTEM: DB1A                                                                                    ACTUAL FROM: 08/01/13 20:56:51.31
 DB2 VERSION: V11

 ---- IDENTIFICATION -------------------------------------------------------------------------------------------------------------
 ACCT TSTAMP: 08/01/13 22:36:37.38    PLANNAME: DB211Jav        WLM SCL: DDFBAT          CICS NET: N/A
 BEGIN TIME : 08/01/13 22:36:15.07    PROD TYP: JDBC DRIVER                              CICS LUN: N/A
 END TIME   : 08/01/13 22:36:37.38    PROD VER: V4 R17M0        LUW NET: G9378921        CICS INS: N/A
 REQUESTER  : ::9.55.137.33           CORRNAME: db2jcc_a        LUW LUN: D8D9
 MAINPACK   : DB211Jav                CORRNMBR: ppli            LUW INS: CBBFAC1O0F8O    ENDUSER : ClientUser_012#1
 PRIMAUTH   : DB2R1                   CONNTYPE: DRDA            LUW SEQ:            2    TRANSACT: DB211JavaNewDriver_01234567890#1
 ORIGAUTH   : DB2R1                   CONNECT : SERVER                                   WSNAME  : WorkstationName_#1
```

Example 9-17 shows the JDBC driver correlation section as produced with the SMF records created by the application using the old Java driver.

Example 9-17 JDBC driver correlation: Old Java driver

```
---- INITIAL DB2 COMMON SERVER OR UNIVERSAL JDBC DRIVER CORRELATION ------------------------------------------------------------
 PRODUCT ID     : JDBC DRIVER
 PRODUCT VERSION: V4 R13M0
 CLIENT PLATFORM: WorkstationName_01
 CLIENT APPLNAME: DB211JavaNewDriver_0
 CLIENT AUTHID  : ClientUs
 DDCS ACC.SUFFIX: ClientAccountingInformation_012345678901234567890123456789012345678901234567890123456789012345678901234567890123
                  678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123
```

Example 9-18 shows the same section of the report for the records created using the new Java driver. There is no difference in both cases.

Example 9-18 JDBC driver correlation: New Java driver

```
---- INITIAL DB2 COMMON SERVER OR UNIVERSAL JDBC DRIVER CORRELATION ------------------------------------------------------------
 PRODUCT ID     : JDBC DRIVER
 PRODUCT VERSION: V4 R17M0
 CLIENT PLATFORM: WorkstationName_01
 CLIENT APPLNAME: DB211JavaNewDriver_0
 CLIENT AUTHID  : ClientUs
 DDCS ACC.SUFFIX: ClientAccountingInformation_012345678901234567890123456789012345678901234567890123456789012345678901234567890123
                  678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123
```

However, using the new Java driver results in an expended identification section, as shown in Example 9-19. This example shows how the longer client field information fields are truncated in the OMPE report. This section is not available if the application was executed with the old Java driver.

Example 9-19 OMPE Accounting Trace report, identification section

```
---- IDENTIFICATION --------------------------------------------------------------------------------------------------------
 ACCT TSTAMP: 08/01/13 22:36:37.38    PLANNAME: DB211Jav        WLM SCL: DDFBAT          CICS NET: N/A
 BEGIN TIME : 08/01/13 22:36:15.07    PROD TYP: JDBC DRIVER                              CICS LUN: N/A
 END TIME   : 08/01/13 22:36:37.38    PROD VER: V4 R17M0        LUW NET: G9378921        CICS INS: N/A
 REQUESTER  : ::9.55.137.33           CORRNAME: db2jcc_a        LUW LUN: D8D9
 MAINPACK   : DB211Jav                CORRNMBR: ppli            LUW INS: CBBFAC100F80    ENDUSER : ClientUser_012#1
 PRIMAUTH   : DB2R1                   CONNTYPE: DRDA            LUW SEQ:            2    TRANSACT: DB211JavaNewDriver_01234567890#1
 ORIGAUTH   : DB2R1                   CONNECT : SERVER                                   WSNAME  : WorkstationName_#1

    ------------------------------------------------------------------------------------------------------------------------
    |TRUNCATED VALUE                   FULL VALUE                                                                          |
    |ClientUser_012#1                  ClientUser_012345678901234567890123456789012345678901234567890123456789012345678901|
    |                                  23456789012345678901234567890123456                                                 |
    |DB211JavaNewDriver_01234567890#1 DB211JavaNewDriver_012345678901234567890123456789012345678901234567890123456789012345678901234567890123|
    |                                  45678901234567890123456789012345678901234567890123456789012345678901234567890123456|
    |                                  78901234567890123456789012345678901234567890                                        |
    |WorkstationName_#1                WorkstationName_012345678901234567890123456789012345678901234567890123456789012345678901234567890123456|
    |                                  78901234567890123456789012345678901234567890123456789012345678901234567890123456789|
    |                                  01234567890123456789012345678901234567890                                           |
    ------------------------------------------------------------------------------------------------------------------------
```

Application and SQL release incompatibility considerations

This section covers Application and SQL release incompatibility considerations that apply to the DB2 11 changes related to client information fields changes.

Truncated client information values

In previous versions of DB2, client information values were truncated and padded to the maximum length. In DB2 11, trailing blanks are removed.

Attention: In DB2 11 New Function Mode (NFM), special registers for client information fields might return different length values

When the application compatibility of a package is set to `V11R1`, the application might receive a different length client information value than in previous versions of DB2. The value is no longer padded to the supported maximum length and trailing blanks are removed.

You can find more information about application compatibility in 6.6, “SET CURRENT APPLICATION COMPATIBILITY” on page 114.

SYSPROC.ADMIN_COMMAND_DB2 stored procedure

The DB2 provided **SYSPROC.ADMIN_COMMAND_DB2** stored procedure executes DB2 commands on a connected DB2 subsystem, or on a DB2 data sharing group member. This stored procedure also returns the command output messages.

Starting in DB2 11 conversion mode (CM), the **ADMIN_COMMAND_DB2** result set row returned in the created **SYSIBM.DB2_THREAD_STATUS** global temporary table when processing-type = **“**`THD`**”** has changed. The column data type and maximum lengths for **WORKSTATION**, **USERID**, **APPLICATION**, and **ACCOUNTING** has changed. Existing applications now receive a **VARCHAR** data type and possibly a different length client information value. The length is no longer padded to the supported maximum length.

Client information special registers length

In DB2 11 NFM, special registers for client information fields might return different length values. The values in these special registers change:

- CURRENT CLIENT_USERID
- CURRENT CLIENT_WRKSTNAME
- CURRENT CLIENT_APPLNAME
- CURRENT CLIENT_ACCTNG

In addition, the value of these special register change based on the application compatibility level. Whereas in previous version of DB2 special register values were truncated and padded, trailing blanks are removed in DB2 11. In consequence, when the application compatibility for a package is set to `V11R1`, the application might receive a different length client information value than they did previously.

Special registers

A *special register* is a storage area that is defined for an application process by DB2 and is used to store information that can be referenced in SQL statements. A reference to a special register is a reference to a value provided by the current server.

The following special registers related to the client information are available:

- **`CURRENT CLIENT_ACCTNG`** contains the value of the accounting string from the client information that is specified for the connection.
- **`CURRENT CLIENT_APPLNAME`** contains the value of the application name from the client information that is specified for the connection.
- **`CURRENT CLIENT_CORR_TOKEN`** contains the value of the client correlation token from the client information that is specified for the connection.
- **`CURRENT CLIENT_USERID`** contains the value of the client user ID from the client information that is specified for the connection.
- **`CURRENT CLIENT_WRKSTNNAME`** contains the value of the workstation name from the client information that is specified for the connection.

Resource Limit Facility

Resource limit tables can be used to limit the amount of resources used by SQL statements that run on middleware servers. Statements can be limited based on this client information:

- Application name
- User ID
- Workstation ID
- IP address

Resource limits apply only to dynamic SQL statements. The resource limit facility does not control static SQL statements regardless of whether they are issued locally or remotely.

The RLF table **`DSNRLMTxx`** columns are changed to support the longer lengths for client information fields, as summarized in Table 9-2.

Table 9-2 DSNRLMTxx longer columns in DB2 11

Column name	DB2 11	Comment
RLFEUAN	VARCHAR(255)	Specifies an application name
RLFEUID	VARCHAR(128)	Specifies a user ID
RLFEUWN	VARCHAR(255)	Specifies a user's workstation name

Column name	DB2 11	Comment
RLFIP	CHAR(254)	The IP address of the location where the request originated

The DDL use to create the RLMT table (DSNRLMTxx) are provided in the DB2 **DSNTIJSG** installation job. DB2 11 provides the long field version of the table in **DSNTIJSG**.

Note: The DB2 11 **DSNTIJSG** installation job provides the DDL for the long field version of the RLMT table, but long fields can be used in NFM only

Example 9-20 shows the long field version DDL as provided in **DSNTIJSG** in DB2 11. The DDL is commented out in the job.

Example 9-20 DDL to create the RLMT table DSNRLMT01, DB2 11 version

```
//*********************************************************************
//* USE THE FOLLOWING DDL TO CREATE AN OPTIONAL RLST AND INDEX FOR
//* RLF GOVERNING BASED ON END-USER ID, APPLICATION NAME, WORKSTATION
//* ID, AND IP ADDRESS.  SEE THE DB2 PERFORMANCE MONITORING AND TUNING
//* GUIDE FOR MORE INFORMATION ABOUT THIS TABLE.
//*
//*   CREATE TABLE DSNRLMT01
//*         (RLFFUNC        CHAR(1)      NOT NULL WITH DEFAULT,
//*          RLFEUAN        VARCHAR(255) NOT NULL WITH DEFAULT,
//*          RLFEUID        VARCHAR(128) NOT NULL WITH DEFAULT,
//*          RLFEUWN        VARCHAR(255) NOT NULL WITH DEFAULT,
//*          RLFIP          CHAR(254)    NOT NULL WITH DEFAULT,
//*          ASUTIME        INTEGER,
//*          RLFASUERR      INTEGER,
//*          RLFASUWARN     INTEGER,
//*          RLF_CATEGORY_B CHAR(1)      NOT NULL WITH DEFAULT)
//*      IN DSNRLST.DSNRLS01;
//*
//*   CREATE UNIQUE INDEX DSNMRL01
//*      ON DSNRLMT01
//*         (RLFFUNC, RLFEUAN DESC, RLFEUID DESC,
//*          RLFEUWN DESC, RLFIP DESC)
//*      CLUSTER CLOSE NO;
//*********************************************************************
```

For comparison, Example 9-21 shows the **DSNRLMT01** DDL as provided in **DSNTIJSG** in DB2 10. The DDL is commented out in the job.

Example 9-21 DDL to create the RLMT table DSNRLMT01, DB2 10 version

```
//*********************************************************************
//* USE THE FOLLOWING DDL TO CREATE AN OPTIONAL RLST AND INDEX FOR
//* RLF GOVERNING BASED ON END-USER ID, APPLICATION NAME, WORKSTATION
//* ID, AND IP ADDRESS.  SEE THE DB2 PERFORMANCE MONITORING AND TUNING
//* GUIDE FOR MORE INFORMATION ABOUT THIS TABLE.
//*
//*   CREATE TABLE DSNRLMT01
//*         (RLFFUNC        CHAR(1)      NOT NULL WITH DEFAULT,
//*          RLFEUAN        CHAR(32)     NOT NULL WITH DEFAULT,
//*          RLFEUID        CHAR(16)     NOT NULL WITH DEFAULT,
//*          RLFEUWN        CHAR(18)     NOT NULL WITH DEFAULT,
//*          RLFIP          CHAR(254)    NOT NULL WITH DEFAULT,
```

```
//*          ASUTIME        INTEGER,
//*          RLFASUERR      INTEGER,
//*          RLFASUWARN     INTEGER,
//*          RLF_CATEGORY_B CHAR(1)      NOT NULL WITH DEFAULT)
//*      IN DSNRLST.DSNRLS01;
//*
//*   CREATE UNIQUE INDEX DSNMRL01
//*      ON DSNRLMT01
//*         (RLFFUNC, RLFEUAN DESC, RLFEUID DESC,
//*          RLFEUWN DESC, RLFIP DESC)
//*      CLUSTER CLOSE NO;
//*********************************************************************
```

Tip: Verify that the value in **RLFAUTH** results as the table creator when running these DDL statements. This system parameter defines the authorization ID of the DB2 governor RLF

Table 9-3 summarizes the column definition differences between the DB2 10 and DB2 11 versions of the **DSNRLMTxx** tables.

Table 9-3 DSNRLMTxx column difference summary DB2 10 versus DB2 11

Column name	DB2 10	DB2 11	Comment
RLFEUAN	CHAR(32)	VARCHAR(255)	Specifies an application name
RLFEUID	CHAR(16)	VARCHAR(128)	Specifies a user ID
RLFEUWN	CHAR(18)	VARCHAR(255)	Specifies a user's workstation name
RLFIP	CHAR(254)	CHAR(254)	The IP address of the location where the request originated

Special considerations with DSNRLMT01 and DB2 versions

Special considerations apply for different combinations of RLMT table and DB2 versions. This section describes the different behaviors observed for different scenarios.

Example 9-22 shows the command used for starting the resource limit facility using the set of RLF tables 01.

Example 9-22 Start Resource Limit Facility command

```
-STA RLIMIT ID=01
```

For DB2 11 NFM with DB2 10 version of **DSNRLMT01**, Example 9-23 shows the output of the **START RLIMIT** command.

Example 9-23 Successful start of RLF

```
DSNT704I  -DB1A SYSIBM.DSNRLST01 HAS BEEN STARTED FOR THE RESOURCE
LIMIT FACILITY
DSNT704I  -DB1A SYSIBM.DSNRLMT01 HAS BEEN STARTED FOR THE RESOURCE
LIMIT FACILITY
DSN9022I  -DB1A DSNTCSTR 'START RLIMIT' NORMAL COMPLETION
***
```

RLF starts correctly when running DB2 11 and the DB2 10 version of the RLF tables.

For DB2 10 NFM with DB2 11 version of **DSNRLMT01**, Example 9-24 shows the output of the START RLIMIT command.

Example 9-24 Starting RLIMIT in DB2 10 with DSNRLMT01 version DB2 11

```
DSNT707I  -DBOA COLUMN RLFEUAN IN TABLE SYSIBM.DSNRLMT01 IS
INCORRECTLY DEFINED. THE COLUMN DEFINITION IS VARCHAR WITH LENGTH 255
AND NULL ATTRIBUTE N. THE COLUMN DEFINITION SHOULD BE CHAR WITH
LENGTH 32 AND NULL ATTRIBUTE N. THE START RLIMIT COMMAND WILL NOT USE
THIS TABLE.
DSNT707I  -DBOA COLUMN RLFEUID IN TABLE SYSIBM.DSNRLMT01 IS
INCORRECTLY DEFINED. THE COLUMN DEFINITION IS VARCHAR WITH LENGTH 128
AND NULL ATTRIBUTE N. THE COLUMN DEFINITION SHOULD BE CHAR WITH
LENGTH 16 AND NULL ATTRIBUTE N. THE START RLIMIT COMMAND WILL NOT USE
THIS TABLE.
DSNT707I  -DBOA COLUMN RLFEUWN IN TABLE SYSIBM.DSNRLMT01 IS
INCORRECTLY DEFINED. THE COLUMN DEFINITION IS VARCHAR WITH LENGTH 255
AND NULL ATTRIBUTE N. THE COLUMN DEFINITION SHOULD BE CHAR WITH
LENGTH 18 AND NULL ATTRIBUTE N. THE START RLIMIT COMMAND WILL NOT USE
THIS TABLE.
DSNT704I  -DBOA SYSIBM.DSNRLST01 HAS BEEN STARTED FOR THE RESOURCE
LIMIT FACILITY
DSNT727I  -DBOA TABLE SYSIBM.DSNRLMT01 WILL NOT BE USED BY THE
RESOURCE LIMIT FACILITY
DSN9022I  -DBOA DSNTCSTR 'START RLIMIT' NORMAL COMPLETION
***
```

This example shows that the DB2 11 version of the RLMT table is not usable in a DB2 10 subsystem. As a result, RLF is partially started and there is no support for the RLMT table in this case. Example 9-25 shows this evidence, this is the output of the **-DISPLAY RLIMIT** command.

Example 9-25 -DIS RLIMIT: RLF partially started

```
DSNT700I  -DBOA SYSIBM.DSNRLST01 IS THE ACTIVE RESOURCE LIMIT
SPECIFICATION TABLE
DSN9022I  -DBOA DSNTCDIS 'DISPLAY RLIMIT' NORMAL COMPLETION
***
```

For DB2 11 CM with DB2 11 version of **DSNRLMT01**, Example 9-26 shows the output of the **START RLIMIT** command.

Example 9-26 Starting RLIMIT in DB2 11 CM with DSNRLMT01 version DB2 11

```
DSNT728I  -DBOB THE FORMAT OF TABLE SYSIBM.DSNRLMT01 IS NOT SUPPORTED
IN COMPATIBILITY MODE. THE START RLIMIT COMMAND WILL NOT USE THIS
TABLE.
DSNT704I  -DBOB SYSIBM.DSNRLST01 HAS BEEN STARTED FOR THE RESOURCE
LIMIT FACILITY
DSNT727I  -DBOB TABLE SYSIBM.DSNRLMT01 WILL NOT BE USED BY THE
RESOURCE LIMIT FACILITY
DSN9022I  -DBOB DSNTCSTR 'START RLIMIT' NORMAL COMPLETION
***
```

The DB2 11 of the RLMT table is not compatible with DB2 11 CM. There is no support for RLMT in this scenario. Example 9-27 shows that RLF is partially active.

Example 9-27 -DIS RLIMIT: RLF partially started

```
DSNT700I  -DB0B SYSIBM.DSNRLST01 IS THE ACTIVE RESOURCE LIMIT
SPECIFICATION TABLE
DSN9022I  -DB0B DSNTCDIS 'DISPLAY RLIMIT' NORMAL COMPLETION
***
```

Converting resource limit facility (RLF) tables

When migrated to DB2 11 NFM, you have to convert the resource limit facility (RLF) tables to take advantage of the longer client information fields. Follow these steps to alter and existing RLF table. Verify the status of RLF by executing a -DIS RLIMIT command. An output example is shown in Example 9-28.

Example 9-28 -DIS RLIMIT output example

```
DSNT700I  -DB1A SYSIBM.DSNRLST01 IS THE ACTIVE RESOURCE LIMIT
SPECIFICATION TABLE
DSNT700I  -DB1A SYSIBM.DSNRLMT01 IS THE ACTIVE RESOURCE LIMIT
SPECIFICATION TABLE
DSN9022I  -DB1A DSNTCDIS 'DISPLAY RLIMIT' NORMAL COMPLETION
***
```

After stopping RLF you can alter the RLMT table to the new format. Example 9-29 shows the SQL used for altering the RLMT table to the DB2 11 NFM supported format.

Example 9-29 ALTER TABLE SYSIBM.DSNRLMT01

```
ALTER TABLE SYSIBM.DSNRLMT01 ALTER COLUMN RLFEUAN
  SET DATA TYPE VARCHAR(255);
ALTER TABLE SYSIBM.DSNRLMT01 ALTER COLUMN RLFEUID
  SET DATA TYPE VARCHAR(128);
ALTER TABLE SYSIBM.DSNRLMT01 ALTER COLUMN RLFEUWN
  SET DATA TYPE VARCHAR(255);
```

Converting resource limit facility (RLF) tables, no RLF outage

If your installation cannot afford to run without RLF active, this alternative method allows for a RLMT migration without RLF outage. Using the DB2 11 DDL in **DSNTIJSG**, create a new RMT table with a non -used ID. For example, create DSNRLMT02 where the current RLMT table is **DSNRLMT01**. If needed, create an ID=02 of the table **DSNRLSTxx** as well.

You can copy the RLF definitions from the ID=01 to the ID=02 tables using SQL. Example 9-30 documents the SQL used to copy data from the old to the new version of the RLMT table.

Example 9-30 Copying RLMT data to a DB2 11 version of the table

```
INSERT INTO SYSIBM.DSNRLMT02
  ( RLFFUNC, RLFEUAN, RLFEUID, RLFEUWN, RLFIP,
    ASUTIME, RLFASUERR, RLFASUWARN, RLF_CATEGORY_B )
SELECT
    RLFFUNC, RLFEUAN, RLFEUID, RLFEUWN, RLFIP,
    ASUTIME, RLFASUERR, RLFASUWARN, RLF_CATEGORY_B
  FROM  SYSIBM.DSNRLMT01;
```

Example 9-31 shows the SQL used to copy data between RLST tables.

Example 9-31 Copying RLST data to a DB2 11 version of the table

```
INSERT INTO SYSIBM.DSNRLST02
  ( AUTHID, PLANNAME, ASUTIME, LUNAME, RLFFUNC,
    RLFBIND, RLFCOLLN, RLFPKG, RLFASUERR,
    RLFASUWARN, RLF_CATEGORY_B )
SELECT
    AUTHID, PLANNAME, ASUTIME, LUNAME, RLFFUNC,
    RLFBIND, RLFCOLLN, RLFPKG, RLFASUERR,
    RLFASUWARN, RLF_CATEGORY_B
  FROM  SYSIBM.DSNRLST01;
```

After the new RLST and RLMT tables are populated with the original data, you can switch the RLF definitions without outage. In this example, you can issue the **START RLIMIT** command using ID=02, as shown in Example 9-32, even if RLF is active with ID=01.

Example 9-32 -STA RLIMIT command

```
-STA RLIMIT ID=02
```

Tip: While RLF is active, you can switch RLF tables by executing the **START RLIMIT** command using a new ID.

Example 9-33 shows the output of this command. Note the **DSNT709I** message. This message informs that a **START RLIMIT** command was entered and that the facility was already active. The facility remains active and switches from using the old table name to the new one.

Example 9-33 Starting RLIMIT on a new set of RLF tables

```
DSNT709I  -DB1A SYSIBM.DSNRLST02 NOW ACTIVE. SYSIBM.DSNRLST01 WAS OLD
RESOURCE LIMIT SPECIFICATION TABLE
DSNT709I  -DB1A SYSIBM.DSNRLMT02 NOW ACTIVE. SYSIBM.DSNRLMT01 WAS OLD
RESOURCE LIMIT SPECIFICATION TABLE
DSN9022I  -DB1A DSNTCSTR 'START RLIMIT' NORMAL COMPLETION
***
```

Profile monitoring for remote threads and connections

Profile tables identify contexts in which DB2 takes particular actions such resource monitoring, subsystem parameter customization, and dynamic SQL stabilization. The contexts might identify statements, threads, or connections based on information about the originating application, system, or user.

A profile is a set of criteria that identifies a particular context 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 **DSNTIJSG** job during DB2 installation or migration.

A complete set of profile tables and related indexes includes the following objects:

- **SYSIBM.DSN_PROFILE_TABLE**
- **SYSIBM.DSN_PROFILE_HISTORY**
- **SYSIBM.DSN_PROFILE_ATTRIBUTES**
- **SYSIBM.DSN_PROFILE_ATTRIBUTES_HISTORY**
- **SYSIBM.DSN_PROFILE_TABLE_IX_ALL**
- **SYSIBM.DSN_PROFILE_TABLE_IX2_ALL**
- **SYSIBM.DSN_PROFILE_ATTRIBUTES_IX_ALL**

Tip: Refer to*DB2 11 for z/OS Managing Performance*, SC19-4060 for details about the profile tables

The monitoring functions are defined by inserting rows in the **DSN_PROFILE_ATTRIBUTES** table. You can specify the following values in the **KEYWORD** column of the **DSN_PROFILE_ATTRIBUTES** table:

- **MONITOR THREADS**
- **MONITOR CONNECTIONS**
- **MONITOR IDLE THREADS**

MONITOR THREADS indicates that the profile monitors the total number of concurrent active remote threads according to the following filter criteria defined on **SYSIBM.DSN_PROFILE_TABLE**:

- **LOCATION**
- **IPADDR**
- **PRDID**
- **ROLE AUTHID**
- **COLLID**
- **PKGNAME**

The system-wide threshold that is defined by the value of the **MAXDBAT** subsystem parameter continues to apply.

MONITOR CONNECTIONS indicates that the profile monitors the total number of remote connections from TCP/IP requesters. The only filtering criteria is the **LOCATION** column in the SYSIBM. **DSN_PROFILE_TABLE**. Nevertheless, you can specify either an IP address or a domain name for its value. The system-wide threshold that is defined by the value of the **CONDBAT** subsystem parameter continues to apply.

MONITOR IDLE THREADS indicates that the profile monitors the approximate time (in seconds) that an active server thread is allowed to remain idle. A zero value means that matching threads are allowed to remain idle indefinitely.

At a glance, system profile monitoring allows to tailor the values of the following otherwise subsystem-level parameters to the need of any application:

- **MAXDBAT**
- **CONDBAT**
- **IDTHTOIN**

Important: By making client information fields longer, DB2 11 provides greater granularity for managing DDF connections, threads, and idle thread timeout.

After profiling is correctly defined, use the **-START PROFILE** DB2 command to start profile monitoring. Issue a -STOP PROFILE command. to disable all profile functions.

The following **DSN_PROFILE_TABLE** columns are defined as **VARCHAR(255)** in DB2 10 for z/OS, but their values are truncated as follows:

- **CLIENT_APPLNAME**: truncated to 32 bytes
- **CLIENT_USERID**: truncated to 16 bytes
- **CLIENT_WRKSTNNAME**: truncated to 18 bytes

There is no change in behavior in DB2 11 conversion mode. The **-START PROFILE** command continues to truncate the client information field's information in the **DSN_PROFILE_TABLE**, as in DB2 10.

In DB2 11 New Function Mode (NFM), the complete value of columns **CLIENT_APPLNAME** and **CLIENT_WRKSTNNAME**, up to 255 bytes, is honoured. The value in column **CLIENT_USERID** is considered up to 128 bytes.

Important: IBM Data Server Driver or Client level DB2 10.5 Fix Pack 2 is required to exploit the enhanced client information fields. Previous levels do not take advantage, even if the DB2 server is running DB2 11 for z/OS NFM

Workload management

For mixed workloads, the general recommendation is to use multiple WLM service classes to differentiate users and applications that have different levels of importance for the business.

A WLM Service class describes a group of work within a workload with similar performance characteristics. A service class is a key construct for WLM. Each service class has at least one period, and each period has one goal. Address spaces and transactions are assigned to service classes using classification rules. Within a workload, a group of work with similar performance requirements can share the same service class.

WLM Report Classes refers to an aggregate set of work for reporting purposes. You can use report classes to analyze the performances of individual workloads running in the same or different service classes. Work is classified into report classes using the same classification rules that are used for classification into service classes. A useful way to contrast report classes to service classes is that report classes are used for monitoring work; service classes are primarily to be used for managing work.

As an example, the following general considerations might apply to your workload environment:

- Use WLM service classes with percentile response time goals for early periods that have frequent completions of short consumption work.
- Use WLM service classes with velocity goals for later periods containing work having less-frequent completions and larger, perhaps more varying, resource consumption characteristics.
- Potentially use a discretionary goal for the last period, which might not be applicable in OLTP environments with high CPU utilization, because it can result in severe DB2 locking issues.

The design of a WLM strategy must match the workload characteristics. For example, operational BI queries are typically numerous and small CPU consumers. Therefore, they have WLM service classes with response time goals and fall into early periods. Alternatively, data mining activity might be less frequent, long-running, and have wide variability in resource consumption. It is therefore likely to be targeted for WLM service classes with velocity goals and later periods.

WLM Classification rules are the filters that WLM uses to associate a transaction's external properties (also called *work qualifiers*, such as LU name or user ID) with a WLM service class. As a preferred practice, classify each distributed request within WLM. If you do not classify your DDF transactions into specific WLM service classes, they are assigned to the default service class for the DDF workload.

Optionally, you can assign incoming work to a report class. Report classes can be used to report on a subset of transactions running in a single service class but also to combine transactions running in different service classes within one report class.

Figure 9-1 shows a LPAR level CPU utilization report for a given DB2 workload. This report is based on the RMF Workload (type 72) records. In this example, it is not possible to identify how the CPU utilization is distributed by application.

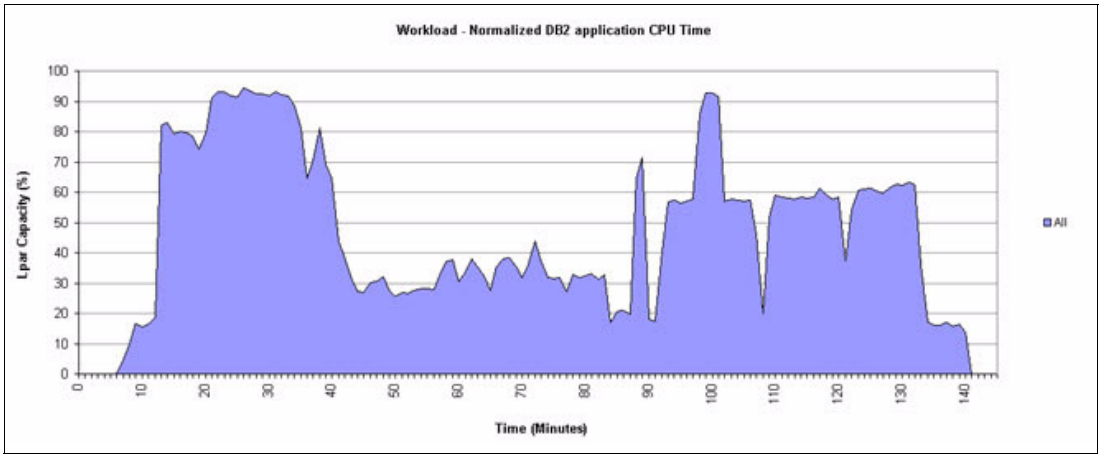


Figure 9-1 RMF Workload, LPAR CPU utilization

For the same workload, it is possible to obtain a more detailed overview of the CPU utilization by using WLM Report Class. Figure 9-2 shows a graph of the same scenario when the CPU utilization report is plotted by Report Class. In this case it is possible to identify which part of the application is active a different periods.

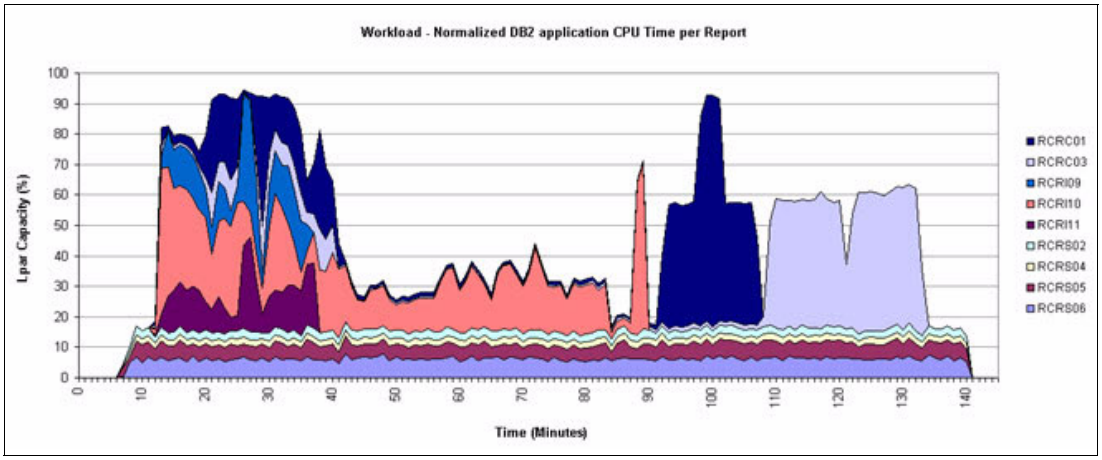


Figure 9-2 RMF Workload, LPAR CPU utilization per WLM Report Class

WLM Classification Rules are the rules you define to categorize work into service classes, and optionally report classes, based on work qualifiers. A work qualifier is what identifies a work request to the system. The first qualifier is the subsystem type that receives the work request.

Table 9-4 list the WLM classification attributes, or qualifiers, that pertain to DB2 DDF threads in z/OS 1.13.

Table 9-4 DDF - WLM classification attributes in z/OS 1.13

Attribute	Classify workload based on
AI	Accounting information
CAI	Client accounting information

Attribute	Classify workload based on
CI	DB2 correlation ID of the DDF thread
CN	Collection name
LU	IBM VTAM® LUNAME
NET	VTAM NETID
PK	Name of the first package accessed
PN	Plan name
PR	Stored procedure name
SI	Subsystem instance
SSC	Subsystem collection name
UI	User id

z/OS 2.1 introduces a new WLM classify work request macro service, IWM4CLSY. this new macro supports the new longer lengths client information fields to classify DDF server threads. There are new WLM classification attributes to support the longer lengths client information fields. These new attributes are listed in Table 9-5.

Table 9-5 New DDF - WLM classification attributes in z/OS 2.1

Attribute	Classify workload based on	Maximum length
CUI	Client user ID	128 bytes
CWN	Client workstation name	255 bytes
CTN	Client transaction (application) name	255 bytes
CIP	Client IP address	39 bytes
CAI	Client accounting information	512 bytes
PC	Process name	32 bytes
SPM	Subsystem parameter	255 bytes

Note: Refer to the IBM publication “z/OS MVS Planning Workload Management” for more details about WLM and its components.

The SPM qualifier has a maximum length of 255 bytes. The first 16 bytes contain the client's user ID. The next 18 bytes contain the client's workstation name. The remaining 221 bytes are reserved. If the length of the client's user ID is less than 16 bytes, SPM uses blanks after the user ID to pad the length. If the length of the client's workstation name is less than 18 bytes, SPM uses blanks after the workstation name to pad the length. The values of the client user ID and client workstation name are the truncated to 16 bytes and 18 bytes respectively. The full lengths, 128 bytes and 255 bytes respectively, are specified in the CUI and CWN attributes, as listed in Table 9-5.

Example 9-34 shows an example of the WLM ISPF panel Modify Rules for the Subsystem Type in a z/OS 2.1 LPAR.

Example 9-34 WLM Modify Rules for the Subsystem Type panel

```
. . . . . .
  Subsystem-Type  Xref  Notes  Options  Help
-------------------------------------------------------------------------------
                Modify Rules for the Subsystem Type       Row 1 to 2 of 2
Command ===> _____________________________________________ Scroll ===> PAGE

Subsystem Type . : DDF         Fold qualifier names?   N  (Y or N)
Description  . . . DDF clasificatoin rules

Action codes:   A=After     C=Copy        M=Move     I=Insert rule
                B=Before    D=Delete row  R=Repeat   IS=Insert Sub-rule
                                                        More ===>
          --------Qualifier--------               -------Class--------
Action    Type       Name     Start                Service     Report
                                              DEFAULTS: DDFUNKWN    REPUNKWN
 ____  1  SI         DB1D*    ___                  DDFDEFLT    REPDEFLT
 ____  2    AI         Bxls*    56                 DDFHI       REPBXLS
```

In this example, if a DDF work request comes in from an DB2 subsystem other than DB1D, then it is assigned to the default Service and Report Class DDFUNKWN and REPUNKWN. A request coming from DB1D with Accounting Information starting with the string **Bxls** gets assigned to DDFHI and REPBXLS. The order of the nesting and the order of the level 1 qualifiers, determine the hierarchy of the classification rules.

Example 9-35 uses a Java application to drive the distributed workload. In the program, the accounting information is set.

Example 9-35 Setting accounting information in a Java program

```
con = DriverManager.getConnection(url);
DB2Connection db2con = (DB2Connection) con;
con.setAutoCommit(false);
db2con.setClientInfo("ClientAccountingInformation", "BxlsAPP_ChocoO1");
```

Important: The start of the Client Accounting Information string has to be 56 in the WLM ISPF panel to match the beginning of the string used in the program

In the “Fold qualifier names” option, set to the default Y, means that the qualifier names is folded to uppercase as soon as you type them and press Enter. If you set this option to N, the qualifier names remains in the case they are typed in. Leave this option set to Y unless you know that you need mixed case qualifier names in your classification rules.

You can use masking and wild card notation to group qualifiers that share a common substring. For work qualifiers that run longer than eight characters, you can use a start position to indicate how far to index into the character string. The name field for work qualifiers is 8 characters long. You can use nesting for the work qualifiers that run longer than 8 characters. Use the following fields:

- Accounting information
- Client accounting information
- Client IP address
- Client transaction name
- Client user ID

- Client workstation name
- Collection name
- Correlation information
- Package name
- Procedure name
- Process name
- Scheduling environment
- Subsystem parameter
- zEnterprise service class name

From this list, the fields Scheduling environment and zEnterprise service class name are not applicable for DDF workloads.

For example, for DDF workload, Accounting Information is the value of the DB2 accounting string associated with the DDF server thread. Because DB2 supports more than 8 characters in accounting information, and the WLM ISPF application allows only 8 characters per rule, the application allows “nesting” for accounting information.

By nesting work qualifiers you can exploit the longer client information fields introduced in DB2 11. Example 9-36 shows an example.

Example 9-36 WLM classification rules: nesting accounting information

```
   Subsystem-Type  Xref  Notes  Options  Help
------------------------------------------------------------------------------
                Modify Rules for the Subsystem Type       Row 1 to 3 of 3
Command ===> _____________________________________________ Scroll ===> PAGE

Subsystem Type . : DDF         Fold qualifier names?   N  (Y or N)
Description  . . . DDF clasificatoin rules

Action codes:   A=After     C=Copy        M=Move     I=Insert rule
                B=Before    D=Delete row  R=Repeat   IS=Insert Sub-rule
                                                        More ===>
          --------Qualifier--------               -------Class--------
Action    Type       Name     Start                Service     Report
                                          DEFAULTS: DDFUNKWN    REPUNKWN
 ____  1  SI         DB1D*    ___                  DDFDEFLT    REPDEFLT
 ____  2    AI         BxlsAPP* 56                  DDFHI       REPBXLS
 ____  3      AI         Choco*   63                DDFHI       REPCHOCO
```

In this configuration, and application with accounting information starting with the string `BxlsAPP` is classified in the DDFHI service class and REPBXLS report class. If the application has an accounting information field starting with `BxlsAPP_Choco` uses the **`REPCHOCO`** report class instead.

To activate the WLM changes use the Utilities menu entry, and then option 1 Install definition, as shown in Example 9-37.

Example 9-37 installing WLM definitions

```
File  Utilities  Notes  Options  Help
----- +-----------------------------------------------+ ----------------
Funct | 1  1. Install definition                      |  Appl LEVEL029
Comma |    2. Extract definition                      | _______________
      |    3. Activate service policy                 |
Defin |    4. Allocate couple data set                |
      |    5. Allocate couple data set using CDS values |
Defin |    6. Validate definition                     |
```

```
Descr +-----------------------------------------------+ _

Select one of the
following options. . . . . ___  1.   Policies
                               2.   Workloads
                               3.   Resource Groups
```

When installation is successful, the system provide the feedback shown in Example 9-38

Example 9-38 WLM Service definition installation successful

```
Service definition was installed. (IWMAM038)
```

Changes to the WLM policy have to be activated to be made effective.

Example 9-39 shows how the WLM policy activation can be done using the WLM ISPF panels.

Example 9-39 Activating WLM definitions

```
File  Utilities  Notes  Options  Help
 ----- +-----------------------------------------------+ ----------------
 Funct | 3  1. Install definition                      |  Appl LEVEL029
 Comma |    2. Extract definition                      | _______________
       |    3. Activate service policy                 |
 Defin |    4. Allocate couple data set                |
       |    5. Allocate couple data set using CDS values |
 Defin |    6. Validate definition                     |
 Descr +-----------------------------------------------+ _

 Select one of the
 following options. . . . . ___  1.   Policies
                                2.   Workloads
                                3.   Resource Groups
```

As confirmation, the **IWM001I** system message is written in to system console, as shown in Example 9-40.

Example 9-40 WLM Policy activated

```
IWM001I WORKLOAD MANAGEMENT POLICY DB211TO NOW IN EFFECT
```

There are many ways of verifying of the WLM classification is working as expected. A simple and quick way is to explore the DDF activity in the RMF Enclave Report panel. The Enclave report provides detailed information about the activities of enclaves.

Example 9-41 shows an example of the RMF Enclave Report panel in z/OS 2.1.

Example 9-41 RMF Enclave Report panel

```
RMF V2R1   Enclave Report                 Line 1 of 5
Command ===>                                          Scroll ===> CSR

Samples: 100     System: SC76  Date: 08/07/13  Time: 19.36.40  Range: 100   Sec

Current options:   Subsystem Type: ALL                          -- CPU Util --
                   Enclave Owner:                              Appl%   EAppl%
                   Class/Group:                                  0.2      1.9
```

```
Enclave   Attribute  CLS/GRP  P Goal    % D X   EAppl%   TCPU    USG  DLY  IDL

*SUMMARY                                        0.811
ENC00002             SYSSTC   1 N/A       Y     0.736  12.62     34  3.0  0.0
ENC00004             DDFHI    2        50         0.040  0.100   100  0.0  0.0
ENC00003             DDFHI    2        50 W       0.036  0.083   0.0  0.0  0.0
ENC00001             SYSSTC   1 N/A       Y     0.000  0.025   0.0  0.0  0.0
```

This example shows the **ENC00004** and **ENC00003** enclaves running on the DDFHI service class. Selecting one of the enclaves provide access to the Enclave Classification Attributes panel where you can obtain more details, as shown in Example 9-42.

Example 9-42 Enclave details in RMF Enclave report

```
                        RMF V2R1   Enclave Report                 Line 1 of 5
 Command ===>                                                 Scroll ===> CSR

 Samples:  +-----------------------------------------------------------------+
           |                  RMF Enclave Classification Attributes          |
 Current o |                                                                 |
           | The following details are available for enclave ENC00004        |
           | Press Enter to return to the Report panel.                      |
           |                                                                 |
 Enclave   |                                                    More:     +  |
           | Subsystem Type: DDF     Owner: DB1DDIST   System: SC76          |
 *SUMMARY  | Accounting Information . . . :                                  |
 ENC00002  |    JCC04170192.168.150.1                                BxlsA   |
 ENC00004  |    PP_Choco01                                                   |
 ENC00003  |                                                                 |
 ENC00001  | Collection Name  . . . . . . : NULLID                           |
           | Connection Type  . . . . . . : SERVER                           |
           | Correlation Information  . . : db2jcc_appli                     |
           | LU Name  . . . . . . . . . . :                                  |
           | Net ID . . . . . . . . . . . :                                  |
           | Plan Name  . . . . . . . . . : DISTSERV                         |
           | Priority . . . . . . . . . . :                                  |
           | Process Name . . . . . . . . : db2jcc_application               |
           | Transaction/Job Class  . . . :                                  |
           | Transaction/Job Name . . . . :                                  |
           | User ID  . . . . . . . . . . : DB2RS1                           |
```

Navigating down this panels provide access to the Client Information Field values. A partial example is shown in Example 9-43.

Example 9-43 RMF Enclave Classification Attributes

```
RMF V2R1   Enclave Report                 Line 1 of 5
  Command ===>                                              Scroll ===> CSR

  Samples:  +-----------------------------------------------------------------+
            |                  RMF Enclave Classification Attributes          |
  Current o |                                                                 |
            | The following details are available for enclave ENC00004        |
            | Press Enter to return to the Report panel.                      |
            |                                                                 |
  Enclave   |                                                  More:   - +    |
            |                                                                 |
  *SUMMARY  |                                                                 |
  ENC00002  | Client IP Address  . . . . . :                                  |
  ENC00004  |    0000:0000:0000:0000:0000:0000:9.55.137.                      |
```

```
ENC00003  | Client User ID . . . . . . . :                                  |
ENC00001  |    db2rs1                                                       |
          |                                                                 |
          |                                                                 |
          | Client Transaction Name  . . :                                  |
          |    db2jcc_application                                           |
          |                                                                 |
          |                                                                 |
          |                                                                 |
          |                                                                 |
          | Client Workstation/Host Name :                                  |
          |    192.168.150.1                                                |
```

As discussed in this chapter, the driver version can influence the length of the client information field that is sent to the DB2 server. For example, a Java program setting the client accounting information with a string of 255 characters sends the complete string when using a DB2 driver 10.5 fix pack 2. The same application executed with a lower version of the driver sends the string truncated to 200 characters. WLM classification rules taking advantage of the longer client information fields introduced in DB2 11 behave different if the driver used by the applications is not the one required to fully exploit the DRDA changes in DB2 11 (9.5 fix pack 2).

Important: With DB2 11 longer client information fields, WLM classification rules might behave inconsistently, depending on the driver version used by the applications.

Setting Client info fields

This section discusses these topics related to how to change the Client info values:

- **WLM_SET_CLIENT_INFO** stored procedure
- Setting Client info in Java applications
- Setting Client info in DB2 command line processor scripts

WLM_SET_CLIENT_INFO stored procedure

WLM_SET_CLIENT_INFO is a DB2 provided, WLM established, stored procedure. It allows the caller to set client information that is associated with the current connection at the DB2 for z/OS server. It is of particular use for connections where DB2 for z/OS is the requester because in such situations there is no other way of setting the client information values. These DB2 for z/OS client special registers can be changed by calling **WLM_SET_CLIENT_INFO**:

- **CURRENT CLIENT_ACCTNG**
- **CURRENT CLIENT_USERID**
- **CURRENT CLIENT_WRKSTNNAME**
- **CURRENT CLIENT_APPLNAME**

Important: The **WLM_SET_CLIENT_INFO** stored procedure that is shipped with DB2 11 for z/OS does not allow to change the value of the new client information register **CURRENT CLIENT_CORR_TOKEN**, nor to update the system built-in session global variable **SYSIBM.CLIENT_IPADDR**.

Example 9-44 shows the **WLM_SET_CLIENT_INFO** call syntax.

Example 9-44 WLM_SET_CLIENT_INFO syntax

```
>>-WLM_SET_CLIENT_INFO--(--+-client_userid-+--,--+-client_wrkstnname-+--,-->
                           '-NULL----------'     '-NULL--------------'
```

```
>--+-client_applname-+--,--+-client_acctstr-+--)--------------><
   '-NULL------------'     '-NULL-----------'
```

The **WLM_SET_CLIENT_INFO** procedure uses the following parameters:

client_userid A **VARCHAR(255)** input parameter that specifies the user ID for the client. If NULL is specified, the value remains unchanged. If an empty string is specified, the user ID for the client is reset to the default value. If the value specified exceeds 128 bytes, it is truncated to 128 bytes.

client_wrkstnname A **VARCHAR(255)** input parameter that specifies the workstation name for the client. If NULL is specified, the value remains unchanged. If an empty string is specified, the workstation name for the client is reset to the default value.

client_applname A **VARCHAR(255)** input parameter that specifies the application name for the client. If NULL is specified, the value remains unchanged. If an empty string is specified, the application name for the client is reset to the default value.

client_acctstr A **VARCHAR(255)** input parameter that specifies the accounting string for the client. If NULL is specified, the value remains unchanged. If an empty string is specified, the accounting string for the client is reset to the default value.

Setting Client info in Java applications

Table 9-6 summarizes the client information properties values for Java type 4 connectivity to DB2 for z/OS. It also shows the default values and the DB2 special register that can be used to read this information at run time.

Table 9-6 Client information property values for type 4 connectivity to DB2 for z/OS

Name	Max length	Default value	Special register
ApplicationName	255 bytes	The string “db2jcc_application”	CURRENT CLIENT_APPLNAME
ClientAccountingInformation	255	JCCversionclient-ip	CURRENT CLIENT_ACCTNG
ClientCorrelationToken	255	Data server generated LUWID	CURRENT CLIENT_CORR_TOKEN
ClientHostname	255	The string “db2jcc_local”	CURRENT CLIENT_WRKSTNNAME
ClientUser	128	The user ID that was specified when the connection was established	CURRENT CLIENT_USERID

These properties can be modified using the **setClientInfo** Java method. Example 9-45 shows how to setup the client user ID using this method. In this example, db2con is an established connection to DB2 for z/OS.

Example 9-45 Using the setClientInfo Java method

```
db2con.setClientInfo("ClientUser",
"ClientUser_01234567890123456789012345678901234567890123456789012345678901234567890123456789
012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012
234567890123456789012345678901234567890123456789012345678901234567890123456789");
```

Example 9-46 shows a fully functional Java program that can be used to test the new lengths provided by DB2 11. This example sets 4 client information fields and execute some SQL code.

Example 9-46 Java program: setting client information fields

```
import com.ibm.db2.jcc.DB2Connection;
import java.sql.*;

public class DB211NewDriver {

    public static Connection con = null;
    public static CallableStatement cstmt;
    public static ResultSet results;
    public static boolean debug = true;

    public static void main(String args[]) throws Exception {
        String url = "jdbc:db2://redbook8:38420/DB1A"
                + ":user=db2r1;password=********;"
                + "traceLevel="
                + (com.ibm.db2.jcc.DB2BaseDataSource.TRACE_ALL) + ";"
                +
"traceFile=c:\\work\\Author\\Redbook#8\\DRDA_traces\\DB211NewDriver.trace;";
        try {
            Class.forName("com.ibm.db2.jcc.DB2Driver");
        } catch (java.lang.ClassNotFoundException e) {
            System.err.print("ClassNotFoundException: ");
            System.err.println(e.getMessage());
        }
        con = DriverManager.getConnection(url);
        DB2Connection db2con = (DB2Connection) con;
        con.setAutoCommit(false);

        db2con.setClientInfo("ClientUser",
"ClientUser_0123456789012345678901234567890123456789012345678901234567890123456789012345678
9012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789
0123456789012345678901234567890123456789");
        db2con.setClientInfo("ClientAccountingInformation",
"ClientAccountingInformation_01234567890123456789012345678901234567890123456789012345678901
2345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012
345678901234567890123456789012345678901234567890123456789");
        db2con.setClientInfo("ClientHostname",
"WorkstationName_01234567890123456789012345678901234567890123456789012345678901234567890123
4567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
567890123456789012345678901234567890123456789");
        db2con.setClientInfo("ApplicationName",
"DB211JavaNewDriver_01234567890123456789012345678901234567890123456789012345678901234567890
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901
234567890123456789012345678901234567890123456789");
        db2con.setClientInfo("ClientCorrelationToken", "BXLS001");

        query1(con);
        query2(con);
    }

    public static void query1(Connection con) {
        System.out.println("Starting DB211NewDriver V1");
        Statement stmt;
        ResultSet rs;
        String planname;
        try {
```

```
            stmt = con.createStatement();
            rs = stmt.executeQuery("SELECT NAME FROM SYSIBM.SYSPLAN WHERE PROGAUTH = 'D'");
            System.out.println("--> Query executed. resultset follows");
            while (rs.next()) {
                planname = rs.getString(1);
                System.out.println("PLAN NAME = " + planname);
                // Force 1 second sleep for easier monitoring
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException ex) {
                    Thread.currentThread().interrupt();
                }
            }
            System.out.println("--> Resultset exhausted");
            rs.close();
            stmt.close();
        } catch (SQLException e) {
            System.out.println("==> SQLException = " + e);
            System.out.println("==> SQLCODE = " + e.getErrorCode());
            System.out.println("==> SQLSTATE = " + e.getSQLState());
            System.out.println("==> Text of Error Message = " + e.getMessage());
        }
    }

    public static void query2(Connection con) {
        System.out.println("Starting query2");
        Statement stmt;
        ResultSet rs;
        String currclntacctng;
        try {
            stmt = con.createStatement();
            rs = stmt.executeQuery("select CURRENT CLIENT_ACCTNG from sysibm.sysdummy1;");
            System.out.println("--> Query executed. resultset follows");
            while (rs.next()) {
                currclntacctng = rs.getString(1);
                System.out.println("CURRENT CLIENT_ACCTNG = " + currclntacctng);
                // Force 1 second sleep for easier monitoring
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException ex) {
                    Thread.currentThread().interrupt();
                }
            }
            System.out.println("--> Resultset exhausted");
            rs.close();
            stmt.close();
        } catch (SQLException e) {
            System.out.println("==> SQLException = " + e);
            System.out.println("==> SQLCODE = " + e.getErrorCode());
            System.out.println("==> SQLSTATE = " + e.getSQLState());
            System.out.println("==> Text of Error Message = " + e.getMessage());
        }
    }
}
```

Tip: To keep an active connection with DB2 during the complete execution of the program and to make easier to monitor it online using commands, disable **Autocommit** by modifying the connection with **con.setAutoCommit(false);**

Example 9-47 shows the output of the execution of this program in this example test environment.

Example 9-47 Java sample program output

```
Starting DB211NewDriver V1
--> Query executed. resultset follows
PLAN NAME = DSNTIA11
PLAN NAME = DSNREXX
PLAN NAME = DSNESPCS
....
PLAN NAME = ADB2RIP
PLAN NAME = ADB2WCL
PLAN NAME = ADB27SPC
--> Resultset exhausted
Starting query2
--> Query executed. resultset follows
CURRENT CLIENT_ACCTNG =
ClientAccountingInformation_012345678901234567890123456789012345678901234567890123456789012
345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012
456789012345678901234567890123456789012345678901234567890123456789
--> Resultset exhausted
```

Example 9-48 shows the result of the DB2 command **-DIS THD(*) DETAIL** when the type 4 driver is DB2 10.5 fix pack 2.

Example 9-48 -DIS THD(*) DETAIL

```
NAME     ST A   REQ ID           AUTHID   PLAN     ASID TOKEN
SERVER   RA *     2 db2jcc_appli DB2R1    DISTSERV 0131   621
 V437-WORKSTATION=WorkstationName_012345678901234567890123456789012345
        678901234567890123456789012345678901234567890123456789012345678
        890123456789012345678901234567890123456789012345678901234567890
        012345678901234567890123456789012345678901234567890123456789
      USERID=ClientUser_012345678901234567890123456789012345678901234567
        678901234567890123456789012345678901234567890123456789012345678
        890123456
      APPLICATION NAME=DB211JavaNewDriver_012345678901234567890123456789
        890123456789012345678901234567890123456789012345678901234567890
        012345678901234567890123456789012345678901234567890123456789012
        234567890123456789012345678901234567890123456789012345678901234
 V441-ACCOUNTING=ClientAccountingInformation_012345678901234567890123
      567890123456789012345678901234567890123456789012345678901234567
      901234567890123456789012345678901234567890123456789012345678901
      345678901234567890123456789012345678901234567890123456789
 V442-CRTKN=BXLS001
 V482-WLM-INFO=DDFBAT:1:3:1
 V445-G9378921.D8D9.CBBFAC100F80=621 ACCESSING DATA FOR
  (  1)::9.55.137.33
 V447--INDEX SESSID           A ST TIME
 V448--(  1) 38420:55513      W R2 1321318355007
```

DB2 **DSNV436I** message contains detail output from the **DISPLAY THREAD** command, which is part of the **DSNV401I** message. If a thread is processing an SQL statement, the output includes the following information about the SQL statement and the program that contains the statement.

Example 9-49 illustrates the output of the **-DIS THD(*) DETAIL** command while a Java application was executing a SQL statement. This results in the inclusion of message V436 in the output.

Example 9-49 DIS THD(*) DETAIL and message V436

```
-DB1A DIS THD(*) DET
DSNV401I  -DB1A DISPLAY THREAD REPORT FOLLOWS - DSNV402I  -DB1A ACTIVE THREADS - 717
NAME     ST A   REQ ID           AUTHID   PLAN     ASID TOKEN
SERVER   RA *     2 db2jcc_appli DB2R6    DISTSERV 00A2   839
 V437-WORKSTATION=9.55.137.146
      USERID=db2r6
      APPLICATION NAME=db2jcc_application
 V441-ACCOUNTING=JCC04170955.137.146
 V436-PGM=NULLID.SYSLH200, SEC=1, STMNT=0, THREAD-INFO=DB2R6:9.55.137.
      146:db2r6:db2jcc_application:DYNAMIC:59:*:<BRXLS_APPCRIS>
 V442-CRTKN=BRXLS_APPCRIS
 V482-WLM-INFO=DDFDEF:2:4:*
 V445-G9378992.CB66.CBF19D605478=839 ACCESSING DATA FOR
  (  1)::9.55.137.146
 V447--INDEX SESSID           A ST TIME
 V448--(  1) 38420:52070        S2 1325311562154
```

In V436 message, the section **THREAD-INFO** provides information about the thread presented in a colon-delimited list that contains the following segments:

- The primary authorization ID that is associated with the thread.
- The name of the user's workstation.
- The ID of the user.
- The name of the application.
- The statement type for the currently executing statement: dynamic or static.
- The statement identifier for the currently executing statement, if available. The statement identifier can be used to identify the particular SQL statement.
  - For static statements, the statement identifier correlates to the **STMT_ID** column in the **SYSIBM.SYSPACKSTMT** table.
  - For dynamic statements, the statement identifier correlates to the **STMT_ID** column in the **DSN_STATEMENT_CACHE_TABLE** table
- The name of the role that is associated with the thread.
- The correlation token that can be used to correlate work at the remote system with work that runs at the DB2 subsystem. The default correlation token, if available, is enclosed in < and > characters, and contains three components, which are separated by periods:
  - A 3 - 39 character IP address
  - A 1 - 8 character port address
  - A 12 character unique identifier

Example 9-50 shows the output of **-DIS THD(*) DETAIL** for the same scenario, with the original application still running but not executing a query in DB2. This results in V436 not be presented in the output.

Example 9-50 DIS THD(*) DETAIL and message V436 missing

```
-DB1A DIS THD(*) DET DSNV401I  -DB1A DISPLAY THREAD REPORT FOLLOWS - DSNV402I  -DB1A ACTIVE
THREADS - 721
NAME     ST A   REQ ID           AUTHID   PLAN     ASID TOKEN
SERVER   RA *     3 db2jcc_appli DB2R6    DISTSERV 00A2   839
```

```
V437-WORKSTATION=9.55.137.146
     USERID=db2r6
     APPLICATION NAME=db2jcc_application
V441-ACCOUNTING=JCC041709.55.137.146
V442-CRTKN=BRXLS_APPCRIS
V482-WLM-INFO=DDFDEF:2:4:20
V445-G9378992.CB66.CBF19D605478=839 ACCESSING DATA FOR
 (  1)::9.55.137.146
V447--INDEX SESSID           A ST TIME
V448--(  1) 38420:52070      N R2 1325311562639
```

9.2 Cancel thread and cancel SQL statement improvements

This section discusses the following improvements for distributed application accessing DB2 11 for z/OS:

- Changes in Cancel DDF thread
- Changes in SQL statement interruption processing

9.2.1 Changes in Cancel DDF thread

The DB2 **CANCEL THREAD** command cancels processing for specific local or distributed threads. The DDF option is used to identify distributed threads for which you want to cancel processing.

In previous versions of DB2 for z/OS, the DDF cancel command and the SQL Cancel command might not work to cancel hung threads or interrupt long running SQL statements. DB2 11 for z/OS introduces the following improvements:

- Enhance the DDF cancel command to use a new z/OS function to terminate a preemptable SRB
- Remove the restrictions preventing the SQL Cancel from interrupting any long running SQL statement

The DB2 for z/OS **CANCEL THREAD** command is a reactive command by design. A DB2 thread is flagged only as being canceled. The thread processing continues until it reaches a cancel detection point where the thread reacts by abnormally terminating itself. These cancel detection points are numerous and strategically distributed in the DB2 code.

This reactive cancel behavior is usually sufficient and successful, but there are cases where the continued processing of the thread might be such that a cancel detection point might not be encountered in a timely manner, or not at all, including in the following cases:

- Relatively tight loops in DB2 processing due to the extensive processing nature of the a SQL statement
- Relatively tight loops in DB2 processing due to a DB2 logic error

In these cases, the reactive nature of the DB2 **CANCEL THREAD** command is ineffective and a more proactive cancel behavior would be more reliable.

To satisfy this requirement, a new z/OS function allows to terminate DBAT related SRB and Enclave processing in a way that allows for DB2 functional recovery. A new z/OS **CALLRTM TYPE=SRBTERM** service is provided in z/OS 1.13 to allow the DB2 **CANCEL THREAD** command processing to proactively cancel the thread when it is executing under an SRB.

Important: z/OS **CALLRTM TYPE=SRBTERM** service is provided in z/OS 2.1 or z/OS 1.13 retrofitted through APAR OA39392.

To allow for this, the **CANCEL THREAD** command in DB2 11 includes a new **FORCE** option. Example 9-51 shows the syntax of the **CANCEL THREAD** command in DB2 11. Note the addition of the new **FORCE** option.

Example 9-51 CANCEL THREAD command in DB2 11

```
>>-CANCEL--+-THREAD(token)-----------+--+------+--+-------+----->
           '-DDF THREAD(-+-luwid-+-)-'  '-DUMP-'  '-LOCAL-'
                         '-token-'

>--+-----------+--+-------+-----------------------------------><
   '-NOBACKOUT-'  '-FORCE-'
```

Use the **FORCE** option to instruct DB2 to attempt to purge the thread of a remote connection in the DB2 server. The **FORCE** option is accepted only after a request to **CANCEL THREAD** is issued without the **FORCE** option.

Attention: The **FORCE** option can potentially affect theDB2 subsystem. Use it to cancel threads that impact the DB2 subsystem and cannot be canceled without **FORCE**.

Important: Even with the **FORCE** option, sensitive processing is still shielded to protect the subsystem. In these situations, **FORCE** might not work as expected.

You can use the **DISPLAY THREAD** command to display information by location. Example 9-52 shows the results of the **DISPLAY THREAD(*) LOCATION(*)** command.

Example 9-52 -DISPLAY THREAD(*) LOCATION(*) command

```
DSNV401I  -DB1A DISPLAY THREAD REPORT FOLLOWS -
DSNV402I  -DB1A ACTIVE THREADS -
NAME     ST A   REQ ID           AUTHID   PLAN     ASID TOKEN
SERVER   RA *     4 db2jcc_appli DB2R1    DISTSERV 0130    60
 V437-WORKSTATION=9.55.137.139
      USERID=db2r1
      APPLICATION NAME=db2jcc_application
 V442-CRTKN=::9.55.137.139.52107.CBC10606A950
 V445-G937898B.CB8B.CBC10606A950=60 ACCESSING DATA FOR
  ::9.55.137.139
DISPLAY ACTIVE REPORT COMPLETE
DSN9022I  -DB1A DSNVDT '-DISPLAY THREAD' NORMAL COMPLETION
***
```

In this example, the thread token that is assigned to the thread is 60, as shown under the **TOKEN** keyword. The **CANCEL DDF THREAD** command syntax accepts either a thread token or a thread **luwid** as input, as shown in Example 9-53.

Example 9-53 CANCEL DDF THREAD command syntax

```
-CANCEL DDF THREAD (token or luwid)
```

As from DB2 11, the **CANCEL DDF THREAD** commands accepts the **FORCE** option. For this example, when using the thread token as input parameter, this command can be written as shown in Example 9-54.

Example 9-54 CANCEL DDF THD FORCE example

```
-CAN DDF THD(60) FORCE
```

Note: The **CANCEL THREAD** command has no effect if the thread is not active or suspended in DB2.

Example 9-55 shows the DB2 feedback after the execution of this command.

Example 9-55 DB2 11 new message DSNV519I

```
DSNV519I  -DB1A DSNLCNCL CANCEL THREAD COMMAND WITH FORCE OPTION FOR
'60' HAS COMPLETED WITH RETURN CODE X'0001'
***
```

DB2 11 introduces the **DSNV519I** message. This message is provided when the **CANCEL THREAD FORCE** command is issued and indicates the success or failure of the command through a return code. That is: to determine the successful or failure of this command, you have to interpret the return code provided by **DSNV519I**.

Example 9-56 shows the structure of the message **DSNV519I**.

Example 9-56 Structure of DB2 message DSNV519I

```
CANCEL THREAD COMMAND WITH FORCE OPTION FOR token-id HAS COMPLETED WITH RETURN CODE return-code
```

The **DSNV519I** message provides this information:

token-id	Either a thread identifier or a logical unit of work identifier (**luwid**) returned from the **DISPLAY THREAD** command
return-code	A numeric value that indicates the success or failure of the **CANCEL THREAD** command

The **return-code** can use the following possible values:

X'0000'	The **CANCEL THREAD** command successfully completed.
X'0001'	The **CANCEL THREAD** command was not accepted. The **FORCE** option is not allowed until a **CANCEL THREAD** without the **FORCE** option is first attempted.
X'0002'	The **CANCEL THREAD** command was not accepted. The **CANCEL THREAD** command with the **FORCE** option for the same token-id cannot be repeated.
X'0003'	The **CANCEL THREAD** command was not accepted. The token-id cannot be found.
X'0004'	The **CANCEL THREAD** command was not accepted. The token-id is associated with a DDF disconnected DBAT on the DB2 server.

Example 9-55 receives an X'0001' return code when executing the **CANCEL THD** command in Example 9-54. This return code is a consequence of using the **FORCE** option before executing a non-**FORCE** command. At this point, no action against the target thread has been performed by DB2. In this scenario the only next option to cancel this tread is then to execute a non-**FORCE CANCEL THD** command as shown in Example 9-57.

Example 9-57 CANCEL DDF THREAD command

```
-CAN DDF THD(60)
```

Example 9-58 shows the DB2 feedback on the execution of this command.

Example 9-58 CANCEL THREAD command output example

```
DSNL010I  -DB1A DDF THREAD '60' HAS BEEN CANCELLED
***
```

In this example, the thread was effectively and immediately cancelled. This can be confirmed by a **DISPLAY THREAD** command, or by inspecting the DB2 MSTR address space feedback. Example 9-59 shows how the termination of the thread is reported in the MSTR address space of this example DB2 subsystem.

Example 9-59 Cancelled thread: DB2 MSTR feedback

```
20.24.21 STC06973  DSNL027I  -DB1A SERVER DISTRIBUTED AGENT WITH  220
   220                        LUWID=G937898B.CB8B.CBC10606A950=60
   220
   220             THREAD-INFO=DB2R1:9.55.137.139:db2r1:db2jcc_application:*:*:*:<::9.55
   220             .137.139.52107.CBC10606A950>
   220                        RECEIVED ABEND=04E
   220                        FOR REASON=00D3001A
20.24.21 STC06973  DSNL028I  -DB1A G937898B.CB8B.CBC10606A950=60  221
   221                        ACCESSING DATA FOR
   221                          LOCATION ::9.55.137.139
   221                          IPADDR ::9.55.137.139
20.24.21 STC06973  DSNL511I  -DB1A DSNLIENO TCP/IP CONVERSATION FAILED  222
   222                        TO LOCATION ::9.55.137.139
   222                        IPADDR=::9.55.137.139 PORT=52107
   222                        SOCKET=SENDMSG RETURN CODE=3448 REASON CODE=00000000
```

The 00D3001A **reason code** indicates that a **CANCEL DDF THREAD** command naming a distributed thread caused the thread to be terminated. If the thread not being cancelled because of the reason discussed previously in this section, a **CANCEL DDF THREAD** option **FORCE** would be accepted by DB2.

9.2.2 Changes in SQL statement interruption processing

Prior to DB2 for z/OS Version 8, the only way to interrupt SQL statement processing that was executing on behalf of a remote application was for the remote application to terminate its connection to the DB2 for z/OS server. This interrupted the SQL statement by terminating the entire DB2 for z/OS server thread (DBAT) and all SQL in the transaction were also aborted.

To allow applications to remain connected to the DB2 for z/OS sever, DB2 for z/OS V8 introduced the ability to interrupt the operation of individual SQL statements. DB2 returns an SQLCODE indicating that the specific SQL statement was canceled, while maintaining the connection with the remote application and the effects of all previous SQL in the transaction.

Application driver environments typically have the following property settings that determine which form of SQL Interruption are used:

- To interrupt the SQL statement
- To interrupt the entire connection

The default client driver behavior is the more granular approach to interrupt just the SQL statement, as opposed to terminating the connection.

For example, the **interruptProcessingMode** property specifies the behavior of the IBM Data Server Driver for JDBC and SQLJ when an application executes the **Statement.cancel method**. Possible values are:

- DB2BaseDataSource.INTERRUPT_PROCESSING_MODE_DISABLED (0)

  Interrupt processing is disabled. When an application executes **Statement.cancel,** the IBM Data Server Driver for JDBC and SQLJ does nothing

- DB2BaseDataSource.INTERRUPT_PROCESSING_MODE_STATEMENT_CANCEL (1)

  This is the default value. When an application executes **Statement.cancel**, the IBM Data Server Driver for JDBC and SQLJ cancels the currently executing statement. If the data server does not support interrupt processing, the driver throws an **SQLException**.

- DB2BaseDataSource.INTERRUPT_PROCESSING_MODE_CLOSE_SOCKET (2)

  When an application executes **Statement.cancel**, the driver drops the underlying socket

Example 9-60 shows a portion of a JCC trace for a Java application connecting to a DB2 11 for z/OS server.This example highlights the default **queryTimeoutInterruptProcessingMode=1** property value. This value indicates that the driver is working with interrupt processing.

Example 9-60 JCC trace and the default interrupt processing mode

```
[jcc] pureQuery present = false
[jcc] END TRACE_DRIVER_CONFIGURATION
[jcc] BEGIN TRACE_CONNECTS
[jcc] Attempting connection to redbook8:38420/DB1A
[jcc] Using properties: { maxStatements=0, currentPackagePath=null, currentLockTimeout=-2147483647,
timerLevelForQueryTimeOut=0, optimizationProfileToFlush=null, timeFormat=1, monitorPort=0, sendCharInputsUTF8=0,
...
currentSchema=null, CR_LOCKBLOB=null, traceLevel=-1, enableRowsetSupport=0, clientDebugInfo=null, dataSourceName=null,
enableAlternateServerListFirstConnect=0, maxRetriesForClientReroute=-1, fetchSize=-1, queryDataSize=0,
queryTimeoutInterruptProcessingMode=1, alternateGroupServerName=null, clientRerouteAlternateServerName=null,
DBTEMP=/tmp, enableT2zosLBF=0, SUBQCACHESZ=10, ssid=null, maxConnCachedParamBufferSize=1048576,
fullyMaterializeInputStreamsOnBatchExecution=0, alternateGroupPortNumber=null,
...
defaultIsolationLevel=2, deferPrepares=true, currentDegree=null, DUMPMEM=null, memberConnectTimeout=0 }
[jcc] END TRACE_CONNECTS
```

The SQL statement Interruption technique is the more granular and preferred operation with respect to remote applications. Nevertheless, the statement interruption processing is not completely reliable in some scenarios. As a consequence, there is a strong recommendation for users to use the more drastic, but more effective, INTERRUPT_PROCESSING_MODE_CLOSE_SOCKET method that terminates the connection.

However, it is often difficult for users to modify the client driver property to use the more reliable form. As a consequence, until the DB2 SQL statement Interrupt processing can be made more reliable, DB2 compensates by changing its SQL Interruption processing to behave as though the connection had been terminated.

This way, SQL statement interruption is more reliable at a DB2 for z/OS server system, but at the expense of terminating the connection with the remote application. This condition should be handled by the application.

In DB2 11, when DB2 receives a DRDA SQL Interrupt from a remote client, it closes the connection and terminate the thread under which the statement is running, instead of interrupting just the statement and returning an **SQLCODE=-952**.

Depending on whether the remote client has enabled Sysplex Workload Balancing (**sysplexWLB**) and if the application has resources that need to persist across transactions preventing the connection from being reused by a different application at the end of a transaction, the remote application might receive the following SQLCODEs:

-30081 An application gets this SQLCODE if the client does not support **sysplexWLB** or the connection cannot be reused. The client has to reconnect to DB2 before executing the application again

-30108 An application gets this SQLCODE if the client supports **sysplexWLB** and the connection can be reused but the client cannot retry the failed statement seamless to the application. The client, however, reconnects to DB2 before returning the SQLCODE to the application so that the application can retry the failed transaction immediately

0 An application can get this SQLCODE if the client supports **sysplexWLB**, the connection can be reused and the client seamlessly retried the failed statement which completed successfully

9.3 Continuous block fetching

DB2 11 introduces package-based continuous block fetch. It can improve performance for retrieval of large, read-only result sets from a remote DB2 for z/OS server.

Important: DB2 11for z/OS provides improved performance for distributed applications that return large result sets

Like the previously existing SQL-based continuous block fetch, package-based continuous block fetch causes fewer messages to be transmitted from the requester to retrieve the entire result set. However, package-based continuous block fetch is easier to configure. It requires only that you bind your applications with the new **DBPROTOCOL(DRDACBF)** option. You do not need to modify your applications or set subsystem parameters to indicate the maximum number of blocks to be returned for a remote request. This change is available in NFM and requires `APPLCOMPAT = V11R1` to be set.

The new package-based continuous block fetch is more efficient than SQL-based continuous block fetch. With package-based continuous block fetch, the requester opens a secondary connection to the DB2 server for each read-only cursor. The DB2 server returns extra query blocks until the all rows for the cursor have been retrieved. When the cursor is closed, the secondary connection is implicitly closed.

Figure 9-3 shows a representation of how SQL Based Continuous Block Fetch works. Using this technique, DB2 can send numerous query blocks per request. A single connection is used for all SQL. The implication for the single connection is that other SQL, outside of the cursors, cannot use the connection while the cursor driven blocks are using the connection.

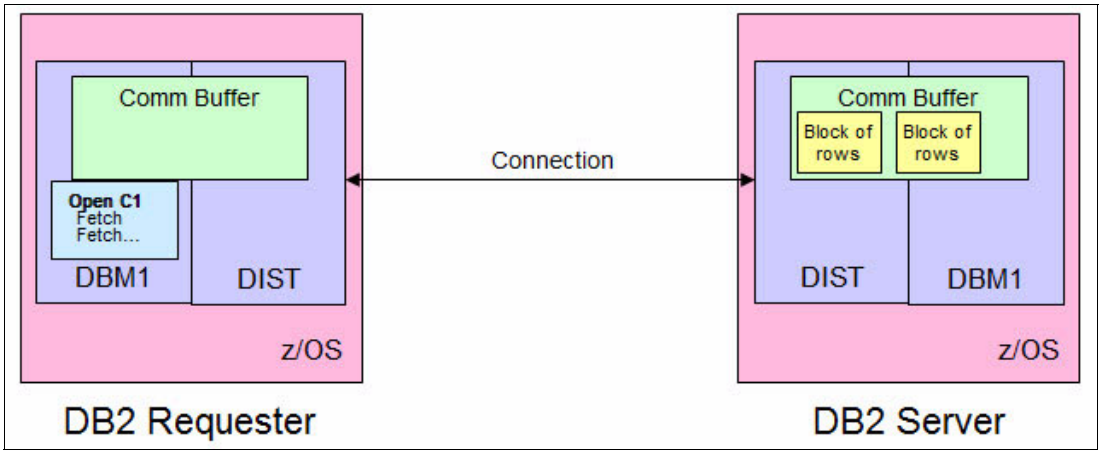


Figure 9-3 SQL based continuous block fetch

Figure 9-4 shows a representation of the Package Based Continuous Block Fetch, introduced in DB2 11. With this method, query blocks flow on a secondary connection until the cursor is exhausted. As a consequence, the network latency is significantly improved. When the result set or cursor is exhausted, the DB2 server terminates the connection and the thread is immediately pooled.

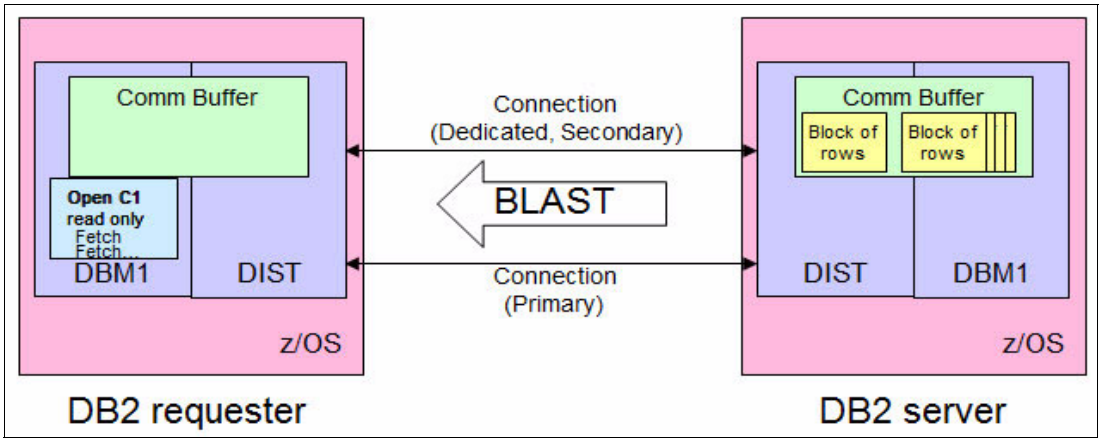


Figure 9-4 Package based continuous block fetch

Example 9-61 shows the changes to the SPUFI **REBIND** panels to indicate the **DRDACBF** option.

Example 9-61 SPUFI panel DSNEBP11, defaults for REBIND PACKAGE

```
DSNEBP11               DEFAULTS FOR REBIND PACKAGE          SSID: DB1A
COMMAND ===>

Change default options as necessary.
----------------- Use the UP/DOWN keys to access all options ------------------
                                                                More:   -
                                              UNICODE, or ccsid)
11  OPTIMIZATION HINT ....... ===> SAME     >  (SAME or 'hint-id')
12  IMMEDIATE WRITE.......... ===> SAME           (SAME, NO, YES,
```

```
                                                      or INHERITFROMPLAN)
13  DBPROTOCOL ............... ===> SAME        (SAME, DRDA, DRDACBF)
14  DYNAMIC RULES ............ ===> SAME        (SAME, RUN, BIND,
                                                 DEFINERUN, DEFINEBIND,
                                                 INVOKERUN or INVOKEBIND)
15  PLAN MANAGEMENT ......... ===> DEFAULT     (DEFAULT, BASIC, EXTENDED, OFF)
16  ACCESS PATH REUSE ........ ===> DEFAULT     (DEFAULT, ERROR, NONE, or WARN)
17  ACCESS PATH COMPARISON .. ===> DEFAULT     (DEFAULT, ERROR, NONE, or WARN)
18  ACCESS PATH RETAIN DUPS . ===> DEFAULT     (DEFAULT, NO, or YES)
19  SYSTEM_TIME SENSITIVE ... ===> SAME        (SAME, NO, or YES)
20  BUSINESS_TIME SENSITIVE . ===> SAME        (SAME, NO, or YES)
21  ARCHIVE SENSITIVE ........ ===> SAME        (SAME, NO, or YES)
22  APPLICATION COMPATIBILITY ===> SAME        (SAME, V10R1, or V11R1)
------------------------------------------------------------------------------
PRESS:  ENTER to continue   UP/DOWN to scroll   RETURN to EXIT
```

Example 9-62 shows an example of **REBIND PACKAGE** command using the **DBPROTOCOL(DRDACBF)** option.

Example 9-62 REBIND PACKAGE with DBPROTOCOL(DRDACBF) option

```
//DRDACBF1 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//STEPLIB  DD DSN=DB1AT.SDSNLOAD,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
 DSN SYSTEM(DB1A)
 REBIND PACKAGE(DSN8BH11.DSN8BC3) DBPROTOCOL(DRDACBF)
/*
```

Example 9-63 shows the output of the execution of this **REBIND** command.

Example 9-63 REBIND output

```
1READY
  DSN SYSTEM(DB1A)
 DSN
  REBIND PACKAGE(DSN8BH11.DSN8BC3) DBPROTOCOL(DRDACBF)
 WARNING, ONLY IBM-SUPPLIED COLLECTION-IDS SHOULD BEGIN WITH  "DSN"
 WARNING, ONLY IBM-SUPPLIED PACKAGE-IDS SHOULD BEGIN WITH  "DSN"
 DSNT254I  -DB1A DSNTBRB2 REBIND OPTIONS FOR
            PACKAGE = DB1A.DSN8BH11.DSN8BC3.()
            ACTION
            OWNER         SYSADM
            QUALIFIER     SYSADM
            VALIDATE      RUN
            EXPLAIN       NO
            ISOLATION     CS
            RELEASE
            COPY
            APREUSE
            APCOMPARE
            APRETAINDUP   YES
            BUSTIMESENSITIVE YES
            SYSTIMESENSITIVE YES
            ARCHIVESENSITIVE YES
            APPLCOMPAT V11R1
 DSNT255I  -DB1A DSNTBRB2 REBIND OPTIONS FOR
             PACKAGE = DB1A.DSN8BH11.DSN8BC3.()
            SQLERROR      NOPACKAGE
            CURRENTDATA   YES
```

```
            DEGREE        1
            DYNAMICRULES
            DEFER
            REOPT         NONE
            KEEPDYNAMIC   NO
            IMMEDWRITE    INHERITFROMPLAN
            DBPROTOCOL    DRDACBF
            OPTHINT
            ENCODING      EBCDIC(00037)
            PLANMGMT      EXTENDED
            PLANMGMTSCOPE STATIC
            CONCURRENTACCESSRESOLUTION
            EXTENDEDINDICATOR
            PATH
DSNT232I  -DB1A SUCCESSFUL REBIND FOR
            PACKAGE = DB1A.DSN8BH11.DSN8BC3.()
DSN
END
READY
END
```

Package-based continuous block fetch provides a performance advantage for a DB2 for z/OS application with the following characteristics:

- The application queries only remote sites
- The application does not contain INSERT, UPDATE, DELETE or MERGE statements
- No statement in the application creates a unit of recovery on the remote site. This situation results in an SQL error when the application package is bound for package-based continuous block fetch

Table 9-7 shows the results observed during IBM conducted preliminary internal lab performance tests. In this table, results are expressed as delta %.

Table 9-7 Preliminary internal lab performance results

Delta %	Class 1 Elapsed Time	Class 2 Elapsed Time	Class 1 CPU Time	Class 2 CPU Time
Server	-29,5	-8,3	-20	-5,8
Requester	-31,1	-31,1	-13	-13

9.4 Support for global variables

This section describes the use of global variables.

DB2 supports the following types of distributed protocols an application can use to execute a statement on a remote system:

- When the application explicitly connects to the remote server, a*pplication-directed access*
- When the application implicitly connects to the remote server by using three-part name references, *system-directed access*

Application-directed access

When using application-directed access, the location and platform of the system executing the statement is known to the application. The application, which had connected to DB2 and

possibly run some statements, now connects to another location. In doing so, the application must then establish the session environment it needs to run any of its statements while connected to this new location. Thus, any session information, special registers and user-defined session global variables, are maintained between the application itself and the current serving location. If the application decides to temporarily move away from this location to another, the session information would be preserved at this serving location until the connection between the requesting and serving locations was terminated.

System-directed access

When using system-directed access, the location and platform of the system executing the statement is transparent to the application. The application connects to DB2 and runs the statement as though executing on the local system. You can create aliases for remote objects which allows you to reference remote objects without any change to your application. Users access them with the same syntax and application environment as local objects. When DB2 parses the statement and determines the object is on a remote system, DB2 implicitly connects to the system and forwards the statement to the other system.

For example, an application connects to the EAST location and queries the DEPT table. If the DEPT table is moved to the WEST location, you can create an alias on the EAST location for the DEPT table on the WEST location, the application can continue to issue the query without change. Even if the application directly references the DEPT table at the WEST location by using an explicit 3-part name reference, the application is still just referencing a table/view object without regard to the fact that communications have to be established to the WEST location to access the DEPT table

To maintain location transparency, DB2 ensures the application execution environment is maintained across systems. Special registers used to store information that can be referenced in SQL statements and set by the application are maintained by DB2 on both the local and remote systems transparent to the application. For example, if an application issues the **SET CURRENT PRECISION** statement to assign a value to the **CURRENT PRECISION** special register, The **CURRENT PRECISION** register value is propagated to the remote system prior to executing the statement. When the statement is executed, special register settings set by the application are used to process the statement on the other system.

To maintain location transparency for user-defined session global variables, DB2 ensures the global variables and their values set by the application context are maintained and persist across systems. Because any statement executed locally can reference or alter any user-defined session global variable, an instance of any global variable with its last updated value must be created on the remote system prior to executing the statement. Thus, global variables values are maintained by DB2 on both the local and remote systems for the application process.

For example, if an application sets some global variables and then executes a system-directed SQL statement, the global variable settings are propagated to the remote system. Prior to the execution of the statement, the remote system uses the global variable definitions and values sent from the requesting system to create instances of the user-defined global variables. The statement is then executed on the remote system.

If any changes are made to user-defined session global variables on the remote system by the just executed statement, the definitions and updated values of the changed user-defined session global variables are returned to the requesting system. The requesting system then updates any already instantiated user-defined session global variables or creates instances of newly set user-defined session global variables. For this all to work correctly, the definitions of the global variables, that is CREATE VARIABLE statements, must be identical on both the requesting and serving systems or SQLCODE -30045 is issued.

Important: The global variable definitions must exist at every remote location that is accessed by an application, and the definitions must be identical. Otherwise, the application receives SQLCODE -30045.

Example 9-64 shows the structure of the SQL error code -30045.

Example 9-64 Structure of SQLCODE -30045

```
EXECUTION FAILED BECAUSE THE DEFINITION OF OBJECT object-name OF TYPE object-type BEING
ACCESSED AT server-name-1 DIFFERS FROM THE DEFINITION OF THE OBJECT AT server-name-2
```

As a result of the error reported by the SQL error code -30045, the statement cannot be processed. Refer to the IBM documentation “DB2 for z/OS Codes” for more details about this SQL error code.

Finally, an application can intermix application-directed and system-directed statements on the same connection. How session information is maintained is dependent on the persistence of any connection created between the two locations. Mixing protocols that utilize user-defined session global variables can result in unexpected behavior and is prevented. If an application intermixes distributed protocols where a statement first used application-directed protocols and was then followed by a statement using system-directed protocols, an SQLCODE -30047 exception is generated.

For example, an application issues a statement that uses an ALIAS to refer to a table at another location and then calls a procedure which issues a CONNECT statement and issues statements to the same location, the connection has executed statements using both protocols.

Persistence of connections is governed by the SQL **RELEASE** statement and how the plan was bound as follows:

- For z/OS applications, the **DISCONNECT** bind plan option determines when connections are dropped during commit operations. The default value is **EXPLICIT**. If **EXPLICIT** is used, the application must issue a **RELEASE** statement prior to a **COMMIT** to have a connection dropped during commit processing.

  If no **RELEASE** statements are issued, the connections persist until the application ends. Another possible option is **AUTOMATIC**. When the **AUTOMATIC** bind option is in control, all connections to remote servers from the requester is dropped when a COMMIT is processed. The final value of the **DISCONNECT** bind option is **CONDITIONAL**. It behaves similarly to **AUTOMATIC** with one exception. If a **WITH HOLD** cursor is still open against a location, the connection is not dropped when a **COMMIT** is processed. If the application eventually closes the cursor, a subsequent **COMMIT** then causes the connection to be dropped.
- For IBM DB2 Connect™ applications, the **EXPLICIT** behavior is used and cannot be changed.
- The above connection persistence rules apply whether or not the application was prepared (not bound) under connect type-1 or connect type-2 rules.

Based on the above connection persistence rules, SQL statements that are processed at a location under application-directed protocols behave as follows:

- All statement references to user-defined session global variables, both input and output, refer to the user-defined session global variables at the serving location. User-defined session global variables at the requesting location are neither updated nor referenced as a result of either dynamic or static SQL statements in this scenario.

- The content of the user-defined session global variables at the serving location persists until the connection is dropped. Whether connect type-1 or connect type-2 rules are used, the connection type is not be a factor and doesn't affect when a connection is dropped. Any subsequent references to user defined session global variables at the location where the connection had been dropped cause the user defined session global variables to be instantiated with default values again; otherwise, subsequent references to the user-defined session global variables use the values last updated.

Based on these connection persistence rules, SQL statements that are processed using system-directed protocols behave as follows:

- Contents of the user-defined session global variables that are instantiated at the requesting location are sent to the serving location, such that the same user-defined session global variable values are used during the processing of the SQL statement at both locations.
- User-defined session global variables that are the target of the output process from the SQL statement are made to both the user-defined session global variables at both locations.
- For static SQL statements referencing objects and user-defined session global variables that use system-directed access, a package must be bound at both locations. If at the time of the static bind, different definition exists on the requester location and server location for the same user defined session global variable, the executable runtime structures are generated differently. At execution time, DB2 issues an -30045 exception to indicate a mismatch of user-defined session global variable definitions if referenced on both sites.
- For dynamic statements referencing objects and user-defined session global variables at remote locations through system-directed distributed processing, there is no corresponding restriction as static statements. However, if the SQL statement uses a user-defined session global variable and DB2 determines there is a definition mismatch, DB2 issues the -30045 error.

If the application connected to DB2 issues statements that use both system-directed protocols and application directed protocols to the same location, they share the same connection. For example, an application connects to the HDQ location and issues a statement that uses an ALIAS to query a table at the MFG location. The next statement calls a procedure on the HDQ location. If procedure issues a **CONNECT** statement to the MFG location, the same connection is used. Mixing protocols which use user-defined global variables can cause nondeterministic results. Mixing distributed protocol statements on the same connection that use user-defined session variables is prevented by DB2 issuing an SQLCODE -30047 exception.

For DB2 Connect clients that are **sysplexWLB** enabled which performs transaction level load balancing across a data sharing group, connections persist across different members of the data sharing group. To support user defined session global variables an upgrade of the client is needed with this feature enabled. DB2 returns changed user-defined session global variables to the client driver to allow the client to replay them when the application connection is transparently moved to a different member of the data sharing group.

Global variables in SQL statements referencing remote servers

In the case of static statements referencing a 3-part remote object, or a statically bound statement executed when the **CURRENT SERVER** is a remote server, DB2 marks the current section as a distributed-section at the requester site, and all SQL processing occurs at the target server site.

This means, the created global variables at the local requester site are not used in the processing of the SQL statement, because the processing occurs at the server.

DB2 requires the package to exist at both the requester and the server sites for DRDA communication protocol. Thus, the packages on all sites need to be created first using some form of **BIND PACKAGE** command. However, because the bind occurs on different sites, the global variables might not be all created, or if created might not share the same definition nor **DEFAULT** expressions. Incongruous definitions or instantiation of global variables can result in different outcomes when the same SQL statement is executed locally versus remotely

Global variables scope with Thread-Reuse

DB2 Distributed Data Facility (DDF) can employ thread-reuse to enhance performance when multiple connections are made to the DB2 server. If a connection (or thread) in DB2 qualifies for reuse, then it is returned to the reusable thread pool at **COMMIT** or **ROLLBACK**, waiting for the next connection request. The next connection request can be from a different application, or it can be the continuation of the previous application. Because global variables are not affected by **COMMIT** nor **ROLLBACK**, the content must persist across **COMMIT** and **ROLLBACK**, and therefore, across reusable threads.

When a thread is reused for an application that referenced global variables, all instantiated variables are “replayed” for the reused thread such that all values recorded from the previous thread are copied over to the current reused thread. This ensures the persistence of the instantiated global variable across reusable threads.

9.5 Local stored procedure execution improvement

DB2 11 delivers performance optimization for processing stored procedure calls from local ODBC and JDBC applications by improving stored procedure result set processing. This is beneficial for customers who call stored procedures from a local JDBC or ODBC environment, such as WebSphere on z/OS or MessageBroker on z/OS, encapsulating SQL in stored procedures. The enhancements do not require changes to the application and are available in CM.

The enhancements are in the following areas:

- The communication between the ODBC or JDBC/SQLJ driver and DB2 to execute the CALL statement.

  Bundling **CALL** and **DESCRIBE PROCEDURE** and bundling **ALLOCATE CURSOR** and **DESCRIBE CURSOR** to reduce trips from ODBC/JDBC driver to DBM1.
- The communication between the ODBC or JDBC/SQLJ driver and DB2 to return the result set metadata.
- The processing of the result sets returned from the called stored procedure using limited block fetch and progressive streaming (which is better performing than multi-row fetch).
- The communication between ODBC or JDBC/SQLJ driver and DB2 by implicitly closing the result sets at their termination (SQLCODE +100).
- Support of 64bit private variables area for in/out parameters.

  Allows the exchange of parameter larger than 32 KB, such as parameter of data type LOB (with usage of 64-bit DB2VAR for input/output parameters).
- More efficient way to describe stored procedure parameters.

Similar enhancements had been introduced with DB2 10 for local ODBC/JDBC, but not for stored procedures.

Figure 9-5 summarizes the enhancements.

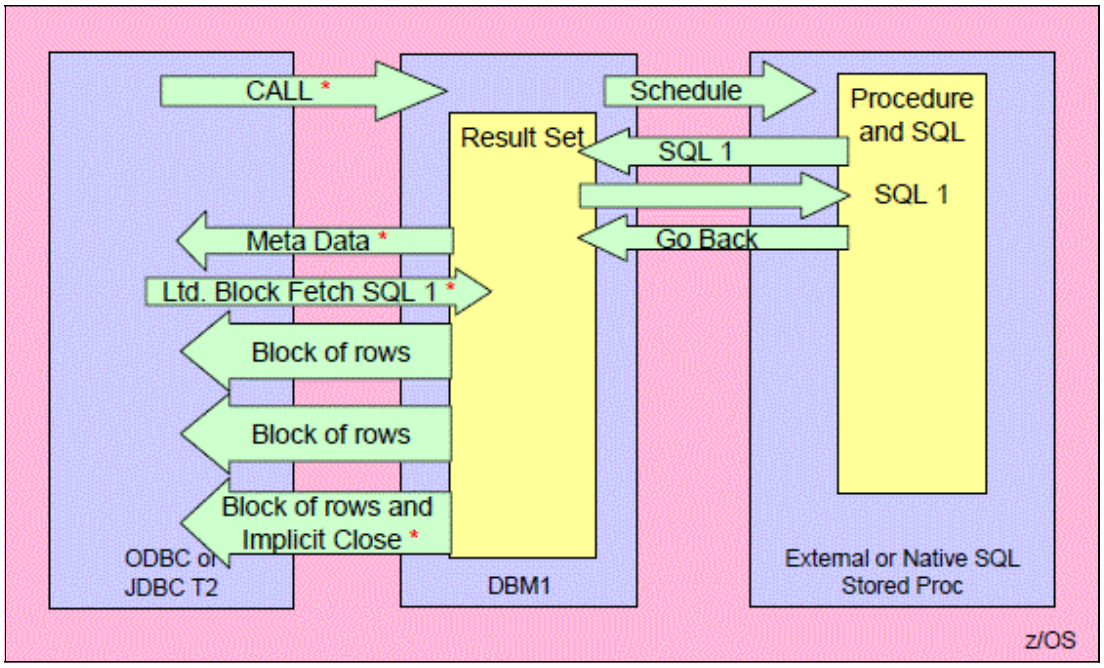


Figure 9-5 Summary of local stored procedure improvements

The ODBC driver supports the optimization of stored procedure result set processing by enabling block fetch through the **LIMITEDBLOCK**, **QUERYDATASIZE**, and **STREAMBUFFERSIZE** keywords in the initialization file data source section. The keyword values are read from the data source stanza following a successful connect.

The LIMITEDBLOCK keyword specifies if the driver is to attempt a block fetch when fetching a result set at the connected data source (server). The acceptable keyword values are 0 and 1:

0	No block fetch.
1 (default)	The driver would attempt a block fetch and return as many rows as can fit in a data block in a single fetch provided that blocking is supported at the server for the result set being fetched. The driver currently does not perform block fetch if any of the columns in the result set is a LOB, XML, or file reference.

QUERYDATASIZE specifies the size of the data block in bytes. The default for **QUERYDATASIZE** is 32 KB (32767). The maximum data block size is 1048575 in 32 KB increments.

STREAMBUFFERSIZE is the threshold value, in bytes (default 1 MB) to return LOB or XML as inline data or as internal token:

- If size of LOB or XML object <= **STREAMBUFFERSIZE**, data returned inline
- If size of LOB or XML object >= **STREAMBUFFERSIZE**, progressive reference returned

The JDBC driver supports stored procedure optimization transparently.

- The **queryDataSize** property also used for stored procedure result sets

  32 KB, up to 1 MB in 32 KB increments
- **FET_BUF_SIZE** (64 KB) keyword can be used to limit rows per buffer

The tests have shown better performance for local ODBC and Type 2 applications that call local stored procedures because of:

- More efficient blocking of data in returned result sets
- More efficient retrieval of LOB and XML result sets
- Reduced traffic for implicit close

9.6 Multi-threaded Java stored procedure environment

DB2 11 adds support for running Java stored procedures in a 64-bit Java virtual machines (JVM). Earlier versions of DB2 run Java stored procedures in 31-bit JVM only, and each JVM can run only one Java stored procedure at a time.

With DB2 10, the behavior is single threaded JVM for Java stored procedures can be summarized as follows:

- WLM stored procedure address space (WLM-SPAS)
- 1 JVM per TCB in WLM-SPAS
  - Large storage footprint per TCB
  - Overhead on starting JVMs
- The recommended **NUMTCB** is 8 or less per WLM application environment
  - NUMTCB used is typically 2-5
- There are performance and scalability implications for Java stored procedures
- They use a 31-bit JVM

With DB2 11, see Figure 9-6, Java stored procedures use multi-threaded JVMs. DB2 11 can concurrently run multiple Java stored procedures in 64-bit JVMs. Therefore, more Java stored procedures can run in a single stored procedure address space than in earlier DB2 versions.

- One 64-bit JVM per WLM-SPAS
  - Less overhead to start JVM
  - Smaller JVM storage footprint
- NUMTCB of 25 or more per WLM application environment
  - Better scalability
- It requires JDK 1.6
  - 64-bit JDK
  - IBM Data Server Driver for JDBC and SQLJ

The multi-threaded JVM executes a new **DSNX9WJM** module in the WLM application environment which is specified on the start-up JCL.

The existing application environments need to be modified or new application environments defined to take advantage of more TCBs (larger **NUMTCB**).

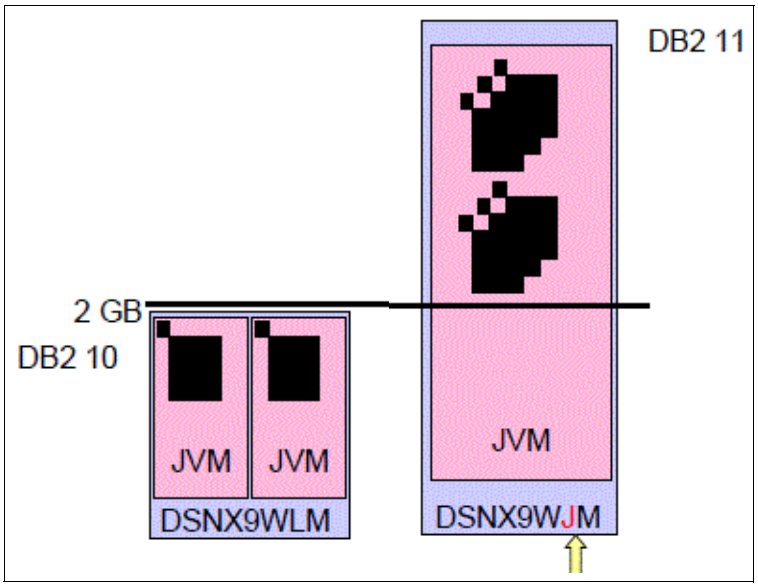


Figure 9-6 Moving to multi-threaded JVM environment

Most existing Java stored procedures can be altered to run in new multi-threaded environment. If the Java stored procedure invokes a native method through JNI calls, the dynamic link library (DLL) for native functions must be compiled and linked in 64-bit mode.

Note: Native non-Java code must be rebuilt and tested for the 64-bit environment.

While in data sharing coexistence, multi-threaded Java stored procedures can be used in CM mode but all members must use new **DSNX9WJM** module for that WLM environment if stored procedures uses native JNI calls.

9.7 ADMIN_COMMAND_MVS stored procedure

DB2 provides stored procedures that you can call in your application programs to perform administrative functions.

You can use the DB2 provided **SYSPROC.ADMIN_COMMAND_MVS** stored procedure to issue the following z/OS commands:

- **QUERY COPYPOOL**
- **LIST COPYPOOL**
- **DB2 START**
- **DB2 STOP**
- **DUMP**
- **DISPLAY WLM**

SYSPROC.ADMIN_COMMAND_MVS (also retrofitted to DB2 10 by APAR PM93773) extends the list to the following available command related stored procedures:

- **ADMIN_COMMAND_DB2**
- **ADMIN_COMMAND_DSN**
- **ADMIN_COMMAND_UNIX**

This stored procedure runs in a WLM-established stored procedures address space, and all of the libraries that are specified in the **STEPLIB DD** statement must be APF-authorized. Example 9-65 shows the syntax for calling **ADMIN_COMMAND_MVS**.

Example 9-65 Syntax CALL ADMIN_COMMAND_MVS

```
>>-CALL--ADMIN_COMMAND_MVS--(----type---,----------------------->

>--+-command_prefix-+-,--+-remote_system-+-,--+-jobname-+-,----->
   '-NULL-----------'    '-NULL----------'    '-NULL----'

>----command---,--+-parameters-+-,--+-subparameters-+-,--------->
                  '-NULL-------'    '-NULL----------'

>--+-wait_timeout-+--return-code,--command_completion_code,----->
   '-NULL---------'

>--+-message-+--)------------------------------------------><
   '-NULL----'
```

The call parameter **type** cannot be NULL, and it accepts these values:

- HSM
- DB2
- DUMP
- WLM

This stored procedure returns the following output parameters:

return-code	Provides the return code from the stored procedure. Possible values are 0,4,8, and 12.
command_completion_code	Indicates the completion status of the command. Possible values are 0,4,8,12 and 16.
message	Contains messages that describe the error that was encountered by the stored procedure.

Tip: Refer to *DB2 11 for z/OS Administration Guide*, SC19-4050 for details about **ADMIN_COMMAND_MVS** parameters.

Execution example: display WLM

To display the WLM application environments using **ADMIN_COMMAND_MVS** you have to specify these parameters:

- Type: Use the value **WLM**
- Command: Use the value **DISPLAY**
- Parameters: Specify either **APPLENV=*name*** or **APPLENV=***

All the other parameters have to be defined as NULL. Figure 9-7 shows an example of calling **ADMIN_COMMAND_MVS** from Data Studio.

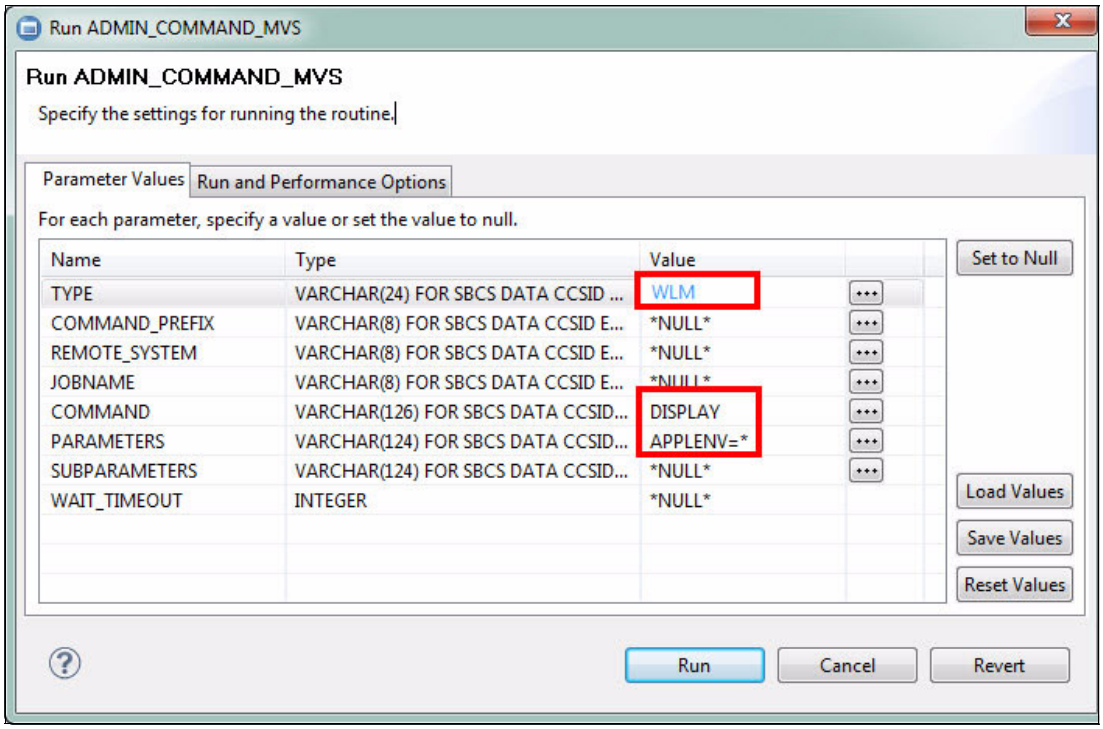


Figure 9-7 Calling ADMIN_COMMAND_MVS from Data Studio

For parameter that are not filled in, use the **Set to Null** button in this dialog box. Otherwise, the execution of the stored procedure fails with return code 12 and **DSNA601I** message.

Example 9-66 is an illustration of the feedback received when using an invalid **COMMAND_PREFIX** parameter.

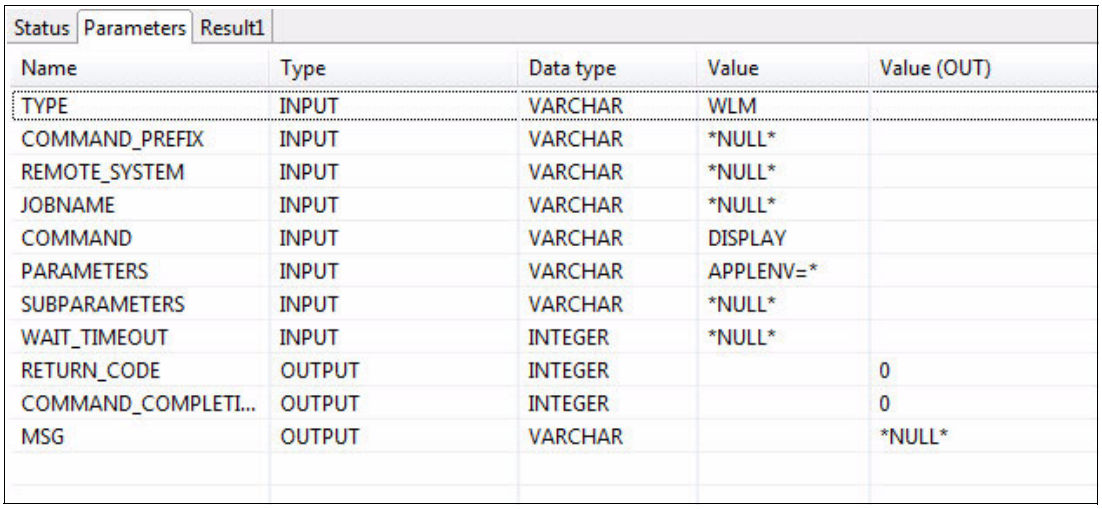
Example 9-66 Message DSNA601I

```
DSNA601I  DSNADMCM THE PARAMETER COMMAND_PREFIX IS NOT VALID. INVALID REASON CODE=5
```

DSNA601I indicates that the parameter specified in the message is not valid. The cause of the invalid error is identified by the INVALID REASON CODE value in the message text. The specified parameter is not valid for the indicated reason, as follows:

- REASON CODE=1: Value is not an acceptable value
- REASON CODE=2: Value is not unique
- REASON CODE=3: Value is null
- REASON CODE=4: Value is blank
- REASON CODE=5: Value is not null
- REASON CODE=6: Value is too long
- REASON CODE=7: Named parameter is not known
- REASON CODE=8: Named parameter is missing

Figure 9-8 shows the parameters panel in Data Studio after successful execution. Note the parameters `RETURN_CODE = 0` and **`COMMAND_COMPLETION_CODE = 0`**. **`MSG`** is `Null`.

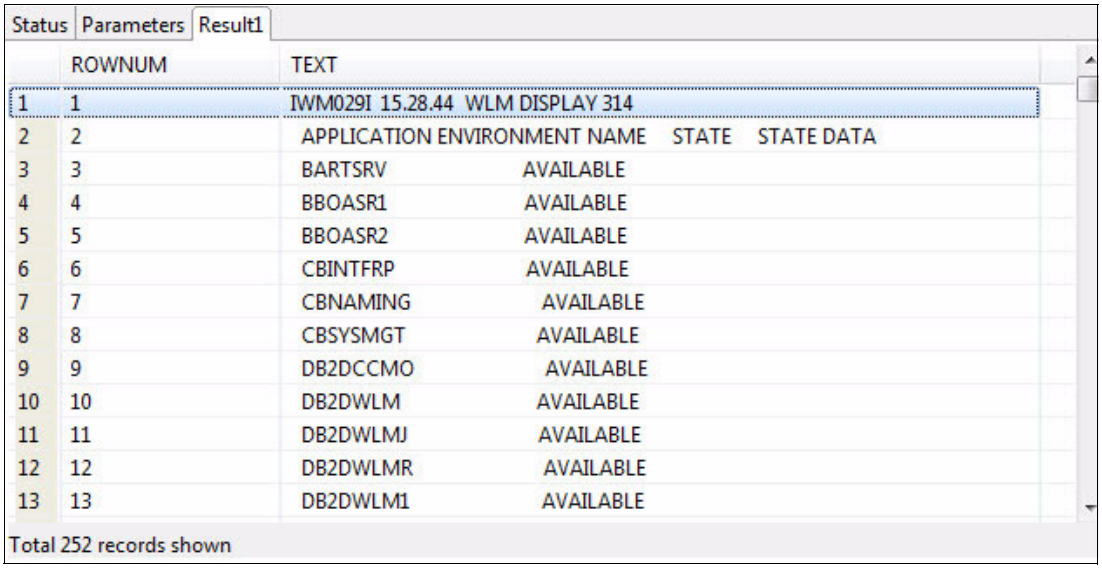


Status | Parameters | Result1

Name	Type	Data type	Value	Value (OUT)
TYPE	INPUT	VARCHAR	WLM	
COMMAND_PREFIX	INPUT	VARCHAR	*NULL*	
REMOTE_SYSTEM	INPUT	VARCHAR	*NULL*	
JOBNAME	INPUT	VARCHAR	*NULL*	
COMMAND	INPUT	VARCHAR	DISPLAY	
PARAMETERS	INPUT	VARCHAR	APPLENV=*	
SUBPARAMETERS	INPUT	VARCHAR	*NULL*	
WAIT_TIMEOUT	INPUT	INTEGER	*NULL*	
RETURN_CODE	OUTPUT	INTEGER		0
COMMAND_COMPLETI...	OUTPUT	INTEGER		0
MSG	OUTPUT	VARCHAR		*NULL*

Figure 9-8 ADMIN_COMMAND_MVS parameters in Data Studio

The actual WLM **`DISPLAY`** results are reported in the Result1 panel, as shown in Figure 9-9.



Status | Parameters | Result1

	ROWNUM	TEXT
1	1	IWM029I 15.28.44 WLM DISPLAY 314
2	2	APPLICATION ENVIRONMENT NAME STATE STATE DATA
3	3	BARTSRV AVAILABLE
4	4	BBOASR1 AVAILABLE
5	5	BBOASR2 AVAILABLE
6	6	CBINTFRP AVAILABLE
7	7	CBNAMING AVAILABLE
8	8	CBSYSMGT AVAILABLE
9	9	DB2DCCMO AVAILABLE
10	10	DB2DWLM AVAILABLE
11	11	DB2DWLMJ AVAILABLE
12	12	DB2DWLMR AVAILABLE
13	13	DB2DWLM1 AVAILABLE

Total 252 records shown

Figure 9-9 Calling ADMIN_COMMAND_MVS: Result1 panel

In addition, the WLM **`DISPLAY`** output command is listed in the system log, as shown in Example 9-67.

Example 9-67 ADMIN_COMMAND_MVS and WLM DISPLAY: system log messages

```
15:52:51.67 STC06093 00000090  ICH70001I DB2R1    LAST ACCESS AT 15:39:32 ON FRIDAY, JULY 26, 2013
15:52:51.70 STC06093 00000290  IEA630I  OPERATOR DSNADMCM NOW ACTIVE,   SYSTEM=SC63    , LU=DSNADMCM
15:52:51.70 DSNADMCM 00000290  DISPLAY WLM,APPLENV=*
15:52:51.73 DSNADMCM 00000090  IWM029I  15.52.51  WLM DISPLAY 337
                337 00000090    APPLICATION ENVIRONMENT NAME     STATE     STATE DATA
                337 00000090    BARTSRV                          AVAILABLE
                337 00000090    BBOASR1                          AVAILABLE
                337 00000090    BBOASR2                          AVAILABLE
                337 00000090    CBINTFRP                         AVAILABLE
```

```
 337 00000090    CBNAMING                        AVAILABLE
...
```

In this example, **STC06093** is the WLM address space that supports the execution of the stored procedure. **DSNADMCM** is the name of the extended MCS console that issue the requested command.

Security considerations

To execute the **CALL** statement, the owner of the package or plan that contains the CALL statement must have one or more of the following privileges:

- The **EXECUTE** privilege on the stored procedure
- Ownership of the stored procedure
- **SYSADM** authority

The load module for **ADMIN_COMMAND_MVS** is named **DSNADMCM**. This name can be used in the definition of some specific RACF resources as a way of increasing security.

The caller of **ADMIN_COMMAND_MVS** must have **READ** access to the RACF **MVS.MCSOPER.*** or to the **MVS.MCSOPER.DSNADMCM** resource profile of the **OPERCMDS** class. RACF perform access checks starting by the most restrictive resource profile. If **MVS.MCSOPER.DSNADMCM** is not defined, RACF checks for **MVS.MCSOPER.***.

You can use the RACF ISPF panels to investigate the access defined on the resource **MVS.MCSOPER.*** following these steps:

1. Open the RACF ISPF main menu to receive the SERVICES OPTION MENU.
2. Select option **2. GENERAL RESOURCE PROFILES**.
3. You are now in the GENERAL RESOURCE PROFILE SERVICES panel. Select option **S. SEARCH**.
4. Press Enter to access to the GENERAL RESOURCE SERVICES - SEARCH panel.
5. Enter OPERCMDS in the CLASS option, as shown in Example 9-68.

Example 9-68 RACF - GENERAL RESOURCE SERVICES panel

```
RACF - GENERAL RESOURCE SERVICES -  SEARCH
  OPTION ===>

  ENTER THE FOLLOWING PROFILE INFORMATION:

     CLASS     ===> OPERCMDS

     PROFILE   ===>
```

6. Press Enter to access the SEARCH FOR GENERAL RESOURCE PROFILES panel. Fill-in MASK1 with MVS, and MASK2 with MCSOPER, as shown in Example 9-69. Press Enter to continue.

Example 9-69 RACF SEARCH FOR GENERAL RESOURCE PROFILES panel

```
RACF - SEARCH FOR GENERAL RESOURCE PROFILES
 COMMAND ===>

 ENTER MASK(S) OR FILTER (OPTIONAL):

    MASK1      ===> MVS
```

```
                                    <= end of data

    MASK2       ===> MCSOPER


                                    <= end of data
```

7. In next panel, SEARCH FOR GENERAL RESOURCE PROFILES, just press Enter to continue.
8. Example 9-70 shows the RACF command output.

Example 9-70 RACF COMMAND OUTPUT, MVS.MCSOPER

```
BROWSE - RACF COMMAND OUTPUT----------------------- LINE 00000000 COL 001 080
 COMMAND ===>                                                 SCROLL ===> CSR
******************************** Top of Data ********************************
MVS.MCSOPER.ABC
MVS.MCSOPER.AOPAOP1C
MVS.MCSOPER.PLUGH
MVS.MCSOPER.* (G)
******************************** Bottom of Data *****************************
```

The **MVS.MCSOPER.DSNADMCM** profile is not defined in this environment. Access is controlled by the profile **MVS.MCSOPER.***, a RACF generic profile. Using the RACF ISPF panel GENERAL RESOURCE SERVICES - DISPLAY you can browse the access definitions on this resource. Example 9-71 illustrates the RACF command output obtained when using this panel.

*Example 9-71 RACF COMMAND OUTPUT, resource MVS.MCSOPER.**

```
BROWSE - RACF COMMAND OUTPUT----------------------- LINE 00000000 COL 001 080
 COMMAND ===>                                                 SCROLL ===> CSR
******************************** Top of Data ********************************
CLASS      NAME
-----      ----
OPERCMDS   MVS.MCSOPER.* (G)

LEVEL  OWNER      UNIVERSAL ACCESS  YOUR ACCESS  WARNING
-----  --------   ----------------  -----------  -------
 00    TRAUNER        CONTROL          CONTROL    NO

...
******************************** Bottom of Data *****************************
```

The RACF Universal Access Authority (UACC) is assigned by default to a user id unless there is a more restrictive access definition. In this example, every user receives CONTROL access on the resource MVS.MCSOPER.* and, in consequence, there is no RACF restriction on the execution of the **ADMIN_COMMAND_MVS** stored procedure.

A way to increase the security control is to create the **MVS.MCSOPER.DSNADMCM** resource and to administer granular access to it. Example 9-72 shows the RACF command. Note that the access by default is NONE, that is, initially no user ID has access to the resource.

Example 9-72 RACF define resource MVS.MCSOPER.DSNADMCM

```
RDEFINE OPERCMDS MVS.MCSOPER.DSNADMCM UACC(NONE)
```

Example 9-73 shows the resulting output at execution.

Example 9-73 RACF define resource output example

```
RACLISTED PROFILES FOR OPERCMDS WILL NOT REFLECT THE ADDITION(S) UNTIL A SETROPTS REFRESH
IS ISSUED.
***
```

The RACF class has to be refreshed to activate the changes. Example 9-74 shows the command to execute a **RACF SETROPTS REFRESH** command.

Example 9-74 RACF SETROPTS REFRESH command

```
SETROPTS RACLIST(OPERCMDS) REFRESH
```

Example 9-75 shows the resulting resource definitions after the execution of these RACF commands.

Example 9-75 RACF resources search result

```
BROWSE - RACF COMMAND OUTPUT----------------------- LINE 00000000 COL 001 080
 COMMAND ===>                                                 SCROLL ===> CSR
******************************** Top of Data ********************************
MVS.MCSOPER.ABC
MVS.MCSOPER.AOPAOP1C
MVS.MCSOPER.DSNADMCM
MVS.MCSOPER.PLUGH
MVS.MCSOPER.* (G)
****************************** Bottom of Data ******************************
```

Example 9-76 shows the details of the **MVS.MCSOPER.DSNADMCM** RACF resource.

Example 9-76 RACF resource MVS.MCSOPER.DSNADMCM

```
BROWSE - RACF COMMAND OUTPUT----------------------- LINE 00000000 COL 001 080
 COMMAND ===>                                                 SCROLL ===> CSR
******************************** Top of Data ********************************
CLASS      NAME
-----      ----
OPERCMDS   MVS.MCSOPER.DSNADMCM

LEVEL  OWNER      UNIVERSAL ACCESS  YOUR ACCESS  WARNING
-----  --------   ----------------  -----------  -------
 00    DB2R1           NONE               NONE    NO

...
****************************** Bottom of Data ******************************
```

This example shows that both universal access and user id DB2R1's access is NONE. As a result, any attempt to execute the **ADMIN_COMMAND_MVS** by the user id DB2R1, or any other user not explicitly authorized in RACF, fails. Under these settings, the calling application receives Return Code 12 at call. Example 9-77 shows the accompanying error message.

Example 9-77 Error message DSNA628I

```
DSNA628I  DSNADMCM THE STORED PROCEDURE SYSPROC.ADMIN_COMMAND_MVS ENCOUNTERED AN ERROR
WHILE USING THE EXTENDED MCS CONSOLE TO ISSUE THE MVS SYSTEM COMMAND DISPLAY WLM,APPLENV=*.
EMCS activation failed. Macro MCSOPER: RC=0C,RSN=00
```

At failure, RACF writes a **ICH408I** error message in the system console, as shown in Example 9-78.

Example 9-78 RACF message ICH408I

```
IEA631I  OPERATOR DSNADMCM NOW INACTIVE, SYSTEM=SC63    , LU=DSNADMCM
ICH70001I DB2R1    LAST ACCESS AT 17:53:40 ON FRIDAY, JULY 26, 2013
ICH408I USER(DB2R1   ) GROUP(SYS1    ) NAME(PAOLO BRUNI         ) 481
  MVS.MCSOPER.DSNADMCM CL(OPERCMDS)
  INSUFFICIENT ACCESS AUTHORITY
  ACCESS INTENT(READ   )  ACCESS ALLOWED(NONE   )
```

This message shows that RACF is not allowing the user id DB2R1 to access the RACF resource **MVS.MCSOPER.DSNADMCM**. Because this is a requirement for the execution of the **ADMIN_COMMAND_MVS** stored procedure, the call fails. To provide access to this resource, you can use the RACF **PERMIT** command as shown in Example 9-79.

Example 9-79 RACF PERMIT MVS.MCSOPER.DSNADMCM

```
PERMIT MVS.MCSOPER.DSNADMCM CLASS(OPERCMDS) ACC(READ) ID(DB2R1)
```

The execution of this command has to be followed by a RACF **SETROPTS** command to activate changes, as shown in Example 9-80.

Example 9-80 RACF SETROPTS RACLIST(OPERCMDS) REFRESH command

```
SETROPTS RACLIST(OPERCMDS) REFRESH
```

Example 9-81 shows that DB2R1 has READ access on the resource **MVS.MCSOPER.DSNADMCM**. The call of the **ADMIN_COMMAND_MVS** stored procedure by DB2R1 is now allowed by RACF.

Example 9-81 RACF MVS.MCSOPER.DSNADMCM resource details

```
BROWSE - RACF COMMAND OUTPUT----------------------- LINE 00000000 COL 001 080
 COMMAND ===>                                                SCROLL ===> CSR
******************************** Top of Data ********************************
CLASS      NAME
-----      ----
OPERCMDS   MVS.MCSOPER.DSNADMCM

LEVEL  OWNER      UNIVERSAL ACCESS  YOUR ACCESS  WARNING
-----  --------   ----------------  -----------  -------
 00    DB2R1           NONE              READ     NO

...
****************************** Bottom of Data ******************************
```

9.8 Drivers, clients, and connectivity requirements

Distributed clients communicate to DB2 11 for z/OS using the IBM Distributed Relational Database Architecture™ (DRDA) protocol. DRDA is an open, vendor-independent architecture that supports the connectivity between a client and database servers. It was initially developed by IBM and then adopted by The Open Group as an industry standard interoperability protocol.

In DRDA terms, the Application Requester function accepts SQL requests from an application and redirect them to an Application Server for processing. The Application Server function receives requests from Application Requesters and processes them. The Application Server can process part of the request and forwards the applicable portion of the request to a database server.

In a distributed application environment accessing DB2 for z/OS, the Application Requester function is supported by a DB2 Client, by a DB2 driver, or by a DB2 Connect server. The Application Server function is integrated in DB2 for z/OS.

Figure 9-10 shows a schematic representation of the AR and AS DRDA components involved in a client to DB2 communication.

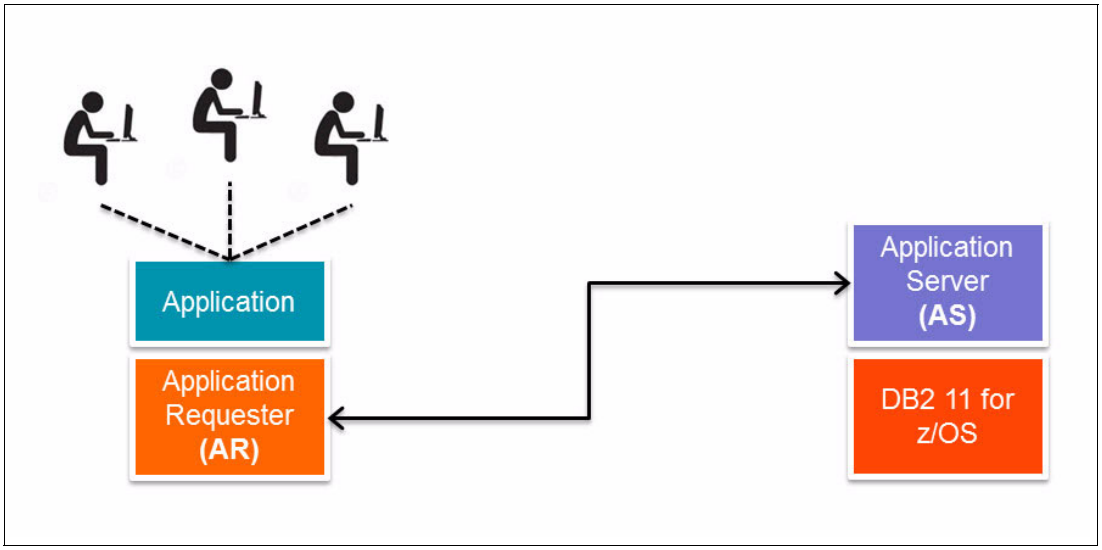


Figure 9-10 AR and AS DRDA components

Improvements related to the distributed access to DB2 11 to z/OS might require changes at the Client, driver or DB2 Connect side.

DB2 Clients, Drivers and DB2 Connect DB2 10.5 FP2 are required to fully take advantage of DB2 11 for z/OS distributed features, such as:

- CALL with array type arguments
- Larger **CLIENT INFO** properties (including new correlation token)
- Implicit **COMMIT** for stored procedures
- Sysplex support for Global Session Variables

The DRDA protocol implements DRDA levels to group improvements and features. A down level DRDA Client works with DB2 11, but it cannot use all the benefits of DB2 11. Any in-service level of DB2 Client, DB2 Drivers, or DB2 Connect server should work with DB2 11 CM and DB2 11 NFM. At the moment of this writing, versions 9.5 and later are in-service. DB2 Connect V9.5 is planned to be out of service at April-2014.

DB2 Connect drivers seamlessly handle the migration path from DB2 10 for z/OS to DB2 11 CM, and from them to DB2 11 NFM. In a data sharing environment, applications continue to function as members are migrated one by one.

During migration, these considerations apply:

- While in DB2 11 CM, applications continue to see DB2 10 function level.
- After migrating to DB2 11 NFM, new connections see DB2 11 function level when using **APPLCOMPAT** set to V11R1.

Verifying the level of DB2 Clients and DB2 Drivers

To exploit the latest DB2 11 distributed access improvements, you have to work with a DB2 driver or client DB2 10.5 fix pack 2.

Note: At the moment of this writing, the latest drivers and clients are available at the web page “Download initial DB2 10.5 clients and drivers” at http://www.ibm.com/support/docview.wss?uid=swg21385217

Use the **db2level** command to show the current version and service level of a DB2 client or DB2 Connect server. Example 9-82 shows the execution of db2level on a Windows machine. Among other details, this example shows that this Client in DB2 10.5 with Fix Pack 0.

Example 9-82 Using the db2level command

```
C:\Program Files\IBM\SQLLIB_03\BIN>db2level
DB21085I  This instance or install (instance name, where applicable: "DB2_03")
uses "64" bits and DB2 code release "SQL10050" with level identifier
"0601010E".
Informational tokens are "DB2 v10.5.0.420", "s130528", "NTX64105", and Fix Pack
"0".
Product is installed at "C:\PROGRA~1\IBM\SQLLIB~3" with DB2 Copy Name
"DB2V10R5".

C:\Program Files\IBM\SQLLIB_03\BIN>
```

The db2level command is not available for Java drivers. For JDBC or SQLJ applications, if you are using the IBM DB2 driver for SQLJ and JDBC, you can determine the level of the driver by running the db2jcc utility, as shown in Example 9-83.

Example 9-83 Running the db2jcc utility

```
java com.ibm.db2.jcc.DB2Jcc -version
```

Example 9-84 shows the db2jcc output in this example test environment. There is no information about the driver version nor Fix Pack level. This can be an inconvenient because the DB2 11 for z/OS requirements for Clients and Drivers are expressed on these terms.

Example 9-84 Db2jcc utility output

```
C:\Program Files\IBM\SQLLIB_03\BIN>java com.ibm.db2.jcc.DB2Jcc -version
IBM DB2 JDBC Universal Driver Architecture 3.66.46

C:\Program Files\IBM\SQLLIB_03\BIN>
```

There is no direct way to discern which JDBC driver (JCC) version corresponds with each DB2 release and Fix Pack level. Nevertheless, there is a way to map the driver architecture, provided by db2jcc -version, with that information.

Note: To find the correlation between the Java driver architecture and the driver version visit the page “DB2 JDBC driver Versions” at
http://www.ibm.com/support/docview.wss?rs=71&uid=swg21363866

Figure 9-11 shows a partial view of the *DB2 JDBC driver Versions* web page. It highlights how to match the driver’s architecture to the DB2 version and Fix Pack level.

The DB2 Universal JDBC driver versions are included with DB2 products as shown in the table below. Notes on how to check for the version can be found at the bottom of this page.

DB2 Version 10.5

DB2 Level	Build Number	JDBC 3.0 driver		JDBC 4.0 Driver		APAR List
		Driver version	Size (bytes)	Driver version	Size (bytes)	
v10.5 FP0 (GA)		3.66.46	3656105	4.16.53	3866524	N/A

DB2 Version 10.1

DB2 Level	Build Number	JDBC 3.0 driver		JDBC 4.0 Driver		APAR List
		Driver version	Size (bytes)	Driver version	Size (bytes)	
v10.1 FP0 (GA)	s120403	3.63.123	3516375	4.13.127	3714904	N/A
v10.1 FP1	s120826	3.64.104	3600047	4.14.111	3801427	
v10.1 FP2	s121127	3.65.77	3611423	4.15.82	3816252	

DB2 Version 9.7

DB2 Level	Build Number	JDBC 3.0 driver		JDBC 4.0 driver		APAR List
		Driver version	Size (Bytes)	Driver version	Size (Bytes)	
v9.7 FP0 (GA)	s090521	3.57.82	3146716	4.7.85	3312885	Not Applicable
v9.7 FP1	s091114	3.58.82	3226620	4.8.87	3395609	v9.7 FP1 APAR List
v9.7 FP2	s100514	3.59.81	3295950	4.9.78	3472296	v9.7 FP2 APAR List
v9.7 FP3	special_25384	3.61.65	3348030	4.11.69	3527921	v9.7 FP3 APAR List
v9.7 FP3a	s101006	3.61.75	3348681	4.11.77	3528544	v9.7 FP3 APAR List

Figure 9-11 DB2 JDBC driver Versions web page

The IBM DB2 JDBC Universal Driver Architecture 3.66.46, as shown in Example 9-84, correspond to DB2 DB2 10.5 FP0 (GA).

Example 9-85 shows how the JDBC connection string can be used to activate a JDBC trace.

Example 9-85 JDBC connection url String with TRACE_ALL

```
String url = "jdbc:db2://redbook8:38420/DB1A" +
        ":user=db2r1;password=******;" +
        "traceLevel=" +
        (com.ibm.db2.jcc.DB2BaseDataSource.TRACE_ALL) + ";" +
        "traceFile=c:\\work\\Author\\Redbook#8\\DRDA_traces\\DB2110ldDriver.trace;";
```

Example 9-86 shows a partial example of the JDBC trace output as a result of the syntax in Example 9-85.

Example 9-86 JDBC trace output

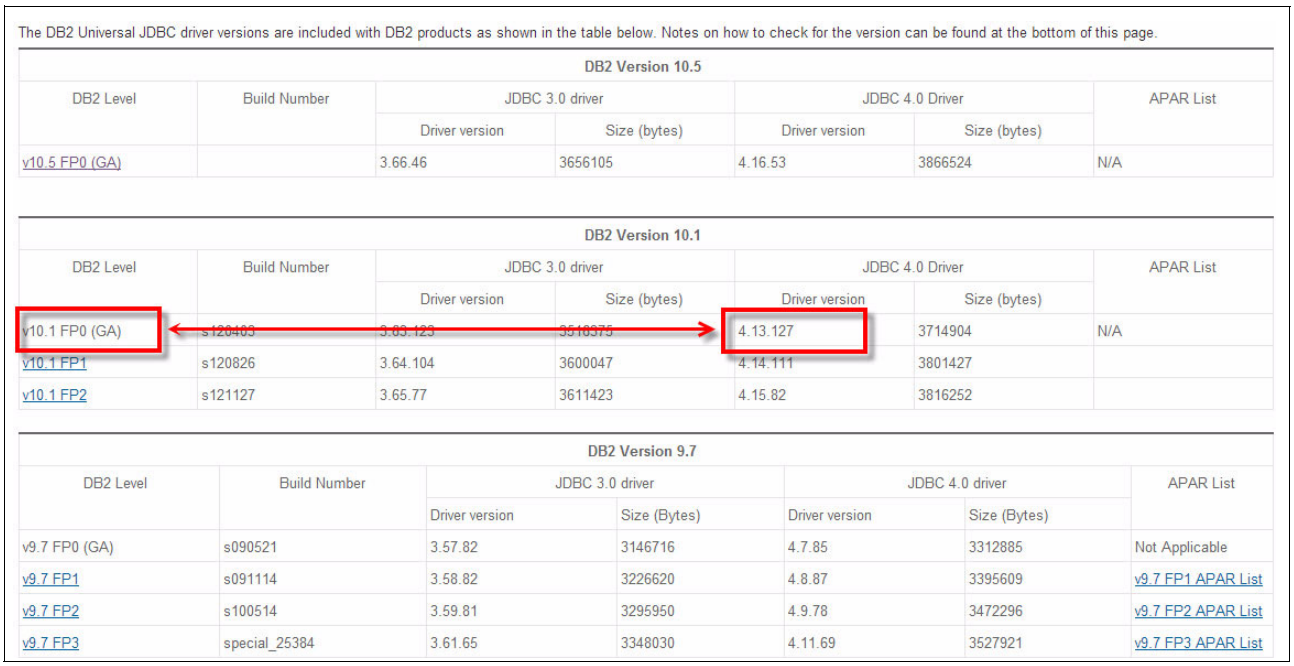
```
[jcc] BEGIN TRACE_XML_CONFIGURATION_FILE
[jcc] dsdriverConfigFile=null
[jcc] END TRACE_XML_CONFIGURATION_FILE

[jcc] BEGIN TRACE_DRIVER_CONFIGURATION
[jcc] Driver: IBM Data Server driver for JDBC and SQLJ 4.13.127
[jcc] Compatible JRE versions: { 1.6, 1.7 }
```

```
[jcc] Target server licensing restrictions: { z/OS: enabled; SQLDS: enabled
[jcc] License editions: { O: not found; ZS: not found; IS: not found; AS: n
[jcc] Range checking enabled: true
[jcc] Bug check level: 0xff
[jcc] Default fetch size: 64
[jcc] Default isolation: 2
[jcc] Collect performance statistics: false
[jcc] No security manager detected.
[jcc] Detected local client host: x1/9.55.137.33
```

Figure 9-12 shows a portion of the “DB2 JDBC driver Versions” at:

http://www.ibm.com/support/docview.wss?rs=71&uid=swg21363866

This figure highlights the link between the driver version and the DB2 level of the driver.



The DB2 Universal JDBC driver versions are included with DB2 products as shown in the table below. Notes on how to check for the version can be found at the bottom of this page.

DB2 Version 10.5

DB2 Level	Build Number	JDBC 3.0 driver		JDBC 4.0 Driver		APAR List
		Driver version	Size (bytes)	Driver version	Size (bytes)	
v10.5 FP0 (GA)		3.66.46	3656105	4.16.53	3866524	N/A

DB2 Version 10.1

DB2 Level	Build Number	JDBC 3.0 driver		JDBC 4.0 Driver		APAR List
		Driver version	Size (bytes)	Driver version	Size (bytes)	
v10.1 FP0 (GA)	s120403	3.63.123	3516375	4.13.127	3714904	N/A
v10.1 FP1	s120826	3.64.104	3600047	4.14.111	3801427	
v10.1 FP2	s121127	3.65.77	3611423	4.15.82	3816252	

DB2 Version 9.7

DB2 Level	Build Number	JDBC 3.0 driver		JDBC 4.0 driver		APAR List
		Driver version	Size (Bytes)	Driver version	Size (Bytes)	
v9.7 FP0 (GA)	s090521	3.57.82	3146716	4.7.85	3312885	Not Applicable
v9.7 FP1	s091114	3.58.82	3226620	4.8.87	3395609	v9.7 FP1 APAR List
v9.7 FP2	s100514	3.59.81	3295950	4.9.78	3472296	v9.7 FP2 APAR List
v9.7 FP3	special_25384	3.61.65	3348030	4.11.69	3527921	v9.7 FP3 APAR List

Figure 9-12 db2 JDBC driver versions web page

Data Studio

IBM Data Studio provides database developers and database administrators with an integrated, modular environment for development and productive administration of DB2 databases. IBM Data Studio is a fully licensed product available at no charge and with no time restrictions.

Important: IBM Data Studio supports DB2 11 for z/OS with V4.1 or later, which can be downloaded at no additional charge from:

http://www.ibm.com/developerworks/downloads/im/data/

Figure 9-13 shows that IBM Data Studio V3.2 identifies a DB2 11 NFM database as a DB2 10 NFM subsystem.

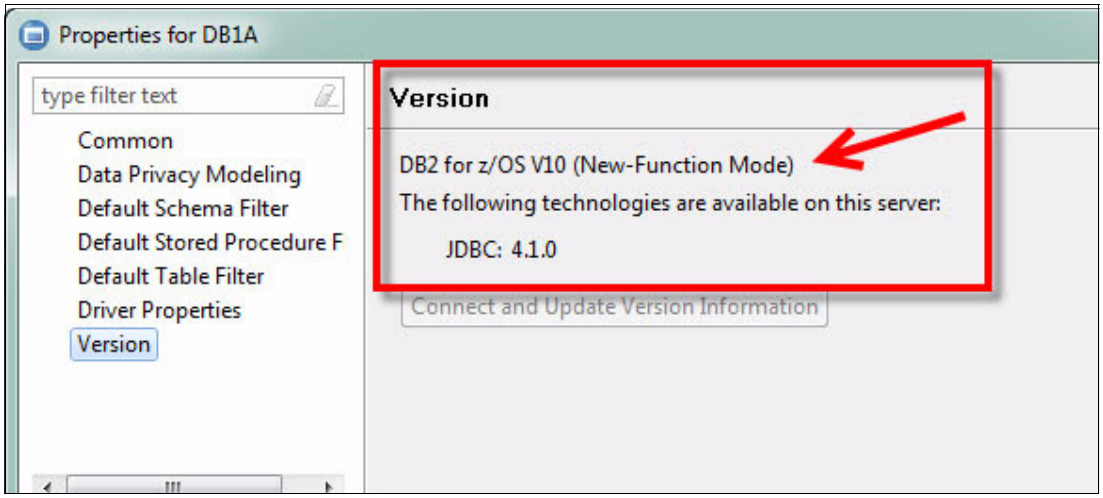


Figure 9-13 IBM Data Studio 3.2 and DB2 11

Figure 9-14 shows that IBM Data Studio 4.1 correctly identifies the server as a DB2 11 NFM.

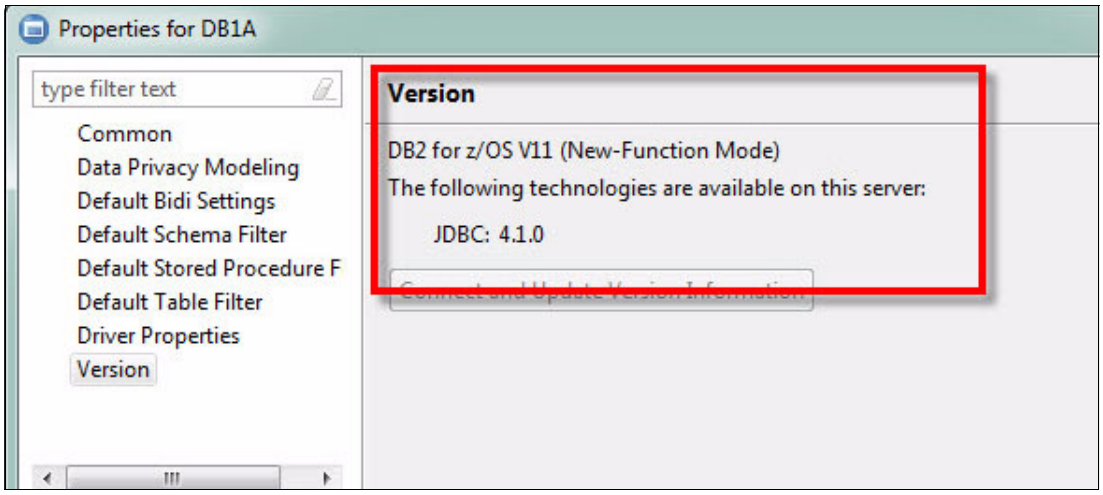


Figure 9-14 IBM Data Studio 4.1 and DB2 11

Using the Data Studio application menu, select **Help** → **About IBM Data Studio** to verify the Data Studio version. Figure 9-15 shows the About IBM Data Studio window with the version information.

Figure 9-15 How to identify the Data Studio version

You can use Data Studio 4.1 with a DB2 10 for z/OS database. In that case, the PTF UK91146 must be applied to the DB2 10 for z/OS data server that you want to connect to prevent connectivity problems. Refer to the technote (troubleshooting) “Connecting to DB2 z/OS 10 with Data Studio V4.1 or InfoSphere Data Architect V9.1 results in SQL error code -4499 or -1224” available at:

http://www.ibm.com/support/docview.wss?uid=swg21641377

How to catalog a DB2 for z/OS database using commands

To access a DB2 for z/OS server using a Client, you have to catalog it. The DB2 Client version 10 does not comes with the Configuration Assistant. In previous versions, the Configuration Assistance, a GUI tool, can be used to catalog a DB2 for z/OS database as a ODBC data source in a Windows system machine. With DB2 10, the configuration has to be done using commands. This section describes the steps involved on the process.

Start by getting the host database configuration information by issuing the **-DIS DDF DETAIL** command. Example 9-87 shows the output of this command.

Example 9-87 -DIS DDF output example

```
DSNL080I  -DB1A DSNLTDDF DISPLAY DDF REPORT FOLLOWS:
DSNL081I STATUS=STARTD
DSNL082I LOCATION           LUNAME            GENERICLU
DSNL083I DB1A               USIBMSC.SCPDB1A   -NONE
DSNL084I TCPPORT=38420 SECPORT=38422 RESPORT=38421 IPNAME=-NONE
DSNL085I IPADDR=::9.12.6.70
DSNL086I SQL    DOMAIN=wtsc63.itso.ibm.com
DSNL090I DT=I  CONDBAT=  10000 MDBAT=  200
DSNL092I ADBAT=    1 QUEDBAT=      0 INADBAT=      0 CONQUED=      0
DSNL093I DSCDBAT=      1 INACONN=      2
DSNL105I CURRENT DDF OPTIONS ARE:
```

```
DSNL106I PKGREL = COMMIT
DSNL099I DSNLTDDF DISPLAY DDF REPORT COMPLETE
```

Note the following information:

- Location, provided in message DSNL083I. **DB1A** in this example.
- TCP/IP port, provided in message DSNL084I. **38420** in this example.

The IP address provided in message **DSNL085I** might not work as a target IP from your workstation, depending from where it was obtained. Refer to the documentation of message DSNL085I for more details. In this example environment, the DB2 server can be reached by using the 9.12.6.70 IP address. This information is used in commands within the DB2 CLP. The CLP is part of the DB2 client, and it is included in other DB2 offerings as well.

If you have installed a DB2 client, you can use a DB2 CLP command window to catalog a DB2 for z/OS database in a windows server or workstation. Example 9-88 shows the initial contents when opening a DB2 Client 10.5.0 CLP window.

Example 9-88 DB2 Command Line Processor initial contents

```
Command Line Processor for DB2 Client 10.5.0

You can issue database manager commands and SQL statements from the command
prompt. For example:
    db2 => connect to sample
    db2 => bind sample.bnd

For general help, type: ?.
For command help, type: ? command, where command can be
the first few keywords of a database manager command. For example:
 ? CATALOG DATABASE for help on the CATALOG DATABASE command
 ? CATALOG          for help on all of the CATALOG commands.

To exit db2 interactive mode, type QUIT at the command prompt. Outside
interactive mode, all commands must be prefixed with 'db2'.
To list the current command option settings, type LIST COMMAND OPTIONS.

For more detailed help, refer to the Online Reference Manual.

db2 =>
```

The first step is to catalog a TCP/IP node using the CLP. Cataloging a TCP/IP node adds an entry to the Data Server Client node directory that describes the remote node. This entry specifies the chosen alias (`node_name`), the host name (or `ip_address`), and the `svcename` (or `port_number`) that the client uses to access the remote host.

Example 9-89 shows the commands that can be used.

Example 9-89 DB2 catalog TCP/IP node example

```
catalog tcpip node SC63 remote 9.12.6.70 server 38420 ostype mvs
```

In general, it is a preferred practice to use machine names instead if IP addresses when cataloging a remote server. Using names means fewer configuration points to maintain in case of an IP address change. You can use a DNS (domain name server) for mapping a server name to an IP address to make this information available to your network. If the scope is your own workstation, it is in general simpler to just maintain a hosts file. In Windows 7, the hosts files are located at `C:\Windows\System32\drivers\etc\hosts`.

Example 9-90 shows a sample Windows hosts file that is customized with the information about the System z server that runs the target DB2 11 for z/OS.

Example 9-90 Sample Windows hosts file

```
# This is a sample HOSTS file used by Microsoft TCP/IP for Windows.
#
# This file contains the mappings of IP addresses to host names. Each
# entry should be kept on an individual line. The IP address should
# be placed in the first column followed by the corresponding host name.
# The IP address and the host name should be separated by at least one
# space.
#
# localhost name resolution is handled within DNS itself.
#  127.0.0.1       localhost
#  ::1             localhost
9.12.6.70         redbook8
```

After updating the hosts file as shown in this example, a reference to redbook8 is redirected to the IP address 9.12.6.70. In consequence, the DB2 catalog TCP/IP node can be simplified as shown in Example 9-91.

Example 9-91 DB2 catalog TCP/IP node example using an hosts file entry

```
catalog tcpip node SC63 remote redbook8 server 38420 ostype mvs
```

Example 9-92 illustrates the output of this command.

Example 9-92 DB2 catalog TCP/IP node output example

```
db2 => catalog tcpip node SC63 remote redbook8 server 38420 ostype mvs
DB20000I  The CATALOG TCPIP NODE command completed successfully.
DB21056W  Directory changes may not be effective until the directory cache is
refreshed.
db2 =>
```

The **DB21056W** message indicates that changes might not be effective immediately, and that a directory refresh might be required to make the updates effective. Execute the **terminate** command, as illustrated in Example 9-93.

Example 9-93 DB2 terminate example

```
db2 => terminate
DB20000I  The TERMINATE command completed successfully.
C:\Program Files\IBM\SQLLIB_03\BIN>
```

Tip: To refresh the CLP directory cache, issue a db2 **terminate** command. To refresh the directory information for another application, stop and restart that application. To refresh the directory information for the database, stop (db2stop) and restart (db2start) the database

The node directory is created and maintained on each database client. The directory contains an entry for each remote database partition server having one or more databases that the client can access. The DB2 client uses the communication endpoint information in the node directory whenever a database connection or instance attachment is requested. The entries in the directory also contain information about the type of communication protocol to be used

to communicate from the client to the remote database partition. Cataloging a local database partition creates an alias for an instance that resides on the same computer.

Example 9-94 illustrates the execution of a **list node directory** command in this example test environment. Use this command to verify the successful addition of the node.

Example 9-94 DB2 list node directory command example

```
C:\Program Files\IBM\SQLLIB_03\BIN>db2 list node directory

 Node Directory

 Number of entries in the directory = 1

Node 1 entry:

 Node name                      = SC63
 Comment                        =
 Directory entry type           = LOCAL
 Protocol                       = TCPIP
 Hostname                       = redbook8
 Service name                   = 38420

C:\Program Files\IBM\SQLLIB_03\BIN>
```

The **catalog database** command stores database location information in the system database directory. The database can be located either on the local workstation or on a remote database partition server. Example 9-95 illustrates the command used in this example test environment to catalog the DB2 11 target database.

Example 9-95 DB2 catalog database command example

```
catalog database DB1A as DB1A at node SC63 authentication SERVER_ENCRYPT
```

Example 9-96 shows the results of executing this command.

Example 9-96 DB2 catalog database command output example

```
db2 => catalog database DB1A as DB1A at node SC63 authentication SERVER_ENCRYPT
DB20000I  The CATALOG DATABASE command completed successfully.
DB21056W  Directory changes may not be effective until the directory cache is
refreshed.
db2 =>
```

The **list database directory** command lists the contents of the system database directory. Use this command to verify the addition of a database, as illustrated in Example 9-97.

Example 9-97 DB2 list database directory command output example

```
db2 => list database directory

 System Database Directory

 Number of entries in the directory = 3

Database 1 entry:

 Database alias                       = BLUDB01
 Database name                        = BLUDB01
 Local database directory             = C:
```

```
 Database release level              = 10.00
 Comment                             =
 Directory entry type                = Indirect
 Catalog database partition number   = 0
 Alternate server hostname           =
 Alternate server port number        =

Database 2 entry:

 Database alias                      = DB1A
 Database name                       = DB1A
 Node name                           = SC63
 Database release level              = 10.00
 Comment                             =
 Directory entry type                = Remote
 Authentication                      = SERVER_ENCRYPT
 Catalog database partition number   = -1
 Alternate server hostname           =
 Alternate server port number        =

Database 3 entry:

 Database alias                      = SAMPLE
 Database name                       = SAMPLE
 Local database directory            = C:
 Database release level              = 10.00
 Comment                             =
 Directory entry type                = Indirect
 Catalog database partition number   = 0
 Alternate server hostname           =
 Alternate server port number        =

db2 =>
```

This example shows that the DB2 Client can be used to connect to two local databases (SAMPLE and BLUDB01) and to the remote database DB1A, the target. Applications connect using the database alias value that is provided in this command.

Finally, connect to the target database using the CLP for verification, as shown in Example 9-98.

Example 9-98 Connect to a DB2 for z/OS database using the CLP

```
db2 => connect to DB1A user db2r1
Enter current password for db2r1:

   Database Connection Information

 Database server        = DB2 z/OS 11.1.5
 SQL authorization ID   = DB2R1
 Local database alias   = DB1A

db2 =>
```

This example shows a connection to the target DB1A, which is DB2 11 for z/OS database.

A data source, in ODBC (Open Database Connectivity) terminology, is a user-defined name for a specific database or file system. That name is used to access the database or file system through ODBC APIs. Either user or system data sources can be cataloged. A user

data source is only visible to the user who cataloged it, whereas a system data source is visible to and can be used by all other users. The **CATALOG ODBC DATA SOURCE** command is used to catalog a user or system ODBC data source. Example 9-99 shows the command to be executed in this example environment.

Example 9-99 DB2 catalog ODBC data source command example

```
catalog odbc data source DB1A
```

Example 9-100 illustrates the execution results in this environment.

Example 9-100 DB2 catalog ODBC data source command output example

```
db2 => catalog odbc data source DB1A
DB20000I  The CATALOG USER ODBC DATA SOURCE command completed successfully.
db2 =>
```

Use the **list ODBC data sources** command to confirm the changes, as shown in Example 9-101.

Example 9-101 DB2 LIST ODBC DATA SOURCES command example

```
db2 => list odbc data sources
               User ODBC Data Sources

Data source name                Description
------------------------------- ----------------------------------------
MS Access Database              Microsoft Access driver (*.mdb)
Excel Files                     Microsoft Excel driver (*.xls)
dBASE Files                     Microsoft dBase driver (*.dbf)
DZA1                            IBM DB2 ODBC DRIVER - DB2COPY1
BRUXLS                          IBM DB2 ODBC DRIVER - DB2COPY1
DB1A                            IBM DB2 ODBC DRIVER - DB2V10R5
db2 =>
```

Part 3

Operations and performance

Integration of DB2 11 with z/OS Security Server (RACF) helps in operational compliance (both regulatory and governance) and separation of duty and more flexibility is added to masking functions.

DB2 Utilities Suite for z/OS V11 (program number 5655-W87) delivers full support for the significant enhancements in DB2 11 and delivers improvements in availability (online **REORG** granularity) and zIIP eligibility.

DB2 11 can improve performance by taking advantage of the platform functions, reducing path length in several situations, and further reducing virtual storage requirements below the bar. In addition, installation and migration functions use the established conversion mode and New Function Mode (NFM) statuses and allow new options to reduce incompatibility situations.

This part includes the following chapters:

10

Security

DB2 includes the following main security topics:

- DB2 enhancements for exit authorization checking
  - DB2 provides the accessor environment element (ACEE) of the package owner for authorization checking when the access control authorization exit is active
  - DB2 also refreshes the cache entries of the package authorization, the routine authorization, the DDF user authorization, and the dynamic statement when a user profile or resource access is changed in RACF and the access control authorization exit is active.
- DB2 enhancements for program authorization

  DB2 provides the capability to check whether an application program is authorized to use a plan.
- DB2 enhancement to the masking functions

  DB2 removes some restrictions related to aggregation of data while using the masking functions.

This chapter describes these changes in the following sections:

- Enhancements for exit authorization checking
- Enhancements to program authorization
- Column masking enhancements

10.1 Enhancements for exit authorization checking

The Access Control Authorization Exit (DSNX@XAC) enables to use external security such as RACF for authorization checking for DB2 objects, authorities, commands, and utilities. There are certain instances where the authorization checking done by RACF is different from the authorization checking done by DB2. Due to new regulations and separation of duties requirements, customers who are moving to use RACF for authorization consider the differences as serious limitations in adopting RACF authorization.

Figure 10-1 provides an overview of the relationship between DB2 10 and the RACF Access Control Authorization Exit Authorization.

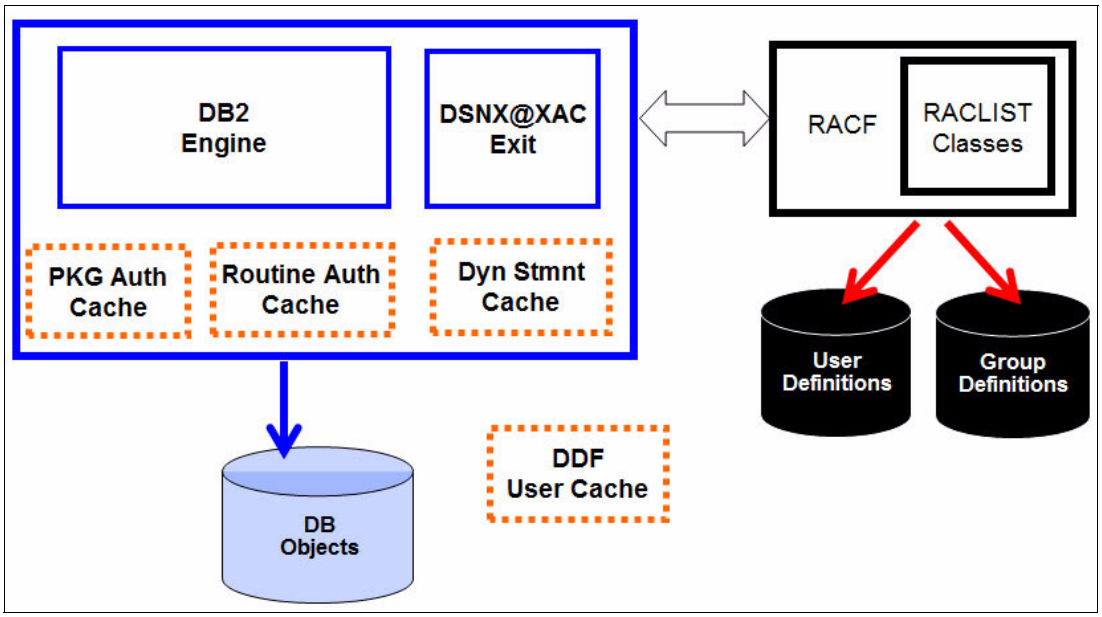


Figure 10-1 DB2 10 and RACF Access Control Authorization Exit Authorization

The authorization process, when DB2 security is in RACF, flows as follows:

- DB2 obtains RACF information through the DSNX@XAC exit
- DB2 caches the successful authorization for packages, routines and dynamic statements execution in the three caches. They are represented with dotted boxes in Figure 10-1.

As a result of the DB2 security in RACF implementation in DB2 10, the following concerns can appear:

- The **OWNER** keyword is not honored when RACF exit is used to control authorization
  - RACF checks the invoker (primary authorization ID) not the owner during **BIND**, **REBIND**, and **AUTOBIND**
  - **AUTOBIND** fails if the invoker is not authorized
- Use of the **DYNAMICRULES(BIND)** option with dynamic SQL

  RACF checks the invoker ID regardless of **DYNAMICRULES** specification. RACF authorization fails if the invoker is not authorized to execute the SQL statements in the package.
- **AUTOBIND**

  Invoker is checked for authorization to execute the SQL statements. RACF authorization fails if the invoker is not authorized to execute the SQL statements in the package.

- DB2 10 static packages not invalidated following a RACF **REVOKE**

  RACF authorization checked at **BIND/REBIND**
- DB2 10 dynamic statement cache and authorization caches never flushed at RACF change
  - DB2 cache can retain privileges after RACF **REVOKE**
  - DBAs must take action (**GRANT/REVOKE/RUNSTATS**) to sync with RACF
- DB2 10 caches are refreshed every 3 minutes

  Flushing the cache takes time and resources

DB2 11 introduces changes that address these issues. The following sections describe these enhancements.

10.1.1 Use owner privileges for authorization

During **AUTOBIND**, DB2 provides the ACEE of the runner for exit authorization. Most of the time, runner does not have the privilege to execute the SQL statements in the package, which causes **AUTOBIND** to fail.

Similarly, when **BIND OWNER** option is specified, RACF always checks authorization on the runner. Also, when embedded dynamic SQL in packages bound with **DYNAMICRULES(BIND)** option is executed, it is expected that the `auth-id` checked for authorization is the package owner. However, DB2 provides the ACEE of the runner. Thus, every user executing the package has to be provided access.

Previous to DB2 11, when using Access Control Authorization Exit (DSNX@XAC) for authorization:

- **BIND** and **REBIND** commands do not support the **OWNER** keyword.
- The **AUTOBIND** fails, which causes application outage.
- For packages bound with **DYNAMICRULES(BIND)**, the dynamic SQL statement authorization requires the runner to have the privilege.

DB2 11 addresses this limitation by providing the ACEE of the package owner to perform authorization checking when processing auto bind, the **BIND** and **REBIND** commands; and the ACEE of the authorization ID to perform dynamic SQL authorization checking, when **DYNAMICRULES** value other than **RUN** is in effect. For dynamic SQL authorization checking, the **DYNAMICRULES** behavior determines whether the authorization ID is the package owner, the routine definer or the routine invoker.

DB2 11 for z/OS, when using DSNX@XAC exit authorization, is enhanced to:

- Support of the **OWNER** keyword for the **BIND** and **REBIND** commands.
- Support owner authorization during **AUTOBIND**.
- Support **DYNAMICRULES(BIND)** behavior for dynamic SQL statements.

DB2 11 introduces the capability to provide the owner ACEE for authorization checking, when access control authorization exit is active. The owner can be a valid user or a group in ACF, which allows the package owner to be used for authorization checking during auto bind and **BIND** or **REBIND** command processing. This function also allows package owner or routine definer or routine invoker as determined by the **DYNAMICRULES** behavior to be used for dynamic SQL authorization checking, when the **DYNAMICRULES** bind option value other than **RUN** is in effect and that **DYNAMICRULES** authorization ID is cached in the dynamic statement cache.

DB2 11 introduces the **AUTHEXIT_CHECK** installation parameter, which allows you to specify whether the owner or the primary authorization ID is to be used for authorization checks when the access control authorization exit is active. This parameter takes the following values:

DB2	Specifies that DB2 provides the ACEE of the package owner to perform authorization checking when processing the **AUTOBIND**, **BIND**, and **REBIND** commands. DB2 provides the ACEE of the authorization ID as determined by the **DYNAMICRULES** option to perform dynamic SQL authorization checking. The access control authorization exit uses the ACEE for XAPLUCHK for authorization checking. The XAPLUCHK authorization ID can be a user or a group in RACF
PRIMARY	Specifies that DB2 provides the ACEE of the primary authorization ID to perform all authorization checks. The primary authorization ID must be permitted access to the resources in RACF.

The default value is **PRIMARY**. This install parameter is not online updatable. It is part of the DSN6SPRM macro. This parameter in added to the DB2 Protection panel, DSNTIPP.

If system parameter, **AUTHEXIT_CHECK** is set to DB2, then DB2 provides the ACEE of the package owner to perform authorization checking when processing auto bind, the BIND and REBIND commands; and the ACEE of the authorization ID to perform dynamic SQL authorization checking, when **DYNAMICRULES** value other than RUN is in effect. For dynamic SQL authorization checking, the **DYNAMICRULES** behavior determines whether the authorization ID is the package owner, the routine definer or the routine invoker. Hence, this owner authorization ID has to be permitted access to resources in RACF in the previously described scenarios.

To ensure successful authorization checks with the owner ACEE, the owner authorization ID in **XAPLUCHK** must be permitted access to the resources in RACF. If the owner is a group in RACF, you need to permit the group access to the resource associated with the connection in the RACF **DSNR** class. You can issue the **PERMIT** command to grant a group access to the resource **BATCH** in the DSNR class. Example 10-1 illustrates the RACF command to permit access on the DB2 subsystem DB1A to the **DB2GRP** group.

Example 10-1 RACF permit ACCESS(READ) on CLASS(DSNR)

```
PERMIT DB1A.BATCH CLASS(DSNR) ID(DB2GRP) ACCESS(READ)
```

Also, this owner RACF group must have authorization to execute all the statements embedded in the package for successful processing of **BIND/REBIND** command, auto bind, and dynamic SQL authorization when **DYNAMICRULES** value other than **RUN** is in effect.

10.1.2 Refresh DB2 cache entries when RACF permissions change

Previous to DB2 11, when DB2 caches are enabled and RACF permissions change in RACF, then the package authorization cache, routine authorization cache and dynamic statement cache are not refreshed to reflect the change. To refresh the cache entries, SQL **GRANT** and **REVOKE** statements have to be issued or to invalidate the entry from dynamic statement cache, **RUNSTATS** utility has to be executed.

DB2 11 introduces the capability to refresh the DB2 following cache entries when access control authorization exit is active and RACF permissions change:

- Package
- Authorization cache
- Routine authorization cache
- Dynamic statement cache

DB2 11 implements this enhancement by listening to the following ENF signals sent by RACF and refreshing the DB2 authorization related cache entries:

- Type 71 ENF signals when a user's permission is changed in RACF
- Type 79 ENF signals when a user's permission to access a resource is changed in RACF
- Type 62 ENF signals when RACF options are refreshed

Important: DB2 only listens to the ENF signals sent by RACF. If other vendor products are used for access control, the caches cannot be refreshed.

Figure 10-2 is a schema of the relationship between DB2 11 and RACF. Changes made in RACF (point 1 in the figure) are communicated to DB2 to take action accordingly (point 2 in the figure).

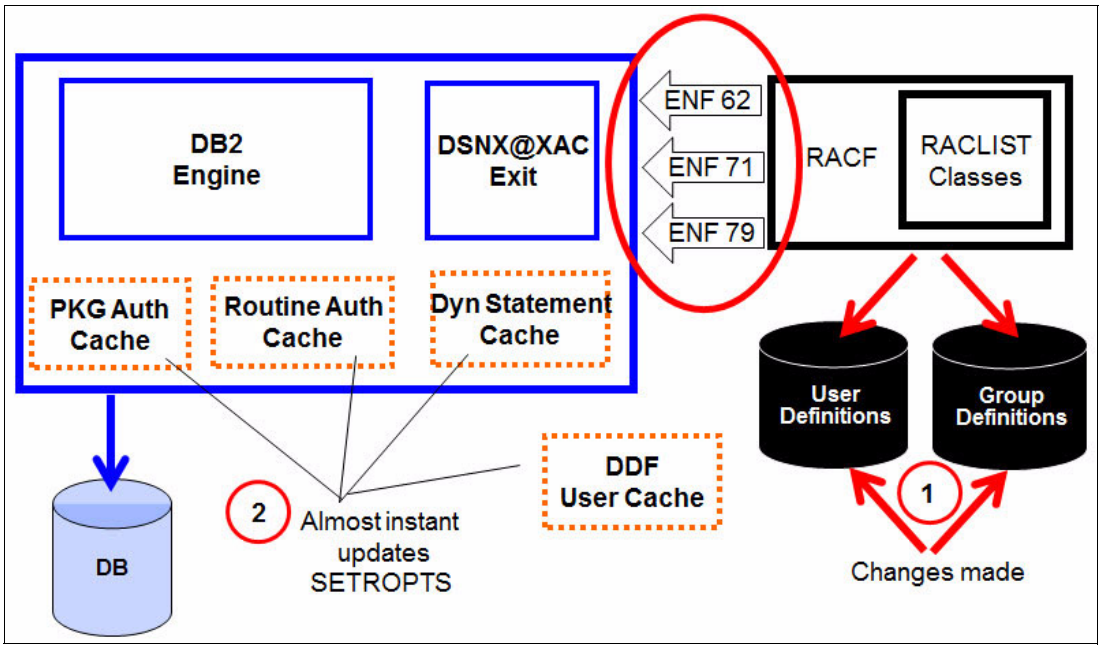


Figure 10-2 DB2 11 and RACF Access Control Authorization Exit Authorization

ENF 71 is issued for change in a user or group profile. When ENF 71 is issued, DB2 refreshes the cache entries for the affected user or group.

ENF 79 is issued for change in a user's or group's authorization to resources. When ENF 79 is issued, DB2 caches the resource changes, as the resource changes will not take effect until the **`SETROPTS RACLIST REFRESH`** command is issued.

ENF 62 is issued for the **`SETROPTS RACLIST REFRESH command`**. When ENF 62 is issued, DB2 refreshes the cache entries for the resources that are cached during ENF 79 notification.

DB2 11 listens to the type 71 ENF signal issued by RACF for the following RACF commands:

- **ALTUSER**
- **CONNECT**
- **DELUSER**
- **DELGROUP**
- **REMOVE**

Note: For **ALTUSER** and **CONNECT** command notifications, DB2 processes the signal only when REVOKE option is specified.

DB2 listens to the type 79 ENF signal issued by RACF for the following RACF commands:

- **PERMIT** options:
  - **DELETE**
  - **ACCESS(NONE)**
  - **RESET**
  - **WHEN(CRITERIA(SQLROLE ….))**
- **RALTER** options:
  - **UACC(NONE)**
  - **DELMEM**
- **RDELETE**

Attention: The cache entries are not refreshed when the **RDELETE** command is issued to delete general resource profiles for **DSNADM** and **MDSNSM/GDSNSM** classes without a profile name

ENF 79 signal is issued only for classes that have been defined in the RACF Class Descriptor Table with the SIGNAL=YES option. The SIGNAL=YES option is enabled, by default, for the following IBM supplied RACF resource classes for DB2:

- **MDSNPK / GDSNPK**
- **MDSNTB / GDSNTB**
- **MDSNSP / GDSNSP**
- **MDSNSQ / GDSNSQ**
- **DSNADM** and **MDSNSM / GDSNSM**
- **MDSNUF / GDSNUF**
- **MDSNGV / GDSNGV**

Tip: If you are defining classes for DB2 objects and administrative authorities and not using IBM supplied RACF resource classes for DB2, then you need to enable the SIGNAL=YES option for these classes.

The class names for DB2 objects in both single-subsystem scope and multiple-subsystem scope are supported.

This list shows the RACF classes and the corresponding caches that are impacted by the revocation:

- **MDSNPK**, Package Authorization Cache
- **MDSNTB**, **MDSNSQ**, and **MDSNGV**, Dynamic Statement Cache
- **MDSNSP**, Routine Authorization Cache
- **MDSNUF**, Routine Authorization Cache, Dynamic Statement Cache

- **DSNADM** and **MDSNSM**, Package and Routine Authorization Caches, Dynamic Statement Cache

The RACF discrete and generic resource names in the profile are supported with the following restrictions:

- Generic character ampersand (&) indicates that RACF is to use a profile in the **RACFVARS** class to determine the actual values to use for that part of the profile name. The ENF signal notification for profile that contains ampersand is ignored and not processed by DB2 for cache refresh
- Generic character % is not supported in the privilege part of the profile for cache refresh. The ENF signal notification is ignored. Example 10-2 illustrates this situation.

Example 10-2 RACF PERMIT command and % generic resource character

```
PERMIT SYS1.SCHM1.PROCA2.EXE%UTE ID(CRISO3) DELETE CLASS(MDSNSP)
```

- Generic characters in profile names for all classes other than **DSNADM** class: If a profile has less number of parts than supported by the **CLASS** parameter and contains generic characters * or **, then depending on the specification of the generic character all objects or all privileges for the specified **CLASS** parameter can be considered for cache refresh. As an illustration, the command in Example 10-3 results in all the entries in the package authorization cache for user USER01 being deleted.

Example 10-3 RACF PERMIT DELETE command

```
PERMIT SYS1.** ID(USER01) DELETE CLASS(MDSNPK)
```

- Revoking **DSNADM** class authority or **MDSNSM** class **SQLADM** authority:
  - ID(*) is not supported for cache refresh. The ENF signal notification is ignored.
  - When ID (`auth-id`) is specified, all the entries in the caches for the specified `auth-id` can be deleted.

DB2 also checks for static package dependency and invalidates the package when one of the following resource class permissions is removed from the user:

- **INSERT**, **UPDATE**, **DELETE**, or **SELECT** on a table
- **USAGE** on a sequence
- **EXECUTE** on a stored procedure
- **EXECUTE** on an UDF (Dependent packages are marked inoperative.)
- **READ** or **WRITE** on a global variable

If **EXECUTE** on a package is revoked from the user, DB2 will check for plan dependency and invalidates the plan.

A package can be invalidated only when DB2 is active during ENF notification and if the name of the affected RACF profile contains discrete characters. ENF notification ignores a profile if it is associated with the **DSNADM** class or if its name contains any generic characters (*, **, &, %).

If the package owner is a user (not a RACF group) and if the user is associated with a group that had the required privileges when the package was bound, you need to explicitly permit the user all the privileges required for invalidating the package and then delete the permissions in RACF.

Note: Static package invalidation for the revoked privilege is supported with some restrictions, if a group associated with the user allowed access.

DB2 11 introduces the **AUTHEXIT_CACHEREFRESH** installation parameter to support the implementation of this function. This system parameter specifies whether the cache entries of the package authorization, the routine authorization, the DDF user authentication, and the dynamic statement are refreshed and whether the dependent packages are invalidated when a user profile or resource access is changed in RACF. The cache entries are refreshed only when the access control authorization exit (DSNX@XAC) is active

This parameter supports the following values:

ALL	Specifies that DB2 refreshes the cache entries of the package authorization, the routine authorization, and the dynamic statement and invalidates dependent packages when the user profile or resource access is changed in RACF.
NONE	Specifies that DB2 does not refresh the cache entries of the package authorization, the routine authorization, and the dynamic statement or invalidate dependent packages when the user profile or resource access is changed in RACF.

The default value is NONE. This install parameter is not online updatable. It is part of the **DSN6SPRM** macro. This parameter in added to the DB2 Protection panel, DSNTIPP.

When access control authorization exit is active and system parameter, **AUTHEXIT_CACHEREFRESH** is set to YES, then DB2 listens to the type 62, type 71, and type 79 ENF signals issued by RACF for user profile or resource access changes and refreshes the DB2 cache entries accordingly. If you are defining classes for DB2 objects and administrative authorities and not using IBM supplied RACF resource classes for DB2, then you need to enable the SIGNAL=YES option for these classes.

RACF access control module (DSNXRXAC) support

To support new functionality, DB2 11 introduces the following changes in the RACF access control module (**DSNXRXAC**):

- Support the Global Variable privileges, **READ (READAUTH)** and **WRITE (WRITEAUTH)**. IBM supplied RACF resource class for global variable is **MDSNGV/GDSNGV**.
- Return the **RACLISTED** classes at DB2 start in the new XAPL field, **XAPLCLST**.
- Support all authorization checks that are associated with AUTOBIND requests for user-defined functions. This removes the return code 8 and reason code 17 issued for the authorization failures associated with **AUTOBIND** requests for user-defined functions.

DB2 11 introduces the IFCID 386 for the serviceability of RACF ENF signal processing. This trace is written when DB2 receives the ENF signal that is processed by DB2.

The explanation of the SQLCODE -551 (auth-id DOES NOT HAVE THE PRIVILEGE TO PERFORM OPERATION operation ON OBJECT object-name) is enhanced as shown in Example 10-4.

Example 10-4 SQLCODE -551 explanation and RACF changes

```
This error might occur for packages that are bound with the DYNAMICRULES(BIND)
option when authorization caching, statement caching, or both are enabled and if
the following conditions exist:
   The access control authorization exit routine is active
   The AUTHEXIT_CHECK system parameter is set to PRIMARY
```

```
   The authorization ID of the process does not have the necessary privileges.
If the access control authorization exit is active and the AUTHEXIT_CHECK system
parameter is set to DB2, this error might occur if ACEE cannot be created for the
authorization ID auth-id.
```

The “Explanation” and “System programmer response” sections are updated in the **DSNT210I**, **DSNT235I**, **DSNT241I**, and **DSNX101I** message. Example 10-5 shows the changes in the **DSNT210I** message.

Example 10-5 Updates to DB2 message DSNT210I

```
...
Explanation
The indicated authorization ID does not have the indicated privilege and therefore
cannot invoke the indicated BIND subcommand against the indicated application
plan. If the access control authorization exit is active and the AUTHEXIT_CHECK
system parameter is set to DB2, this error might occur if ACEE cannot be created
for the indicated authorization ID.
....
System programmer response
If the indicated privilege is BINDADD, then the privilege to invoke the BIND
subcommand with the ACTION(ADD) option must be granted to the indicated
authorization ID. If the indicated privilege is BIND, the privilege to invoke a
BIND subcommand against the indicated application plan must be granted to the
indicated authorization ID.

If you use the access control authorization exit, ensure that the indicated
authorization ID is defined in RACF and granted the indicated privilege in RACF.
```

DB2 introduces the following DB2 messages:

- **DSNX235I**
- **DSNX236I**
- **DSNX237I**

Their contents are listed in Example 10-6, Example 10-7, and Example 10-8.

Example 10-6 DSNX235I

```
REGISTRATION OF THE RACF ENF MESSAGE LISTENER EXIT WITH THE ENF FACILITY FOR ENF
enf-signal FAILED WITH RETURN CODE return-code.

Explanation
The process that registers DB2 with the RACF Event Notification Facility (ENF)
message listener exit for listening to the specified ENF signal returned an
unexpected return code.

enf-signal
The possible values are 62, 71, and 79.

return-code
Hexadecimal value of the z/OS ENFREQ macro return code.

System action
DB2 is not registered to listen to the specified ENF signal notification. DB2
continues its startup.
```

System administrator response
Notify the security administrator. Restart DB2 after the ENF signal notification problem is fixed.

Security administrator response
Make sure that DB2 is registered to listen to the specified ENF signal notification.

Operator response
No action is required.

Example 10-7 DSNX236I

DSNX236I
A RESOURCE resource-name TYPE OF RESOURCE resource-type FOR PROCESSING ENF SIGNAL FOR AUTHID authid OPERATION operation ON OBJECT object-name IS NOT AVAILABLE FOR REASON reason-code. ENF SIGNALS RECEIVED FOR CLASS class-name ARE NOT PROCESSED FOR PACKAGE INVALIDATION.

Explanation
The ENF signal process for package invalidation has failed because a required resource resource-name is not available. The ENF signals received for the class class-name are not processed.

resource-name
The name of the resource.

resource-type
The type of the resource.

authorization-ID
The authorization identifier that is identified in the message. The authorization-ID can be a role.

operation
The operation that is performed.

object-name
The name of the object. If the operation is EXECUTE PACKAGE, the object name consists of the collection ID and the package name. For all other operations, the object name consists of the schema name and the object name.

reason-code
A numeric value that indicates the reason for the failure of the operation.

class-name
The name of the RACF resource class for DB2 objects and administrative authorities.

System action
The ENF signal processing continues.

System administrator response

```
Ensure that the required resource is available for the ENF signal process.
Manually restart the package invalidation process. Notify the security
administrator.

Security administrator response
Identify the RACF commands that were issued to remove resource access for the
specified RACF class. Permit the user access to the identified resources and then
delete the permissions in RACF.

Operator response
No action is required.
```

Example 10-8 DSNX237I

```
DSNX237I
AN ABEND HAS OCCURRED DURING ENF SIGNAL PROCESSING. ENF SIGNALS RECEIVED FOR CLASS
class-name ARE NOT PROCESSED.

Explanation
An abend has occurred in DB2 when processing the ENF signal received from the
security server. The ENF signals received for the class-name are not processed.

class-name
RACF resource class for DB2 objects and administrative authorities

System action
The ENF signal processing continues.

System administrator response
Manually restart the cache refresh and package invalidation processes. Notify the
security administrator.

Security administrator response
Identify the RACF commands that were issued to remove resource access for the
specified RACF class. Permit the user access to the identified resources and then
delete the permissions in RACF.

Operator response
No action is required.
```

Performance expectations

Performance is expected to improve when package owner is used for dynamic SQL authorization checking when **DYNAMICRULES(BIND)** is in effect, because the package owner ID along with the authorization information will be cached with the dynamic SQL statements for subsequent executions after initial invocations, DB2 will no longer need to go through RACF to check runner's SQL ID at run time. It is expected that, performance wise, using DB2 authorization is now similar to `AUTHEXIT_CHECK = DB2` and slower than `AUTHEXIT_CHECK = PRIMARY`. Also some improvements are expected in both statement level cache and package cache.

10.2 Enhancements to program authorization

DB2 11 provides an approach to verify that an application program using a DB2 application plan is correct when accessing DB2. An DB2 plan relates an application process to a local instance of DB2 and specifies processing options. One of the options is a list of package names that can be used by the application plan.

This list controls packages that can be used by any program that uses the plan. It is provided by the owner of the plan when the plan is bound to DB2. If any user is granted execute privilege on the plan, the user can execute any program using the plan and any package identified in the package list.

Plan owners might not know in advance which programs or packages that might use a plan. In these cases, the plan owner must create a plan that allows any collection or any package to be used by any program executing the plan. If a user has execute authority to run the plan, the user can accidentally invoke the wrong program or change the application program to execute different packages. A user can execute packages the plan owner never intended to be called. To protect from these types of mistakes, a new bind option is added to the **BIND PLAN** and **REBIND PLAN** commands. When the new option is set, DB2 performs an extra check when the program identifies to DB2. The check verifies if the program is a valid program that can execute the plan.

Important: DB2 11 program authorization provides an approach to verify that an application program using a DB2 application plan is correct when accessing DB2.

Program authorization is a useful technique when you do not know all of the programs and packages that might use a plan. In addition, program authorization lets you determine at the time that a program is loaded whether it has been modified. Program authorization is performed in addition to package authorization.

Some restrictions apply. Programs that run in the following environments do not support program authorization:

- RRSAF applications that issue **CREATE THREAD** with a collection name and, therefore, use the special default plan name **?RRSAF**.
- Multicontext ODBC applications that use the RRSAF attachment facility and the plan name DSNACLI
- Programs that run in stored procedure address spaces

Enabling program authorization

Program authorization is enabled for a program and its plan if the following conditions are true:

- The plan is bound with the new **BIND PLAN** and **REBIND PLAN** option **PROGAUTH(ENABLE)**
- The **SYSIBM.DSNPROGAUTH** table contains a row for the program and the plan.

BIND and REBIND option PROGAUTH

The **PROGAUTH BIND** and **REBIND** option specifies whether DB2 performs program authorization checking to determine whether DB2 can execute a plan. It accepts the following values:

DISABLE	Specifies that program authorization checking is not performed.
ENABLE	Specifies that program authorization checking is performed.

The default value for **BIND PLAN** is **DISABLE**. The default value for **REBIND PLAN** is the existing value. This option is not applicable for **BIND** and **REBIND PACKAGE**.

Figure 10-3 shows a partial representation of the DB2 11 **REBIND PLAN** command, including the new **PROGAUTH** option.

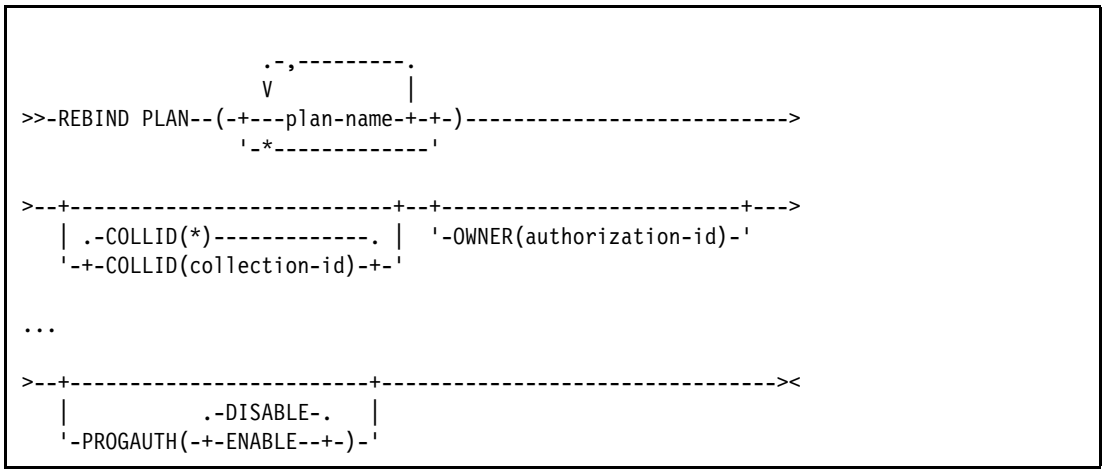


Figure 10-3 DB2 11 REBIND PLAN command and PROGAUTH

DB2 11 updates the **SYSIBM.SYSPLAN** catalog table to include the new **PROGAUTH** column. This column is defined as **CHAR(1) NOT NULL WITH DEFAULT 'D'**. It indicates DB2 to check program association with the plan. It takes the following values:

E	Verify if program is enabled to execute the plan
D	Disabled

The default value is D, disabled.

Example 10-9 shows a simple example of a **REBIND** using a DB2 **PLAN**, **DSNTIA11**, to implement DB2 program authorization by using the **PROGAUTH REBIND** option.

Example 10-9 REBIND to enable PROGAUTH

```
//YOUR_JOB_CARD_COMES_HERE...
//*--------------------------------------------------------------------
//REBIND   EXEC PGM=IKJEFT01,DYNAMNBR=20
//STEPLIB  DD   DISP=(SHR),DSN=DB1AT.SDSNLOAD
//SYSTSPRT DD   SYSOUT=*
//SYSPRINT DD   SYSOUT=*
//SYSOUT   DD   SYSOUT=*
//SYSTSIN  DD   *
DSN  SYSTEM(DB1A)

REBIND PLAN      (DSNTIA11)    +
     PROGAUTH    (E)
END
/*
```

Table SYSIBM.DSNPROGAUTH

The **SYSIBM.DSNPROGAUTH** table enables program authorization with or without program data integrity checking. Add a row in the **SYSIBM.DSNPROGAUTH** table for each program and plan combination for which the plan is bound with **PROGAUTH(ENABLE)**.

The program name that you need to insert in the row depends on the attachment facility that the program uses to connect to DB2, as follows:

- If the program uses the TSO attachment facility, the program name is the name that you specify in the DSN **RUN** subcommand.
- If the program uses any other attachment facility, the program name is the name of the module that is executed first under the job step TCB.

The **DSNTIJSG** job contains a sample **INSERT** statement for a **SYSIBM.DSNPROGAUTH** row. You can modify the **INSERT** statement and execute it to add a row for a program and plan. The **SYSIBM.DSNPROGAUTH** program authorization table is used to verify that a program is authorized to use a plan. The table is created by the DB2 provided the **DSNTIJSG** installation job.

Example 10-10 illustrates a DDL that can be used for creating this table.

Example 10-10 DDL for creating the table SYSIBM.DSNPROGAUTH

```
CREATE DATABASE DSNMDCDB STOGROUP SYSDEFLT CCSID UNICODE;

CREATE TABLESPACE DSNMDCTS IN DSNMDCDB
      BUFFERPOOL BPO LOCKSIZE ROW LOCKMAX SYSTEM
       CLOSE NO CCSID UNICODE  USING STOGROUP SYSDEFLT;

CREATE TABLE SYSIBM.DSNPROGAUTH
       ( "PROGNAME"              VARCHAR(24)   NOT NULL
       , "PLANNAME"              VARCHAR(24)   NOT NULL
       , "PROGMDCVAL"            CHAR(16)      FOR BIT DATA
                                               NOT NULL
                                               WITH DEFAULT
                                 X'00000000000000000000000000000000'
       , "PROGMDCPAD"            CHAR(1)       NOT NULL
                                               WITH DEFAULT '2'
                                               CHECK(PROGMDCPAD = '2'
                                                  OR PROGMDCPAD = '4')
       , "CREATOR"               VARCHAR(128)  NOT NULL
                                               WITH DEFAULT
                                               CURRENT SQLID
       , "ENABLED"               CHAR(1)       NOT NULL
                                               WITH DEFAULT 'N'
                                               CHECK(ENABLED = 'Y'
                                                  OR ENABLED = 'N')
       , "CREATETS"              TIMESTAMP     NOT NULL WITH DEFAULT
       , "REMARKS"               VARCHAR(762)
       )
       IN DSNMDCDB.DSNMDCTS CCSID UNICODE;

CREATE UNIQUE INDEX SYSIBM.DSNPROGAUTH_IDX1
       ON SYSIBM.DSNPROGAUTH
       ( "PROGNAME"
       , "PLANNAME"
       )
       BUFFERPOOL BPO  CLOSE NO  USING STOGROUP SYSDEFLT;
```

Table 10-1 shows the **SYSIBM.DSNPROGAUTH** column's description.

Table 10-1 SYSIBM.DSNPROGAUTH fields description

Column name	Description
PROGNAME	Name of the application program enabled to run the plan

Column name	Description
PLANNAME	Name of the application plan that the program can execute
PROGMDCVAL	Reserved
PROGMDCPAD	Reserved
CREATOR	Authorization ID that inserted or last modified the row
ENABLED	Indicates whether the program authorization is enabled. This column can have one of the following values: Y Program authorization is enabled N Program authorization is disabled
CREATETS	Time when the row was inserted or updated
REMARKS	Comments about this program authorization record

Table 10-2 shows the **SYSIBM.DSNPROGAUTH** column's data type and default values.

Table 10-2 SYSIBM.DSNPROGAUTH fields description

Column name	Data type and default value
PROGNAME	VARCHAR(24) NOT NULL
PLANNAME	VARCHAR(24) NOT NUL
PROGMDCVAL	CHAR(16) NOT NULL FOR BIT DATA WITH DEFAULT
PROGMDCPAD	CHAR(1) NOT NULL WITH DEFAULT '2'
CREATOR	ARCHAR(128) NOT NULL WITH DEFAULT CURRENT SQLID
ENABLED	CHAR(1) NOT NULL WITH DEFAULT 'N'
CREATETS	TIMESTAMP NOT NULL WITH DEFAULT
REMARKS	VARCHAR(762)

SYSIBM.DSNPROGAUTH is a user maintained table. A sample **INSERT** statement is provided in the DB2 provided **DSNTIJSG** job, as shown in Example 10-11.

Example 10-11 DSNTIJSG extract: sample INSERT in SYSIBM.DSNPROGAUTH

```
//* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
//* Here is a sample insert statement for the DSNPROGAUTH table.
//* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
//*  INSERT INTO SYSIBM.DSNPROGAUTH
//*              ( "PROGNAME"
//*              , "PLANNAME"
//*              , "PROGMDCVAL"
//*              , "PROGMDCPAD"
//*              , "CREATOR"
//*              , "ENABLED"
//*              , "CREATETS"
//*              , "REMARKS"
//*              )
//*        VALUES( 'DSNTIAD'
//*              , 'DSNTIA!!'
//*              , X'00000000000000000000000000000000'
//*              , '2'
```

```
//*              , CURRENT SQLID
//*              , 'N'
//*              , CURRENT TIMESTAMP
//*              , 'EXAMPLE DSNPROGAUTH ENTRY (DISABLED)'
//*              );
//*
```

Problem determination

To support the program authorization functionality, DB2 11 adds the following error reason codes:

00F3003A An error occurred while processing the Program Name parameter. This parameter was provided by the attachment facility on a request to allocate a DB2 plan to the application. Either an abend occurred accessing the Program Name or the starting character of the parameter string is out of range. As a result, the request is not processed.

00F3003B The program authorization is enabled by specifying **PROGAUTH** option when the plan was bound. The program name associated with this connection is not authorized to use the specified plan name. The request to allocate a plan to the program name is denied.

00E70026 The program authorization is enabled by specifying **PROGAUTH** option when the plan was bound. The **SYSIBM.DSNPROGAUTH** program name validation table or **SYSIBM.DSNPROGAUTH_INDX1** index do not exist. The request to allocate the plan is not processed.

00E70028 The program authorization is enabled by specifying the **PROGAUTH** option when the plan was bound. The **SYSIBM.DSNPROGAUTH** program name validation table or **SYSIBM.DSNPROGAUTH_INDX1** index is not defined correctly. The request to allocate the plan is not processed.

The DB2 DSNT252I **message** is updated to display the **PROGAUTH** option. This message shows the **BIND** or **REBIND** options that were used for the plan during bind or rebind processing. Example 10-12 shows the message **DSNT252I**.

Example 10-12 REBIND PLAN output showing PROGAUTH enabled

```
READY
DSN  SYSTEM(DB1A)
DSN
DSN
REBIND PLAN      (DSNTIA11)    PROGAUTH    (E)
WARNING, ONLY IBM-SUPPLIED PLAN SHOULD BEGIN WITH  "DSN"
DSNT252I  -DB1A DSNTBRB REBIND OPTIONS FOR PLAN DSNTIA11
           ACTION
           OWNER         SYSADM
           VALIDATE      RUN
           ISOLATION     CS
           ACQUIRE       USE
           RELEASE       COMMIT
           EXPLAIN       NO
           DYNAMICRULES  RUN
           PROGAUTH      ENABLE
DSNT253I  -DB1A DSNTBRB REBIND OPTIONS FOR PLAN DSNTIA11
           NODEFER       PREPARE
           CACHESIZE     3072
           QUALIFIER     SYSADM
           CURRENTSERVER
```

```
           CURRENTDATA   YES
           DEGREE        1
           SQLRULES      DB2
           DISCONNECT    EXPLICIT
           REOPT         NONE
           KEEPDYNAMIC   NO
           IMMEDWRITE    NO
           DBPROTOCOL    DRDA
           OPTHINT
           ENCODING      EBCDIC(00037)
           CONCURRENTACCESSRESOLUTION
           PATH
DSNT200I  -DB1A  REBIND FOR PLAN DSNTIA11 SUCCESSFUL
DSN
END
READY
END
```

Implementation example

This section describes the steps involved in enabling a DB2 11 program authorization for a existing PLAN with the following objectives:

- To limit and secure the packages that can be executed through a PLAN
- To guaranty that a package is not altered

The DB2 provided JCL **DSNTIJTM**, that is part of the DB2 installation stream, performs the **BIND** of the **DSNTIAD** package and its **PLAN**, **DSNTIA11** for DB2 11. Example 10-13 shows the JCL used for this example.

Example 10-13 BIND PLAN DSNTIA11 in job DSNTIJTM

```
//DSNTIAS EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//SYSTSPRT DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//SYSTSIN  DD  *
  DSN SYSTEM(DB1A)
  BIND PACKAGE(DSNTIA11) MEM(DSNTIAD) -
       ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC) -
       LIB('DB1AM.DBRMLIB.DATA')
  BIND PLAN(DSNTIA11) PKLIST(DSNTIA11.DSNTIAD) -
       ACTION(REPLACE) RETAIN +
       ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
  END
//*
```

Example 10-14 shows a portion of the output of the **BIND PLAN DSNTIA11** command. This example exposes the defaults used in this example environment. It also highlights the **PROGAUTH** keyword in the **DSNT252I** message. Because **PROGAUTH** was not specified during **BIND**, the resulting value for this option is **DISABLE**, as shown in this example.

Example 10-14 BIND PLAN DSNTIA11 output

```
DSN
  BIND PLAN(DSNTIA11) PKLIST(DSNTIA11.DSNTIAD)
  ACTION(REPLACE) RETAIN ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
WARNING, ONLY IBM-SUPPLIED PLAN SHOULD BEGIN WITH  "DSN"
DSNT252I  -DB1A DSNTBCM1 BIND OPTIONS FOR PLAN DSNTIA11
           ACTION        REPLACE RETAIN
```

```
           OWNER         DB2R1
           VALIDATE      RUN
           ISOLATION     CS
           ACQUIRE       USE
           RELEASE       COMMIT
           EXPLAIN       NO
           DYNAMICRULES  RUN
           PROGAUTH      DISABLE
DSNT253I  -DB1A DSNTBCM1 BIND OPTIONS FOR PLAN DSNTIA11
           NODEFER       PREPARE
           CACHESIZE     3072
           QUALIFIER     DB2R1
           CURRENTSERVER
           CURRENTDATA   YES
           DEGREE        1
           SQLRULES      DB2
           DISCONNECT    EXPLICIT
           NOREOPT       VARS
           KEEPDYNAMIC   NO
           IMMEDWRITE    NO
           DBPROTOCOL    DRDA
           OPTHINT
           ENCODING      EBCDIC(00037)
           CONCURRENTACCESSRESOLUTION
           PATH
DSNT200I  -DB1A  BIND FOR PLAN DSNTIA11 SUCCESSFUL
```

A SQL query on the DB2 **SYSPLAN** catalog table allows to check the value of **PROGAUTH**, as shown in Example 10-15. The query result of this example is consistent with the DB2 **DSNT252I** message obtained during the **BIND PLAN** command execution.

Example 10-15 SYSPLAN query to show a PLAN's PROGAUTH value

```
---------+---------+---------+---------+---------+---------+---------+---------+---------+
SELECT
  NAME, VALID,OPERATIVE,
  CAST(QUALIFIER AS CHAR(10)) AS QUALIFIER,
  RELBOUND, PROGAUTH, PLENTRIES
FROM SYSIBM.SYSPLAN
WHERE NAME = 'DSNTIA11'
WITH UR
---------+---------+---------+---------+---------+---------+---------+---------+---------+
NAME                 VALID  OPERATIVE  QUALIFIER   RELBOUND  PROGAUTH  PLENTRIES
---------+---------+---------+---------+---------+---------+---------+---------+---------+
DSNTIA11             Y      Y          DB2R1       P         D                1
DSNE610I NUMBER OF ROWS DISPLAYED IS 1
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
---------+---------+---------+---------+---------+---------+---------+---------+---------+
```

In this example, the **PLENTRIES** column indicates that the number of package list entries in the **SYSIBM.SYSPACKLIST** table is 1 for this plan. The **SYSIBM.SYSPACKLIST** table contains one or more rows for every local application plan bound with a package list. Each row represents a unique entry in the plan's package list.

Example 10-16 shows an example of SQL query on **SYSIBM.SYSPACKLIST** to find the relationship between a PLAN and a package list (or collection).

Example 10-16 SQL query on SYSIBM.SYSPACKLIST

```
SELECT
```

```
  PLANNAME, CAST(COLLID AS CHAR(10)) AS COLLID, NAME
FROM SYSIBM.SYSPACKLIST
WHERE PLANNAME = 'DSNTIA11'
WITH UR;
---------+---------+---------+---------+---------+---------+
PLANNAME                 COLLID      NAME
---------+---------+---------+---------+---------+---------+
DSNTIA11                 DSNTIA11    DSNTIAD
DSNE610I NUMBER OF ROWS DISPLAYED IS 1
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
```

The **COLLID** column indicates the collection name for the package. An asterisk (*) indicates that the collection name is determined at run time. This example explicitly defines the collection **DSNTIA11**. The column NAME indicates the name of the package. An asterisk (*) indicates an entire collection. Example 10-17 shows an example of SQL query on SQL query on **SYSIBM.SYSPACKAGE**.

Example 10-17 SQL query on SYSIBM.SYSPACKAGE

```
---------+---------+---------+---------+---------+---------+
SELECT
  CAST(COLLID AS CHAR(10)) AS COLLID,
  CAST(NAME AS CHAR(10)) AS NAME,
  VALID, OPERATIVE, APPLCOMPAT
FROM SYSIBM.SYSPACKAGE
WHERE COLLID = 'DSNTIA11'
WITH UR;
---------+---------+---------+---------+---------+---------+
COLLID      NAME        VALID  OPERATIVE  APPLCOMPAT
---------+---------+---------+---------+---------+---------+
DSNTIA11    DSNTIAD     Y      Y          V11R1
DSNE610I NUMBER OF ROWS DISPLAYED IS 1
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
---------+---------+---------+---------+---------+---------+
```

To enable program authorization control, the following minimum information for the **SYSIBM.DSNPROGAUTH** table is required:

PROGNAME	The program name
PLANNAME	The PLAN name
ENABLED	Insert value **Y** to enable program authorization

The following columns are not explicitly populated for this example:

- **PROGMDCVAL**: Modification detection code (MDC) value of the application program. The default value (a string of binary zeros) specifies that MDC is not to be used.
- **PROGMDCPAD**: The type of padding used to calculate the MDC value. Not applicable in this example, as per the value for **PROGMDCVAL**.
- **CREATOR**: The default for this column, **CURRENT SQLID**, is OK for the purposes of this example.
- **CREATETS**: The default for this column, **CURRENT TIMESTAMP**, is OK for the purposes of this example.
- The column **REMARKS** can be used for documentation purposes.

Example 10-18 shows the SQL used for this test.

Example 10-18 INSERT SQL on SYSIBM.DSNPROGAUTH

```
INSERT INTO SYSIBM.DSNPROGAUTH
            ( "PROGNAME"
            , "PLANNAME"
            , "REMARKS"
            )
      VALUES( 'DSNTIAD'
            , 'DSNTIA11'
            , 'DB2 11 program authorization test1'
            );
```

Example 10-19 shows the **INSERT** results as reported with a SQL query on **SYSIBM.DSNPROGAUTH**. Notice the value N for the **ENABLED** column.

Example 10-19 SQL query on SYSIBM.DSNPROGAUTH

```
---------+---------+---------+---------+---------+---------+---------+---
SELECT
  CAST(PROGNAME AS CHAR(10)) AS PROGRNAME,
  CAST(PLANNAME AS CHAR(10)) AS PLANNAME,
  PROGMDCVAL, PROGMDCPAD,
  CAST(CREATOR AS CHAR(10)) AS CREATOR,
  ENABLED
FROM SYSIBM.DSNPROGAUTH
WITH UR;
---------+---------+---------+---------+---------+---------+---------+---
PROGRNAME   PLANNAME    PROGMDCVAL        PROGMDCPAD  CREATOR     ENABLED
---------+---------+---------+---------+---------+---------+---------+---
DSNTIAD     DSNTIA11    .................  2           DB2R1       N
DSNE610I NUMBER OF ROWS DISPLAYED IS 1
```

Example 10-20 shows the JCL used to invoke **DSNTIAD** for testing the program authentication functionality.

Example 10-20 Testing program authentication with DSNTIAD

```
//*--------------------------------------------------------------------
//RUNTIAD  EXEC PGM=IKJEFT01,DYNAMNBR=20
//STEPLIB  DD DISP=(SHR),DSN=DB1AT.SDSNLOAD
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
 DSN SYSTEM(DB1A)
 RUN  PROGRAM(DSNTIAD) PLAN(DSNTIA11) -
       LIB('DB1AM.RUNLIB.LOAD')
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN    DD *
  SET CURRENT SQLID = 'SYSADM';
  CREATE VARIABLE CRISTIAN.TEMPVAR01 CHAR(10) DEFAULT 'NOT_INIT';
/*
```

The execution of the program fails as reported in Example 10-21.

Example 10-21 Program authentication preventing execution

```
READY
 DSN SYSTEM(DB1A)
DSN
 RUN  PROGRAM(DSNTIAD) PLAN(DSNTIA11)        LIB('DB1AM.RUNLIB.LOAD')
```

```
PLAN DSNTIA11 NOT AUTHORIZED FOR SUBSYSTEM DB1A AND AUTH ID DB2R1
DSN
END
 DSNTIAD  - SAMPLE DYNAMIC SQL PROGRAM 2.0

           SET CURRENT SQLID = 'SYSADM'
 SQL ERROR DURING EXECUTE IMMEDIATE
DSNT408I SQLCODE = -924, ERROR:  DB2 CONNECTION INTERNAL ERROR, 0001, 0008, 00F3003B
DSNT418I SQLSTATE   = 58006 SQLSTATE RETURN CODE
DSNT415I SQLERRP    = DSNAET03 SQL PROCEDURE DETECTING ERROR
```

To allow the execution of the program, the **ENABLED** column of the **SYSIBM.DSNPROGAUTH** table has to be changed to Y, as shown in Example 10-22.

Example 10-22 Updating SYSIBM.DSNPROGAUTH to allow program execution

```
---------+---------+---------+---------+---------+---------+---------+
UPDATE SYSIBM.DSNPROGAUTH
SET ENABLED = 'Y'
WHERE PROGNAME = 'DSNTIAD';
---------+---------+---------+---------+---------+---------+---------+
DSNE615I NUMBER OF ROWS AFFECTED IS 1
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 0
---------+---------+---------+---------+---------+---------+---------+
```

Example 10-23 shows the results. Note the value Y for **ENABLED**.

Example 10-23 SQL query on SYSIBM.DSNPROGAUTH, ENABLED = 'Y'

```
SELECT
  CAST(PROGNAME AS CHAR(10)) AS PROGRNAME,
  CAST(PLANNAME AS CHAR(10)) AS PLANNAME,
  PROGMDCVAL, PROGMDCPAD,
  CAST(CREATOR AS CHAR(10)) AS CREATOR,
  ENABLED
FROM SYSIBM.DSNPROGAUTH
WITH UR;
---------+---------+---------+---------+---------+---------+---------+---------+
PROGRNAME   PLANNAME    PROGMDCVAL        PROGMDCPAD  CREATOR     ENABLED
---------+---------+---------+---------+---------+---------+---------+---------+
DSNTIAD     DSNTIA11    .............  2           DB2R1       Y
DSNE610I NUMBER OF ROWS DISPLAYED IS 1
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
```

The execution of the program is now allowed, and the results are shown in Example 10-24.

Example 10-24 Program authentication allowing execution

```
READY
 DSN SYSTEM(DB1A)
DSN
 RUN  PROGRAM(DSNTIAD) PLAN(DSNTIA11)        LIB('DB1AM.RUNLIB.LOAD')
DSN
END
 DSNTIAD  - SAMPLE DYNAMIC SQL PROGRAM 2.0

           SET CURRENT SQLID = 'SYSADM'
DSNT400I SQLCODE = 000,  SUCCESSFUL EXECUTION
```

Troubleshooting: table DSNPROGAUTH not available

This section describes the DB2 behavior when a **PLAN** is enabled for program authentication, but the DB2 **SYSIBM.DSNPROGAUTH** table is not available. The query in Example 10-25 shows the database and table space name of the **SYSIBM.DSNPROGAUTH** table as created in this test environment.

Example 10-25 SQL to find DBNAME and TSNAME of SYSIBM.DSNPROGAUTH

```
---------+---------+---------+---------+---------+---------+
SELECT
  DBNAME, TSNAME
FROM SYSIBM.SYSTABLES WHERE NAME = 'DSNPROGAUTH'
  AND OWNER = 'SYSIBM'
WITH UR;
---------+---------+---------+---------+---------+---------+
DBNAME                   TSNAME
---------+---------+---------+---------+---------+---------+
DSNMDCDB                 DSNMDCTS
DSNE610I NUMBER OF ROWS DISPLAYED IS 1
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
---------+---------+---------+---------+---------+---------+
```

To force a resource not available error, this example stops the **SYSIBM.DSNPROGAUTH** table space. The unavailability of the table space is confirmed with a display command. Example 10-26 shows the output of this command.

Example 10-26 Display status of SYSIBM.DSNPROGAUTH table space status

```
DSNT360I  -DB1A *********************************
DSNT361I  -DB1A *  DISPLAY DATABASE SUMMARY
                *    GLOBAL
DSNT360I  -DB1A *********************************
DSNT362I  -DB1A     DATABASE = DSNMDCDB  STATUS = RW
                    DBD LENGTH = 4028
DSNT397I  -DB1A
NAME     TYPE PART  STATUS            PHYERRLO PHYERRHI CATALOG  PIECE
-------- ---- ----- ----------------- -------- -------- -------- -----
DSNMDCTS TS         STOP
******* DISPLAY OF DATABASE DSNMDCDB ENDED      **********************
DSN9022I  -DB1A DSNTDDIS 'DISPLAY DATABASE' NORMAL COMPLETION
***
```

At this point, the execution of any program defined with program authentication fails, as shown in Example 10-27.

Example 10-27 SYSIBM.DSNPROGAUTH not available prevents program execution

```
READY
 DSN SYSTEM(DB1A)
DSN
 RUN  PROGRAM(DSNTIAD) PLAN(DSNTIA11)     LIB('DB1AM.RUNLIB.LOAD')
SUBSYSTEM RESOURCE NEEDED FOR PLAN DSNTIA11, AUTH ID DB2R1, AND SUBSYSTEM DB1A IS NOT
AVAILABLE
FEEDBACK - REASON CODE X'00C90081' TYPE X'00000200' RESOURCE NAME- DSNMDCDB.DSNMDCTS
DSN
END
 DSNTIAD  - SAMPLE DYNAMIC SQL PROGRAM 2.0

         SET CURRENT SQLID = 'SYSADM'
 SQL ERROR DURING EXECUTE IMMEDIATE
```

```
DSNT408I SQLCODE = -923, ERROR:  CONNECTION NOT ESTABLISHED: DB2  ACCESS, REASON 00C90081,
TYPE 00000200, NAME
       DSNMDCDB.DSNMDCTS
DSNT418I SQLSTATE   = 57015 SQLSTATE RETURN CODE
DSNT415I SQLERRP    = DSNAET03 SQL PROCEDURE DETECTING ERROR
```

The DB2 MSTR address space reports message 00C90081, as shown in Example 10-28. The 00C90081 message indicates that an attempt was made to allocate a resource that is stopped for all access.

Example 10-28 DB2 MSTR message 00C90081

```
DSNT501I  -DB1A DSNIDBET RESOURCE UNAVAILABLE 734
           CORRELATION-ID=DB2R1
           CONNECTION-ID=TSO
           LUW-ID=USIBMSC.SCPDB1A.CBB6B8D881E6=829
           REASON 00C90081
           TYPE 00000200
           NAME DSNMDCDB.DSNMDCTS
```

Important: A **PLAN** with program authentication enabled will fail the execution of a package if DB2 cannot verify the contents of **SYSIBM.DSNPROGAUTH**.

Troubleshooting: contention on table DSNPROGAUTH

To simulate locking contention on the table SYSIBM.DSNPROGAUTH, this example performs an update on the table using **SPUFI** with **AUTOCOMMIT** set to **NO**, as shown in Example 10-29.

Example 10-29 SPUFI option AUTOCOMMIT = NO

```
SPUFI                              SSID: DB1A
===>

Enter the input data set name:        (Can be sequential or partitioned)
 1  DATA SET NAME ... ===> 'DB2R1.UTIL.SECURITY(DSNPROG)'
 2  VOLUME SERIAL ... ===>             (Enter if not cataloged)
 3  DATA SET PASSWORD ===>             (Enter if password protected)

Enter the output data set name:       (Must be a sequential data set)
 4  DATA SET NAME ... ===> 'DB2R1.SPUFIOUT'

Specify processing options:
 5  CHANGE DEFAULTS   ===> YES        (Y/N - Display SPUFI defaults panel?)
 6  EDIT INPUT ...... ===> YES        (Y/N - Enter SQL statements?)
 7  EXECUTE ......... ===> YES        (Y/N - Execute SQL statements?)
 8  AUTOCOMMIT ...... ===> NO         (Y/N - Commit after successful run?)
 9  BROWSE OUTPUT ... ===> YES        (Y/N - Browse output data set?)

For remote SQL processing:
10  CONNECT LOCATION  ===>


PRESS:  ENTER to process    END to exit              HELP for more information
```

The SQL query used in this test is shown in Example 10-30. Notice the DB2 **DSNE6141** message.

Example 10-30 SQL to update SYSIBM.DSNPROGAUTH

```
---------+---------+---------+---------+---------+---------+
   UPDATE SYSIBM.DSNPROGAUTH
   SET ENABLED = 'Y'
   WHERE PROGNAME = 'DSNTIAD';
---------+---------+---------+---------+---------+---------+
DSNE615I NUMBER OF ROWS AFFECTED IS 1
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 0
---------+---------+---------+---------+---------+---------+
DSNE614I AUTOCOMMIT IS NO; NO CHANGES COMMITTED
---------+---------+---------+---------+---------+---------+-
```

As a result of the SPUFI **AUTOCOMMIT = NO** option, there are locks held by SPUFI on the **SYSIBM.DSNPROGAUTH** table until the user explicitly commits the changes. A batch program running a **PLAN** with program authentication enabled attempts to access the table previous allowing the execution of a package.

The locks are shown with a **DISPLAY** command, of which output can be seen in Example 10-31.

Example 10-31 Locks on SYSIBM.DSNPROGAUTH table space

```
DSNT360I  -DB1A **********************************
DSNT361I  -DB1A *  DISPLAY DATABASE SUMMARY
                *    GLOBAL LOCKS
DSNT360I  -DB1A **********************************
DSNT362I  -DB1A     DATABASE = DSNMDCDB  STATUS = RW
                   DBD LENGTH = 4028
DSNT397I  -DB1A
NAME     TYPE PART  STATUS            CONNID   CORRID       LOCKINFO
-------- ---- ----- ----------------- -------- ------------ ---------
DSNMDCTS TS         RW                BATCH    DB2R1S       H-IS,S,C
    -                  AGENT TOKEN 851
DSNMDCTS TS         RW                TSO      DB2R1        H-IX,S,C
    -                  AGENT TOKEN 844
3        TB                           BATCH    DB2R1S       H-IS,T,C
    -                  AGENT TOKEN 851
3        TB                           TSO      DB2R1        H-IX,T,C
    -                  AGENT TOKEN 844
******* DISPLAY OF DATABASE DSNMDCDB ENDED      **********************
DSN9022I  -DB1A DSNTDDIS 'DISPLAY DATABASE' NORMAL COMPLETION
***
```

This example show the locks originated by the TSO user DB2R1, using SPUFI, and the batch correlation ID DB2R1S. The batch program fails as shown in Example 10-32.

Example 10-32 Program failure due to locks on SYSIBM.DSNPROGAUTH

```
READY
 DSN SYSTEM(DB1A)
DSN
 RUN  PROGRAM(DSNTIAD) PLAN(DSNTIA11)     LIB('DB1AM.RUNLIB.LOAD')
SUBSYSTEM RESOURCE NEEDED FOR PLAN DSNTIA11, AUTH ID DB2R1, AND SUBSYSTEM DB1A IS NOT
AVAILABLE
FEEDBACK - REASON CODE X'00C9008E' TYPE X'00000304' RESOURCE NAME-
DSNMDCDB.DSNMDCTS.X'000002' '.X'01'
DSN
END
 DSNTIAD  - SAMPLE DYNAMIC SQL PROGRAM 2.0
```

```
         SET CURRENT SQLID = 'SYSADM'
 SQL ERROR DURING EXECUTE IMMEDIATE
DSNT408I SQLCODE = -923, ERROR:  CONNECTION NOT ESTABLISHED: DB2  ACCESS, REASON 00C9008E,
TYPE 00000304, NAME
       DSNMDCDB.DSNMDCTS.X'000002' '.X'01'
DSNT418I SQLSTATE   = 57015 SQLSTATE RETURN CODE
DSNT415I SQLERRP    = DSNAET03 SQL PROCEDURE DETECTING ERROR
```

Note the DB2 **00C9008E** message. It indicates that a lock request cannot be granted, and the request waited for a period longer than the maximum specified by the installation. As indicated by **NAME** in this message, the lock was not obtained on the **SYSIBM.DSNPROGAUTH** table.

Attention: Contention on the **SYSIBM.DSNPROGAUTH** table can cause application performance or application availability problems

10.3 Column masking enhancements

Row and column access control enables you to manage access to a table at the level of a row, a column, or both. You can implement row access control through row permissions and column access control through column masks.

A column mask is a database object that describes a specific column access control rule for a column. In the form of an SQL **CASE** expression, the rule specifies the condition under which a user, group, or role can receive the masked values that are returned for a column.

If the **SEPARATE_SECURITY** system parameter is set to **YES**, you must have the **SECADM** authority to create a column mask. If **SEPARATE_SECURITY** is set to **NO**, you must have the **SECADM** or **SYSADM** authority.

Example 10-33 shows a sample DDL and DML code that you can use to test column masks. This code creates a simple table and add some sample records into it.

Example 10-33 Sample table and data for column mask example

```
CREATE TABLE CLIENTS
      (NAME      CHAR(10),
       COUNTRY   CHAR(10),
       PHONE#    CHAR(10));

INSERT INTO CLIENTS VALUES ('CRISTIAN','BELGIUM','+321234567');
INSERT INTO CLIENTS VALUES ('FERNANDO','SPAIN  ','+331234567');
INSERT INTO CLIENTS VALUES ('TATIANA ','BELGIUM','+341234567');
INSERT INTO CLIENTS VALUES ('MARTINA ','ITALY  ','+391234567');
INSERT INTO CLIENTS VALUES ('KATRIN  ','GERMANY','+321234567');
```

Example 10-34 shows the SQL that can be used to create 2 column masks on the table. With these examples, DB2 will mask the values of the columns **COUNTRY** and **PHONE#** for users other than CRIS.

Example 10-34 Creating a column mask

```
CREATE MASK COUNTRY_MASK ON CLIENTS
    FOR COLUMN COUNTRY
    RETURN
```

```
    CASE WHEN CURRENT SQLID = 'CRIS'
         THEN COUNTRY
         ELSE CHAR('----------')
         END
    ENABLE;

CREATE MASK PHONE#_MASK ON CLIENTS
    FOR COLUMN PHONE#
    RETURN
    CASE WHEN CURRENT SQLID = 'CRIS'
         THEN PHONE#
         ELSE SUBSTR(PHONE#, 1, 3) || CHAR('-XXX-XX')
         END
    ENABLE;
```

To activate column access control on this table, issue the ALTER statement shown in Example 10-35.

Example 10-35 Activating column access control

```
ALTER TABLE CLIENTS ACTIVATE COLUMN ACCESS CONTROL;
```

Example 10-36 shows the results of executing the same **SELECT** statement with different SQLIDs on a table with column access control activated.

Example 10-36 Column access control effects on SELECT

```
---------+---------+---------+---------+---------+---------+---------+
SET CURRENT SCHEMA = 'CRIS';
---------+---------+---------+---------+---------+---------+---------+
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 0
---------+---------+---------+---------+---------+---------+---------+
SET CURRENT SQLID =  'CRIS';
---------+---------+---------+---------+---------+---------+---------+
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 0
---------+---------+---------+---------+---------+---------+---------+
SELECT * FROM CLIENTS;
---------+---------+---------+---------+---------+---------+---------+
NAME        COUNTRY     PHONE#
---------+---------+---------+---------+---------+---------+---------+
CRISTIAN    BELGIUM     +321234567
FERNANDO    SPAIN       +331234567
TATIANA     BELGIUM     +341234567
MARTINA     ITALY       +391234567
KATRIN      GERMANY     +321234567
DSNE610I NUMBER OF ROWS DISPLAYED IS 5
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
---------+---------+---------+---------+---------+---------+---------+

SET CURRENT SQLID = 'TOTO';
---------+---------+---------+---------+---------+---------+---------+
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 0
---------+---------+---------+---------+---------+---------+---------+
SELECT * FROM CLIENTS;
---------+---------+---------+---------+---------+---------+---------+
NAME        COUNTRY     PHONE#
---------+---------+---------+---------+---------+---------+---------+
CRISTIAN    ----------  +32-XXX-XX
FERNANDO    ----------  +33-XXX-XX
TATIANA     ----------  +34-XXX-XX
```

```
MARTINA     ----------  +39-XXX-XX
KATRIN      ----------  +32-XXX-XX
DSNE610I NUMBER OF ROWS DISPLAYED IS 5
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
---------+---------+---------+---------+---------+---------+---------+
```

To deactivate column access control on this example table, issue the **ALTER** statement shown in Example 10-37.

Example 10-37 Deactivating column access control

```
ALTER TABLE CLIENTS DEACTIVATE COLUMN ACCESS CONTROL;
```

10.3.1 Remove column access control restrictions for GROUP BY

In DB2 10 for z/OS, column access control (column mask) is restricted for GROUP BY in the statements for some scenarios. After definition and activation of column access control on a table, users might receive SQLCODE -20478 reason codes 3, 4, 5, or 6 at query execution. An abstract of the SQLCODE -20478 message definition is shown in Example 10-38.

Example 10-38 SQLCODE -20478

```
THE STATEMENT CANNOT BE PROCESSED BECAUSE COLUMN MASK mask-name (DEFINED FOR
COLUMN column-name) EXISTS AND THE COLUMN MASK CANNOT BE APPLIED OR THE DEFINITION
OF THE MASK CONFLICTS WITH THE REQUESTED STATEMENT. REASON CODE reason-code.
```

In DB2 10, the user can receive one of the following reason codes:

- **3:** The column is also referenced in a grouping expression of the **GROUP BY** clause.
- **4:** When a table contains a column that has a column mask defined on it, each column that is in the table must be referenced as a simple column reference in the **GROUP BY** clause. Such columns must not be referenced in a grouping expression in the **GROUP BY** clause.
- **5:** The select list includes an aggregate function and the column that is identified by column-name is specified as an input argument to the function. In this case, the column mask must not reference a scalar fullselect or an aggregate function.
- **6:** A column in the select list maps directly or indirectly to the column column-name of a table expression, view, or common table expression. Such a column must be referenced only as an argument to an aggregate function in the select list.

Use the following possible solutions for these reason codes:

- **reason code 3:** Either do not reference the column in the select list or do not reference the column in an expression in the **GROUP BY** clause.
- **reason code 4:** Either do not reference the column in the select list or change the GROUP BY clause to include only a simple column reference for each column of the same table that is referenced in the mask-name mask.
- **reason code 5:** Either do not reference the column in the select list or do not specify the column as an argument to the aggregate function.
- **reason code 6:** Either change the query to specify the column as an argument to an aggregate function or remove the **GROUP BY** clause.

DB2 11 removes column access control restrictions for **GROUP BY** as follows:

- Remove SQLCODE -20478 reason code 5: if column mask contains a scalar fullselect or an aggregate function, allow to apply the column mask under an aggregate function
- Remove SQLCODE -20478 reason codes 3, 4, or 6: if column mask contains non-grouping columns, allow to apply the column mask when there is a GROUP BY clause

The support for GROUP BY column access control starts in DB2 11 New Function Mode (NFM) when `APPLCOMPAT='V11R1'`. No **`REBIND`** is needed to get the **`GROUP BY`** support.

10.3.2 Correct implementation of aggregate function with DISTINCT

In DB2 10, after APAR PM61099, DB2 disallows a column mask on a column that is the input to an aggregate **`DISTINCT`** function by issuing -20478 reason code 26.

Example 10-39 shows a sample query that can show this behavior.

Example 10-39 Aggregate function with DISTINCT in SQL

```
SELECT COUNT(DISTINCT PHONE#) FROM CLIENTS;
```

Table 10-3 summarizes the changes in behavior when comparing DB2 10 to DB2 11.

Table 10-3 DISTINCT and aggregation with column masking in DB2 10 versus DB2 11

	DISTINCT	**Aggregation**
DB2 10	On masked value	On masked value
DB2 11	On unmasked value	On masked value

Support for aggregate function with DISTINCT column starts in DB2 11 NFM when APPLCOMPAT = 'V11R1'. To avoid the inconsistent result during migration and fallback, the correction is retrofitted to DB2 10 NFM. Rebind is needed to correct the aggregate functions with the DISTINCT keyword.

10.3.3 Column access control for UNION

DB2 10 restricted the column access control (column masks) from all set operations, like UNION, INTERSECT, and EXCEPT, by issuing SQLCODE -20478 reason code 1 and 2.

DB2 11 supports the column access control for **`UNION DISTINCT`** and **`UNION ALL`** in DB2 11 NFM when `APPLCOMPAT = 'V11R1'`.

The expression corresponding to the *n*th column in R1 and R2 can reference columns with column masks. The nth column of the result of the union can be derived from the masked values in R1 or R2.

With **`UNION DISTINCT`**, the duplication elimination is based on the unmasked values in R1 and R2. Because each row of the result table of the union is either a row from R1 or a row from R2, the output values in the result table of the union might vary. For example, if a row in R1 is derived from the masked value but a row in R2 is derived from the unmasked value, and if the row in the result table of the union is from R1, the masked value is returned, but if the row in the result table of the union is from R2, the unmasked value is returned.

The following examples illustrate when the values in the result table of the union can vary:

- The expression corresponding to the nth column in R1 references columns with column masks but the expression corresponding to the nth column in R2 does not, or vise versa.
- Both expressions corresponding to the nth column in R1 and R2 reference columns with column masks but they are different column masks.
- The column mask definition references columns that are not the same column for which the column mask is defined and those columns are not part of the UNION DISTINCT operation. It is recommended not to reference other columns in the column mask definition.

EXCEPT and **INTERSECT** can be intermixed with **UNION**, as long as the rows in R1 and R2 for **EXCEPT** and **INTERSECT** do not reference columns with column masks.

11

Utilities

IBM DB2 Utilities Suite for z/OS is a comprehensive set of tools for managing all DB2 data maintenance tasks. DB2 11 includes a variety of improvements to utilities. These utilities are enhanced to support all new functions in DB2 11. The support also includes more widespread use of the System z platform functions, such as more zIIP exploitation. Finally, DB2 11 utilities show the trend to simplifying data management, resource consumption, and maximize availability.

The best utility is the one that you do not need to schedule, or even better you do not need to run. One example is the function described at 4.10, “Idle thread break-in” on page 82, which relieves the need to run **REORG INDEX** for the purpose of removing the pseudo deleted index entries.

This chapter describes enhancements to utilities and includes the following topics:

- Online REORG enhancements
- Enhanced statistics
- Backup and recovery enhancements
- LOAD and UNLOAD enhancements
- Compression dictionaries for Change Data Capture
- General enhancements
- Deprecated functions

For more information, see *DB2 11 for z/OS Utility Guide and Reference*, SC19-4067.

11.1 Online REORG enhancements

Online **REORG** is an availability enhancement to DB2 introduced in DB2 V5. It increases availability of a table space or index while it is being reorganized. Online **REORG** reloads the reorganized table space and rebuilds the indexes into shadow table spaces and indexes. It then switches the original data sets and the new ones at the end of the **REORG** making the objects unavailable for a short time. There is just a small unavailability when the applications are drained in the final log iteration phase (**CHANGE** only) and the switch phase. By using drain and retry options, the draining process can be controlled without leading to resource unavailability conditions for the applications and the process can be repeated until the **REORG** finishes successfully.

By *online REORG*, we mean **REORG.... SHRLEVEL REFERENCE or REORG.... SHRLEVEL CHANGE**. **REORG .... SHRLEVEL NONE** continues to delete and rebuild the DB2 objects.

Online **REORG** is enhanced throughout several DB2 releases to bring more usability and help users to reach continuous availability.

This section describes the following DB2 11 improvements to online REORG:

- Improve performance of partition-level REORG with non partitioned secondary indexes
- SWITCH phase impact reduction
- Physically delete empty partition-by-growth partitions
- Automated REORG mapping table management
- REORG without SORTing data
- Partition-level inline image copy
- Improved REORG LISTDEF processing
- REBALANCE enhancements
- REORG of LOB enhancements
- Improved REORG serviceability
- REORG change of defaults to match preferred practices

11.1.1 Improve performance of partition-level REORG with non partitioned secondary indexes

Since the removal of the BUILD2 phase for partition-level **REORG** in DB2 9, the performance of **REORG** was degraded in some cases due to the cost of building shadow non partitioned secondary indexes (NPSIs). Shadow NPSIs are populated initially with keys of partitions which are not in the scope of the **REORG** during the **UNLOAD** phase. Then keys from parts within the scope of the **REORG** are sorted and inserted into the shadow NPSI during the **SORT** and **REBUILD** phases, respectively.

Significant performance improvement can be achieved by sorting all keys of the NPSI in the same sort operation and rebuilding the index from the entire set of sorted keys.

DB2 11 modifies the processing of NPSIs for **REORG TABLESPACE PART SHRLEVEL CHANGE/REFERENCE** when NPSIs are defined on the table space. Processing of NPSIs in this case is done in one of the following ways:

- During **UNLOAD**, one or more subtasks unload NPSI keys from partitions not within the scope of the **REORG** and build the shadow NPSI. Keys from partitions within the scope of the **REORG** are generated from the reorganized data rows, sorted, and inserted in the shadow index.
- During **UNLOAD**, one or more subtasks process NPSI keys from partitions not within the scope of the **REORG**. These keys are routed to a sort process to be sorted with the keys

from partitions within the scope of the **REORG**. The shadow NPSI is built from this sorted set of keys.

This function can improve performance and leaves the previous behavior intact. It also allows the user to control the behavior through the **DSNZPARM REORG_PART_SORT_NPSI** value with a **SORTNPSI** keyword. See Figure 11-1.

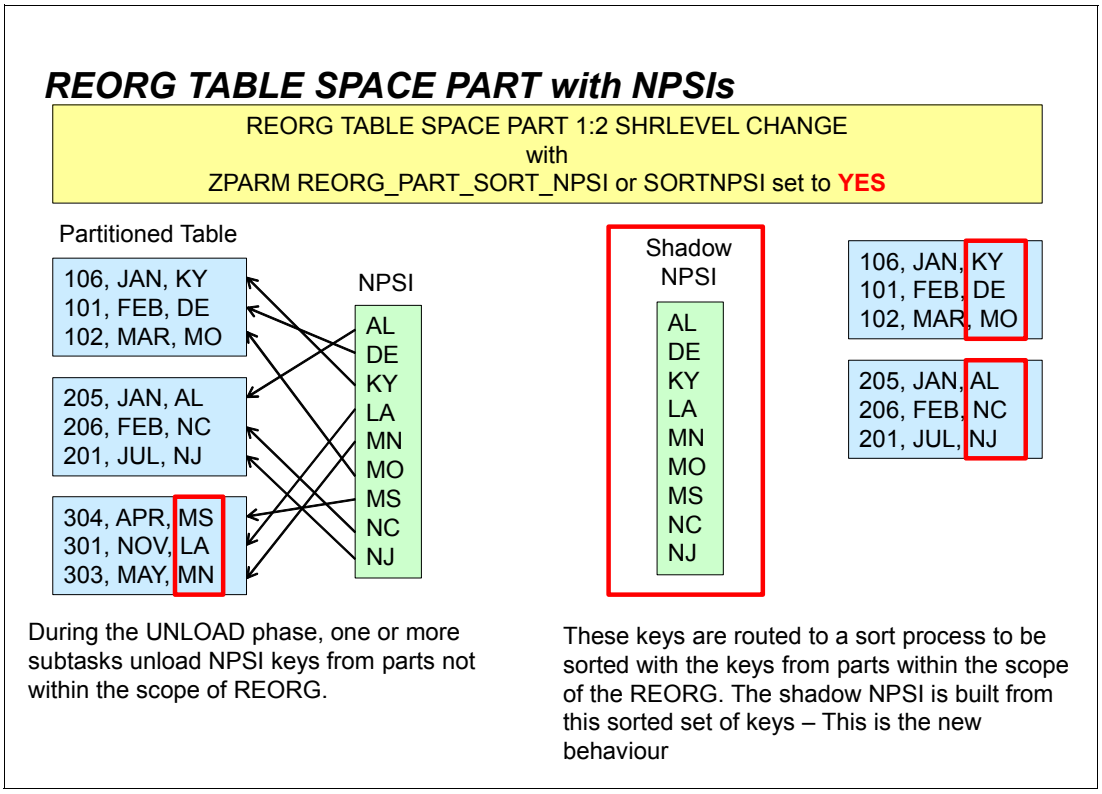


Figure 11-1 REORG TABLE SPACE PART with NPSIs

This keyword is ignored for a **REORG** that is not partition-level or a **REORG** with no NPSIs.

Figure 11-2 shows the following possible set of options for the **SORTNPSI** keyword:

- When **SORTNPSI** is specified as **YES** or **AUTO**, all keys can be sorted.
- When **SORTNPSI** is not specified, and **REORG_PART_SORT_NPSI** is set to **YES** or **AUTO**, all keys can be sorted as well.
- If any of the two parameters are set to **NO**, the previous method is used.

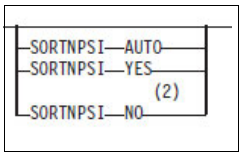
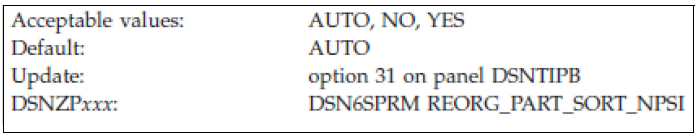


Figure 11-2 New SORTNPSI keyword

If the **SORTNPSI** keyword is not specified, the value is determined by the **DSNZPARM REORG_PART_SORT_NPSI** value (default is **NO**). The **REORG_PART_SORT_NPSI** default is changeable online and has member scope in data sharing.

Figure 11-3 shows the options for the **DSNZPARM REORG_PART_SORT_NPSI** value.



Acceptable values:	AUTO, NO, YES
Default:	AUTO
Update:	option 31 on panel DSNTIPB
DSNZP*xxx*:	DSN6SPRM REORG_PART_SORT_NPSI

Figure 11-3 New DSNZPARM REORG_PART_SORT_NPSI

The following options are available for the **DSNZPARM REORG_PART_SORT_NPSI** value:

AUTO Specifies that if sorting all keys of the non-partitioned secondary indexes improves the elapsed time and CPU performance, all keys are sorted.It uses catalog statistics and RTS information.

YES Specifies that if sorting all keys of the non-partitioned secondary indexes improves the elapsed time, all keys are sorted.

NO Specifies that only keys of the non-partitioned secondary indexes that are in the scope of the REORG are sorted.

The following new message is issued when all keys of an NPSI are sorted during a partition-level **REORG TABLESPACE**.

```
DSNU1242I
csect-name ALL KEYS OF A NON-PARTITIONED SECONDARY INDEX WILL BE SORTED
```

Example 11-1 shows an example of **REORG TABLESPACE PART WITH SORTNPSI YES**.

Example 11-1 REORG TABLESPACE PART WITH SORTNPSI YES

```
/*JOB
//PROCLIB  JCLLIB ORDER=DB1AM.PROCLIB
//REORG EXEC DSNUPROC,SYSTEM=DB1A,
//             LIB='DB1AT.SDSNLOAD',
//             UID='OREORG' UTPROC='PREVIEW'
//DSNUPROC.SYSIN DD *
          TEMPLATE COPY DSN 'DB2R2.&DB..&TS..P&PA..T&TIME.'
           DISP (NEW,CATLG,DELETE)
           UNIT SYSDA
           SPACE (5,5) CYL
          TEMPLATE UT1  DSN 'DB2R2.&DB..&TS..P&PA..SYSUT1'
           DISP (NEW,DELETE,DELETE)
           UNIT SYSDA
           SPACE (5,5) CYL
          TEMPLATE SRTOUT DSN 'DB2R2.&DB..&TS..P&PA..SORTOUT'
           DISP (NEW,DELETE,DELETE)
           UNIT SYSDA
           SPACE (5,5) CYL
           TEMPLATE REC    DSN 'DB2R2.&DB..&TS..P&PA..SYSREC'
            DISP (NEW,DELETE,DELETE)
UNIT SYSDA
           SPACE (5,5) CYL
          REORG TABLESPACE DSN8D11A.DSN8S11E PART(1:4)
                   SORTNPSI YES
          COPYDDN(COPY) UNLDDN REC
            LOG NO
            SHRLEVEL REFERENCE
```

Example 11-2 shows the job output.

Example 11-2 REORG TABLESPACE PART WITH SORTNPSI YES job output

```
DSNU000I    205 14:13:42.62 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = OREORG

DSNU050I    205 14:13:42.64 DSNUGUTC -  REORG TABLESPACE DSN8D11A.DSN8S11E PART(1:4)
SORTNPSI YES COPYDDN(COPY) UNLDDN REC LOG NO SHRLEVEL REFERENCE
 DSNU2904I -DB1A 205 14:13:43.45 DSNURFTB - DATA RECORDS WILL BE UNLOADED VIA TABLE SPACE SCAN
 DSNU1242I -DB1A 205 14:13:43.45 DSNURFUI - ALL KEYS OF A NON-PARTITIONED SECONDARY INDEX WILL BE SORTED
 DSNU2903I   205 14:13:43.45 DSNURORG - PARTITION LEVEL INLINE COPY DATASETS WILL BE ALLOCATED

DSNU010I    205 14:13:46.19 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

The measurements results show that REORG of 40% of partitions showed 55% reduction on elapsed time and an increase of 22% in CPU time. DB2 **SORT** gives additional reduction on elapsed time and reduces CPU time. In general, when reorganizing more than 50% of your table, this option is a good option.

Note: This function was retrofitted by APAR PM55051 to DB2 9 with PTF UK78231 and DB2 10 with PTF UK78229.

11.1.2 SWITCH phase impact reduction

This new feature provides relief for online **REORG** when acquiring **DRAIN** during the **SWITCH** phase (for example, in a situation where users have a partitioned table space with 10 partitions and need to **REORG** the first five partitions). DB2 10 utility requests a **DRAIN** for partition 1, then partition 2, and so on until gets the **DRAIN** for the five partitions. However, often this process runs out of the specified **DRAIN_WAIT** time before it arrives on part five. This process causes the utility to abend and leaves the table space partition in **UTRO** access, with the updates failing with -904 reason code.

DB2 11 sets a flag to prevent new **CLAIM** for the PARTs specified on **REORG**, before starting **DRAIN** for part 1. Thus, new application threads need to wait until **REORG** acquires all the **DRAIN**s to proceeded, as illustrated in Figure 11-4.

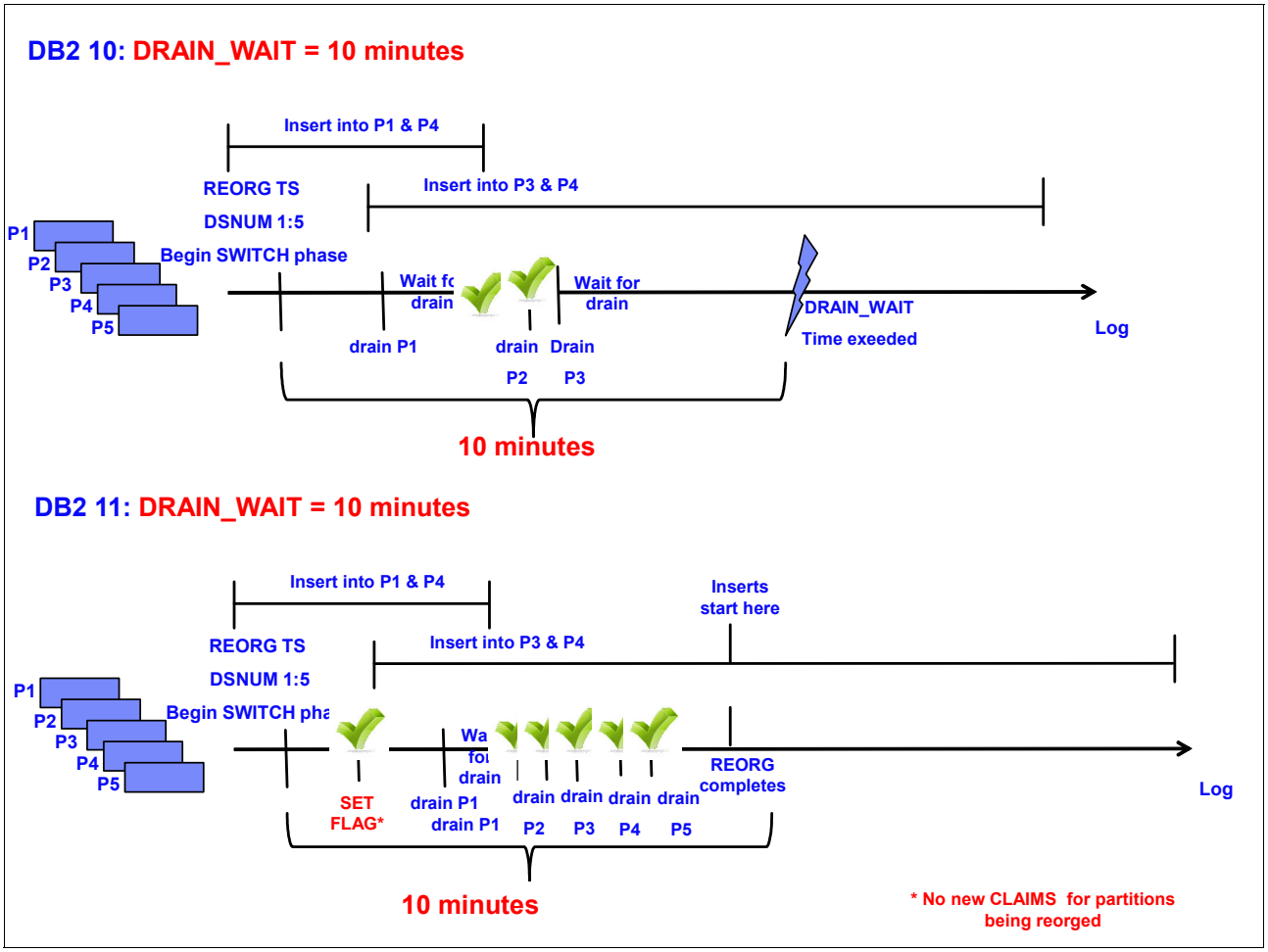


Figure 11-4 Switch phase impact reduction

This technique solves the issue on **DRAIN PART**s but it can still be a problem when draining the NPSI. When **REORG** is requesting DRAIN for part 1 to 5, an application can come in and request a row that is on PART 6. **REORG** lets the application get this **CLAIM**, but more importantly is that the application also gets a **CLAIM** on the NPSI. Later, the application wants a row that is on PART 3 and waits for **REORG** to finish, but **REORG** is waiting to the application to finish as well and release the NPSI. In this case, a deadlock situation occurs.

To solve this issue, DB2 11 provides the new **DRAIN_ALLPARTS YES** option, that tells DB2 to obtains the table space level drain on the entire partitioned table space temporarily first, then **DRAIN** the NPSIs, release the drain on the entire partitioned table space, and start draining the target data partitions and the indexes.

This process provides relief by eliminating **DRAIN** timeout or deadlocks caused by the reverse order of object-draining by **REORG** and object-claiming by DML statement.

The **DRAIN_ALLPARTS** option specifies the action to take during a part level **REORG TABLESPACE SHRLEVEL REFERENCE** or **CHANGE** when a non partitioned secondary index is defined on a partitioned table space. It supports the following values:

NO — **REORG** drains the target data partitions serially followed by the non partitioned secondary indexes. This option is the default behavior.

YES — **REORG** obtains the table space level drain on the entire partitioned table space first, before draining the target data partitions and the indexes. This option can provide relief by eliminating drain timeout or deadlocks caused by the reverse order of object-draining by **REORG** and object-claiming by DML statements.

Example 11-3 shows an example of **REORG TABLESPACE PART** with **DRAIN_ALLPARTS YES**.

Example 11-3 REORG TABLESPACE PART WITH DRAIN_ALLPARTS YES

```
/*JOB
//PROCLIB  JCLLIB ORDER=DB1AM.PROCLIB
//REORG EXEC DSNUPROC,SYSTEM=DB1A,
//             LIB='DB1AT.SDSNLOAD',
//             UID='OREORG' UTPROC='PREVIEW'
//DSNUPROC.SYSIN DD *
          TEMPLATE COPY DSN 'DB2R2.&DB..&TS..P&PA..T&TIME.'
           DISP (NEW,CATLG,DELETE)
           UNIT SYSDA
           SPACE (5,5) CYL
          TEMPLATE UT1  DSN 'DB2R2.&DB..&TS..P&PA..SYSUT1'
           DISP (NEW,DELETE,DELETE)
           UNIT SYSDA
           SPACE (5,5) CYL
          TEMPLATE SRTOUT DSN 'DB2R2.&DB..&TS..P&PA..SORTOUT'
           DISP (NEW,DELETE,DELETE)
           UNIT SYSDA
           SPACE (5,5) CYL
           TEMPLATE REC    DSN 'DB2R2.&DB..&TS..P&PA..SYSREC'
            DISP (NEW,DELETE,DELETE)
            UNIT SYSDA
            SPACE (5,5) CYL
          REORG TABLESPACE DSN8D11A.DSN8S11E PART(1:4)
          DRAIN_ALLPARTS YES
          COPYDDN(COPY) UNLDDN REC
            LOG NO
            SHRLEVEL REFERENCE
```

Example 11-4 shows the job output.

Example 11-4 REORG TABLESPACE PART WITH DRAIN_ALLPARTS YES job output

```
DSNU000I    206 14:39:24.36 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = OREORG

ODSNU050I    206 14:39:24.39 DSNUGUTC -  REORG TABLESPACE DSN8D11A.DSN8S11E PART(1:4) DRAIN_ALLPARTS YES COPYDDN(
 COPY) UNLDDN REC LOG NO SHRLEVEL REFERENCE

-DB1A 206 14:39:26.89 DSNURLOG - DRAIN ALL WITH START TIME 2013-07-25-14.39.26.899227 HAS COMPLETED SUCCESSFULLY
 DSNU1139I   206 14:39:26.94 DSNURLGD - FINAL LOG ITERATION STATISTICS. NUMBER OF LOG RECORDS = 0
 DSNU386I    206 14:39:26.94 DSNURLGD - LOG PHASE STATISTICS. NUMBER OF ITERATIONS = 1, NUMBER OF LOG RECORDS = 0
 DSNU385I    206 14:39:26.94 DSNURLGD - LOG PHASE COMPLETE, ELAPSED TIME = 00:00:00
 DSNU387I    206 14:39:27.11 DSNURSWT - SWITCH PHASE COMPLETE, ELAPSED TIME = 00:00:00
```

```
DSNU428I    206 14:39:27.12 DSNURSWT - DB2 IMAGE COPY SUCCESSFUL FOR TABLESPACE DSN8D11A.DSN8S11E PARTITION 1
DSNU428I    206 14:39:27.12 DSNURSWT - DB2 IMAGE COPY SUCCESSFUL FOR TABLESPACE DSN8D11A.DSN8S11E PARTITION 2
DSNU428I    206 14:39:27.12 DSNURSWT - DB2 IMAGE COPY SUCCESSFUL FOR TABLESPACE DSN8D11A.DSN8S11E PARTITION 3
DSNU428I    206 14:39:27.12 DSNURSWT - DB2 IMAGE COPY SUCCESSFUL FOR TABLESPACE DSN8D11A.DSN8S11E PARTITION 4
DSNU010I    206 14:39:27.97 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

Note: This feature does not affect **REORG** of the whole table space because it is only one **DRAIN** on the whole table space.

Timing of SWITCH phase with MAXRO DEFER

During an online **REORG SHRLEVEL CHANGE**, DB2 allows changes during each iteration of the **LOG** phase, which can cause the **LOG** phase to never end. Thus, the **REORG** utility must at some time switch to read-only access during the **LOG** phase. Then, it has to apply only the changes accumulated during the previous iteration before it enters the **SWITCH** phase. Because the switching to read-only access impacts your applications, the **REORG** utility allows you to decide and specify, using the **MAXRO** utility, how long can you tolerate a read-only period in your environment.

Based on the changes for the previous iteration, the **REORG** utility estimates how long the next iteration takes. If its estimate is lower than or equal to the value specified using the **MAXRO** parameter, the **REORG** utility switches to read-only access or even to no access allowed, depending on what you have requested, and the last iteration takes place.

The **MAXRO DEFER** option causes log processing to continue indefinitely until you change **MAXRO** using the **ALTER UTILITY** command, or the last iteration is forced by **LONGLOG**, or the reorganization is terminated.

DB2 11 allows you to govern the timing of drain and switch for long-running **REORG**s without the need to schedule a separate **ALTER UTILITY** command with a new **SWITCHTIME** parameter to determine the earliest point at which drain processing is attempted. The *timestamp* option determines the time at which the final log iteration of the **LOG** phase is to begin. This time must not have already occurred when **REORG** is run. It simply tells **REORG** when you want to apply the DRAIN and switch without the need of issuing the **ALTER UTILITY** command.

The **SWITCHTIME** option specifies the time for the final log iteration of the **LOG** phase to begin. The final result and all the time stamp calculation of **DEADLINE** is in **TIMESTAMP(6)**. This keyword can be specified in conjunction with the **MAXRO** keyword. **REORG** can enter the final log iteration of the **LOG** phase before the specified **SWITCHTIME** value if the **MAXRO** criteria is met. This option supports the following values:

`NONE`	Does not specify a time for the final log iteration of the **LOG** phase. This option is the default behavior.
`timestamp`	Specifies the time the final log iteration of the **LOG** phase is to begin. This time must not have already occurred when **REORG** is run.
`labeled-duration-expression`	Calculates the time for the final log iteration of **LOG** phase is to begin. The calculation is based on either **CURRENT TIMESTAMP** or **CURRENT DATE**.

To add a labeled duration expression, select to begin your labeled duration expression with either the Current date or the Current time stamp using the following expression:

```
labeled-duration-expression:
CURRENT_DATE
CURRENT_TIMESTAMP WITH TIME ZONE +/- constant YEAR
                                           YEARS
```

```
MONTH
MONTHS
DAY
DAYS
HOUR
HOURS
MINUTE
MINUTES
SECOND
SECONDS
MICROSECOND
MICROSECONDS
```

The **NEWMAXRO** option specifies the maximum amount of time for the last log iteration after **SWITCHTIME** is met. Requires keyword **SWITCHTIME** to be specified. This value overrides the existing **MAXRO** parameter specified. This option supports the following values:

`NONE` Specifies that when the specified **SWITCHTIME** is met, **REORG** proceeds to the last log iteration without taking into log processing time into consideration. Specifying `NONE` results in **REORG** entering the last log iteration almost immediately at or after the specified **SWITCHTIME**. This value is the default.

`integer` Specifies the number of seconds. Specifying a small positive value reduces the length of the period of read-only access, but it might increase the elapsed time for **REORG** to complete. If you specify a huge positive value, it probably ensure **REORG** enters the last log iteration almost immediately at or after the specified **SWITCHTIME**.

You can add or subtract one or more constant values to specify the switch time. This switch time must not have already occurred when **REORG** is run. **CURRENT TIMESTAMP** and **CURRENT DATE** are evaluated once, when the **REORG** statement is first processed. If a list of objects is specified, the same value is in effect for all objects in the list.

11.1.3 Physically delete empty partition-by-growth partitions

REORG a partition-by-growth (PBG) table space prior to DB2 11 can result in empty physical partitions at the end. To avoid this issue, DB2 11 includes the **DSNZPARM REORG_DROP_PBG_PARTS** value. This option specifies whether the **REORG** utility removes trailing empty partitions when operating on an entire PBG table space. It uses the following syntax:

```
Acceptable values: DISABLE, ENABLE
Default: DISABLE
Update: option 31 on panel DSNTIPB
DSNZPxxx: DSN6SPRM REORG_DROP_PBG_PARTS
```

An empty trailing partition occurs when the **REORG** utility moves all data records from a partition into lower numbered partitions. This parameter is meaningful only when the REORG utility is run against an entire PBG table space. It is ignored for the others types of table spaces and for **REORG** of a PBG if you specify PARTs. It cannot be specified at the **REORG** statement level only as a **DSNZPARM** value.

Note: You cannot run a PIT recovery prior to such (pruning) REORG.

11.1.4 Automated REORG mapping table management

During the **REORG LOG** phase, DB2 log records for the changes done to the original table space are applied to the shadow table space. In order to map these changes to the shadow table space, the **REORG** utility uses a mapping table. The mapping table and the index must be created before the **REORG** utility is executed.

The mapping table DDL must change in DB2 11 due to the RBA/LRSN change. In order to help DBAs, DB2 11 can automatically create a mapping table, but if users do not want to pay the cost of creating mapping table automatically during all **REORG**s, the users can continue to create their own mapping tables.

Here are the rules on creating mapping tables:

- If the mapping table is specified and it is in correct format, then honor the specification.
- Else if specified but in the incorrect format, then create a new mapping table in the same database as the original mapping table.
- Else if the mapping table is not specified and the **DSNZPARM REORG_MAPPING_DATABASE** value is specified, then create in the **DSNZPARM** database. The **REORG** utility fails with RC8 and a **DSNU2902I** message occurs, if the specified database name in the keyword or zParm is not found.
- Else create in the implicit database.
- DROP at end of **REORG** or end of last **REORG**, if there are multiple **REORG**s in the job step.

The **MAPPINGDATABASE** option specifies the database in which **REORG** implicitly creates the mapping table and index objects. This keyword overrides the subsystem parameter value in **REORG_MAPPING_DATABASE**. The value cannot be **DSNDB01**, **DSNDB06**, or **DSNDB07**, implicit database, a work file or temporary database.

As an example, consider the following scenario, which executes a **REORG**:

1. Create a mapping table with the DB2 10 format and run **REORG**, as shown in Example 11-5.

Example 11-5 Mapping table with the DB2 10 format and run REORG

```
CREATE TABLE DB2R2.MAPPTBL (
       TYPE CHAR(1) FOR SBCS DATA NOT NULL,
       SOURCE_RID CHAR(5) FOR SBCS DATA NOT NULL,
       TARGET_XRID CHAR(9) FOR SBCS DATA NOT NULL,
       LRSN CHAR(6) FOR SBCS DATA NOT NULL --- DB2 10 format
   )
   IN DSN8D11P.DSN8S11Q
   AUDIT NONE
   DATA CAPTURE NONE
   CCSID EBCDIC;

CREATE UNIQUE INDEX DB2R2.XMAPPTBL
   ON DB2R2.MAPPTBL
   (SOURCE_RIDASC,
     TYPE ASC,
     TARGET_XRIDASC,
     LRSN ASC)
   NOT CLUSTER
   USING STOGROUP DSN8G110
     PRIQTY -1
     SECQTY -1
   BUFFERPOOL BP0
   CLOSE NO
   PIECESIZE 2097152 K;
```

```
//DSNUPROC.SYSIN DD *
REORG TABLESPACE DSN8D11A.DSN8S11E
          COPYDDN(COPY) UNLDDN REC
            LOG NO
            SHRLEVEL CHANGE
            MAPPINGTABLE DB2R2.MAPPTBL
//

1DSNU000I    207 16:33:32.49 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = OREORG

0DSNU050I    207 16:33:32.51 DSNUGUTC -  REORG TABLESPACE DSN8D11A.DSN8S11E COPYDDN(COPY) UNLDDN REC
LOG NO SHRLEVEL
  CHANGE MAPPINGTABLE DB2R2.MAPPTBL
 DSNU2900I -DB1A 207 16:33:32.51 DSNURMAP - MAPPING TABLE IS SPECIFIED WITH A NON-EXPANDED LRSN COLUMN
 DSNU2904I -DB1A 207 16:33:34.59 DSNURFTB - DATA RECORDS WILL BE UNLOADED VIA TABLE SPACE SCAN
 DSNU2901I -DB1A 207 16:33:34.67 DSNURMAP - MAPPING TABLE DB2R2.REORG_MAPTABLE_OREORG_0000
             AND MAPPING INDEX DB2R2.REORG_MAPINDEX_OREORG_0000
             CREATED IN DSN8D11P.RM23B859

DSNU010I    207 16:33:38.47 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

2. Create a mapping table with the DB2 11 format and run **REORG**, as shown in Example 11-6.

Example 11-6 Mapping table with the DB2 11 format and run REORG

```
CREATE TABLE DB2R2.MAPPTBL (
       TYPE CHAR(1) FOR SBCS DATA NOT NULL,
       SOURCE_RID CHAR(5) FOR SBCS DATA NOT NULL,
       TARGET_XRID CHAR(9) FOR SBCS DATA NOT NULL,
       LRSN CHAR(10) FOR SBCS DATA NOT NULL -- DB2 11 format
   )
   IN DSN8D11P.DSN8S11Q
   AUDIT NONE
   DATA CAPTURE NONE
   CCSID EBCDIC;

CREATE UNIQUE INDEX DB2R2.XMAPPTBL
   ON DB2R2.MAPPTBL
   (SOURCE_RIDASC,
     TYPE ASC,
     TARGET_XRIDASC,
     LRSN ASC)
   NOT CLUSTER
   USING STOGROUP DSN8G110
     PRIQTY -1
     SECQTY -1
   BUFFERPOOL BP0
   CLOSE NO
   PIECESIZE 2097152 K;

//DSNUPROC.SYSIN DD *
REORG TABLESPACE DSN8D11A.DSN8S11E
          COPYDDN(COPY) UNLDDN REC
            LOG NO
            SHRLEVEL CHANGE
            MAPPINGTABLE DB2R2.MAPPTBL
//


1DSNU000I    207 16:37:49.77 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = OREORG

ODSNU050I    207 16:37:49.79 DSNUGUTC -  REORG TABLESPACE DSN8D11A.DSN8S11E COPYDDN(COPY) UNLDDN REC
LOG NO SHRLEVEL
  CHANGE MAPPINGTABLE DB2R2.MAPPTBL
 DSNU2904I -DB1A 207 16:37:51.73 DSNURFTB - DATA RECORDS WILL BE UNLOADED VIA TABLE SPACE SCAN
```

```
DSNU010I    207 16:37:55.22 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

3. On the last scenario, there is no mapping table, and **REORG** was run. DB2 automatically creates a mapping table, as shown in Example 11-7.

Example 11-7 Mapping table and run REORG

```
//DSNUPROC.SYSIN DD *
REORG TABLESPACE DSN8D11A.DSN8S11E
          COPYDDN(COPY) UNLDDN REC
            LOG NO
            SHRLEVEL CHANGE
//

1DSNU000I    207 16:04:46.22 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = OREORG

0DSNU050I    207 16:04:46.27 DSNUGUTC -  REORG TABLESPACE DSN8D11A.DSN8S11E COPYDDN(COPY) UNLDDN REC
LOG NO SHRLEVEL
  CHANGE
 DSNU2904I -DB1A 207 16:04:48.53 DSNURFTB - DATA RECORDS WILL BE UNLOADED VIA TABLE SPACE SCAN
 DSNU2901I -DB1A 207 16:04:48.70 DSNURMAP - MAPPING TABLE DB2R2.REORG_MAPTABLE_OREORG_0000
             AND MAPPING INDEX DB2R2.REORG_MAPINDEX_OREORG_0000
             CREATED IN DSN00035.REORGRMA

 DSNU010I    207 16:04:52.69 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

Example 11-8 shows how the database and table space look when automatically created.

Example 11-8 Database and table space format when automatically created by DB2

```
CREATE DATABASE "DSN00035"
   BUFFERPOOL BP0
   INDEXBP BP0
   STOGROUP SYSDEFLT
   CCSID EBCDIC;

CREATE  TABLESPACE "REORGRMA"
   IN DSN00035
   USING STOGROUP SYSDEFLT
     PRIQTY -1
     SECQTY -1
   DSSIZE 4 G
   MAXPARTITIONS 256
   SEGSIZE 32
   BUFFERPOOL BP0
   CCSID EBCDIC
   LOCKMAX SYSTEM
   LOCKSIZE ROW
   MAXROWS 255;
```

Prior to DB2 11, the mapping table needs to be create in a segmented table space which limits the number of rows to REORG. DB2 11 allows the mapping table to be created in segmented and the PBG table space. The PBG definition allows to increase the mapping table max size to 16 TB, and it was also retrofitted to DB2 10 and 9 through APAR PM58177.

Note: DB2 11 New Function Mode (NFM) requires a new format mapping table and conversion mode (CM), CM*, ENFM, and ENFM* support the traditional and new format of mapping tables.

11.1.5 REORG without SORTing data

When **SORTDATA** is specified on the **REORG** utility control statement for a segmented table space, DB2 always unloads rows in physical sequence, that is, table by table, segment by segment, page by page, and row by row. If at least one of the tables in the segmented table space has an explicit clustering index, rows are passed to **DFSORT** for sorting. Rows are sorted in the sequence of the table's explicit or implicit clustering index. If the table does not have an index, rows are not sorted. If a table does not have an explicit clustering index, then the first index created on the table is called the implicit clustering index.

For a partitioned table space, rows of the entire table space or of the partitions to be reorganized are unloaded in physical sequence, that is, page by page and row by row, and passed to **DFSORT** (or an equivalent sort utility) to be sorted in the sequence implied by the partitioning index.

Generally, physically unloading rows and sorting them by **DFSORT** is faster than unloading rows by using a clustering index, especially when the cluster ratio (**CLUSTERRATIOF**) of the index is less than 95%. Thus, so the lower the cluster ratio, the higher is the performance improvement when using **SORTDATA**.

Increasingly, **REORG**s are performed for reasons other than to gain clustering of data. Examples are when **REORG** is used for database conversion, alter segmented size, alter page size, materialize pending changes.

Prior of DB2 11, users do not have the ability to avoid the cost of reclustering.

DB2 11 has implemented the support of **SORTDATA NO** with **SHRLEVEL CHANGE** and also a new **REORG** parameter **RECLUSTER YES/NO** option on **SORTDATA NO**, With **RECLUSTER NO**, **REORG** does not unload data through the clustering index and does not sort data records in clustering order.

If users want to run a **REORG** and they do not care about **CLUSTER**ing data, because this is a conversion table space operation or because the data is already in cluster order, or because the application does not take advantages of clustering data order, you can now specify the option **SORTDATA NO RECLUSTER NO** and DB2 does not sort data. This new feature cuts down the cost of **SORT**ing your data. The option also helps some users that like the use of **SORTDATA NO** but they do not have enough DASD space that **DFSORT** needs to sort your data.

SORTDATA NO tells DB2 to unload using the cluster index or unloading the physical order and pass it to DFSORT. On both ways, your table is in cluster order at the end.

Table 11-1 summarizes the **SORTDATA YES/NO RECLUSTER YES/NO** behavior.

Table 11-1 SORTDATA YES/NO RECLUSTER YES/NO summary

Object description	SORTDATA	SORTDATA NO, RECLUSTER NO	SORTDATA NO, RECLUSTER YES
Segmented table space without index	Unloaded table by table, segment by segment, page by page, and row by row	Unloaded table by table, segment by segment, page by page, and row by row	Unloaded table by table, segment by segment, page by page, and row by row, reclustered following the column sequence

Object description	SORTDATA	SORTDATA NO, RECLUSTER NO	SORTDATA NO, RECLUSTER YES
Partitioned table space without index	Unloaded table by table, segment by segment, page by page, and row by row	Unloaded segment by segment (if UTS), page by page, and row by row	Unloaded table by table, segment by segment, page by page, and row by row, reclustered following the partitioning key
Table space with one clustering index	Rows are passed to DFSORT for sorting, reloaded in sequence of the clustering index	Unloaded table by table, segment by segment (if UTS), page by page, and row by row. Clustering Index is NOT used	Data records are to be reclustered and to be unloaded by the clustering index.
Table space with indexes, but no clustering	Rows are passed to DFSORT for sorting, the index which was created first is used for clustering sequence	Unloaded table by table, segment by segment (if UTS), page by page, and row by row. Clustering Index is NOT used	Data records are to be reclustered and to be unloaded by the 1st index created for any table in the table space

Example 11-9 shows an example of **REORG TABLESPACE SHRLEVEL CHANGE SORTDATA NO RECLUSTER NO**.

Example 11-9 REORG TABLESPACE SHRLEVEL CHANGE SORTDATA NO RECLUSTER NO

```
/*JOB
/*JOBPARM S=SC63,L=9999
//PROCLIB  JCLLIB ORDER=DB1AM.PROCLIB
//REORG EXEC DSNUPROC,SYSTEM=DB1A,
//             LIB='DB1AT.SDSNLOAD',
//             UID='OREORG' UTPROC='PREVIEW'
//DSNUPROC.SYSIN DD *
          TEMPLATE COPY DSN 'DB2R2.&DB..&TS..T&TIME.'
           DISP (NEW,CATLG,DELETE)
           UNIT SYSDA
           SPACE (5,5) CYL
          TEMPLATE UT1  DSN 'DB2R2.&DB..&TS..SYSUT1'
           DISP (NEW,DELETE,DELETE)
           UNIT SYSDA
           SPACE (5,5) CYL
          TEMPLATE SRTOUT DSN 'DB2R2.&DB..&TS..SORTOUT'
           DISP (NEW,DELETE,DELETE)
           UNIT SYSDA
           SPACE (5,5) CYL
           TEMPLATE REC    DSN 'DB2R2.&DB..&TS..SYSREC'
DISP (NEW,DELETE,DELETE)
         UNIT SYSDA
         SPACE (5,5) CYL
        REORG TABLESPACE DSN8D11A.DSN8S11E
        SORTDATA NO RECLUSTER NO
          SORTDEVT SYSDA SORTNUM 32
        COPYDDN(COPY) UNLDDN REC
          LOG NO
          SHRLEVEL CHANGE
```

Example 11-10 shows the job output.

Example 11-10 REORG TABLESPACE SHRLEVEL CHANGE SORTDATA NO RECLUSTER NO job output

```
1DSNU000I    211 14:59:33.41 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = OREORG
 DSNU1044I   211 14:59:33.42 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
0DSNU050I    211 14:59:33.42 DSNUGUTC -  TEMPLATE
 DSNU1035I   211 14:59:33.43 DSNUJTDR - TEMPLATE STATEMENT PROCESSED SUCCESSFULLY
0DSNU050I    211 14:59:33.43 DSNUGUTC -  REORG TABLESPACE DSN8D11A.DSN8S11E SORTDATA NO RECLUSTER NO SORTDEVT SYSDA
 SORTNUM 32 COPYDDN(COPY) UNLDDN REC LOG NO SHRLEVEL CHANGE
 DSNU2904I -DB1A 211 14:59:34.68 DSNURFTB - DATA RECORDS WILL BE UNLOADED VIA TABLE SPACE SCAN
 DSNU2901I -DB1A 211 14:59:34.91 DSNURMAP - MAPPING TABLE DB2R2.REORG_MAPTABLE_OREORG_0000
             AND MAPPING INDEX DB2R2.REORG_MAPINDEX_OREORG_0000
             CREATED IN DSN00043.REORGRMA

 DSNU251I    211 14:59:36.49 DSNURULD - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS UNLOADED=32
 DSNU252I    211 14:59:36.49 DSNURULD - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS UNLOADED=32 FOR TABLESPACE
DSN8D11A.DSN8S11E

 DSNU302I    211 14:59:36.85 DSNURILD - (RE)LOAD PHASE STATISTICS - NUMBER OF INPUT RECORDS PROCESSED=32
 DSNU300I    211 14:59:36.85 DSNURILD - (RE)LOAD PHASE COMPLETE, ELAPSED TIME=00:00:00
 DSNU394I  -DB1A 211 14:59:36.94 DSNURBXA - SORTBLD PHASE STATISTICS - NUMBER OF KEYS=32 FOR INDEX
 DB2R2.REORG_MAPINDEX_OREORG_0000
 DSNU393I  -DB1A 211 14:59:37.00 DSNURBXA - SORTBLD PHASE STATISTICS - NUMBER OF KEYS=32 FOR INDEX DSN81110.XEMP1  PART
1
 DSNU394I  -DB1A 211 14:59:37.06 DSNURBXA - SORTBLD PHASE STATISTICS - NUMBER OF KEYS=32 FOR INDEX DSN81110.XEMP2
 DSNU391I    211 14:59:37.06 DSNURPTB - SORTBLD PHASE STATISTICS. NUMBER OF INDEXES = 3
 DSNU392I    211 14:59:37.06 DSNURPTB - SORTBLD PHASE COMPLETE, ELAPSED TIME = 00:00:00
 DSNU1138I -DB1A 211 14:59:37.38 DSNURLOG - DRAIN ALL WITH START TIME 2013-07-30-14.59.37.382492 HAS COMPLETED
SUCCESSFULLY

 DSNU010I    211 14:59:38.95 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

11.1.6 Partition-level inline image copy

Prior to DB2 11, when you take an inline image copy during a **REORG** of a table space with a lot of partitions, DB2 creates one large inline image copy data set with all the partitions. If later on you decide to recovery a single partition, the recovery time for reading the large image copy to return one partition is much higher than if you have an image copy data set per partition.

DB2 11 takes an image copy per partition when you specify &PA. or &PART on the **TEMPLATE**s. Example 11-11 shows a **REORG TABLESPACE PART WITH INLINE IMAGE COPY**.

Example 11-11 REORG TABLESPACE PART WITH INLINE IMAGE COPY

```
/*JOB
//PROCLIB  JCLLIB ORDER=DB1AM.PROCLIB
//REORG EXEC DSNUPROC,SYSTEM=DB1A,
//             LIB='DB1AT.SDSNLOAD',
//             UID='OREORG' UTPROC='PREVIEW'
//DSNUPROC.SYSIN DD *
          TEMPLATE COPY DSN 'DB2R2.&DB..&TS..P&PA..T&TIME.'
           DISP (NEW,CATLG,DELETE)
           UNIT SYSDA
           SPACE (5,5) CYL
          TEMPLATE UT1  DSN 'DB2R2.&DB..&TS..P&PA..SYSUT1'
           DISP (NEW,DELETE,DELETE)
           UNIT SYSDA
           SPACE (5,5) CYL
          TEMPLATE SRTOUT DSN 'DB2R2.&DB..&TS..P&PA..SORTOUT'
           DISP (NEW,DELETE,DELETE)
           UNIT SYSDA
           SPACE (5,5) CYL
```

```
 TEMPLATE REC    DSN 'DB2R2.&DB..&TS..P&PA..SYSREC'
  DISP (NEW,DELETE,DELETE)
 UNIT SYSDA
 SPACE (5,5) CYL
    REORG TABLESPACE DSN8D11A.DSN8S11E PART(1:4)
COPYDDN(COPY) UNLDDN REC
  LOG NO
  SHRLEVEL REFERENCE
```

Example 11-12 shows the job output.

Example 11-12 REORG TABLESPACE PART WITH INLINE IMAGE COPY job output

```
1DSNU000I    206 14:39:24.36 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = OREORG
 DSNU1044I   206 14:39:24.39 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
0DSNU050I    206 14:39:24.39 DSNUGUTC -  TEMPLATE COPY DSN 'DB2R2.&DB..&TS..P&PA..T&TIME.' DISP(NEW, CATLG, DELETE)
 UNIT SYSDA SPACE(5, 5) CYL
 DSNU1035I   206 14:39:24.39 DSNUJTDR - TEMPLATE STATEMENT PROCESSED SUCCESSFULLY
0DSNU050I    206 14:39:24.39 DSNUGUTC -  TEMPLATE UT1 DSN 'DB2R2.&DB..&TS..P&PA..SYSUT1' DISP(NEW, DELETE, DELETE)
 UNIT SYSDA SPACE(5, 5) CYL
 DSNU1035I   206 14:39:24.39 DSNUJTDR - TEMPLATE STATEMENT PROCESSED SUCCESSFULLY
0DSNU050I    206 14:39:24.39 DSNUGUTC -  TEMPLATE SRTOUT DSN 'DB2R2.&DB..&TS..P&PA..SORTOUT' DISP(NEW, DELETE,
 DELETE) UNIT SYSDA SPACE(5, 5) CYL
 DSNU1035I   206 14:39:24.39 DSNUJTDR - TEMPLATE STATEMENT PROCESSED SUCCESSFULLY
0DSNU050I    206 14:39:24.39 DSNUGUTC -  TEMPLATE REC DSN 'DB2R2.&DB..&TS..P&PA..SYSREC' DISP(NEW, DELETE, DELETE)
 UNIT SYSDA SPACE(5, 5) CYL
 DSNU1035I   206 14:39:24.39 DSNUJTDR - TEMPLATE STATEMENT PROCESSED SUCCESSFULLY
0DSNU050I    206 14:39:24.39 DSNUGUTC -  REORG TABLESPACE DSN8D11A.DSN8S11E PART(1:4) COPYDDN(
 COPY) UNLDDN REC LOG NO SHRLEVEL REFERENCE

DSNU2903I   206 14:39:25.78 DSNURORG - PARTITION LEVEL INLINE COPY DATASETS WILL BE ALLOCATED
 DSNU1038I   206 14:39:25.82 DSNUGDYN - DATASET ALLOCATED.  TEMPLATE=COPY
                        DDNAME=SYS00001
                        DSN=DB2R2.DSN8D11A.DSN8S11E.P00001.T183949
 DSNU1038I   206 14:39:25.87 DSNUGDYN - DATASET ALLOCATED.  TEMPLATE=COPY
                        DDNAME=SYS00002
                        DSN=DB2R2.DSN8D11A.DSN8S11E.P00002.T183949
 DSNU1038I   206 14:39:25.88 DSNUGDYN - DATASET ALLOCATED.  TEMPLATE=COPY
                        DDNAME=SYS00003
                        DSN=DB2R2.DSN8D11A.DSN8S11E.P00003.T183949
 DSNU1038I   206 14:39:25.94 DSNUGDYN - DATASET ALLOCATED.  TEMPLATE=COPY
                        DDNAME=SYS00004
                        DSN=DB2R2.DSN8D11A.DSN8S11E.P00004.T183949

DSNU250I    206 14:39:26.27 DSNURPRD - UNLOAD PHASE COMPLETE, ELAPSED TIME=00:00:00
 DSNU400I    206 14:39:26.41 DSNURBID - COPY PROCESSED FOR TABLESPACE DSN8D11A.DSN8S11E PART 1
                        NUMBER OF PAGES=10
                        AVERAGE PERCENT FREE SPACE PER PAGE =  6.60
                        PERCENT OF CHANGED PAGES =100.00
                        ELAPSED TIME=00:00:00
 DSNU400I    206 14:39:26.42 DSNURBID - COPY PROCESSED FOR TABLESPACE DSN8D11A.DSN8S11E PART 2
                        NUMBER OF PAGES=2
                        AVERAGE PERCENT FREE SPACE PER PAGE =  0.00
                        PERCENT OF CHANGED PAGES =100.00
                        ELAPSED TIME=00:00:00
 DSNU400I    206 14:39:26.42 DSNURBID - COPY PROCESSED FOR TABLESPACE DSN8D11A.DSN8S11E PART 3
                        NUMBER OF PAGES=4
                        AVERAGE PERCENT FREE SPACE PER PAGE = 19.75
                        PERCENT OF CHANGED PAGES =100.00
                        ELAPSED TIME=00:00:00
 DSNU400I    206 14:39:26.44 DSNURBID - COPY PROCESSED FOR TABLESPACE DSN8D11A.DSN8S11E PART 4
                        NUMBER OF PAGES=2
                        AVERAGE PERCENT FREE SPACE PER PAGE =  0.00
                        PERCENT OF CHANGED PAGES =100.00
                        ELAPSED TIME=00:00:00
```

```
DSNU387I    206 14:39:27.11 DSNURSWT - SWITCH PHASE COMPLETE, ELAPSED TIME = 00:00:00
 DSNU428I    206 14:39:27.12 DSNURSWT - DB2 IMAGE COPY SUCCESSFUL FOR TABLESPACE DSN8D11A.DSN8S11E PARTITION 1
 DSNU428I    206 14:39:27.12 DSNURSWT - DB2 IMAGE COPY SUCCESSFUL FOR TABLESPACE DSN8D11A.DSN8S11E PARTITION 2
 DSNU428I    206 14:39:27.12 DSNURSWT - DB2 IMAGE COPY SUCCESSFUL FOR TABLESPACE DSN8D11A.DSN8S11E PARTITION 3
 DSNU428I    206 14:39:27.12 DSNURSWT - DB2 IMAGE COPY SUCCESSFUL FOR TABLESPACE DSN8D11A.DSN8S11E PARTITION 4
 DSNU010I    206 14:39:27.97 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

Figure 11-5 shows the performance improvement for RECOVERY of a single partition of a twenty partitions table space. The elapsed time is reduced by 28% and CPU time is reduced by 49%.

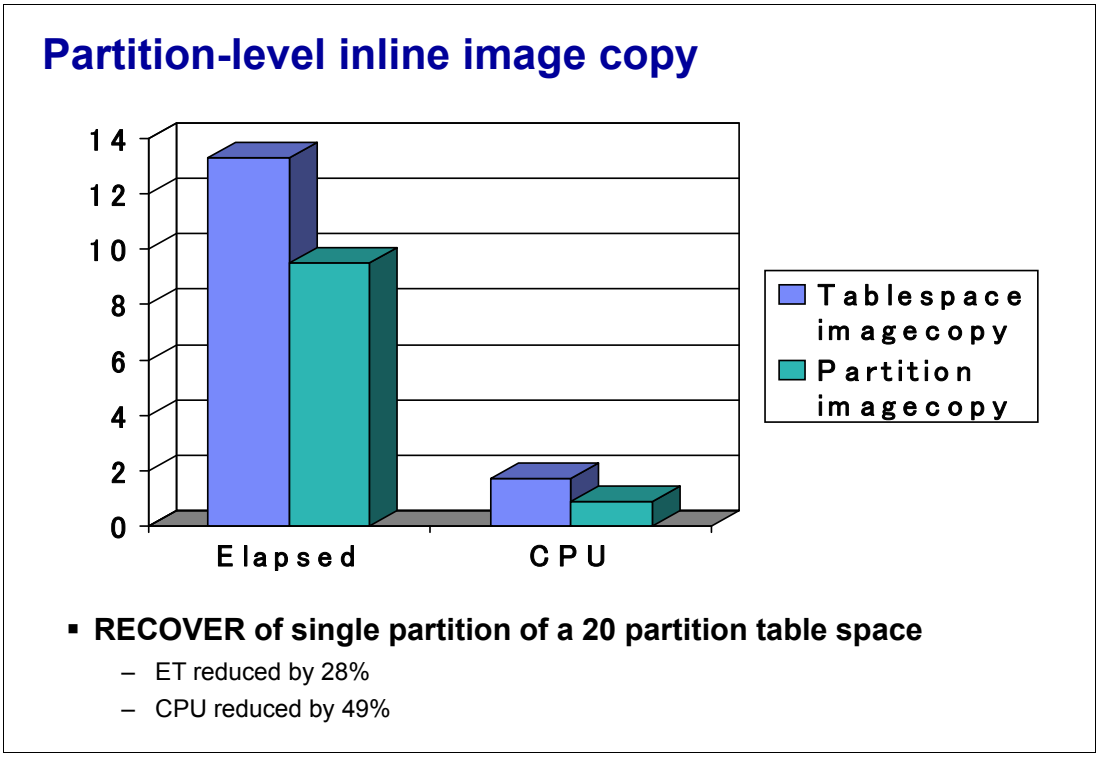


Figure 11-5 Partition-level inline image copy performance

11.1.7 Improved REORG LISTDEF processing

Prior of DB2 9, when you specify a set of partitions on a **REORG**, such as partition 1 to 10, DB2 **REORG**s partition 1, when finished that, then **REORG**s partition 2, and so on until partition 10. On DB2 9 DB2 starts to process all specified partition together, which is much more efficient, but it also needs more DASD space to shadow all PARTs.

In order to give the users an option to choose between all PARTS together or not, DB2 10 implements the **PARALLEL** keyword **YES**|**NO** where **PARALLEL NO** means **REORG** reorganizes one data partition at a time if data partitions are specified at the part level in the LISTDEF. DB2 11 provide a new option, *LISTPARTS n*, where user can determine the n maximum number of data partitions to be reorganized in a single **REORG** execution. This option enables customer to further fine-tune and balance between the resource consumption and performance trade off of the **REORG** utility. If customer does not have enough DASD for all parts but are able to handle say 20% of them.

When **LISTPARTS** keyword is not specified, **REORG** defaults to the setting of the existent **REORG_LIST_PROCESSING** zParm in determining whether to process data partitions together or

not. If **REORG_LIST_PROCESSING** is set to **SERIAL**, this equates to specifying **LISTPARTS 1**. If **REORG_LIST_PROCESSING** is set to **PARALLEL**, then all specified partitions will be reorganized in a single **REORG**.

PARALLEL YES/NO is deprecated but still supported in DB2 11. The following rules apply to existing **REORG** jobs:

- If **PARALLEL YES** is specified, it equates to not specifying **LISTPARTS** but overriding the **REORG_LIST_PROCESSING** zParm value to **PARALLEL** for current **REORG** execution. It means, **REORG** all specified partition together.
- If **PARALLEL NO** is specified, it equates to **LISTPARTS 1**, and **REORG** will process data partitions one at a time as specified in the **LISTDEF**.

The **LISTPARTS** option specifies the maximum number of data partitions to be reorganized in a single **REORG** on a **LISTDEF** that contains **PARTLEVEL** list items. It supports the following values:

n Specifies an integer that represents the maximum number of data partitions to be reorganized at the same time. A valid value is greater than 0. If **LISTPARTS** is not specified, the default value is the setting of the **REORG_LIST_PROCESSING** subsystem parameter.

Example 11-13 shows an example of **REORG TABLESPACE PART WITH LISTPARTS *n***.

Example 11-13 REORG TABLESPACE PART WITH LISTPARTS

```
/*JOBPARM S=SC63,L=9999
//PROCLIB  JCLLIB ORDER=DB1AM.PROCLIB
//REORG EXEC DSNUPROC,SYSTEM=DB1A,
//             LIB='DB1AT.SDSNLOAD',
//             UID='OREORG' UTPROC='PREVIEW'
//DSNUPROC.SYSIN DD *
          TEMPLATE COPY DSN 'DB2R2.&DB..&TS..P&PA..T&TIME.'
           DISP (NEW,CATLG,DELETE)
           UNIT SYSDA
           SPACE (5,5) CYL
          TEMPLATE UT1  DSN 'DB2R2.&DB..&TS..P&PA..SYSUT1'
           DISP (NEW,DELETE,DELETE)
           UNIT SYSDA
           SPACE (5,5) CYL
          TEMPLATE SRTOUT DSN 'DB2R2.&DB..&TS..P&PA..SORTOUT'
           DISP (NEW,DELETE,DELETE)
           UNIT SYSDA
           SPACE (5,5) CYL
           TEMPLATE REC    DSN 'DB2R2.&DB..&TS..P&PA..SYSREC'
            DISP (NEW,DELETE,DELETE)
       UNIT SYSDA
       SPACE (5,5) CYL
 LISTDEF REOLIST INCLUDE TABLESPACE DSN8D11A.DSN8S11E PARTLEVEL 1
 INCLUDE TABLESPACE DSN8D11A.DSN8S11E PARTLEVEL 2
 INCLUDE TABLESPACE DSN8D11A.DSN8S11E PARTLEVEL 3
 INCLUDE TABLESPACE DSN8D11A.DSN8S11E PARTLEVEL 4
  REORG TABLESPACE LIST REOLIST
                LISTPARTS 3
      COPYDDN(COPY) UNLDDN REC
        LOG NO
        SHRLEVEL REFERENCE
```

Example 11-14 shows the job output. Note that, because **LISTPARTS 3** was specified, DB2 processes the first three parts and, when it finishes, DB2 starts part four.

Example 11-14 REORG TABLESPACE PART WITH LISTPARTS job output

```
1DSNU000I    211 13:38:50.09 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = OREORG
 DSNU1044I   211 13:38:50.11 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
0DSNU050I    211 13:38:50.11 DSNUGUTC -  TEMPLATE
DSNU1035I   211 13:38:50.11 DSNUJTDR - TEMPLATE STATEMENT PROCESSED SUCCESSFULLY
ODODSNU050I    211 13:38:50.11 DSNUGUTC -  LISTDEF REOLIST INCLUDE TABLESPACE DSN8D11A.DSN8S11E PARTLEVEL 1 INCLUDE
TABLESPACE DSN8D11A.DSN8S11E PARTLEVEL 2
INCLUDE TABLESPACE DSN8D11A.DSN8S11E PARTLEVEL 3
INCLUDE TABLESPACE DSN8D11A.DSN8S11E PARTLEVEL 4
 DSNU1035I   211 13:38:50.11 DSNUILDR - LISTDEF STATEMENT PROCESSED SUCCESSFULLY
0DSNU050I    211 13:38:50.11 DSNUGUTC -  REORG TABLESPACE LIST REOLIST LISTPARTS 3 COPYDDN(COPY) UNLDDN REC LOG NO
 SHRLEVEL REFERENCE
 DSNU1039I   211 13:38:50.13 DSNUGULM - PROCESSING LIST ITEM: TABLESPACE DSN8D11A.DSN8S11E PARTITION 1 -- only 3 PARTS
 DSNU1039I   211 13:38:50.13 DSNUGULM - PROCESSING LIST ITEM: TABLESPACE DSN8D11A.DSN8S11E PARTITION 2
 DSNU1039I   211 13:38:50.13 DSNUGULM - PROCESSING LIST ITEM: TABLESPACE DSN8D11A.DSN8S11E PARTITION 3
 DSNU2904I -DB1A 211 13:38:51.15 DSNURFTB - DATA RECORDS WILL BE UNLOADED VIA TABLE SPACE SCAN
 DSNU1242I -DB1A 211 13:38:51.15 DSNURFUI - ALL KEYS OF A NON-PARTITIONED SECONDARY INDEX WILL BE SORTED
 DSNU2903I   211 13:38:51.15 DSNURORG - PARTITION LEVEL INLINE COPY DATASETS WILL BE ALLOCATED

DSNU251I  -DB1A 211 13:38:51.50 DSNURPUT - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS UNLOADED=32 FOR TABLESPACE
 DSN8D11A.DSN8S11E PART 1
 DSNU251I  -DB1A 211 13:38:51.50 DSNURPUT - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS UNLOADED=0 FOR TABLESPACE
 DSN8D11A.DSN8S11E PART 2
 DSNU251I  -DB1A 211 13:38:51.50 DSNURPUT - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS UNLOADED=10 FOR TABLESPACE
 DSN8D11A.DSN8S11E PART 3
 DSNU252I  -DB1A 211 13:38:51.50 DSNURPUT - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS UNLOADED=42 FOR TABLESPACE
 DSN8D11A.DSN8S11E
 DSNU250I    211 13:38:51.51 DSNURPRD - UNLOAD PHASE COMPLETE, ELAPSED TIME=00:00:00


 DSNU1138I -DB1A 211 13:38:52.44 DSNURLOG - DRAIN ALL WITH START TIME 2013-07-30-13.38.52.447315 HAS COMPLETED
SUCCESSFULLY
 DSNU1139I   211 13:38:52.49 DSNURLGD - FINAL LOG ITERATION STATISTICS. NUMBER OF LOG RECORDS = 0
 DSNU386I    211 13:38:52.49 DSNURLGD - LOG PHASE STATISTICS. NUMBER OF ITERATIONS = 1, NUMBER OF LOG RECORDS = 0
 DSNU385I    211 13:38:52.49 DSNURLGD - LOG PHASE COMPLETE, ELAPSED TIME = 00:00:00
 DSNU387I    211 13:38:52.63 DSNURSWT - SWITCH PHASE COMPLETE, ELAPSED TIME = 00:00:00
 DSNU428I    211 13:38:52.63 DSNURSWT - DB2 IMAGE COPY SUCCESSFUL FOR TABLESPACE DSN8D11A.DSN8S11E PARTITION 1
 DSNU428I    211 13:38:52.63 DSNURSWT - DB2 IMAGE COPY SUCCESSFUL FOR TABLESPACE DSN8D11A.DSN8S11E PARTITION 2
 DSNU428I    211 13:38:52.63 DSNURSWT - DB2 IMAGE COPY SUCCESSFUL FOR TABLESPACE DSN8D11A.DSN8S11E PARTITION 3
 DSNU1039I   211 13:38:53.77 DSNUGULM - PROCESSING LIST ITEM: TABLESPACE DSN8D11A.DSN8S11E PARTITION 4 -- Then parts 4
 DSNU2904I -DB1A 211 13:38:54.34 DSNURFTB - DATA RECORDS WILL BE UNLOADED VIA TABLE SPACE SCAN
 DSNU2903I   211 13:38:54.34 DSNURORG - PARTITION LEVEL INLINE COPY DATASETS WILL BE ALLOCATED
 DSNU1038I   211 13:38:54.38 DSNUGDYN - DATASET ALLOCATED.  TEMPLATE=COPY
                      DDNAME=SYS00005
                      DSN=DB2R2.DSN8D11A.DSN8S11E.P00004.T173915
 DSNU3345I   211 13:38:54.38 DSNURPCT - MAXIMUM UTILITY PARALLELISM IS 6 BASED ON NUMBER OF PARTITIONS AND INDEXES
DSNU251I  -DB1A 211 13:38:54.58 DSNURPUT - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS UNLOADED=0 FOR TABLESPACE
DSN8D11A.DSN8S11E PART 4
 DSNU252I  -DB1A 211 13:38:54.58 DSNURPUT - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS UNLOADED=0 FOR TABLESPACE
DSN8D11A.DSN8S11E
 DSNU250I    211 13:38:54.59 DSNURPRD - UNLOAD PHASE COMPLETE, ELAPSED TIME=00:00:00
 DSNU400I    211 13:38:54.65 DSNURBID - COPY PROCESSED FOR TABLESPACE DSN8D11A.DSN8S11E PART 4
                     NUMBER OF PAGES=2
                     AVERAGE PERCENT FREE SPACE PER PAGE =  0.00
                     PERCENT OF CHANGED PAGES =100.00
                     ELAPSED TIME=00:00:00
 DSNU303I  -DB1A 211 13:38:54.66 DSNURWT - (RE)LOAD PHASE STATISTICS - NUMBER OF RECORDS=0 FOR TABLE DSN81110.EMP PART=4
 DSNU302I    211 13:38:54.66 DSNURILD - (RE)LOAD PHASE STATISTICS - NUMBER OF INPUT RECORDS PROCESSED=0
 DSNU300I    211 13:38:54.66 DSNURILD - (RE)LOAD PHASE COMPLETE, ELAPSED TIME=00:00:00
 DSNU393I  -DB1A 211 13:38:54.68 DSNURBXE - SORTBLD PHASE STATISTICS - NUMBER OF KEYS=0 FOR INDEX DSN81110.XEMP1  PART 4

DSNU385I    211 13:38:55.09 DSNURLGD - LOG PHASE COMPLETE, ELAPSED TIME = 00:00:00
```

```
DSNU387I    211 13:38:55.22 DSNURSWT - SWITCH PHASE COMPLETE, ELAPSED TIME = 00:00:00
DSNU428I    211 13:38:55.22 DSNURSWT - DB2 IMAGE COPY SUCCESSFUL FOR TABLESPACE DSN8D11A.DSN8S11E PARTITION 4
DSNU010I    211 13:38:55.61 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

11.1.8 REBALANCE enhancements

REORG REBALANCE allows the partitions of a partitioned table space to be rebalanced, altering their limit keys, avoiding expensive unloading, recreating, and reloading of the entire table space and, thus, improving its availability.

With versions prior of DB2 11, users are not able to execute **REORG REBALANCE SHRLEVEL CHANGE**. This option is now allowed in DB2 11.

Example 11-15 shows an example of **REORG TABLESPACE REBALANCE SHRLEVEL CHANGE**.

Example 11-15 REORG TABLESPACE REBALANCE SHRLEVEL CHANGE

```
set current sqlid = 'DB2R2';
CREATE DATABASE DB2RDB4
CREATE TABLESPACE DB2RTS1
  IN DB2RDB4
CREATE TABLE DB2R2.DB2RTB4
   (DBNAME               CHAR(8) FOR SBCS DATA NOT NULL,
...
MODECREATED          CHAR(2) FOR SBCS DATA NOT NULL
    WITH DEFAULT)
IN db2rDB4.db2rTS1
PARTITION BY (DBNAME ASC)
 (PARTITION 1 ENDING AT ('AAAAAA'),
  PARTITION 2 ENDING AT ('BBBBBB'),
  PARTITION 3 ENDING AT ('ZZZZZZ'))
;

/*JOBPARM S=SC63,L=9999
//PROCLIB  JCLLIB ORDER=DB1AM.PROCLIB
//REORG EXEC DSNUPROC,SYSTEM=DB1A,
//             LIB='DB1AT.SDSNLOAD',
//             UID='OREORG' UTPROC='PREVIEW'
//DSNUPROC.SYSIN DD *
          TEMPLATE COPY DSN 'DB2R2.&DB..&TS..P&PA..T&TIME.'
           DISP (NEW,CATLG,DELETE)
           UNIT SYSDA
           SPACE (5,5) CYL
          TEMPLATE UT1  DSN 'DB2R2.&DB..&TS..P&PA..SYSUT1'
           DISP (NEW,DELETE,DELETE)
           UNIT SYSDA
           SPACE (5,5) CYL
          TEMPLATE SRTOUT DSN 'DB2R2.&DB..&TS..P&PA..SORTOUT'
           DISP (NEW,DELETE,DELETE)
           UNIT SYSDA
           SPACE (5,5) CYL
           TEMPLATE REC    DSN 'DB2R2.&DB..&TS..P&PA..SYSREC'
            DISP (NEW,DELETE,DELETE)
      UNIT SYSDA
      SPACE (5,5) CYL
                     REORG TABLESPACE DB2RDB4.DB2RTS1
                    REBALANCE
                           COPYDDN(COPY) UNLDDN REC
                             LOG NO
```

Example 11-16 shows the job output.

Example 11-16 REORG TABLESPACE REBALANCE SHRLEVEL CHANGE job output

```
1DSNU000I    217 14:43:26.54 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = OREORG
 DSNU1044I   217 14:43:26.57 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
0DSNU050I    217 14:43:26.58 DSNUGUTC -  TEMPLATE COPY DSN 'DB2R2.&DB..&TS..P&PA..T&TIME.' DISP(NEW, CATLG, DELETE)
 UNIT SYSDA SPACE(5, 5) CYL
 DSNU1035I   217 14:43:26.58 DSNUJTDR - TEMPLATE STATEMENT PROCESSED SUCCESSFULLY

0DSNU050I    217 14:43:26.58 DSNUGUTC -  REORG TABLESPACE DB2RDB4.DB2RTS1 REBALANCE COPYDDN(COPY) UNLDDN REC LOG NO
 SHRLEVEL CHANGE
 DSNU2904I -DB1A 217 14:43:27.15 DSNURFTB - DATA RECORDS WILL BE UNLOADED VIA TABLE SPACE SCAN
 DSNU2901I -DB1A 217 14:43:27.48 DSNURMAP - MAPPING TABLE DB2R2.REORG_MAPTABLE_OREORG_0000
             AND MAPPING INDEX DB2R2.REORG_MAPINDEX_OREORG_0000
             CREATED IN DSN000062.REORGRMA

 DSNU3340I   217 14:43:28.14 DSNUGSRT - UTILITY PERFORMS DYNAMIC ALLOCATION OF SORT DISK SPACE
 DSNU251I    217 14:43:28.21 DSNUGSRT - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS UNLOADED=0 FOR TABLESPACE
 DB2RDB4.DB2RTS1 PART 1
 DSNU251I    217 14:43:28.21 DSNUGSRT - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS UNLOADED=25 FOR TABLESPACE
 DB2RDB4.DB2RTS1 PART 2
 DSNU251I    217 14:43:28.21 DSNUGSRT - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS UNLOADED=260 FOR TABLESPACE
 DB2RDB4.DB2RTS1 PART 3
 DSNU252I    217 14:43:28.21 DSNUGSRT - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS UNLOADED=285 FOR TABLESPACE
 DB2RDB4.DB2RTS1
 DSNU250I    217 14:43:28.21 DSNUGSRT - UNLOAD PHASE COMPLETE, ELAPSED TIME=00:00:00
 DSNU3345I   217 14:43:28.30 DSNURPIB - MAXIMUM UTILITY PARALLELISM IS 3 BASED ON NUMBER OF PARTITIONS AND INDEXES
 DSNU397I    217 14:43:28.30 DSNURPIB - NUMBER OF TASKS CONSTRAINED BY CPUS TO 3
 DSNU3340I   217 14:43:28.30 DSNUGSOR - UTILITY PERFORMS DYNAMIC ALLOCATION OF SORT DISK SPACE
 DSNU2906I -DB1A 217 14:43:28.30 DSNURBAT - REBALANCE PARTITION SUMMARY ON DB2RDB4.DB2RTS1
  LPART   PPART   ROWCOUNT              LIMITKEY
  ------  ------  -------------------  ------------------------------------------------------------
       1       1                   89  X'C4E2D5F0F0F0F2F7'
       2       2                  114  X'C4E2D5F8C4F1F1C1'
       3       3                   82  X'E9E9E9E9E9E9FFFF'
 DSNU303I  -DB1A 217 14:43:28.40 DSNURWT - (RE)LOAD PHASE STATISTICS - NUMBER OF RECORDS=89 FOR TABLE DB2R2.DB2RTB4
 PART=1
 DSNU303I  -DB1A 217 14:43:28.40 DSNURWT - (RE)LOAD PHASE STATISTICS - NUMBER OF RECORDS=114 FOR TABLE DB2R2.DB2RTB4
 PART=2
 DSNU303I  -DB1A 217 14:43:28.40 DSNURWT - (RE)LOAD PHASE STATISTICS - NUMBER OF RECORDS=82 FOR TABLE DB2R2.DB2RTB4
 PART=3
 DSNU304I  -DB1A 217 14:43:28.40 DSNURWT - (RE)LOAD PHASE STATISTICS - NUMBER OF RECORDS=285 FOR TABLE DB2R2.DB2RTB4
 DSNU302I    217 14:43:28.41 DSNURILD - (RE)LOAD PHASE STATISTICS - NUMBER OF INPUT RECORDS PROCESSED=285
 DSNU300I    217 14:43:28.41 DSNURILD - (RE)LOAD PHASE COMPLETE, ELAPSED TIME=00:00:00
DSNU387I    217 14:43:28.67 DSNURSWT - SWITCH PHASE COMPLETE, ELAPSED TIME = 00:00:00
DSNU010I    217 14:43:29.49 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

REORG REBALANCE issue with data distribution

Today, when **REORG REBALANCE** is run on a partitioned table space with high skewed data distribution, often times the **REORG** utility fails with RC8 and the **DSNU1130I – NOT ALL PARTITIONS POPULATED BY REBALANCE – PROCESSING TERMINATES** message. In the extreme cases, the **REBALANCE** option might be suppressed by the **DSNU1128I – FEWER PAGES THAN PARTS – REBALANCE** message ignored. These error scenarios tend to happen on a table space where the number of records or number of unique limit key values are small in relation to the number of partitions being reorganized, resulting in the 'insufficient data' to rebalance type of errors.

To address these issues, DB2 11 **REBALANCE** logic was optimized to become more resilient against failures on table space with skewed data distribution. This includes getting better statistics on the data being reorganized, to leaving trailing empty partitions at successful rebalance operation. Note that this does not mean the **DSNU1130I** and **DSNU1128I** limitations will be eliminated altogether, but rather aim to reduce the likelihood that data rebalancing fails with these symptoms.

Empty partitions with compression dictionary

Traditionally, **REORG** builds a compression dictionary on the data partitions as data got unloaded during the **UNLOAD** phase, which presents a problem to **REORG REBALANCE** execution, where there might not be any or enough data records unloaded from a partition to build a compression dictionary during **UNLOAD**. If data records get loaded into this partition later on as a result of data rebalancing, then none of these data records will get compressed, requiring a subsequent **REORG** to gain compression.

To address this issue, **REORG REBALANCE** will now build a single compression dictionary for all target partitions. The primary reason for this behavior is that today, compression dictionaries are built based on the unloaded records, which do not represent the records that got loaded back into the same partition in reload. Thus, this solution mimics that of partition-by-growth table space, and provides relief for partitions that do not have compression dictionaries built today. If users are satisfied with the existing compression ratio, **KEEPDICTIONARY** should be specified.

REORG SORTCLUSTER option

With DB2 10, **REORG REBALANCE** sorts the data records based on the limit key value. In the event that the limit key columns is not defined as a superset of (or identical to) the clustering index columns, **REORG REBALANCE** sets the affected data partitions in AREO* on successful completion. This behavior was a contradictory statement to many users who have had just run the **REORG** utility, by recommending another subsequent **REORG** to put the data records into clustering order.

With DB2 11, a new keyword option is introduced to specify whether or not **REBALANCE** should also sort the data records in clustering order, in addition to the existing partitioning order. This is an optional keyword because there is additional resource consumption on sort due to the extra sort field for clustering order. Clustering might be not important for the target table space in some situations. Figure 11-6 shows this behavior.

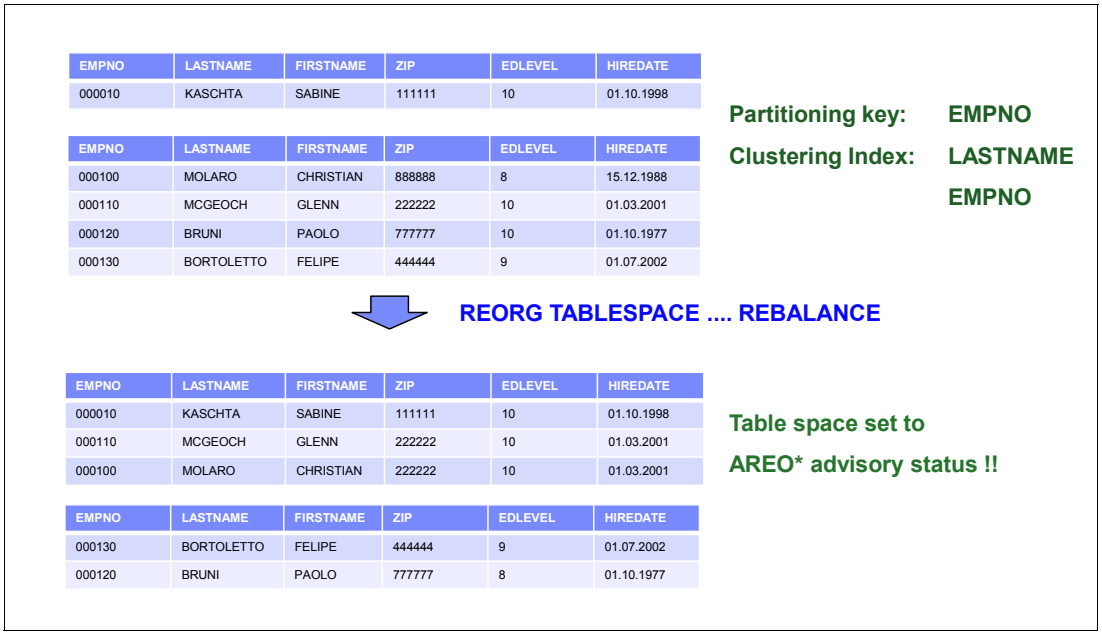


Figure 11-6 AREO status after REORG REBALANCE

The **SORTCLUSTER** option determines if **REBALANCE** is to attempt to sort the data records into clustering order. This option is ignored if no clustering index exists in the table, or when the limit key columns are identical to or are a superset of the clustering index columns. It supports the following values:

NO — Specifies that the data records are not to be explicitly sorted into clustering order. This option is the default behavior. If **SORTCLUSTER NO** is explicitly specified, **AREO*** is not set on the affected data partitions upon **REORG REBALANCE** completion.

YES — Specifies that the data records are to be explicitly sorted into clustering order as needed.

11.1.9 REORG of LOB enhancements

DB2 10 implemented **REORG LOB SHRLEVEL CHANGE** and it also removed **REORG LOB SHRLEVE NONE**. However, in DB2 10 specifying **REORG** with **SHRLEVEL NONE** returns RC0 with **MSGDSNU126**.

DB2 11 changes the return code from RC0 to RC8.

Example 11-17 shows the new return code of **REORG LOB SHRLEVE NONE**.

Example 11-17 REORG LOB SHRLEVE NONE

```
/*JOBPARM S=SC63,L=9999
//PROCLIB  JCLLIB ORDER=DB1AM.PROCLIB
//REORG EXEC DSNUPROC,SYSTEM=DB1A,
//            LIB='DB1AT.SDSNLOAD',
//            UID='OREORG' UTPROC='PREVIEW'
```

```
//DSNUPROC.SYSIN DD *
REORG TABLESPACE ADBDCHG.ADBSCFGL
COPYDDN(COPY) UNLDDN REC
  LOG NO
  SHRLEVEL NONE
```

Example 11-18 shows the job output.

Example 11-18 REORG LOB SHRLEVE NONE job output

```
1DSNU000I    212 10:10:30.78 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = OREORG
 DSNU1044I   212 10:10:30.79 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
DSNU1035I   212 10:10:30.80 DSNUJTDR - TEMPLATE STATEMENT PROCESSED SUCCESSFULLY
DSNU050I    212 10:10:30.80 DSNUGUTC -  REORG TABLESPACE ADBDCHG.ADBSCFGL COPYDDN(COPY) UNLDDN REC LOG NO SHRLEVEL NONE
 DSNU126I  -DB1A 212 10:10:30.80 DSNURFIT - REORG SHRLEVEL NONE ON LOB TABLE SPACE IS NO LONGER SUPPORTED
 DSNU012I    212 10:10:30.80 DSNUGBAC - UTILITY EXECUTION TERMINATED, HIGHEST RETURN CODE=8
```

11.1.10 Improved REORG serviceability

When something goes wrong with **SYSLGRNX**, IBM support can recommend users to run **REORG LOGRANGES NO**; however, this option cannot be execute with **SHRLEVEL CHANGE** prior of DB2 11.

DB2 11allows to use online **REORG** even when **SYSLGRNX** cannot be relied upon, by supporting the **LOGRANGES NO** option for **REORG SHRLEVEL CHANGE**. This option tells **REORG** not to use **SYSLGRNX** information during the **LOGAPPLY** phase. The downside of this option is that it can cause **REORG** to run much longer. In a data sharing environment this option can result in the merging of all logs from all members.

Example 11-19 shows an example of **REORG SHRLEVEL CHANGE LOGRANGES NO**.

Example 11-19 REORG SHRLEVEL CHANGE LOGRANGES NO

```
/*JOBPARM S=SC63,L=9999
//PROCLIB  JCLLIB ORDER=DB1AM.PROCLIB
//REORG EXEC DSNUPROC,SYSTEM=DB1A,
//             LIB='DB1AT.SDSNLOAD',
//             UID='OREORG' UTPROC='PREVIEW'
//DSNUPROC.SYSIN DD *
REORG TABLESPACE DSN8D11A.DSN8S11E
COPYDDN(COPY) UNLDDN REC
  LOG NO
  SHRLEVEL CHANGE
  LOGRANGES NO
```

Example 11-20 shows the job output.

Example 11-20 REORG SHRLEVEL CHANGE LOGRANGES NO job output

```
1DSNU000I    213 18:14:44.47 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = OREORG
 DSNU1044I   213 18:14:44.48 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
0DSNU050I    213 18:14:44.49 DSNUGUTC -  TEMPLATE DSNU1035I
0DSNU050I    213 18:14:44.49 DSNUGUTC -  REORG TABLESPACE DSN8D11A.DSN8S11E COPYDDN(COPY) UNLDDN REC LOG NO SHRLEVEL
  CHANGE LOGRANGES NO
DSNU250I    213 18:14:48.14 DSNUGSRT - UNLOAD PHASE COMPLETE, ELAPSED TIME=00:00:00

DSNU1136I -DB1A 213 18:14:49.00 DSNURLOG - SYSLGRNX IS NOT USED FOR LOG READ DUE TO LOGRANGES NO SPECIFICATION
 D DSNU010I    213 18:14:52.32 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

11.1.11 REORG change of defaults to match preferred practices

To improve experience and to reinforce the preferred practices, the following default behaviors and values of **REORG** are modified:

- **DRAIN** from **WRITERS** to **DRAIN ALL**.

  DRAIN ALL is the new default instead of the existing **DRAIN WRITERS** default setting. In numerous cases **DRAIN ALL** tends to increase the likelihood of online **REORG** breaking-in for the **SWITCH** phase outage. This is applicable to both **REORG TABLESPACE** and **REORG INDEX**.

- **DISCARD** to **DISCARD NOPAD YES** and **UNLOAD EXTERNAL** to **UNLOAD EXTERNAL NOPAD YES**.

  The **NOPAD** option in **UNLOAD EXTERNAL** and also in **DISCARD** clause is the default settings. This option improves performance of the **REORG** utility because padding the variable length data might result in significant overhead. However, because today the only way to ask for padding of the unloaded or discarded data is by not specifying the **NOPAD** keyword, making the **NOPAD** keyword the default does not allow the option to unload or discard data in the padded format. As a result, the **NOPAD** keyword accepts an additional YES | NO parameter. The default is **NOPAD** specified, which has the same behavior as **NOPAD YES**. But users can then specify **NOPAD NO** to request padded data.

11.2 Enhanced statistics

Starting from DB2 10, a portion of the **RUNSTATS** utility is made eligible for zIIP redirection. Less complex statistics (for example, frequency statistics) get most of their execution eligible for zIIP. Complex statistics requiring call to a sort product, such as **DFSORT**, can benefit from zIIP eligibility of the sort product. zIIP eligibility can reach up to 99.9% for **RUNSTATS** with no additional parameters and it goes down for more complex statistics.

Basically, with DB2 10, you find a varying degree of zIIP eligibility when you execute your **RUNSTATS** utility workload and there is no zIIP eligibility for inline statistics in DB2 10.

DB2 11 has the ability to route more **RUNSTATS** workload to zIIP, the complex statistics, and the distribution statistics, reaching up to 80% zIIP-eligible. DB2 11 also can reroute up to 30% of inline statistics to zIIP.

To avoid the need of **RUNSTATS** after a **REORG**, DB2 11 implement the following enhancement for inline statistics:

- Inline statistics collection on NPSIs during **REORG** with **SORTNPSI**
- Inline histogram statistics
- Inline **DSTATS**

This section describes the following **RUNSTATS** enhancements:

- **RUNSTATS RESET ACCESSPATH**
- **RUNSTATS USE PROFILE** usability for **LISTDEF**

11.2.1 RUNSTATS RESET ACCESSPATH

You can use the **RUNSTATS** utility to remove out-of-date access path statistics for DB2 objects. When the **RUNSTATS** utility is invoked over a period of time, statistics are collected incrementally for target objects. The combination of many changes to target objects and many **RUNSTATS** invocations, perhaps with different options, might result in some previously collected statistics becoming outdated. Such out-of-date statistics might cause DB2 to choose inefficient access paths for SQL statements. One solution is to invoke the **RUNSTATS**

utility again to refresh the statistics. However, the task of formulating **RUNSTATS** invocations to solve the problem might prove difficult because of the complicated nature of the many previous **RUNSTATS** invocations.

When this situation occurs, you can invoke the **RUNSTATS** utility to reset the access path statistics for all tables and indexes in a specified table space. When you reset the statistics, the default values are used. No statistics are gathered or reported. Space statistics and real-time statistics are not reset for the specified objects. After your reset access path statistics, the previous values cannot be recovered if no statistics history is available

To reset access path statistics invoke the **RUNSTATS** utility, and specify the following options:

- Specify the **RESET ACCESSPATH** option.
- Optionally specify the **HISTORY ACCESSPATH** option to record that the access path statistics were reset in rows in the **SYSIBM.SYSTABLES_HIST** and **SYSIBM.SYSINDEXES_HIST** statistics tables. This option only records that the reset occurred and does not save the access path statistics values that are reset.

For example, you might issue the following utility control statement:

```
RUNSTATS TABLESPACE db-name.ts-name TABLE table-name RESET ACCESSPATH
```

Statistics are not collected. Instead, the **RUNSTATS** utility resets the access path statistics.

Certain catalog table rows are updated with default values, and rows are deleted from other catalog tables. All updated rows in the catalog tables contain the same **timestamp** value. Real-time statistics and space for the specified object are not reset. However, the dynamic statement cache is invalidated.

Table 11-2 shows the new default values for statistics.

Table 11-2 New default statistics values

Catalog table	Column	Changed value
SYSTABLESPACE	**NACTIVE**	-1
	NACVTIVEF	-1
	STATSTIME	The **TIMESTAMP** value for the reset operation
SYSCOLUMNS	**COLCARD**	-1
	COLCARDF	-1
	HIGH2KEY	Zero-length blank
	LOW2KEY	Zero-length blank
	STATSTIME	The **TIMESTAMP** value for the reset operation
	STATS_FORMAT	Blank

Catalog table	Column	Changed value
SYSTABLES	CARD	-1
	CARDF	-1
	NPAGES	-1
	NPAGESF	-1
	PCTPAGES	-1
	PCTROWCOMP	-1
	STATSTIME	The **TIMESTAMP** value for the reset operation
SYSINDEXES	CLUSTERED	'N'
	NLEAF	-1
	NLEVELS	-1
	FIRSTKEYCARD	-1
	FULLKEYCARD	-1
	FIRSTKEYCARDF	-1
	FULLKEYCARDF	-1
	CLUSTERRATIO	0
	CLUSTERRATIOF	0
	DATAREPEATFACTORF	-1
	STATSTIME	The **TIMESTAMP** value for the reset operation
SYSKEYTARGETS	CARDF	-1
	HIGH2KEY	Zero-length blank
	LOW2KEY	Zero-length blank
	STATSTIME	**TIMESTAMP**
	STATS_FORMAT	Blank

Applicable rows are deleted from the following catalog tables for the specified objects:

- **SYSIBM.SYSTABSTATS**
- **SYSIBM.SYSCOLSTATS**
- **SYSIBM.SYSINDEXSTATS**
- **SYSIBM.SYSCOLDIST**
- **SYSIBM.SYSCOLDISTSTATS**
- **SYSIBM.SYSKEYTARGETSTATS**
- **SYSIBM.SYSKEYTGTDIST**
- **SYSIBM.SYSKEYTGTDISTSTATS**

After resetting the statistics, you might want to invoke the **RUNSTATS** utility again with different options to capture new statistic.

Example 11-21 shows an example of **RUNSTATS RESET** option.

Example 11-21 RUNSTATS RESET option

```
Statistics before reset
select NACTIVE
 ,STATSTIME
   from sysibm.SYSTABLESPACE
  where NAME = 'DSN8S11E';
    NACTIVE  STATSTIME
          15 2013-08-01-19.37.32.770296

select COLCARD
 ,HIGH2KEY
 ,LOW2KEY
  from sysibm.SYSCOLUMNS
  where tbcreator = 'DSN81110' and TBNAME = 'EMP';
   COLCARD  HIGH2KEY    LOW2KEY
        32  000330      000020
        30  WILLIAM     CHRISTINE
        19  W           A
        31  WALKER      BROWN
         8   E11        B01
        32   9001       0942
         31
         8   PRES       CLERK
         8
         2   M          F
         30
        32   0           0
         8   0 ?         0
        32   0           0

select CARD
 ,NPAGES
 ,PCTPAGES
 ,PCTROWCOMP
   from sysibm.SYSTABLES
   where creator = 'DSN81110' and NAME = 'EMP';
CARD       NPAGES  PCTPAGES  PCTROWCOMP
32            1         6         100

 select CLUSTERED
  ,NLEAF
  ,NLEVELS
  ,FIRSTKEYCARD
  ,FULLKEYCARD
  ,CLUSTERRATIO
   from sysibm.SYSINDEXES
   where creator = 'DSN81110' and TBNAME = 'EMP';
CLUSTERED        NLEAF  NLEVELS  FIRSTKEYCARD  FULLKEYCARD  CLUSTERRATIO
Y                    1        2            32           32           100
N                   -1       -1            -1           -1             0
DSNE610I NUMBER OF ROWS DISPLAYED IS 2


/*JOBPARM S=SC63,L=9999
//PROCLIB  JCLLIB ORDER=DB1AM.PROCLIB
//REORG EXEC DSNUPROC,SYSTEM=DB1A,
//             LIB='DB1AT.SDSNLOAD',
//             UID='ORUN' UTPROC='PREVIEW'
```

```
//DSNUPROC.SYSIN DD *
RUNSTATS TABLESPACE DSN8D11A.DSN8S11E RESET ACCESSPATH

Statistics after reset
select NACTIVE
 ,STATSTIME
   from sysibm.SYSTABLESPACE
  where NAME = 'DSN8S11E';
    NACTIVE  STATSTIME
         -1  2013-08-05-22.02.23.547981
DSNE610I NUMBER OF ROWS DISPLAYED IS 1

select COLCARD
 ,HIGH2KEY
 ,LOW2KEY
  from sysibm.SYSCOLUMNS
  where tbcreator = 'DSN8111O' and TBNAME = 'EMP';
    COLCARD  HIGH2KEY  LOW2KEY
     -1
     -1
     -1
     -1
     -1
     -1
     -1
     -1
     -1
     -1
     -1
     -1
     -1
     -1
DSNE610I NUMBER OF ROWS DISPLAYED IS 14

select CARD
 ,NPAGES
 ,PCTPAGES
 ,PCTROWCOMP
   from sysibm.SYSTABLES
   where creator = 'DSN8111O' and NAME = 'DSN8S11E';
CARD       NPAGES  PCTPAGES  PCTROWCOMP
-1           -1        -1          -1

 select CLUSTERED
  ,NLEAF
  ,NLEVELS
  ,FIRSTKEYCARD
  ,FULLKEYCARD
  ,CLUSTERRATIO
   from sysibm.SYSINDEXES
   where creator = 'DSN8111O' and TBNAME = 'EMP';
CLUSTERED        NLEAF  NLEVELS  FIRSTKEYCARD  FULLKEYCARD  CLUSTERRATIO
N                   -1       -1            -1           -1             0
N                   -1       -1            -1           -1             0
DSNE610I NUMBER OF ROWS DISPLAYED IS 2
```

Example 11-22 shows the job output.

Example 11-22 RUNSTATS RESET job output

```
1DSNU000I    213 18:49:50.21 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = ORUN
 DSNU1044I   213 18:49:50.22 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
0DSNU050I    213 18:49:50.22 DSNUGUTC -  RUNSTATS TABLESPACE DSN8D11A.DSN8S11E RESET ACCESSPATH
 DSNU1380I -DB1A 213 18:49:50.22 DSNUSRST - SYSCOLSTATS CATALOG ACCESSPATH STATISTICS RESET SUCCESSFUL
 DSNU1380I -DB1A 213 18:49:50.22 DSNUSRST - SYSTABSTATS CATALOG ACCESSPATH STATISTICS RESET SUCCESSFUL
 DSNU1380I -DB1A 213 18:49:50.22 DSNUSRST - SYSCOLDISTSTATS CATALOG ACCESSPATH STATISTICS RESET SUCCESSFUL
 DSNU1380I -DB1A 213 18:49:50.22 DSNUSRST - SYSCOLUMNS CATALOG ACCESSPATH STATISTICS RESET SUCCESSFUL
 DSNU1380I -DB1A 213 18:49:50.22 DSNUSRST - SYSTABLES CATALOG ACCESSPATH STATISTICS RESET SUCCESSFUL
 DSNU1380I -DB1A 213 18:49:50.23 DSNUSRST - SYSCOLDIST CATALOG ACCESSPATH STATISTICS RESET SUCCESSFUL
 DSNU1380I -DB1A 213 18:49:50.23 DSNUSRST - SYSTABLESPACE CATALOG ACCESSPATH STATISTICS RESET SUCCESSFUL
 DSNU1380I -DB1A 213 18:49:50.23 DSNUSRST - SYSINDEXSTATS CATALOG ACCESSPATH STATISTICS RESET SUCCESSFUL
 DSNU1380I -DB1A 213 18:49:50.23 DSNUSRST - SYSINDEXES CATALOG ACCESSPATH STATISTICS RESET SUCCESSFUL
 DSNU620I  -DB1A 213 18:49:50.23 DSNUSRST - RUNSTATS CATALOG TIMESTAMP = 2013-08-01-18.49.50.227714
 DSNU010I    213 18:49:50.23 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

11.2.2 RUNSTATS USE PROFILE usability for LISTDEF

DB2 11 improves **RUNSTATS USE PROFILE** usability for **LISTDEF** processing. If users execute a **RUNSTATS USE PROFILE** for a list of table spaces and one table space in the list does not have a profile, DB2 will gather default statistics, before DB2 11 utility terminate with error.

Example 11-23 shows an example of **RUNSTATS USE PROFILE** with **LISTDEF** when a table spaces in the list does not have a profile.

Example 11-23 RUNSTATS USE PROFILE usability for LISTDEF

```
/*JOBPARM S=SC63,L=9999
//PROCLIB  JCLLIB ORDER=DB1AM.PROCLIB
//REORG EXEC DSNUPROC,SYSTEM=DB1A,
//             LIB='DB1AT.SDSNLOAD',
//             UID='ORUN' UTPROC='PREVIEW'
//DSNUPROC.SYSIN DD *
LISTDEF RUNLIST INCLUDE TABLESPACE DSN8D11A.DSN8S11E PARTLEVEL 1
INCLUDE TABLESPACE DSN8D11A.DSN8S11E PARTLEVEL 2
INCLUDE TABLESPACE DSN8D11A.DSN8S11E PARTLEVEL 3
INCLUDE TABLESPACE DSN8D11A.DSN8S11E PARTLEVEL 4
      RUNSTATS TABLESPACE LIST RUNLIST TABLE ALL USE PROFILE
```

Example 11-24 shows the job output.

Example 11-24 RUNSTATS USE PROFILE usability for LISTDEF job output

```
1DS
1DSNU000I    213 19:37:32.45 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = ORUN
 DSNU1044I   213 19:37:32.46 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
0DSNU050I    213 19:37:32.47 DSNUGUTC -  LISTDEF RUNLIST INCLUDE TABLESPACE DSN8D11A.DSN8S11E PARTLEVEL 1 INCLUDE
 TABLESPACE DSN8D11A.DSN8S11E PARTLEVEL 2 INCLUDE TABLESPACE DSN8D11A.DSN8S11E PARTLEVEL 3 INCLUDE TABLESPACE
 DSN8D11A.DSN8S11E PARTLEVEL 4
 DSNU1035I   213 19:37:32.47 DSNUILDR - LISTDEF STATEMENT PROCESSED SUCCESSFULLY
0DSNU050I    213 19:37:32.47 DSNUGUTC -  RUNSTATS TABLESPACE LIST RUNLIST TABLE ALL USE PROFILE
 DSNU1039I   213 19:37:32.48 DSNUGULM - PROCESSING LIST ITEM: TABLESPACE DSN8D11A.DSN8S11E PARTITION 1
 DSNU1382I -DB1A 213 19:37:32.48 DSNUGPRF - THE STATS PROFILE FOR TABLE EMP NOT FOUND.
 DEFAULT PROFILE STATS COLLECTED
 DSNU1368I   213 19:37:32.48 DSNUGPRB - PARSING STATS PROFILE FOR TABLE EMP
 DSNU1369I   213 19:37:32.48 DSNUGPRB - PARSING STATS PROFILE FOR TABLE EMP COMPLETED
 DSNU610I  -DB1A 213 19:37:32.61 DSNUSUTP - SYSTABLEPART CATALOG UPDATE FOR DSN8D11A.DSN8S11E SUCCESSFUL
 DSNU610I  -DB1A 213 19:37:32.61 DSNUSUPT - SYSTABSTATS CATALOG UPDATE FOR DSN81110.EMP SUCCESSFUL
```

```
DSNU610I  -DB1A 213 19:37:32.62 DSNUSUPC - SYSCOLSTATS CATALOG UPDATE FOR DSN8110.EMP SUCCESSFUL
DSNU610I  -DB1A 213 19:37:32.62 DSNUSUTB - SYSTABLES CATALOG UPDATE FOR DSN8110.EMP SUCCESSFUL
DSNU610I  -DB1A 213 19:37:32.63 DSNUSUCO - SYSCOLUMNS CATALOG UPDATE FOR DSN8110.EMP SUCCESSFUL
DSNU610I  -DB1A 213 19:37:32.63 DSNUSUTS - SYSTABLESPACE CATALOG UPDATE FOR DSN8D11A.DSN8S11E SUCCESSFUL
DSNU610I  -DB1A 213 19:37:32.63 DSNUSUIP - SYSINDEXPART CATALOG UPDATE FOR DSN8110.XEMP1 SUCCESSFUL
DSNU610I  -DB1A 213 19:37:32.63 DSNUSUPI - SYSINDEXSTATS CATALOG UPDATE FOR DSN8110.XEMP1 SUCCESSFUL
DSNU610I  -DB1A 213 19:37:32.63 DSNUSUPC - SYSCOLSTATS CATALOG UPDATE FOR DSN8110.XEMP1 SUCCESSFUL
DSNU610I  -DB1A 213 19:37:32.63 DSNUSUCO - SYSCOLUMNS CATALOG UPDATE FOR DSN8110.XEMP1 SUCCESSFUL
DSNU610I  -DB1A 213 19:37:32.63 DSNUSUIX - SYSINDEXES CATALOG UPDATE FOR DSN8110.XEMP1 SUCCESSFUL
DSNU620I  -DB1A 213 19:37:32.63 DSNUSEOF - RUNSTATS CATALOG TIMESTAMP = 2013-08-01-19.37.32.488783

DSNU010I    213 19:37:32.84 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=4
```

11.3 Backup and recovery enhancements

This section describes the following major backup and recovery enhancements:

- SYSLGRNX recording for catalog and directory table
- VCAT name translation for RESTORE SYSTEM
- Remove the incompatibility of REORG and COPY
- Removal of many point-in-time recovery restrictions

11.3.1 SYSLGRNX recording for catalog and directory table

Prior to DB2 11, the update ranges for some DB2 directory objects are not recorded in **SYSIBM.SYSLGRNX**. This can result in an unnecessary full log scan extended recovery time for critical directory objects. DB2 11 enables **SYSLGRNX** recording for the following objects:

- DSNDB01.SCT02
- DSNDB01.SPT01
- DSNDB01.SYSSPUXA
- DSNDB01.SYSSPUXB
- Indexes over the above table spaces.

The **RECOVER** utility uses the **SYSLGRNX** records to selectively read and apply the log records for ranges of update. Thus, users have a faster catalog and directory recovery process, because DB2 filters the part of log to read and apply using **SYSLGRNX**.

11.3.2 VCAT name translation for RESTORE SYSTEM

With DB2 10, users who clone their DB2 systems within the same **SYSPLEX** using a system-level backup as a base, do not have a way to apply log records to update their data to a desired point in time. This restriction exists because the high-level qualifier, the ICF catalog VCAT alias, on the cloned DB2 system must be different from the high level qualifier on the source DB2 system.

DB2 11 implements a new VCAT name translation for **RESTORE SYSTEM** for system cloning. It also supports log apply. A new sequential input data set that contains the old and new VCAT alias values, identified by using the **SYSVALUEDDN** option is to be provided by the user. Figure 11-7 shows the syntax diagram.

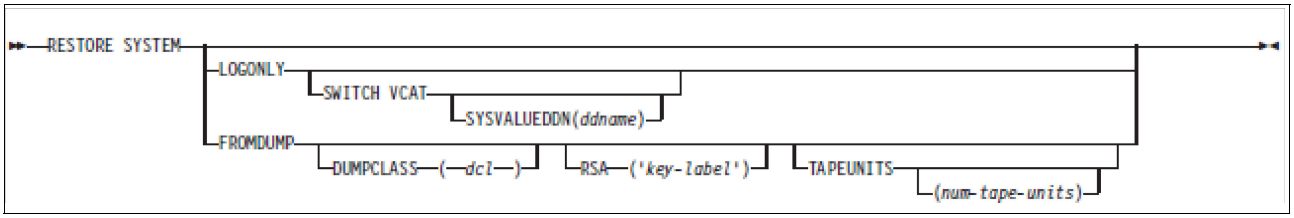


Figure 11-7 RESTORE SYSTEM syntax diagram with SWITCH VCAT and SYSVALUEDDN

The **SYSVALUEDDN** option supports the following values:

SWITCH VCAT Indicates that the integrated catalog facility (ICF) alias (VCAT) names are to be substituted with those names that are provided when the log is processed. Every VCAT encountered in the log must be specified in the **SYSVALUEDDN** data set. This option might be used in the process of cloning a DB2 subsystem.

SYSVALUEDDN ('ddname') Specifies that the DD statement for the control statements specifying the integrated catalog facility (ICF) VCAT aliases used when processing log records.ddname can be up to eight characters, and must start with an alphabetic or national bank character.

The default value is **SYSVALUEDDN(SYSVALUE)**, where **SYSVALUE** identifies the primary data set.

SYSVALUE Data set description VCAT alias values data set Defines a set of records which contain integrated catalog facility (ICF) catalog VCAT alias names. Each record must contain a pair of VCAT alias names separated by only a comma. The first VCAT alias name is the name used when the system level backup was created.

Example 11-25 recovering a backup system after the database volumes have already been restored and VCAT aliases renamed The **LOGONLY** keyword in the following control statement indicates that **RESTORE SYSTEM** is to apply any outstanding log changes to the database, the utility is not to restore the volume copies. In this example, the database volumes have already been restored outside of DB2. Note that **RESTORE SYSTEM** applies log changes; it never restores the log copy pool. The **SWITCH VCAT SYSVALUEDDN(SYSVALUE)** keywords indicate that the **SYSVALUE DD** name data set contains a list of pairs of integrated catalog facility VCAT aliases. The first VCAT alias is the name when the backup was created and the second VCAT alias is the name after any renaming has completed. The VCAT alias **DSNC000** is specified as both the first and second alias since it was not renamed and might be encountered in the log.

Example 11-25 RESTORE SYSTEM LOGONLY SWITCH VCAT

```
//STEP1 EXEC DSNUPROC,TIME=1440,
// UTPROC='',
// SYSTEM='DSN'
//SYSIN DD *
RESTORE SYSTEM LOGONLY SWITCH VCAT SYSVALUEDDN(SYSVALUE)
/*
//SYSVALUE DD *
```

```
VCAT1,VCAT2
VCAT5,Z1234567
DSNC000,DSNC000
/*
```

11.3.3 Remove the incompatibility of REORG and COPY

Some users run online **REORG** for a whole weekend and for business reason they also need to take frequent image copies. DB2 11 removes the incompatibility of **REORG** and **COPY** and it allows **COPY TABLESPACE SHRLEVEL CHANGE** to run at the same time as **REORG TABLESPACE SHRLEVEL CHANGE**, until **REORG** is able to drain the claimers.

For details about which utilities can run concurrently with **COPY** on the same target object, see *DB2 11 for z/OS Utility Guide and Reference*, SC19-4067.

11.3.4 Removal of many point-in-time recovery restrictions

DB2 11 removes many restrictions on point-in-time (PIT), which is recovery to a point prior to the execution of the materializing **REORG**.

PIT recovery restrictions are removed for the following points:

- LOB table spaces
- XML table spaces
- PBR table spaces
- Including when immediate alters have occurred since materializing **REORG**

PIT recovery restrictions are still in place for the following points:

- Table space conversion
- PBG table spaces
- PBG partition pruning
- Online **DROP COLUMN**

There is further information about PIT recovery on 4.1.1, “Scope of enhancements for online schema changes in DB2 11” on page 52.

11.4 LOAD and UNLOAD enhancements

This section describes the following **LOAD** and **UNLOAD** enhancements:

- LOAD SHRLEVEL NONE with PARALLEL option
- LOAD SHRLEVEL CHANGE with PARALLEL option
- Addition of crossloader support for XML
- More offload to zIIP with NPSIs

11.4.1 LOAD SHRLEVEL NONE with PARALLEL option

The best way to **LOAD** is split your input data set into part and load individual data set into part of partition table space. However, splitting your input data set sometimes is not an easy process and you may need to create a batch job to do the work using DFSORT, REXX, COBOL program. DB2 11 deliveries a performance improvement for **LOAD SHRLEVEL NONE** when the input data is in a single input data set with the new **PARALLEL** option, If you specify

the **PARALLEL** keyword, the **LOAD** utility can use multiple parallel subtasks, which can reduce the elapsed time for the load.

This kind of parallelism is useful when the utility is CPU bound. It does not increase I/O parallelism. CPU bottlenecks are typical of CCSID conversion, numeric conversion, compression, complex data types and **VARCHAR** columns. This kind of parallelism is also useful to overlap the synchronous I/Os of multiple indexes. Hence, the more indexes, the greater the benefit.

SHRLEVEL NONE appends to the end of the end of the table instead of trying to store the rows in cluster sequence. Free space searches are not a consideration. However, there is still overhead to insert keys into the index. There can be synchronous I/Os and index splits. Performance is sensitive to whether or not the input data is sorted. If the input data is unsorted, parallelism is of greater value than it would be for presorted input. The CPU overhead of parallelism is also somewhat higher for sorted input. Nevertheless, if the **LOAD** is the type that uses a lot of CPU time, parallelism will help reduce the elapsed time.

With a high degree of parallelism there will be an extra overhead on CPU time, so that should be factored in when determining the degree of parallelism specified with the **PARALLEL** keyword. The recommendation is to specify **PARALLEL 0** or **PARALLEL** without a number specified so DB2 can determine the most optimal degree of parallelism. Whether or not the input data is sorted also affects the performance, because the parallel tasks may suffer from contention.

Measurements show that **LOAD SHRLEVEL NONE PARALLEL** reduced up to 50% of elapsed time with unsorted data, as shown in Figure 11-8.

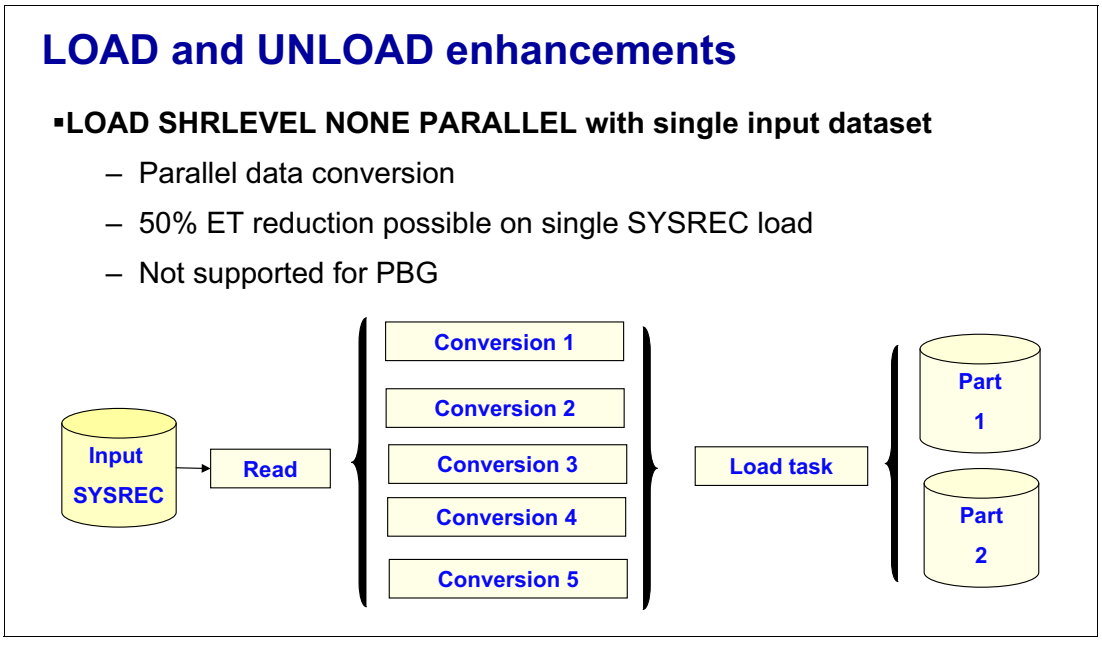


Figure 11-8 LOAD SHRLEVEL NONE PARALLEL

Example 11-26 shows an example of **LOAD SHRLEVEL NONE** with **PARALLEL** option.

Example 11-26 LOAD SHRLEVEL NONE with PARALLEL option

```
/*JOBPARM S=SC63,L=9999
//PROCLIB  JCLLIB ORDER=DB1AM.PROCLIB
//LOAD  EXEC DSNUPROC,SYSTEM=DB1A,
//             LIB='DB1AT.SDSNLOAD',
```

```
//             UID='LOADPP' UTPROC='PREVIEW'
//DSNUPROC.SYSREC DD DSN=DB2R2.SYSREC,DISP=SHR
//DSNUPROC.SYSIN DD *
          TEMPLATE ...
    LOAD DATA INDDN SYSREC
           LOG NO  REPLACE  PARALLEL
           EBCDIC  CCSID(00037,00000,00000)
           WORKDDN(UT1,SRTOUT) DISCARDS 0 DISCARDDN DIS ERRDDN ERR
            COPYDDN COPY STATISTICS TABLE(ALL)  INDEX(ALL)
           INTO TABLE
           "DB2R2".
           "DB2RTB4"
           WHEN(00001:00002) = X'0003'
           NUMRECS                  285
          (COLUMNS..)
```

Example 11-24 shows the job output.

Example 11-27 LOAD SHRLEVEL NONE with PARALLEL option job output

```
1DSNU000I    218 16:43:44.86 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = LOADPP
 DSNU1044I   218 16:43:44.87 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
0DSNU050I    218 16:43:44.88 DSNUGUTC -  TEMPLATE
0DSNU050I    218 16:43:44.88 DSNUGUTC -  LOAD DATA INDDN SYSREC LOG NO REPLACE PARALLEL EBCDIC CCSID(37, 0, 0)
 WORKDDN(UT1, SRTOUT) DISCARDS 0 DISCARDDN DIS ERRDDN ERR COPYDDN COPY STATISTICS TABLE(ALL) INDEX(ALL)
 DSNU650I  -DB1A 218 16:43:44.88 DSNURWI -  INTO TABLE "DB2R2". "DB2RTB4" WHEN(1:2)=X'0003' NUMRECS 285
 DSNU650I  -DB1A 218 16:43:44.88 DSNURWI -   ("DBNAME" POSITION(3:10) CHAR(8),

DSNU650I  -DB1A 218 16:43:44.88 DSNURWI -    "MODECREATED" POSITION(2005:2006) CHAR(2))

 DSNU350I  -DB1A 218 16:43:45.89 DSNURRST - EXISTING RECORDS DELETED FROM TABLESPACE
 DSNU1177I   218 16:43:45.91 DSNURPNP - TABLE SPACE WILL BE LOADED IN PARALLEL, NUMBER OF TASKS = 4
 DSNU400I    218 16:43:45.99 DSNURBID - COPY
 DSNU304I  -DB1A 218 16:43:46.14 DSNURWT - (RE)LOAD PHASE STATISTICS - NUMBER OF RECORDS=285 FOR TABLE DB2R2.DB2RTB4
 DSNU1147I -DB1A 218 16:43:46.14 DSNURWT - (RE)LOAD PHASE STATISTICS - TOTAL NUMBER OF RECORDS LOADED=285 FOR TABLESPACE
 DB2RDB4.DB2RTS1
 DSNU428I    218 16:43:46.14 DSNURILD - DB2 IMAGE COPY SUCCESSFUL FOR TABLESPACE DB2RDB4.DB2RTS1
 DSNU302I    218 16:43:46.14 DSNURILD - (RE)LOAD PHASE STATISTICS - NUMBER OF INPUT RECORDS PROCESSED=285
 DSNU300I    218 16:43:46.14 DSNURILD - (RE)LOAD PHASE COMPLETE, ELAPSED TIME=00:00:01
 DSNU610I  -DB1A 218 16:43:46.43 DSNUSUTP - SYSTABLEPART CATALOG UPDATE FOR DB2RDB4.DB2RTS1 SUCCESSFUL
 DSNU610I  -DB1A 218 16:43:46.43 DSNUSUPT - SYSTABSTATS CATALOG UPDATE FOR DB2R2.DB2RTB4 SUCCESSFUL
 DSNU610I  -DB1A 218 16:43:46.45 DSNUSUPC - SYSCOLSTATS CATALOG UPDATE FOR DB2R2.DB2RTB4 SUCCESSFUL
 DSNU610I  -DB1A 218 16:43:46.45 DSNUSUTB - SYSTABLES CATALOG UPDATE FOR DB2R2.DB2RTB4 SUCCESSFUL
 DSNU610I  -DB1A 218 16:43:46.46 DSNUSUCO - SYSCOLUMNS CATALOG UPDATE FOR DB2R2.DB2RTB4 SUCCESSFUL
 DSNU610I  -DB1A 218 16:43:46.46 DSNUSUTS - SYSTABLESPACE CATALOG UPDATE FOR DB2RDB4.DB2RTS1 SUCCESSFUL
 DSNU620I  -DB1A 218 16:43:46.46 DSNUSEF2 - RUNSTATS CATALOG TIMESTAMP = 2013-08-06-16.43.45.909408
 DSNU010I    218 16:43:46.51 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

11.4.2 LOAD SHRLEVEL CHANGE with PARALLEL option

The **SHRLEVEL CHANGE** option usually has higher CPU time than the **SHRLEVEL NONE** option. Thus, parallelism provides more value for **SHRLEVEL CHANGE** than it does for **SHRLEVEL NONE**.

Unlike **SHRLEVEL NONE**, **SHRLEVEL CHANGE** stores the rows in cluster sequence (rather than appending the rows to the end of the table). As with ordinary SQL inserts, performance is sensitive to space search algorithms and contention between parallel inserts. If the table space has free space, DB2 spends less time searching for space and there is less contention. Whether or not there is contention, parallelism may significantly reduce the elapsed time for

the **LOAD** utility. However, if there is contention, expect a more significant increase in the CPU time, and a lot more CPU increase, than you have with **SHRLEVEL NONE**.

LOAD SHRLEVEL CHANGE PARALLEL for single input data set shows 80% reduction on elapsed time due to multiple parallel subtasks allocation, as shown in Figure 11-9.

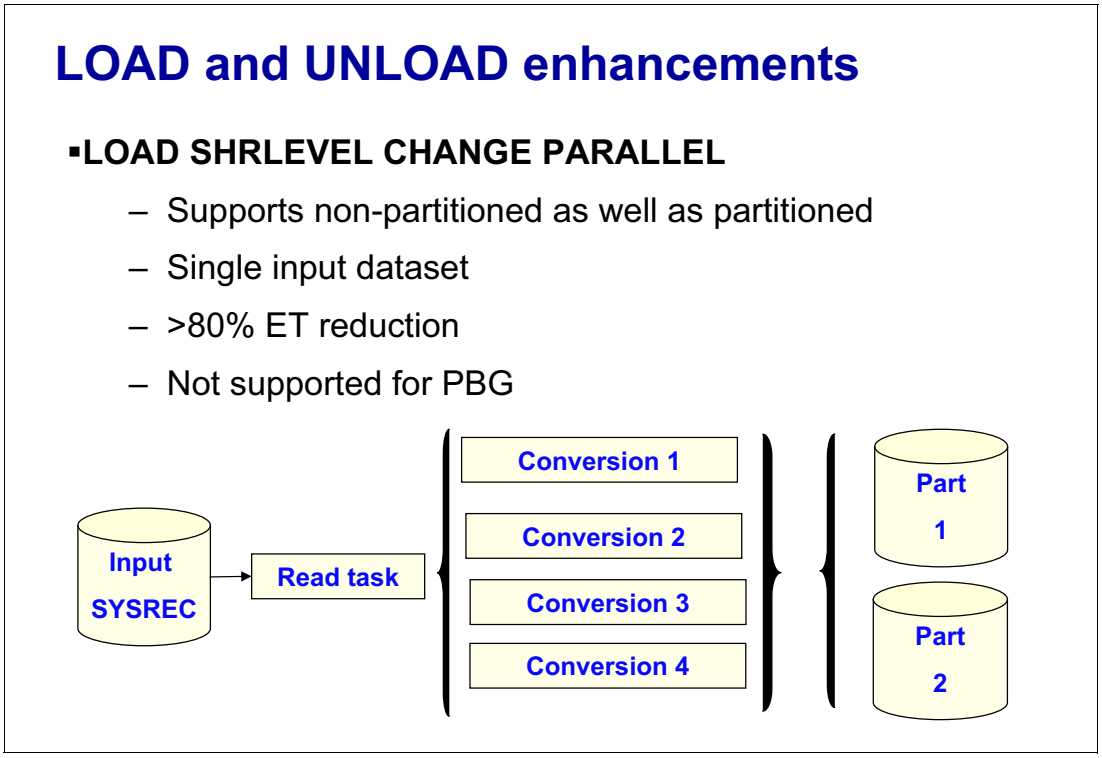


Figure 11-9 LOAD SHRLEVEL CHANGE PARALLEL

Example 11-28 shows an example of **LOAD SHRLEVEL CHANGE** with **PARALLEL** option.

Example 11-28 LOAD SHRLEVEL CHANGE with PARALLEL option

```
/*JOBPARM S=SC63,L=9999
//PROCLIB  JCLLIB ORDER=DB1AM.PROCLIB
//LOAD  EXEC DSNUPROC,SYSTEM=DB1A,
//             LIB='DB1AT.SDSNLOAD',
//             UID='LOADPP' UTPROC='PREVIEW'
//DSNUPROC.SYSREC DD DSN=DB2R2.SYSREC,DISP=SHR
//DSNUPROC.SYSIN DD *
          TEMPLATE ...
      LOAD DATA INDDN SYSREC
          RESUME YES PARALLEL SHRLEVEL CHANGE
          EBCDIC  CCSID(00037,00000,00000)
          WORKDDN(UT1,SRTOUT) DISCARDS 0 DISCARDDN DIS ERRDDN ERR
          INTO TABLE
          "DB2R2".
          "DB2RTB4"
          WHEN(00001:00002) = X'0003'
          NUMRECS                  285
          (COLUMNS..)
```

Example 11-29 shows the job output.

Example 11-29 LOAD SHRLEVEL CHANGE with PARALLEL option job output

```
1DSNU000I    218 16:45:21.01 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = LOADPP
 DSNU1044I   218 16:45:21.05 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
0DSNU050I    218 16:45:21.06 DSNUGUTC -  TEMPLATE

0DSNU050I    218 16:45:21.06 DSNUGUTC -  LOAD DATA INDDN SYSREC RESUME YES PARALLEL SHRLEVEL CHANGE
EBCDIC CCSID(37, 0, 0) WORKDDN(UT1, SRTOUT) DISCARDS 0 DISCARDDN DIS ERRDDN ERR
 DSNU650I  -DB1A 218 16:45:21.06 DSNURWI -  INTO TABLE "DB2R2". "DB2RTB4" WHEN(1:2)=X'0003' NUMRECS 285
 DSNU650I  -DB1A 218 16:45:21.06 DSNURWI -   ("DBNAME" POSITION(3:10) CHAR(8),

DSNU650I  -DB1A 218 16:45:21.06 DSNURWI -    "MODECREATED" POSITION(2005:2006) CHAR(2))

 DSNU1177I   218 16:45:21.17 DSNURPLL - TABLE SPACE WILL BE LOADED IN PARALLEL, NUMBER OF TASKS = 24
 DSNU397I    218 16:45:21.17 DSNURPLL - NUMBER OF TASKS CONSTRAINED BY CPUS TO 24
 DSNU1114I -DB1A 218 16:45:21.20 DSNURWT - (RE)LOAD PHASE STATISTICS - NUMBER OF RECORDS LOADED =285 FOR TABLE
 DB2R2.DB2RTB4
 DSNU1147I -DB1A 218 16:45:21.20 DSNURWT - (RE)LOAD PHASE STATISTICS - TOTAL NUMBER OF RECORDS LOADED=285 FOR TABLESPACE
 DB2RDB4.DB2RTS1
 DSNU302I    218 16:45:21.21 DSNURILD - (RE)LOAD PHASE STATISTICS - NUMBER OF INPUT RECORDS PROCESSED=285
 DSNU300I    218 16:45:21.21 DSNURILD - (RE)LOAD PHASE COMPLETE, ELAPSED TIME=00:00:00
 DSNU010I    218 16:45:21.23 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

Note: WIth the **LOAD PARALLEL** option, enable parallelism for the following table space types:

- Simple
- Segmented
- Classic partitioned
- Partition by range

PBG is not supported.

PARALLEL keyword for faster utilities: DB2 11 improves the algorithm that decides the level of utilities parallelism for faster utilities. You can find further information about the use of **PARALLEL** keyword for utilities in 4.5, “Governing of parallel processing of utilities” on page 70.

11.4.3 Addition of crossloader support for XML

DB2 10 implemented the crossloader support for LOB and DB2 11 adds the support for XML data and it also reduces virtual storage requirement and avoids DSNU1178i errors by exploiting **FETCH CONTINUE** for processing large LOBs and XML data in crossloader.

11.4.4 More offload to zIIP with NPSIs

When users want to clean the whole partition of a table space executing L**OAD REPLACE PART** with dummy input, DB2 11 reroutes up to 100% of the workload to eliminate the rows from NPSIs to zIIP.

11.5 Compression dictionaries for Change Data Capture

For users that use IFI 306, QREP users need to read log records, decompress them and process them. If the users have run a **REORG** or **LOAD** and built a new compression dictionary,

IFI 306 readers still need the old dictionary to read data in the log. DB2 11 stores the old compression dictionary in the log and IFI 306 read automatically retrieves old compression dictionary if necessary. This new feature is transparent for IFI 306 readers, because it was implemented in DB2.

You can find more information about compression dictionary in 4.6, “Compression dictionary availability for CDC tables” on page 72.

11.6 General enhancements

This section describes the following DB2 utilities general enhancements:

- DISPLAY UTILITY additional output
- Improved TEMPLATE for extended format data sets
- DSN1COPY
- Command to externalize RTS statistics
- DSNACCOX

11.6.1 DISPLAY UTILITY additional output

Users match utility ID with the job name to identify the job when executing the **-DISPLAY UTILITY** command. DB2 11 modifies this command output to show JOBNAME and the start time stamp. See Example 11-30.

Example 11-30 DISPLAY UTILITY command output

```
DSNU105I  -DB1A DSNUGDIS - USERID = DB2R2
                MEMBER =
                UTILID = OREORG
                PROCESSING UTILITY STATEMENT 1
                UTILITY = REORG
                PHASE = RELOAD   COUNT = 0
                NUMBER OF OBJECTS IN LIST = 1
                LAST OBJECT STARTED = 1
                STATUS = ACTIVE
                JOBNAME = DB2R2TSC
                TIME STARTED = 2013-08-08-20:30:05
 DSNU347I  -DB1A DSNUGDIS -
                DEADLINE = NONE
 DSNU384I  -DB1A DSNUGDIS -
                MAXRO = 180 SECONDS
                LONGLOG = CONTINUE
                DELAY = 1200 SECONDS
 DSNU111I  -DB1A DSNUGDIS - SUBPHASE = COPY COUNT = 10
 DSN9022I  -DB1A DSNUGCCC '-DIS UTIL' NORMAL COMPLETION
```

11.6.2 Improved TEMPLATE for extended format data sets

Utility **TEMPLATE**s are heavily used by users to automatically allocate utility data sets. As user data has grown, the size of the utility data sets has also increased and it can exceed the maximum size of basic sequential data sets.

Users would like to specify **DSNTYPE** on their **TEMPLATE**s to easily automate the allocation of large format or extended format sequential data sets which have a much larger maximum size than basic sequential data sets. Today users must change their SMS data class routines to use large format or extended format sequential data sets. Also, the date and time variables for TEMPLATE data set names are always resolved to reflect the Coordinated Universal Time (UTC) values rather than the local date and time. This makes it difficult to correlate the utility data set to the local date and time when it was created.

DB2 11 enhances **TEMPLATE**s by adding a new **DSNTYPE** options to support large format and extended format data sets and adds new **TEMPLATE** data set name **DATE** and **TIME** variables to support the options.

DSNTYPE Specifies the type of data set to be allocated.

The new options are as follows:

BASIC	Specifies a basic format data set. No more than 65535 tracks can be allocated.
LARGE	Specifies a large format data set. Greater than 65535 tracks can be allocated.
EXTREQ	Specifies an extended format data set is required.
EXTPREF	Specifies an extended format data set is preferred.
TIME	Specifies time used in expansion of date and time DSN variables. The default TIME value is determined by the **TEMPLATE_TIME** subsystem parameter.
LOCAL	Use local time at the DB2 server in the expansion of date and time in DSN variables.
UTC	Use Coordinated Universal Time (UTC) in the expansion of date and time in DSN variable

Example 11-31 shows a **COPY** utility execution with extended format required and time (of the store clock of the INIT phase) in local time.

Example 11-31 TEMPLATE with DSNTYPE EXTREQ and TIME LOCAL

```
/*JOB
//PROCLIB  JCLLIB ORDER=DB1AM.PROCLIB
//REORG EXEC DSNUPROC,SYSTEM=DB1A,
//             LIB='DB1AT.SDSNLOAD',
//             UID='OCOPY' UTPROC='PREVIEW'
//DSNUPROC.SYSIN DD *
          TEMPLATE COPY DSN 'DB1AD.&DB..&TS..T&TIME.'
           DISP (NEW,CATLG,DELETE) UNIT SYSDA
           DSNTYPE EXTREQ TIME LOCAL
           SPACE (5,5) CYL
          COPY  TABLESPACE DSN8D11A.DSN8S11E
           COPYDDN(COPY)
1DSNU000I    220 20:54:12.94 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = OCOPY
 DSNU1044I   220 20:54:12.95 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
0DSNU050I    220 20:54:12.95 DSNUGUTC -  TEMPLATE COPY DSN 'DB1AD.&DB..&TS..T&TIME.'
DISP(NEW, CATLG, DELETE) UNIT SYSDA DSNTYPE EXTREQ TIME LOCAL SPACE(5, 5) CYL
 DSNU1035I   220 20:54:12.95 DSNUJTDR - TEMPLATE STATEMENT PROCESSED SUCCESSFULLY
0DSNU050I    220 20:54:12.95 DSNUGUTC -  COPY TABLESPACE DSN8D11A.DSN8S11E COPYDDN(COPY)
 DSNU1038I   220 20:54:13.00 DSNUGDYN - DATASET ALLOCATED.  TEMPLATE=COPY
                        DDNAME=SYS00001
                        DSN=DB1AD.DSN8D11A.DSN8S11E.T205412
 DSNU400I    220 20:54:13.07 DSNUBBID - COPY PROCESSED FOR TABLESPACE DSN8D11A.DSN8S11E
```

```
                     NUMBER OF PAGES=15
                     AVERAGE PERCENT FREE SPACE PER PAGE =  4.40
                     PERCENT OF CHANGED PAGES =  0.00
                     ELAPSED TIME=00:00:00
 DSNU428I  -DB1A 220 20:54:13.08 DSNUBAFI - DB2 IMAGE COPY SUCCESSFUL FOR TABLESPACE
DSN8D11A.DSN8S11E
 DSNU010I    220 20:54:13.08 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0

Here is generated dataset.
NONVSAM ------- DB1AD.DSN8D11A.DSN8S11E.T205412
      IN-CAT --- UCAT.DB1ADATA
      HISTORY
        DATASET-OWNER-----(NULL)     CREATION--------2013.220
        RELEASE----------------2     EXPIRATION------0000.000
        ACCOUNT-INFO-----------------------------------(NULL)
      SMSDATA
        STORAGECLASS ---DB1ADATA     MANAGEMENTCLASS---MCDB22
        DATACLASS --------(NULL)     LBACKUP ---0000.000.0000
      VOLUMES
        VOLSER------------SBOXG2     DEVTYPE------X'3010200F'     FSEQN---------
---------0
      ASSOCIATIONS--------(NULL)
      ATTRIBUTES
        STRIPE-COUNT-----------1
        EXTENDED
```

11.6.3 DSN1COPY

Users very often take a **DSN1COPY** of a table space and use that copy to populate another table space in the same or different DB2 subsystem. If an improper process is used to transfer data with **DSN1COPY**, it can cause abends, data corruption or storage overlays in the target table space. The data integrity issue usually is not discovered until data has been stored into the target table space and system failure has occurred.

The typical scenarios are the following:

- A wrong DBID, PSID or OBID is specified while transferring the data using **DSN1COPY OBIDXLAT** option.
- The user copies data from one type of table space and populates it to a different type of table space, except XML table space.
- The user takes a **DSN1COPY** of a table space and populates data to the other table space which has a different data version number and/or table schema definition.
- The user takes a **DSN1COPY** of a table space that is in basic row format (BRF) and copies it to a table space that is in reordered row format (RRF) or vice versa.
- The **DSN1COPY** table has 10 bytes expanded RBA format but the target table space has still the basic 6 bytes RBA format or vice versa.

DB2 11 improves data availability by providing the validation between data and catalog definition during the first physical open of the page set. If any inconsistency is found in the process, access to the data is blocked and an error message is issued. The following data mismatch is detected during physical open:

- DBID, PSID, or OBID
- **SEGSIZE** or **PAGESIZE**
- Table space type definition. This includes the following table space type:

  - Segmented table space
  - Classic partition table space
  - Partition by range universal table space
  - Partition by range universal table space with **MEMBER CLUSTER** attribute
  - Partitioned by growth universal table space
  - Partitioned by growth universal table space with **MEMBER CLUSTER** attribute
  - XML version table space
- Table schema definition if there is only one table in the table space and there is an OBDREC stored in the system page.

In addition, the **REPAIR CATALOG** utility can also correct the following mismatches in the catalog (Record format, RBA format, Data Version number and Hash Data Page value) see 11.6.4, “REPAIR utility” on page 310.

As an example, if the physical open of data set triggered by DML detects that the data is populated by DSN1COPY, the following data is validated.

- DBID, PSID, or OBID on the header page is validated. The OBID on the header page only records the first table on the table space. Therefore, the OBID validation can be done if one table resides in the table space. If error found for this case, a SQLCODE -904 with reason code 00C900E0 is issued.
- The **SEGSIZE** or **PAGESIZE** and table space type definition on the header page is validated.
  - If error found due to mismatch of **SEGSIZE**, a SQLCODE -904 with reason code 00C900E1 is issued.
  - If error found due to mismatch of page size, a SQLCODE -904 with reason code 00C900E2 is issued.
- Table schema definition is validate if there is only one table in the table space and also the system page exists in the page set. If the system page exists, then the latest version of the OBDREC will be retrieved from system page. Total number of columns and each column's data type and its data length is validated with current OBDREC.
  - If any error found due to mismatch of schema, a SQLCODE -904 with reason code 00C900E3

As a result, any error found during this process, a SQLCODE -904 with proper reason code is issued. The physical open process is terminated.

To avoid any performance impact during physical open, the validation of data against catalog definition is not performed if the following condition exists:

- Physical open is triggered by Utility.

  This include REPAIR VERSIONS and REPAIR CATALOG utility
- Restart of the DB2
- Header page has not been formatted yet.
- **REPAIR** utility operates on the header page.

  At the end of **REPAIR** utility, it will close the page set. Therefore, the validation can be done by the subsequent physical open.
- **LOG** apply phase

11.6.4 REPAIR utility

The **REPAIR** utility now checks for and fixes any inconsistencies between the information in the catalog and the data. Specifically the **REPAIR** utility can check the values for DBID, PSID, OBID, **SEGSIZE**, **PAGESIZE**, table space type, table schema, record format, RBA format, data version number, and the hash data page.

The record row format, RRF, BRF or RBA, format such as 6 byte verse 10 byte RBA format will not be validated during physical open because the information about the header page can represent the actual data format. The mismatch between catalog and data will not interfere with the accessing of data.The DB2 **REPAIR** utility will provide the similar capability to detect the data inconsistency as data set open triggered by DML explained above.

There are two new keywords added to the REPAIR Utility:

- **REPAIR CATALOG**
- **REPAIR CATALOG TEST**

REPAIR Utility with CATALOG keyword

Example of statement:

```
REPAIR CATALOG TABLESPACE DBNAMET01.TSNAMET01
```

A new keyword **CATALOG** is introduced. This includes validation of metadata and the original function of **REPAIR VERSION** utility. As the result, the following data is modified in the OBDs.

- RRF/BRF format. During the mainline DML code path or utility path, the reference of record format is always based on the physical data. However, if it is necessary, the REPAIR utility can be used to fix the catalog definition.
- The format now supports 6 bytes RBA format versus 10 bytes RBA format, which is similar to RRF/BRF format.
- Data version information. This process is equivalent to the current **REPAIR VERSION** utility. Unlike data checking during the physical open, the **REPAIR** utility checks the table's schema for all the tables in the table space. However, only the table space has system page and its latest version from the system page will be used to validate its schema with catalog.
- Hash data page value. This is similar to RRF/BRF record format behavior, the mainline or utility code path can tolerate the inconsistency between data and catalog definition. However, this value can effect **REORG** or **LOAD REPLACE** utility if it is not being corrected. As the result, the **REPAIR** utility will modify the catalog be match the data set.

The **REPAIR** utility updates the following catalog columns to match the physical data in the table space, as listed in Table 11-3.

Table 11-3 REPAIR CATALOG utility will update the following catalog columns

Table	Column	Description
SYSTABLEPART	**FORMAT**	Updated to match the Reorder Row Format of the table space
SYSTABLEPART	**RBA_FORMAT**	Updated to match the RBA format of the Table Space
SYSTABLEPART	**HASHDATAPAGES**	Updated with the hash data page found in the PBR Universal Table Space if table has hash organization

Table	Column	Description
SYSTABLESPACE	HASHDATAPAGES	Updated with the hash data page found in the PBG Universal Table Space if table has hash organization
SYSTABLES	VERSION	Updated with the highest version in the table space
SYSTABLEPART	OLDEST_VERSION	Updated with the lowest version in the table space
SYSTABLESPACE	OLDEST_VERSION	Updated with the lowest version in the table space
SYSTABLESPACE	CURRENT_VERSION	Updated the highest version in the table space
SYSCOPY	ICTYPE	Put a new row with ICTYPE='V' to indicate REPAIR touched the object

In addition, the **REPAIR** utility with the **CATALOG** keyword also validates the following information.

- DBID/PSID mismatch. If there is only one table in the table space, the **REPAIR** utility also validates OBID mismatch between page set and catalog.
- Table space type mismatch.
- **SEGSIZE** mismatch.
- **PAGESIZE** mismatch.
- Table definition mismatch. This validation includes the total number of columns, column data type and column defined length of the table.

As a result, If any mismatch is found between the data and catalog, the **REPAIR** utility cannot repair the information in catalog. The **REPAIR** utility fails with a return code 8 and reports the mismatch with a message. You need to take an action to correct the mismatch information.

Note: **REPAIR CATALOG** does not check limit key values.

You cannot specify **CATALOG** for LOB or XML table spaces

REPAIR CATALOG does not make any corrections for indexes. If you or **REPAIR** made corrections to the data or catalog as a result of running **REPAIR CATALOG**, rebuild any indexes on the target tables

REPAIR CATALOG TEST utility

Example of statement:

```
REPAIR CATALOG TABLESPACE DBNAMET01.TSNAMET01 TEST
```

A new keyword **CATALOG TEST** is introduced. The function of this keyword is similar to the **REPAIR CATALOG** except the mismatch information will not be corrected in the catalog. The mismatch information that results in none zero return code will be reported.

11.6.5 Command to externalize RTS statistics

The real-time **SYSIBM.SYSTABLESPACESTATS** and **SYSIBM.SYSINDEXSPACESTATS** statistic tables provide statistical information about the table and index spaces in the database system. The tables are updated every 30 minutes (by default). As a consequence, the statistics are 15 minutes old (on average) when a user or a tool are querying the tables. This can causes

some tools which rely on RTS information, such as DSNACCOX, to provide wrong recommendations on heavily changed objects. DB2 11 implements a new command option added to the DB2 **ACCESS** command, so that the users can trigger the externalization of the in-memory RTS blocks, before calling recommendation tools, such as DSNACCOX.

A new **STATS** keyword value added to the existing **MODE** keyword on the DB2 **ACCESS** command. The **MODE(STATS)** option results in the externalization of the real-time statistics in memory blocks.

The DB2 **ACCESS DATABASE** command forces a physical open of a table space, index space, or partition, or removes the GBP-dependent status for a table space, index space, or partition, and externalizes the real-time statistics in-memory blocks to the real-time statistics tables. The **MODE** keyword specifies the desired action.

MODE(STATS) Externalizes the in-memory statistics to the real-time statistic tables. In data sharing environments, the in-memory statistics are externalized for all members. This mode does not physically open the page sets or change the states of the page sets.

Examples are:

```
ACCESS DB(dbname) SP(spname) MODE(STATS) PART(part)
ACCESS DB(dbname) SP(spname) MODE(STATS)
ACCESS DB(dbname) SP(*) MODE(STATS)
ACCESS DB(*) SP(*) MODE(STATS)
```

The last sample command externalizes all in-memory statistics that are currently held in the system to the real-time statistics table.

11.6.6 DSNACCOX

DSNACCOX is a sample stored procedure that uses data from the **SYSIBM.SYSTABLESPACESTATS** and **SYSIBM.SYSSYSINDEXSPACESTATS** real-time statistics tables to make the following recommendations:

- It recommends when you should reorganize, image copy, or update statistics for table spaces or index spaces.
- Indicates when a data set has exceeded a specified threshold for the number of extents that it occupies.
- Indicates whether objects are in a restricted state.

DB2 11 implements the following enhancement on **DSNACCOX**:

- When the user has a lot of small objects that are almost empty, **DSNACCOX** recommends **REORG** on the objects because the **DATASPACERATIO** threshold had been exceeded, so for small objects Space Allocated/Space Used is always larger than the default **DATASPACERATIO** of 2. DB2 11 changes the default to -1 which turn off this criteria and it does not effect users which had changed the value. If you want to use **DATASPACERATIO**, modify the application to pass a positive value for **DATASPACERATIO** parameter.
- Unable to differentiate between LOB, XML, or base table spaces, **DSNACCOX** used to return all table space as TS (standard table space). DB2 11 returns object type XS and LS to differentiate XML and LOB table spaces from TS for regular table space.
- **DSNACCOX** used to recommend REORG on an NPI even when one or more of the table space partitions have been recommended for **REORG** or **RUNSTATS**. DB2 11 enhances **DSNACCOX** to optionally skip **REORG** recommendation on NPI index when any of the table space partitions had been recommended for **REORG**.

- Better granularity in the evaluation. Many users rely on frequent runs of **DSNACCOX**, but they might be interested only in a certain objects. DB2 10 evaluates every rows in the RTS tables and eliminates them using an SQL **WHERE** clause when the return cursor is opened. DB2 11enhances **DSNACCOX** to parse the **CRITERIA** input parameter, and if possible, apply the **CRITERIA** as a **WHERE** clause when **DSNACCOX** queries RTS tables. This action results in less objects to be evaluated, improving performance.

  Figure 11-10 shows **DSNACCOX** performance when the **DBNAME** criterion is applied.

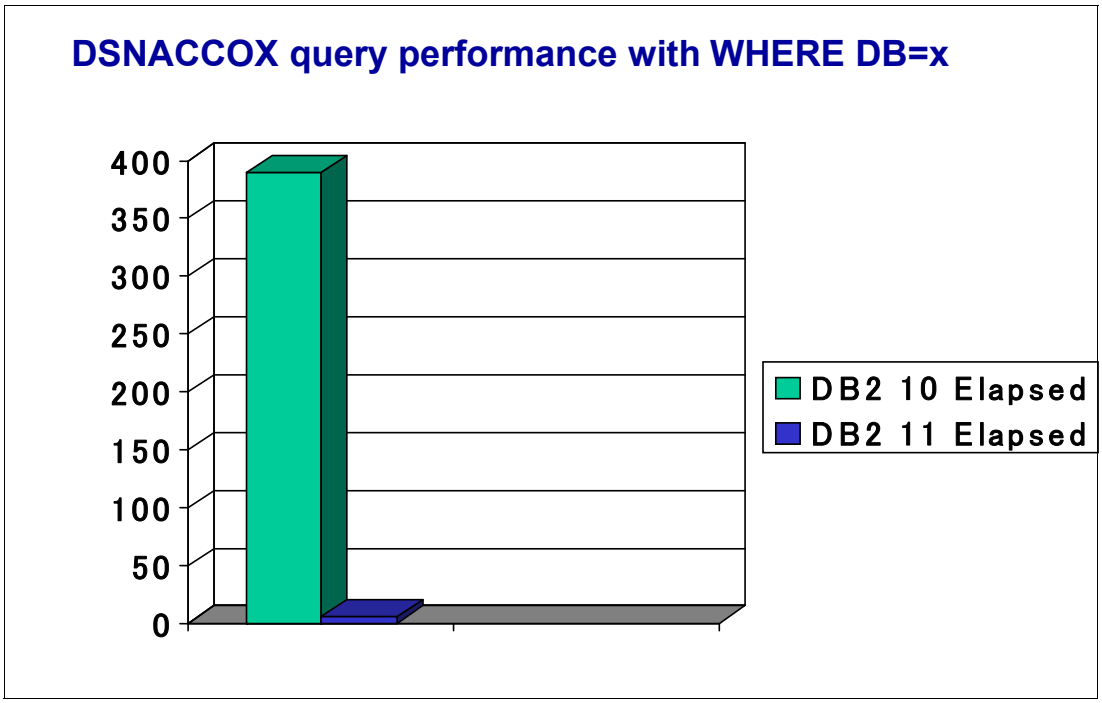


Figure 11-10 DSNACCOX performance

11.7 Deprecated functions

DB2 10 NFM does not support **REORG SHRLEVEL NONE** for LOBs, but it returns RC0 with **MSGDSNU126I**. In DB2 11 when users try to run **REORG SHRLEVEL NONE** for LOBs the return code changed to RC8.

When **DSN1CHKR** is invoked in DB2 11, you will see the same messages as DB2 10 NFM. It will produce following messages:

```
DSN1800I START OF DSN1CHKR FOR JOB - DSN1CHKM, DURING JOBSTEP - xxxxxxxx
DSN1998I INPUT DSNAME = DSNC000.xxxxxx.xxxxxxx.xxxxxxxx.I0001.A001 , VSAM
DSN1810I INPUT DATA SET INVALID, NOT A CHECKABLE SYSTEM TABLESPACE
DSN1816I DSN1CHKR TERMINATED WITH ERRORS, 00000000 PAGES PROCESSED
```

The DB2 11 documentation was updated to indicate that DSN1CHKR is no longer supported, and that **DSN1810I** and **DSN1816I** error messages continue to be generated when it is invoked. The DB2 10 documentation was updated to indicate that **DSN1CHKR** is deprecated.

DB2 11 begins the deprecation process of utility functions which are replaced by a new feature or can now be done by a non utility application. The **INDREFLIMIT**, **OFFPOSLIMIT**, **LEAFDISTLIMIT**, and **CHANGELIMIT** keywords were introduced before tools or stored procedures were available to give recommendations when an object needed to be reorganized or copied.

However, tools and a stored procedure are now available to provide recommendations for **REORG** and **COPY**. Thus, you can use those tools or the **DSNACCOX** stored procedure.

The DB2 Utility options in Table 11-4 are deprecated. Although they are supported in DB2 V11, they will be removed in a later release of DB2.

Table 11-4 DB2 Utility options deprecated

Function	Option
REORG TABLESPACE UNLOAD ONLY	Use the **UNLOAD** utility instead
REORG TABLESPACE UNLOAD PAUSE	Use the **UNLOAD FORMAT INTERNAL** utility instead
REORG TABLESPACE UNLOAD EXTERNAL	Use the **UNLOAD** utility instead
REORG TABLESPACE INDREFLIMIT	Use the **DSNACCOX** to determine whether the object needs to be reorganized
REORG TABLESPACE OFFPOSLIMIT	Use the **DSNACCOX** to determine whether the object needs to be reorganized
REORG TABLESPACE INDREFLIMIT|OFFPOSLIMIT REPORTONLY	**REPORTONLY** valid only with **INDREFLIMIT** or **OFFPOSLIMIT**
REORG INDEX LEAFDISTLIMIT REPORTONLY	**REPORTONLY** valid only when **LEAFDISTLIMIT** is specified
REORG INDEX UNLOAD ONLY	If the function is needed, use **DIAGNOSE** to stop the process
REORG INDEX UNLOAD PAUSE	If function is needed, use **DIAGNOSE** to stop the process
REORG INDEX LEAFDISTLIMIT	Use **DSNACCOX** to determine whether the object needs to be reorganized
LOAD FORMAT UNLOAD	Use **LOAD FORMAT INTERNAL** to load data unloaded with **UNLOAD FORMAT INTERNAL**
COPY CHANGELIMIT	Use **DSNACCOX** to determine whether the object needs to be copied
REPAIR VERSIONS	Use the **REPAIR CATALOG** instead

12

Installation and migration

This chapter provides information to help you evaluate the changes in DB2 11 for z/OS and to plan for a successful installation of or migration to DB2 11 for z/OS. It includes the following topics:

- Currency of versions and migration paths
- Prerequisites for DB2 11
- DB2 11 installation changes and considerations
- Considerations for migrating to DB2 11
- Subsystem parameters
- Release incompatibilities
- Controlling application compatibility

12.1 Currency of versions and migration paths

Figure 12-1 is an overview over the general availability (GA) and end of service (EOS) dates for DB2 for z/OS.

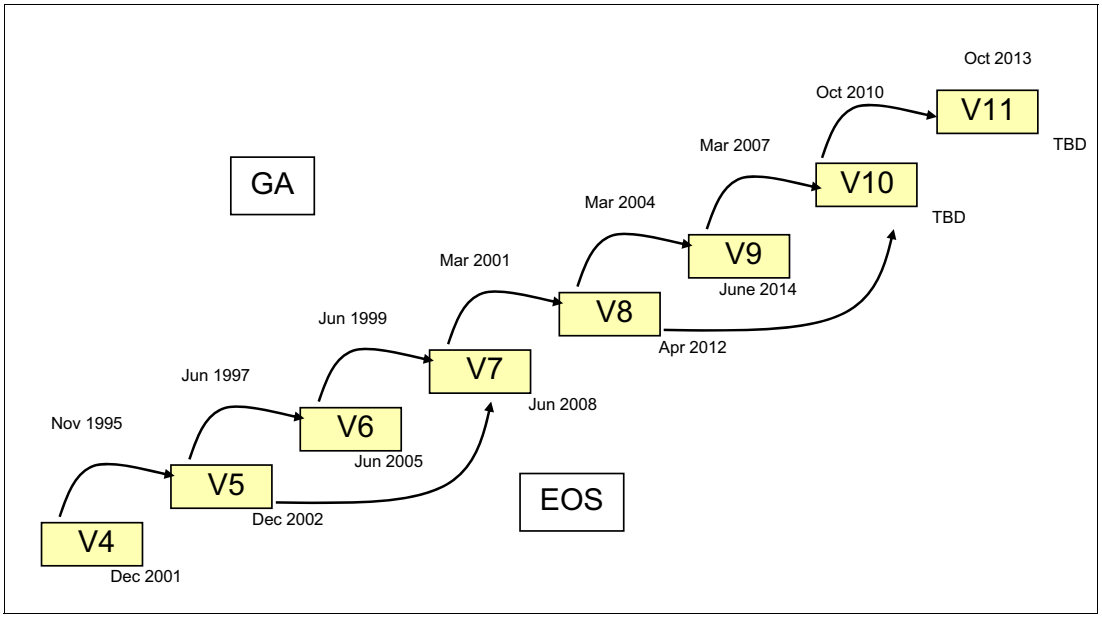


Figure 12-1 Currency of versions

Before you begin the installation or migration process, look at the big picture. You need to be aware of the major requirements to get from your current version of DB2 to DB2 11 for z/OS. You need to know where you are currently and where you need to be before you embark on this process considering DB2, z/OS, and tools.

Figure 12-2 points out the versions, currency dates, and the minimum required z/OS levels.

Version	PID	Generally available	OS prereq	Marketing withdrawal	End of service
V3	5685-DB2	December 1993	MVS V4R3	February 2000	January 2001
V4	5695-DB2	November 1995	MVS V4R3	December 2000	December 2001
V5	5655-DB2	June 1997	MVS V4R3	December 2001	December 2002
V6	5645-DB2	June 1999	OS/390 V1R3	June 2002	June 2005
V7	5675-DB2	March 2001	OS/390 V2R7	March 2007	June 2008
V8	5625-DB2	March 2004	z/OS V1R3	September 2009	April 2012
V9	5635-DB2	March 2007	z/OS V1R7	December 2012	June 2014
V10	5605-DB2	October 2010	z/OS V1R10	TBD	TBD
V11	5615-DB2	October 2013	z/OS V1R13		

Figure 12-2 DB2 versions and required z/OS level

The discussion in this book, and mostly in Chapter 2, “Synergy with System z” on page 7, has described the functions of DB2 that you can enable if your z/OS level is z/OS 2.1, rather than the minimum required level z/OS 1.13. If there are functions you need with z/OS 2.1, you need to take into account the z/OS migration first.

As shown in Figure 12-1, for DB2 10 provides the opportunity to get to it by using a skip-level migration from DB2 V8. For DB2 11 skipping level is not allowed. Thus, if you are on DB2 9, you first have to move to DB2 10 to migrate to DB2 11. As a result, the migration process has fewer modes than for the migration to DB2 10.

The modes that you can use while migrating to DB2 10 are DB2 11 conversion mode (CM), DB2 11 CM*, DB2 11 ENFM, DB2 11 ENFM*, DB2 11 NFM.

Figure 12-3 summarizes the possible migration modes and how you can get to each of them.

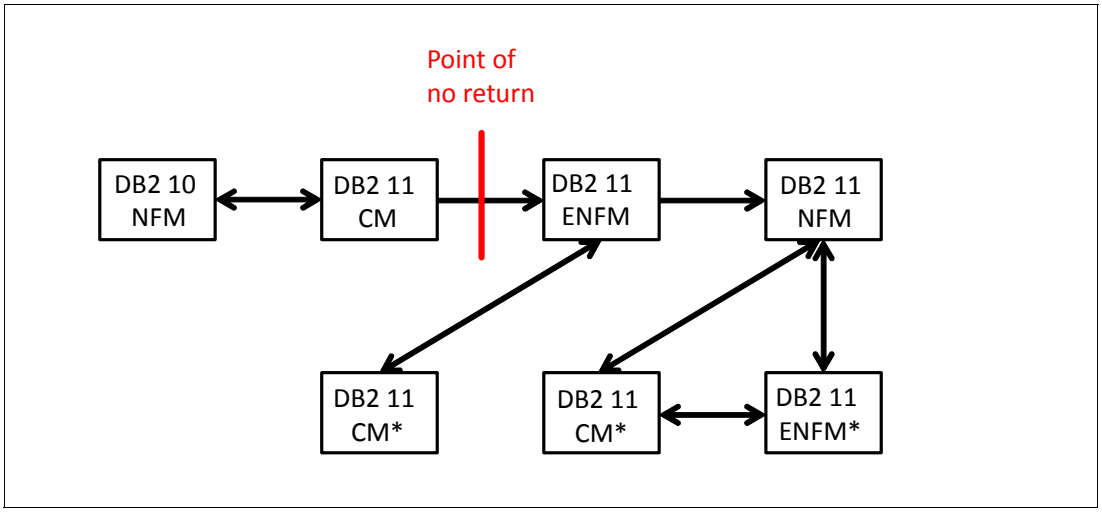


Figure 12-3 Migration modes and paths

DB2 can operate in the following various mode during the migration to DB2 11: =

CM — Conversion mode is the mode DB2 is in when DB2 11 is started for the first time after migration from DB2 10. When DB2 11 is started for the first time, you see messages indicating that the code DB2 is running under is the DB2 11 code, but that the catalog is still in DB2 10. You must fix this mismatch after you started DB2 with V11 code. Job **DSNTIJTC** is the installation job that handles this catalog adjustment. After successful execution of **DSNTIJTC** DB2 still remains in DB2 11 CM. Data sharing systems can be migrated from DB2 10 NFM to DB2 11 CM one member at a time. **DSNTIJTC** needs to be executed only once, because it is in the nature of a DB2 data sharing group that the catalog is common and shared among the members. CM is the only mode that allows for release co-existence between DB2 10 and DB2 11 members.

Important: Fallback to DB2 10 NFM is possible only from CM. This point is illustrated by the vertical bar in Figure 12-3 and is labelled as the *Point of no return*.

ENFM — The Enabling New Function Mode is entered after installation job DSNTIJEN is executed. This job invokes the CATENFM utility with the

	START option, which prepares the DB2 catalog for DB2 11. H describe this step in 12.4.3, “DB2 11 ENFM and NFM” on page 339. DB2 remains in this mode until all the enabling functions are completed. Data sharing systems can only have DB2 11 members in this mode.
NFM	After the catalog migration completes successfully, you can use job DSNTIJNF, which also invokes the CATENFM utility, but this time with the COMPLETE option to reach New Function Mode. This mode indicates that all catalog changes are complete and new functions can now be used.
ENFM*	The ENFM* mode is the same as ENFM, but the * indicates that at one time DB2 was at DB2 11 NFM. Objects that were created when the system was at NFM can still be accessed but no new objects can be created. When the system is in ENFM* it cannot fallback to DB2 10 or coexist with a DB2 10 system.
CM*	This mode is the same as CM, but the * indicates that at one time the subsystem was at a higher level. Objects that were created at the higher level can still be accessed. When DB2 is in CM* it cannot fallback to DB2 10 or coexist with a DB2 10 system.

12.2 Prerequisites for DB2 11

This section describes the prerequisite requirements for hardware and software to successfully install and work with DB2 11 for z/OS.

12.2.1 Processors

DB2 11 operates on IBM z10™ or later processors running z/OS 1.13 or later. The processors must have enough real storage to satisfy the combined requirements of:

- DB2 11 for z/OS
- z/OS
- The appropriate DFSMS storage management subsystem components, access methods, telecommunications, batch requirements and other applications required in your environment.

Tip: DB2 11 requires increased real storage as compared to DB2 10 for z/OS.

12.2.2 Auxiliary storage

The minimum disk space requirement, based on installing DB2 using the panel default values is approximately 1.3 GB. You need additional space for your data.

Note: The default values might not meet your installation’s needs. Over time more disk space might be required for your DB2 subsystems.

12.2.3 Operational requirements

Operational requirements are the products that are required and must be present on the system or the products that are not required but should be present on the system for this product to operate all or part it its functions.

Mandatory requirements

Mandatory operational requisites identify products that are required for DB2 to operate its basic functions. Table 12-1 lists these requirements tor DB2 11.

Table 12-1 Mandatory operational requirements

Program number	Product name and minimum service level
5615-DB2	DB2 11 for z/OS, DB2 base APAR PM93577
5615-DB2	DB2 11 for z/OS, internal resource lock manager (IRLM) 2.3, plus APARs PM84765 and PM85053
Any one of the following:	
5694-A01	z/OS (DFSMS, IBM Language Environment® Base Services, Security Server/RACF) V1.13
5650-ZOS	z/OS (DFSMS, Language Environment Base Services, Security Server/RACF) V2.1

The following functions in DB2 11 require z/OS V2.1

- 2 GB large pages
- 1 MB fixed page frames for DB2 execution code
- Improved performance of batch updates in data sharing
- Improved usability and consistency for security administration

For details, see Chapter 2, “Synergy with System z” on page 7.

Conditional operational requirements

Conditional operational requisites identify products that are not required for DB2 11 to operate its basic functions but are required at run time to operate specific functions. Table 12-2 lists the requirements.

Table 12-2 Target system conditional operational requirements

Program number	Product name and minimum service level	Function
5655-N98	IBM SDK for z/OS, Java 2 Technology Edition	Applications or stored procedures written in Java, such as those using the JDBC or SQLJ interfaces to DB2; Decimal Float data type usage in Java (in a 31-bit environment)
5655-N99	IBM SDK for z/OS, Java 2 Technology Edition	Applications or stored procedures written in Java, such as those using the JDBC or SQLJ interfaces to DB2; Decimal Float data type usage in Java (in a 64-bit environment)

Program number	Product name and minimum service level	Function
5697-A01	z/OS 1.13 Web Deliverable	1 MB pageable for new FRAMESIZE and PGFIX(NO) in DB2 11 for buffer pool
5697-A01	z/OS 1.13 APAR OA40967	2 GB page for new FRAMESIZE option in DB2 11 for buffer pool
5635-A02	Information Management System (IMS) V11.01.00	Transaction Management
5655-M15	Customer Information Control System (CICS) Transaction Server for z/OS V03.01.00 and V03.02.00	Transaction Management For V03.01.00 and V03.02.00 you need APAR PM01800 to return the correct Version and Release number for DB2 11

12.2.4 Optional program requirements

This section describes which version of associated products are tolerated by DB2 11.

Connectivity

DB2 for z/OS supports DRDA as an open interface allowing access from any client.

DB2 Connect Version 10.1 Fixpack 2 or DB2 Connect Version 9.7 Fixpack 6 clients or higher are the minimum required levels for a seamless migration.

However, DB2 Connect Version 10.5 Fixpack 2 is required to support some DB2 11 for z/OS features, such as:

- Array support
- Autocommit performance improvements for procedures and cursors
- Data sharing support for global variables
- Longer client information fields

For details, see Chapter 9, “Connectivity and administration routines” on page 171.

DB2 11 acting as a client supports the following relational database products:

- DB2 Enterprise Server (ESE) for Linux, UNIX and Windows, V9.5 (575-F41) or later
- DB2 Express Edition for Linux, UNIX and Windows, V9.5 (5724-E49) or later
- Database Enterprise Developer Edition V 9.5 (5724-N76) or later
- DB2 for IBM iSeries® V6.1. (5761-SS1) or later
- DB2 Server for VSE & VM V7.3 (5697-F42) or later
- Any other DRDA compliant client or relational DBMS server

Development tools

The following products improve the productivity of database designers, administrators and application developers that are working with DB2 11:

- InfoSphere Optim™ pureQuery™ Runtime for z/OS V3.3 (5655-W92)
- InfoSphere Optim Configuration Manager for DB2 for z/OS V3.1 (5655-AA3)
- IBM Data Studio V4.1

Programming languages

The minimum levels for programming languages are:

- Enterprise COBOL for z/OS V3.4 (5655-G53) or later
- VS Fortran 2.6 (5668-806, 5688-087, 5668-805). New data type and function are not supported since DB2 9.
- Enterprise PL/I for z/OS V3.9 (5655-H31)

IBM DB2 Accessories Suite for z/OS

IBM DB2 Accessories Suite for z/OS, V3.1 (5697-Q04) is a no-charge offering consisting of three features, each bundling components designed to enhance your use of DB2 for z/OS, including the addition of and changes to the following components:

- The DB2 Accessories Suite V11 feature provides spatial functions supporting DB2 11 for z/OS.
- The JSON capability bundles necessary components that enable DB2 10 for z/OS to be used as a JSON document store.
- An update to Data Studio 4.1 delivers health monitoring, single query tuning, and application development tools for DB2 for z/OS.

12.3 DB2 11 installation changes and considerations

As DB2 evolves in its overall improved functionality, there are also several changes which apply to the installation of a DB2 11 subsystem. This section describes the following DB2 11 installation changes and considerations:

- More support of naming standards in install and IVP jobs
- No more EDM calculations
- Modified installation jobs
- New installation job DSNTIJCB
- Miscellaneous

12.3.1 More support of naming standards in install and IVP jobs

When you enter the installation panel through **`CLIST DSNTINST`**, the first panel that you see is **`DSNTIDA1`**. This panel was changed in DB2 10, allowing you to use different prefixes for the SMP/E target library names and for everything else. This enhancement avoids the additional editing on subsequent panels such as panel DSNTIPT, where you can specify all the output library names that you would like to use.

Starting with DB2 11, a new panel DSNTIPG has been added to the install dialog. Figure 12-4 shows the new install panel DSNTIPG. The information that you can provide here gives great flexibility to change user IDs and library name prefixes for the installation and IVP job which are generated through the dialog.

```
DSNTIPG             INSTALL DB2 - INSTALLATION PREFERENCES
 ===>

 Enter authorization IDs for installing DB2-supplied routines:
  1 ROUTINES CREATOR  ===> SYSADM      Authid to create and bind DB2 routines
  2 SEC DEF CREATOR   ===> SYSADM      Authid for routines w/ SECURITY DEFINER

 Enter authorization IDs for other installation and IVP jobs:
  3 INSTALL SQL ID    ===> SYSADM      To process SQL in install jobs
  4 INSTALL PKG OWNER ===>             To own packages bound by install jobs
  5 INSTALL GRANTEE(S)===> PUBLIC    > To be granted access on objects created
                                       by install jobs

 Enter the prefix for data sets created by installation and IVP jobs:
  6 INSTALL IC PREFIX ===> DSN1110           For COPY data sets
  7 INSTALL DS PREFIX ===> DSN1110           For other data sets









  F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
  F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
```

Figure 12-4 DSNTIPG install panel

The following options are available in the new panel:

- 1 ROUTINES CREATOR

  The ROUTINES CREATOR field specifies the CURRENT SQLID setting that is to be used when creating, configuring, and validating most DB2-supplied routines. This field also specifies the default OWNER that is to be used when binding packages for these routines.

  Acceptable values are 1 to 8 characters, the first of which must be an alphabetic character.

  The value that you enter in the ROUTINES CREATOR field is assigned by the installation CLIST as the setting of the AUTHID parameter for installation programs **DSNTRIN** in job **DSNTIJRT** and **DSNTRVFY** in **DSNTIJRV** job.

  The **AUTHID** parameter is used by the **DSNTRIN** program as the **CURRENT SQLID** setting when creating and configuring DB2-supplied routines. The **DSNTRIN** program also uses the **AUTHID** parameter as the default **OWNER** when binding packages for the DB2-supplied routines.

  The **AUTHID** parameter is used by the **DSNTRVFY** program as the CURRENT SQLID setting when validating DB2-supplied routines. The **DSNTRVFY** program also uses the **AUTHID** parameter as the default OWNER when binding packages for validation of these routines.

Note: Specify an authorization ID that has installation system administrator authority. Routines that are created or configured with installation system administrator authority are marked as system-defined.

System-defined routines can be executed by the system DBADM and SQLADM authorities, which fits well with popular SQL tuning tools. For example, IBM Optim Query Tuner requires many of the DB2-supplied stored procedures to be available and accessible by an authority that focuses on SQL tuning activities.

- 2 SEC DEF CREATOR

  The value of the SEC DEF CREATOR field specifies the CURRENT SQLID setting that is to be used when creating and configuring DB2-supplied routines that are defined with the **SECURITY DEFINER** option.

  Acceptable values are 1 to 8 characters, the first of which must be an alphabetic character.

  The value that you enter in the SEC DEF CREATOR field is assigned by the installation CLIST as the setting of the **SECDEFID** parameter for installation program **DSNTRIN** in job **DSNTIJRT**.

  The **SECDEFID** parameter is used by the **DSNTRIN** program as the **CURRENT SQLID** setting when creating and configuring DB2-supplied routines that have the **SECURITY DEFINER** attribute. The default owner of the packages that are bound for these routines is the ID that is specified in the ROUTINES CREATOR field.

  Note: Specify an ID that can be used as a logon ID because it is used by WLM to execute routines that have the **SECURITY DEFINER** attribute.

- 3 INSTALL SQL ID

  The INSTALL SQL ID field specifies the CURRENT SQLID setting that is to be used when SQL is processed by most DB2 installation and sample jobs.

  This field does not apply to the **DSNTIJRT** and **DSNTIJRV** jobs. For these jobs, you can use the **ROUTINES CREATOR** and **SEC DEF CREATOR** fields, which are also on the DSNTIPG panel, to specify the **CURRENT SQLID**.

- 4 INSTALL PACKAGE OWNER

  The INSTALL PKG OWNER field specifies the authorization ID to assign as the owner of packages and plans that are bound by most installation and sample jobs.

  This field does not apply to the **DSNTIJRT** and **DSNTIJRV** jobs. Use installation panels **DSNTIPRA - DSNTIPRP** to specify package owners for stored procedures that are provided by DB2.

- 5 INSTALL GRANTEE(S)

  The INSTALL GRANTEE(S) field specifies the authorization IDs that are to be granted access to objects that are created and bound by most installation and sample jobs.

  This field does not apply to the **DSNTIJRT** and **DSNTIJRV** jobs. Use installation panels **DSNTIPRA - DSNTIPRP** to specify authorization IDs for routines that are provided by DB2.

  Use commas to separate individual IDs. Do not use embedded blanks. You can enter up to 44 characters, including commas.

  To be able to enter more than one ID, type **EXPAND** in the command line, place the cursor on the input field, and hit enter. You then get a screen that allows you to enter various IDs up to a length of 44 bytes in total.

Alternatively, you can also assign **EXPAND** to one of your function keys, place the cursor into the input field, and press Enter to open the **ISPEXPND** screen shown in Figure 12-5.

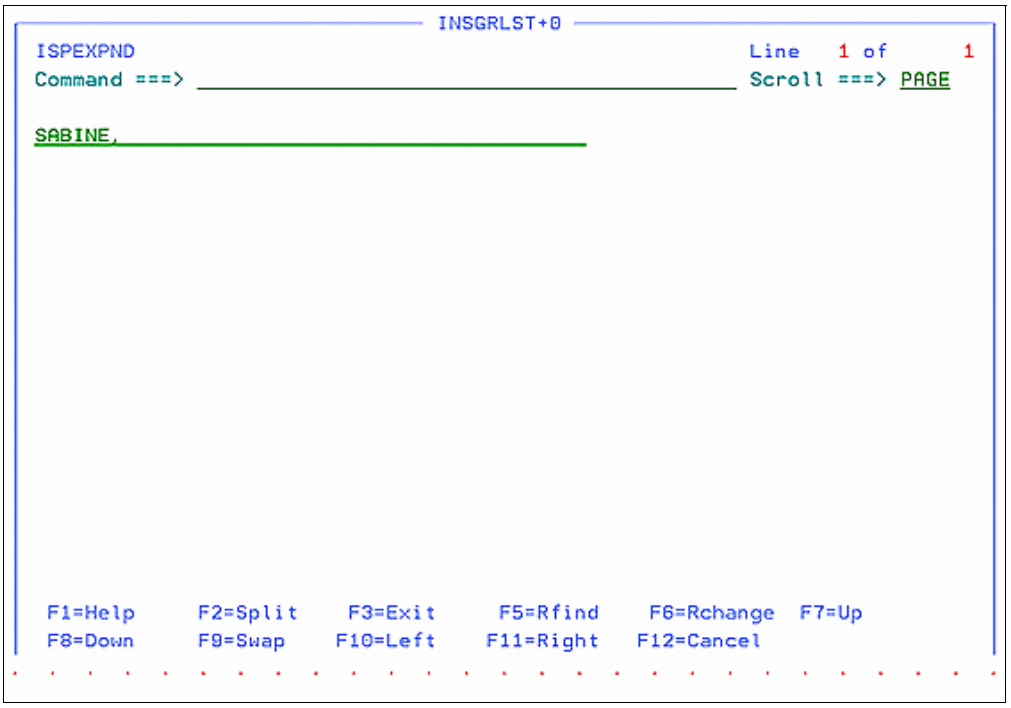


Figure 12-5 EXPAND screen for panel DSNTIPG

The authorization IDs that you enter in this field are granted the following privileges:

- The **USE** privilege for **STOGROUP**s and table spaces that are created by IVP jobs
- The **USE** privilege for buffer pool BP0, the **SYSDEFLT** storage group, and table space **DSNDB04.SYSDEFLT**
- The **DBADM** privilege for databases that are created by IVP jobs
- The **CREATETAB** and **CREATETS** privileges for the temporary database, DSNDB04
- The **DELETE**, **INSERT**, **SELECT**, and **UPDATE** privileges for tables and created global temporary tables that are created by IVP and installation jobs other than DSNTIJRT
- The **EXECUTE** privilege for packages and plans that are bound by IVP and installation jobs other than **DSNTIJRT**
- The **BIND** privilege on most plans that are bound by IVP jobs

- 6 **INSTALL IC PREFIX** field

  The INSTALL IC PREFIX field specifies the prefix for image copy data sets that are created by DB2 installation and IVP jobs.

  This is especially beneficial because the image copy prefix almost always had to be changed in the past in jobs, for example **DSNTEJ0**.

  The value that you can specify here are 1 to 17 characters that form a valid z/OS data set name prefix.

- 7 INSTALL DS PREFIX field

The INSTALL DS PREFIX field specifies the prefix for most data sets that are created by most DB2 installation and IVP jobs. This field does not apply to data sets that are created by job

DSNTIJIN, which applies for example to **TEMPLATE** or **LISTDEFs** that are created as part of the IVP jobs.

Again, the value that you can specify here are 1 to 17 characters that form a valid z/OS data set name prefix.

12.3.2 No more EDM calculations

Traditionally the installation CLIST used linear calculations based on the estimate number of databases, plan, and so on to determine the settings of the various EDM pools. These calculations have changed with the changes in virtual storage use by DB2 and have now been replaced by stepped sizing according to your site's size. These are the five defined stepped sizes:

Small site: About 100 plans, 50 application databases, and 1000 tables
Small-Medium site: About 200 plans, 200 application databases, and 4000 tables
Medium site: About 400 plans, 400 application databases, and 8000 tables
Medium-Large site: About 600 plans, 600 application databases and 12,000 tables
Large site: About 800 plans, 800 application databases, and 16,000 tables

These settings are starting points. You have to check your actual requirements.

Based on the numbers indicated on the installation CLIST panels, Table 12-3 lists pool sizes.

Table 12-3 EDM Pool stepped sizings

System size/ parameter name	EDMDBDC (KB)	EDMSTMTC (KB)	EDM_SKELETON_ POOL (KB)
Small	40960	122880	81920
Small-Medium	102400	307200	204800
Medium	204800	614400	409600
Medium-Large	409600	1228800	819200
Large	819200	2457600	1638400

When you get to the **DSNTIPC** panel, shown in Figure 12-6, you can override the calculated stepped sizes. This example decreases the storage sizes that installation CLIST has picked based on the values entered on previous installation panels.

If you compare the values in Table 12-3 with the values that were assigned to the 3 EDM pools, you can see that they match with the ones for a small site. This, in fact is what you would get if you accept the default values coming from input member DSNTIDXA.

```
DSNTIPC         INSTALL DB2 - CLIST CALCULATIONS - PANEL 1
===>
 You can update the DSMAX, EDMPOOL STATEMENT CACHE (if CACHE DYNAMIC is YES),
 EDM DBD CACHE, SORT POOL, and RID POOL sizes if necessary.
                                      Calculated   Override
 1  DSMAX - MAXIMUM OPEN DATA SETS    =    20000                (1-200000)
 2  DSNT485I  EDM STATEMENT CACHE     =   122880  K            K
 3  DSNT485I  EDM DBD CACHE           =    40960  K            K
 4  DSNT485I  EDM SKELETON POOL SIZE  =    81920  K            K
 5  DSNT485I  EDM LIMIT BELOW THE BAR =        0  K            K
 6  DSNT485I  BUFFER POOL SIZE        =      109  M
 7  DSNT485I  SORT POOL SIZE          =    10000  K            K
 8  DSNT485I  MAX IN-MEMORY SORT SIZE =     1000  K            K
 9  DSNT485I  RID POOL SIZE           =   400000  K            K
10  DSNT485I  DATA SET STORAGE SIZE   =    26000  K
11  DSNT485I  CODE STORAGE SIZE       =    38200  K
12  DSNT485I  WORKING STORAGE SIZE    =    45024  K
13  DSNT486I  TOTAL MAIN STORAGE      =      617  M            M
14  DSNT487I  TOTAL STORAGE BELOW 16M =     1036  K WITH SWA ABOVE 16M LINE
15  DSNT438I  IRLM LOCK MAXIMUM SPACE =     2160  M, AVAILABLE =   2160 M

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

Figure 12-6 Install Panel DSNTIPC

If you are installing DB2 for the first time and not using SAP, use the supplied defaults input member, DSNTIDXA. If you are using SAP, you should specify DSNTIDXB, the SAP-specific input member. If you process the panels several times within a single run of the CLIST, all the previous values that are entered, except edited output data sets, remain the same.

If you use the SAP input member, the calculated storage sizes are quite different as shown in Figure 12-7.

```
DSNTIPC         INSTALL DB2 - CLIST CALCULATIONS - PANEL 1
 ===>

  You can update the DSMAX, EDM STATEMENT CACHE (if CACHE DYNAMIC is YES),
  EDM DBD CACHE, EDM SKELETON POOL, SORT POOL, and RID POOL sizes if necessary.
                                        Calculated   Override
    1  DSMAX - MAXIMUM OPEN DATA SETS    =     20000         20000
(1-200000)
    2  DSNT485I  EDM STATEMENT CACHE     =    300000  K             K
    3  DSNT485I  EDM DBD CACHE           =    150000  K             K
    4  DSNT485I  EDM SKELETON POOL SIZE  =     81920  K             K
    5  DSNT485I  EDM LIMIT BELOW THE BAR =         0  K             K
    6  DSNT485I  BUFFER POOL SIZE        =      2085  M
    7  DSNT485I  SORT POOL SIZE          =     10000  K      64000  K
    8  DSNT485I  MAX IN-MEMORY SORT SIZE =      1000  K             K
    9  DSNT485I  RID POOL SIZE           =    400000  K     100000  K
   10  DSNT485I  DATA SET STORAGE SIZE   =     26000  K
   11  DSNT485I  CODE STORAGE SIZE       =     38200  K
   12  DSNT485I  WORKING STORAGE SIZE    =     17404  K
   13  DSNT486I  TOTAL MAIN STORAGE      =      2566  M       2326  M
   14  DSNT487I  TOTAL STORAGE BELOW 16M =      1269  K WITH SWA ABOVE 16M LINE
   15  DSNT438I  IRLM LOCK MAXIMUM SPACE =   1292763  M, AVAILABLE =   4096  M
  F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
  F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

Figure 12-7 DSNTIPC results when using DSNTIDXB member

12.3.3 Modified installation jobs

Several installation jobs have been changed fromDB2 10 to reflect the product layout in DB2 11. This section lists the changed jobs and provides a few details about the changes:

- **DSNTIJIN**
- **DSNTIJUZ**
- **DSNTIJID, DSNTIJIE, and DSNTIJIF**
- **DSNTIJSG**
- **DSNTIJRT**
- **DSNTIJIC**

DSNTIJIN

During installation, the **DSNTIJIN** job defines VSAM and non-VSAM data sets for DB2. The following groups of changes are applied to this job:

- The **VSAM DEFINE** statement for the BSDS data sets have been modified. The changes are necessary to support for longer RBA/LRSN.
- The job steps that were used to defined the following table space have been removed:
  - **DSNDB06.SYSCOPY**

    The **SYSIBM.SYSCOPY** table now resides in the **DSNDB06.SYSTSCPY** table space.

– **DSNDB06.SYSSTR**

  This table space used to contain the following tables:

  - **SYSIBM.SYSSTRINGS**
  - **SYSIBM.SYSCHECKS**
  - **SYSIBM.SYSCHECKDEP**
  - **SYSIBM.SYSCHECKS2**

  DB2 11 defines one table space for each table. The following new table spaces are listed in the order corresponding to the tables:

  - **DSNDB06.SYSTSSRG**
  - **DSNDB06.SYSTSCHKS**
  - **DSNDB06.SYSTSCHKD**
  - **DSNDB06.SYSTSCHX**

– **DSNDB06.SYSRTSTS**

  This table space used to contain the following tables:

  - **SYSIBM.SYSTABLESPACESTATS**
  - **SYSIBM.SYSINDEXSPACESTATS**

  DB2 11 defines one table space per each of these tables. The following new table spaces are listed in the order corresponding to the tables:

  - **DSNDB06.SYSTSTSS**
  - **DSNDB06.SYSTSISS**

- New job steps have been added for new table spaces and index spaces.

  DB2 11 has 16 new table spaces and 17 new index spaces, including the table spaces and associated index spaces that listed previously.

  The additional four table spaces are:

  – **DSNDB06.SYSTSQRE** for the **SYSIBM.SYSQUERYPREDICATE** table (plus two indexes)
  – **DSNDB06.SYSTSQRS** for the **SYSIBM.SYSQUERYSEL** table (plus two indexes)
  – **DSNDB06.SYSTSIXS** for the **SYSIBM.SYSINDEXES** table (plus two indexes)
  – **DSNDB06.SYSTSSFB** for the **SYSIBM.SYSSTATFEEDBACK** table (plus three indexes)

DSNTIJUZ

The **DSNTIJUZ** job defines the DB2 data-only **DSNZPxxx** subsystem parameter module, the application defaults load module, and the data-only **DSNHMCID** load module.

As with every release, there are multiple changes to this job. Many system parameters were added, some updated and some removed. Refer to the 12.5.1, “New system parameters” on page 346, 12.5.2, “Changed defaults for existing system parameters” on page 354, and 12.5.3, “Removed system parameters” on page 355.

DSNTIJID, DSNTIJIE, and DSNTIJIF

After you define your system data sets and DB2 initialization parameters, you must initialize your system data sets executing these jobs in the shown sequence.

The **DSNTIJID** job records the active log data set names to the BSDS, formats the active log data sets, and initializes the DB2 directory table spaces and indexes. The BSDS is initialized in the basic, pre-DB2 11 format. After you complete your installation, you can optionally use the **DSNTIJCB** job to convert the BSDS to the new format that supports 10-byte RBA and LRSN fields.

The **DSNTIJIE** job initializes the DB2 catalog table spaces and indexes.

The **DSNTIJIF** job initializes the remaining DB2 catalog table spaces and indexes.

Those jobs are adjusted to the removed and new catalog and directory table spaces and indexes. See “DSNTIJIN” on page 327 for additional information about which objects these are.

DSNTIJSG

During installation, the **DSNTIJSG** job binds DB2-supplied packages, plans and creates a few objects such as the RLF database and related objects. This job added the creation of the query optimization database in step **DSNTIJQ** (**EXPLAIN** tables in schema **SYSIBM**.)

In DB2 11 is, this job also creates, the program authorization database, table space, table, and index. The **SYSIBM.DSNPROGAUTH** program authorization table is used to verify that a program is authorized to use a plan. A sample **INSERT** statement is provided, which you can uncomment, customize, and execute to populate the table. See 10.2, “Enhancements to program authorization” on page 250.

DSNTIJRT

DSNTIJRT creates all DB2-provided routines. It is good practice to run this job in CM and again after DB2 enters NFM in case any packages for DB2-supplied routines need to be refreshed. Some new routines might also be added to DB2 11 and not created until NFM.

DSNTIJIC

This job takes image copies of the DB2 11 catalog and directory.

In addition to adjusting the job to the correct table space names, the **SHRLEVEL** option has also been changed to **SHRLEVEL (CHANGE)** for all catalog and directory table spaces. It was **SHRLEVEL(REFERENCE)** in DB2 10.

12.3.4 New installation job DSNTIJCB

The **DSNTIJCB** job is an optional job to convert existing BSDSs to the extended format.

Important: Do not run this job before you are in NFM. The DB2 subsystem must be stopped in order to run this job.

Refer to the discussion in 3.1, “Extended RBA and LRSN” on page 24.

12.3.5 Miscellaneous

In addition to the changes in the installation process that discussed earlier in this chapter, there are some minor but useful things listed in this section:

- How to create a DB2 11 INSTALL member from your DB2 10 member
- BIND PLAN with RETAIN option
- Use of SYNONYMS replaced by ALIASES

How to create a DB2 11 INSTALL member from your DB2 10 member

If you already have one or more DB2 for z/OS subsystems installed on your system, and you need to create one more, it is sometimes convenient to install the new one using the configuration (system parameter) settings of an existing DB2 system. One reason might be that system tests before you actually start migrating an existing subsystem to DB2 11.

To generate customized installation jobs, you have to go through the installation CLIST. The installation CLIST needs an input member containing pre-set values for most of the system parameters. DB2 11 provides an input member with default settings to get started. However you might not want to start with the IBM provided default values, but with values as they are set for one of your subsystems in DB2 10.

One way to get the current values is to manually compare the defaults that DB2 11 provides in member **prefix.sdsnsamp(DSNTIDXA)** and override those with what you currently have available for DB2 10. This process might not be the best method for the following reasons:

- You might overlook something that can cause issues later.
- The **DSNTIDxx** member that you used during your DB2 10 installation or migration might be stale, because after the installation or migration you might have changed several system parameters not going through the official update process but through manual changes in the **DSNTIJUZ** job instead.

You can follow the procedure documented here to convert your stale DB2 10 **DSNTIDxx** member to a DB2 11 **DSNTIDxx** member and start the installation from there.

First, remember the **DSNTIJXZ** job. This job calls the **DSNTXAZP** program to update a stale **DSNTIDxx** member with the current **DSNZPARM** settings. Refer to Figure 12-8. DSNTIJXZ connects to an active DB2 and reads the active system parameter settings. The result is a new **DSNTIDyy** member, which is up-to-date. The job output on the right gives you a listing with information about how the input **DSNTIDxx** differs from the actual settings of this subsystem.

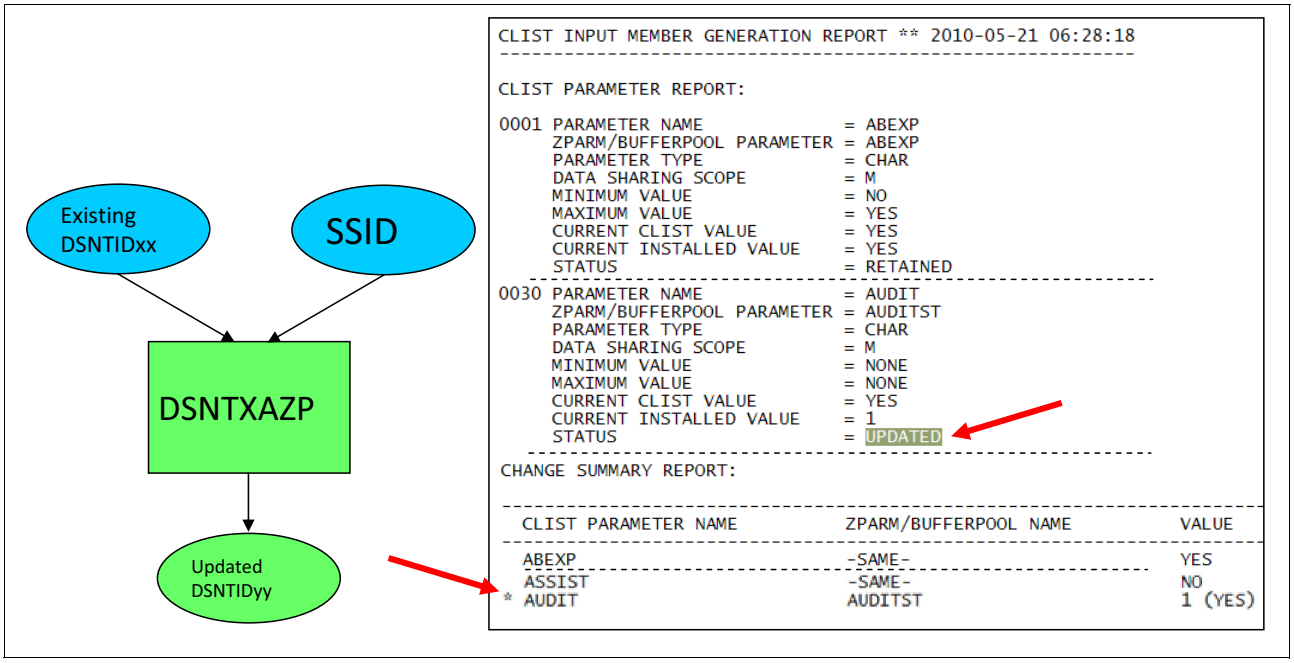


Figure 12-8 DSNTIJXC/DSNTXAZP

The following steps look at the process that involves using this **DSNTIJXC** job. Figure 12-9 assumes that the whole process starts with a stale **DSNTIDxx** and an intact, existing DB2 subsystem. Start on the top left.

1. Run **DSNTIJXZ** on DB2 10 to refresh you DB2 10 **DSNTIDxx** member. Figure 12-9 assumes the output member would be **DSNTIDyy**.

2. Run the DB2 11 install CLIST in **MIGRATE** mode and specify:
   - **DSNTIDyy** as migration input member
   - DB2 11 **DSNTIDXA** as input member
   - A member, for example **DSNTIDzz** to receive the changes
3. Discard the customized migration jobs. They are not needed for the task here.
4. Run the DB2 11 install CLIST in **INSTALL** mode and specify **DSNTIDzz** as the input member.

You are now ready to use the generated, customized install jobs for the installation of a new DB2 subsystem, which uses system parameter settings like the subsystems that you identified as a good one to get started with.

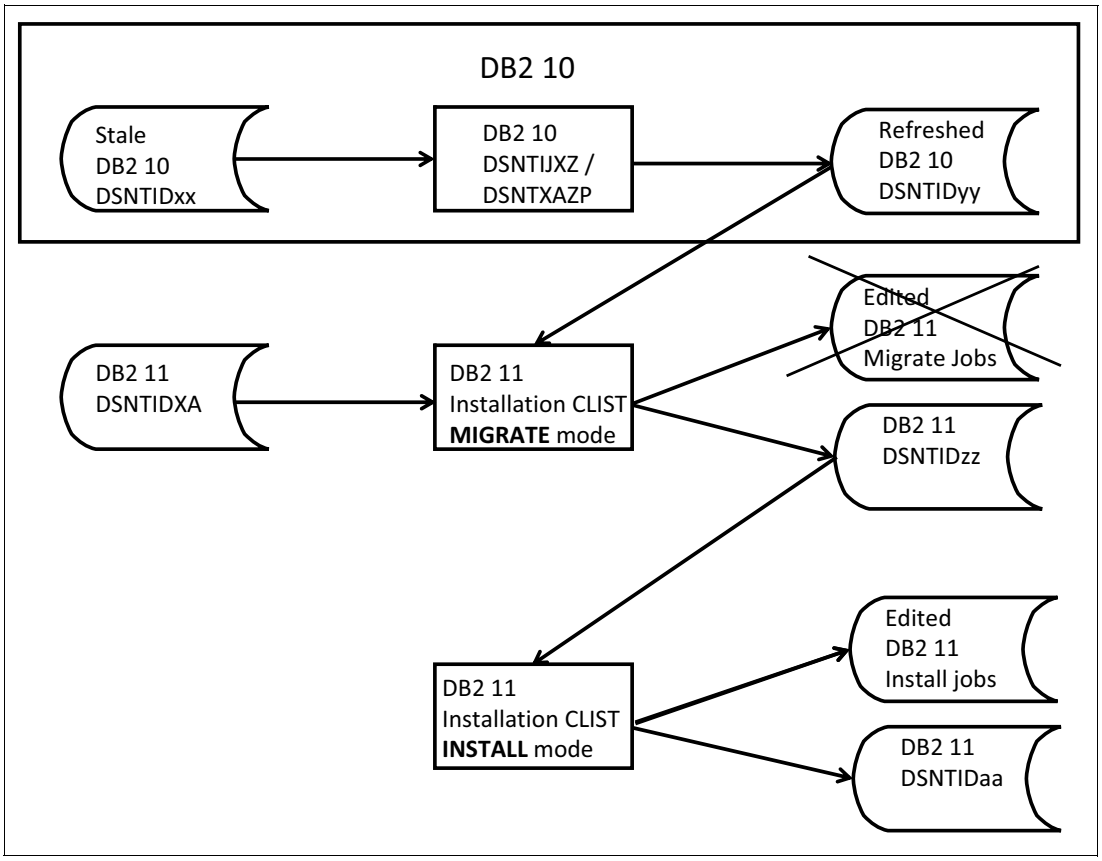


Figure 12-9 CREATE new DB2 11 DSNTIDxx input from old DB2 10 DSNTIDxx

BIND PLAN with RETAIN option

Many installation and IVP jobs bind plans that are sometimes used during the installation or verification process and can also be used for various tasks by different users later.

Up to DB2 10, the **BIND PLAN** statements did not include the **RETAIN®** option on the **ACTION** keyword. The **RETAIN** option preserves **EXECUTE** privileges when you replace a plan. If ownership of the plan changes, the new owner has to grant the privileges **BIND** and **EXECUTE** to the previous owner. **RETAIN** is not the default. If you do not specify **RETAIN**, everyone but the plan owner loses the **EXECUTE** privilege (but not the **BIND** privilege).

In DB2 11 the following Installation and verification jobs now specify **RETAIN** on their **BIND** PLAN statements:

- IVP jobs
  - **DSNTEJ1L**
  - **DSNTEJ1P**
  - **DSNTEJ1S**
  - **DSNTEJ2A**
  - **DSNTEJ2C**
  - **DSNTEJ2D**
  - **DSNTEJ2E**
  - **DSNTEJ2F**
  - **DSNTEJ2H**
  - **DSNTEJ2P**
  - **DSNTEJ3C**
  - **DSNTEJ3P**
  - **DSNTEJ4C**
  - **DSNTEJ4P**
  - **DSNTEJ5C**
  - **DSNTEJ5P**
  - **DSNTEJ6U**
  - **DSNTEJ71**
  - **DSNTEJ73**
  - **DSNTEJ75**
  - **DSNTEJ76**
  - **DSNTEJ77**
  - **DSNTEJ78**
- Installation jobs
  - **DSNTIJSG**
  - **DSNTIJTM**

Use of SYNONYMS replaced by ALIASES

In DB2 10, **SYNONYM**s are deprecated and will not be enhanced starting from DB2 10. Synonyms behave differently with DB2 for z/OS than with the other DB2 family products. Synonyms are not recommended for use when writing new SQL statements or when creating portable applications. Use aliases instead.

Synonyms are no longer used in DB2 11 IVP jobs. The following jobs have changed to use **ALIASES** instead of **SYNONYM**s:

- **DSNTEJ1**

  The **DSNTEJ1** job creates all objects that are to be used by the sample verification jobs.

- **DSNTEJ1U**

  The **DSNTEJ1U** job creates a database, table space, and table with Unicode CCSID.

  DSNTEJ1U loads data into the table from a data set that contains a full range of characters in an EBCDIC Latin-1 code page, which results in a mix of single and double-byte characters in the Unicode table. It then runs DSNTEP2 to select and display the data in hex format.

- **DSNTEJ7**

  The **DSNTEJ7** job demonstrates how to create a LOB table with all the accompanying LOB table spaces, auxiliary tables, and indexes. DSNTEJ7 also demonstrates how to use the DB2 LOAD utility to load a LOB table.

12.4 Considerations for migrating to DB2 11

Migrating a DB2 subsystem means to move from one software version to another. As stated earlier in this chapter, a migration to DB2 11 is possible only if your subsystem is on DB2 10 NFM when you start the migration process. The sequence of jobs shown in Figure 12-10 gives an overview of the migration process, listing changes in the speech bubbles next to the boxes representing individual migration steps.

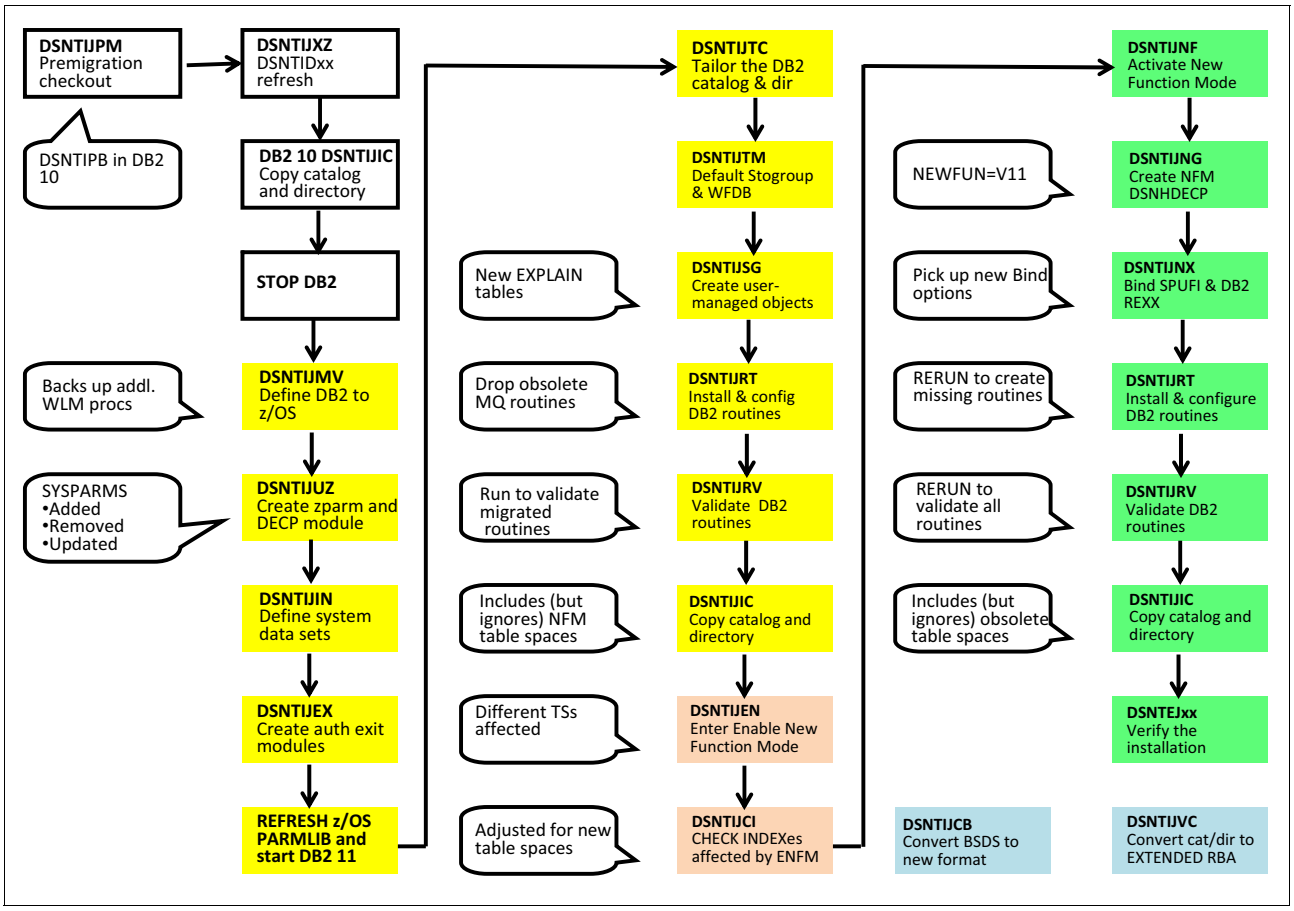


Figure 12-10 DB2 11 migration process at a glance

The next sections discuss what has changed in terms of the following functions:

- Premigration considerations
- DB2 11 CM
- DB2 11 ENFM and NFM

12.4.1 Premigration considerations

Before you actually start your DB2 subsystem in conversion mode using the DB2 11 code, plan for the new version. This planning includes completing the activities listed in this section and carefully reviewing the release incompatibilities discussed in 12.6, “Release incompatibilities” on page 357.

Items deprecated in earlier versions are now eliminated

Each DB2 release deprecates items. To *deprecate* something does not mean that the function does not exist anymore. Instead, no new development effort is spent on these items. Thus, you need to prepare for the removal of the function in a subsequent release. The following items are eliminated in DB2 11:

- Password protection for active log and archive log data sets
- **DSNH CLIST NEWFUN** values of V8 and V9
- Some DB2 supplied routines:
  - **SYSPROC.DSNAEXP**
  - AMI-based DB2 MQ functions, see APAR PK37290 for guidance
  - **DB2MQ1C.***
  - **DB2MQ2C.***
  - **DB2MQ1N.***
  - **DB2MQ2N.***
- **CHARSET** application programming default value is removed in DB2 11.

  CHARSET was a **DSNHDECP** parameter used to specify whether the character set associated with the default EBCDIC CCSID was either **ALPHANUM** or **KATAKANA**. Prior to DB2 8, this function was needed by DB2 parser for parsing in EBCDIC. Beginning in DB2 V8, parser parses statements in Unicode and no longer needs to know whether the character set is alphanumeric or Katakana.
- **BIND PACKAGE** options **ENABLE** and **DISABLE (REMOTE) REMOTE (location-name,…,<luname>,…)**

  In DB2 11, you cannot use the **BIND PACKAGE** options **ENABLE** and **DISABLE (REMOTE) REMOTE (location-name,…,<luname>,…)** to enable or disable specific remote connections. You can use the **ENABLE(REMOTE)** or **DISABLE(REMOTE)** options to enable or disable all remote connections.
- Sysplex query parallelism

  In DB2 11, sysplex query parallelism is no longer supported. Packages that used sysplex query parallelism in releases before DB2 11 use CPU parallelism in DB2 11.

  If you system was allowed to use sysplex query parallelism at all was determined by the setting of system parameter COORDNTR. Because sysplex query parallelism is eliminated from DB2 11, the system parameter is also removed from the **DSNZPARM** module.
- **DSN1CHKR**

  In DB2 11, the **DSN1CHKR** utility is no longer needed and therefore not longer supported. You can use **DSN1CHKR** in versions prior to DB2 10 NFM to scan the specified table space for broken links, broken hash chains, and records that are not part of any link or chain.

  Because DB2 10 New Function Mode (NFM), catalog and directory table spaces do not contain hashes or links. Thus, **DSN1CHKR** is unnecessary.

  The **DSN1810I** and **DSN1816I** messages are issued when the **DSN1CHKR** utility is invoked.

Fallback PTF

In the rare case of a severe error while operating under DB2 11 conversion mode, you might need to return to operation on the previous version. This process is called fallback. After fallback, the catalog remains a DB2 11 CM catalog.

DB2 10 by its nature does not support all the changes that occurred to the catalog during the **CATMAINT** utility execution that made the catalog a DB2 11 CM catalog. This means that a certain maintenance level is required on your DB2 10 code to tolerate these changes.

A fallback PTF plus prerequisite PTFs prepare the DB2 10 code to handle a DB2 11 catalog. DB2 10 must have started at least once with this fallback PTF applied. When you try to start DB2 with the DB2 11 code base for the first time, and your DB2 10 system has never been started with this PTF applied, you receive the error message shown in Example 12-1.

Example 12-1 Missing fallback PTF error message

```
DSNR045I  -DBOB DSNRRPRC DB2 SUBSYSTEM IS STARTING  883
AND
          IT WAS NOT STARTED IN A
          PREVIOUS RELEASE WITH THE FALLBACK SPE APPLIED.
          FALLBACK SPE APAR: PM31841
          NEW RELEASE LEVEL: 0000D780
          KNOWN LEVEL(S): 0000D6700000D6720000D6750000D680000
```

Important: As you can see from message DSNR45i, the V10/V11 Fallback SPE APAR is PM31841, PTF UK96357 for the fallback SPE. You need to install the PTF to prepare your subsystems for the migration to DB2 11.

Premigration checkout job DSNTIJPM

On DB2 11 target data set prefix.SDSNSAMP, that is, not on the customized NEW.SDSNSAMP, you can find job DSNTIJPM. Run this job on DB2 10 prior to your migration to DB2 11. It queries the DB2 catalog to identify conditions that you need to take into account before you attempt to of after you migrated to DB2 11.

To allow customers maximum time to prepare for migration to a new release, the DSNTIJPM job is also shipped under a different name in the previous release. For example:

- V8 DSNTIJPM is shipped as DSNTIJP8 in V7
- V9 DSNTIJPM is shipped as DSNTIJP9 in V8
- V10 DSNTIJPM is shipped as DSNTIJPA in V8 and V9
- V11 DSNTIJPM is shipped as DSNTIJPB in V10

This arrangement permits customers to begin preparing for migration in advance of buying and SMP/E-installing the new version.

The **DSNTIJPB** job is added to DB2 10 through maintenance (APAR PM94057) some time before DB2 11 is generally available (GA), which allows you to run these reports as early as possible so that you have enough time to action on the items that might be found in your catalog.

DSNTIJPB uses DB2 REXX Language Support, which is bound by running installation job **DSNTIJTM**, job step **DSNTIRX**. If you did not bind the packages and plan for DB2 REXX when you migrated to DB2 10 for z/OS, use the **DSNTIJTM** job, the **DSNTIRX** job step, to do so before running **DSNTIJPB**.

At the time this book was written, this job generates 24 reports, that include the following information:

1. Existence of the previous-release sample database. The objects in this database are needed for the IVP jobs that you are supposed to run in CM.
2. User-defined indexes on the DB2 catalog that reside in user-managed page sets. Because these indexes reside on user-managed storage, you need to modify the ENFM catalog conversion job, **DSNTIJEN**, to define shadow data sets for them for use by DB2 online **REORG**. Use the job step descriptions in the prolog of DSNTIJEN to determine the

appropriate placement of the AMS **DELETE** and **DEFINE** statements for each shadow data set you need to add.

3. User-defined indexes on the DB2 catalog that reside in DB2-controlled page sets. **DSNTIJEN** is going to handle those indexes automatically, but before running DSNTIJEN, you should review the current space allocations for the data sets for these indexes and increase the space for any that are approaching capacity to accommodate expansion during catalog conversion.
4. Plans last bound prior to DB2 9. These plans are auto-rebound when they are called for the first time in DB2 11 if your setting for system parameter ABIND is set to YES or COEXIST. If COEXIST. If **ABIND** is set to NO, DB2 V11 returns SQLCODE -908 (SQLSTATE 23510) for all attempts to use any such plan until it is explicitly rebound.
5. Plans last bound prior to DB2 9. Same as 4.
6. EXPLAIN tables, which are not in the expected DB2 10 format. Consider running the **DSNTIJXA** job for those tables.

   Attention: The **DSNTIJXA** job converts all EXPLAIN tables in the system to DB2 10. DB2 10 format explain tables are only allowed encoded in UNICODE. Thus, if the **DSNTIJXA** job expands the format of any EXPLAIN tables from say DB2 9 format to DB2 10 format, but those are still encoded in EBCDIC, they are unusable after the expansion. You must then use the **DSNTIJXB** and **DSNTIJXC** jobs to convert it from EBCDIC to UNICODE!

7. Reserved report.
8. A list of MQTs that are affected by migration job DSNTIJEN. You need to drop these MQTs before starting migration to DB2 V11 ENFM, otherwise CATENFM processing might fail. You can recreate them after you have completed running job DSNTIJEN.
9. AMI IBM MQSeries® functions that were deprecated in V8 and 9 and became obsolete in DB2 10. These are now dropped when you run DSNTIJRT, the job that creates DB2 provided routines and related objects.
10. XML MQSeries functions that were deprecated in V8 and 9 and became obsolete in DB2 10. These are now dropped when you run DSNTIJRT, the job that creates DB2 provided routines and related objects.
11. A list of simple table spaces. Simple table spaces were deprecated with DB2 9. You can still keep them in DB2 11, but you should migrate them to any other table space type, because if you accidently drop a simple table space, you are unable to recreate it as such in DB2 11. The same is true for DB2 10.
12. Trigger packages that have an invalid SECTNOI. These invalid SECTNOI were caused by a bug, which were corrected by PTF UK42129. Drop and re-create those triggers, because they might cause unpredictable results.
13. Views that contain a reference to a temporal table, such as for example:

```
CREATE VIEW VW1 AS (SELECT * FROM POLICY_INFO
                    FOR BUSINESS_TIME AS OF '2008-06-15'
                    WHERE POLICY_ID = 'A123') ;
```

    The use of such views was allowed at DB2 10 GA, but the support was discontinued with APAR PM45015 later. These views should be dropped to avoid migration errors.
14. MQTs with a period specification. See 13 for an explanation
15. SQL functions with a period specification. See 13 for an explanation.

16. Catalog table spaces with version errors. It might occur that the oldest version is larger than the current version. You must correct this problem by running MODIFY RECOVERY, followed by an REORG before you start your catalog migration.
17. A list of packages that reference catalog tables that are stored in one of the following catalog table spaces:
    - **SYSCOPY**
    - **SYSSTR**
    - **SYSRTSTS**

    As discussed earlier in this chapter, these table spaces are dropped and replaced by new table spaces in DB2 11. The first time you touch these tables in DB2 11, DB2 executes an automatic rebuild if **ABIND** is set to **YES** or **COEXIST**. If `ABIND = NO`, then SQLCODE -908 occurs.
18. The **SYSIBM.SYSCOPY** and **SYSIBM.SYSOBDS** catalog tables can contain orphaned rows due to a bug that is now fixed. These rows do not cause trouble while you are on DB2 10 but might interfere with the catalog migration. If report 18 returns a few rows, you must run the **REPAIR** utility as follows for each of the RIDs listed in this report:

    ```
    REPAIR OBJECT
      LOCATE TABLESPACE DSNDB06.SYSALTER RID X'<rid>' DELETE
    ```
19. Report orphaned rows in **SYSTABSTATS**.
20. Report orphaned rows in **SYSCOLAUTH**.
21. Report inconsistent version numbers in the DB2 catalog.
22. Report plan dependencies on table spaces processed by ENFM.
23. Report package dependencies on table spaces processed by DSNTIJTC.
24. Report plan dependencies on table spaces processed by DSNTIJTC.

12.4.2 DB2 11 CM

Conversion mode is the first mode you enter when you migrate to DB2 11. It is an important step in the migration process, because after you are in DB2 11 CM, all operation is done using the new code base. Refer to *DB2 11 for z/OS Installation and Migration*, SC19-4056 for a detailed description about all jobs and the sequence to use to execute those to migrate your DB2 subsystems to CM.

This section focuses on the changes in this process from DB2 10 for the following jobs:

- **DSNTIJMV**
- **DSNTIJUZ**
- **DSNTIJRT**
- **DSNTIJIC**

Consider using the DB2 10 **DSNTIJXZ** job to refresh your DB2 10 **DSNTIDxx** member before you use it as migration input to the DB2 11 installation CLIST. Refer to 12.3.5, “Miscellaneous” on page 329 for information about how to create a member for installing a new DB2 11 based on an existing DB2 10.

DSNTIJMV

The **DSNTIJMV** migration job is not new in DB2 11. **DSNTIJMV** first renames existing members containing started task JCL, such as **xxxxMSTR**, **xxxxDBM1**, and so on.

In a second step, it updates JCL, which can, for example, contain release dependent data set names for data sets, such as **SDSNLOAD** and so on. The JCL is generated into the procedure library that you indicated on the data set names panel of the installation CLIST.

Since DB2 10, the installation CLIST also generates the procedure JCL for the core WLM environments which are associated with the many DB2 supplied stored procedures. The library names used in these JCL members might also require changes. For this purpose, the DB2 11 CLIST now also renames the members containing the WLM procedure JCL. This task adds 11 additional **RENAME** statements to this job.

DSNTIJUZ

DSNTIJUZ creates the **DSNZPARM** and **DSNHDECP** load modules. If you compare the DB2 10 version of this job with DB2 11, you find that there are many new and removed system parameters. System parameters are discussed in 12.5.1, “New system parameters” on page 346, 12.5.2, “Changed defaults for existing system parameters” on page 354, and 12.5.3, “Removed system parameters” on page 355.

DSNTIJRT

Execute this job twice when you are migrating from DB2 10 to DB2 11. The first time is when you are in conversion mode and the second time when you are in NFM.

DSNTIJRT creates stored procedure **ADMIN_COMMAND_MVS** in DB2 11 CM (if it was not already created before migrating from DB2 10). The DB2-supplied stored procedure is described at 9.7, “ADMIN_COMMAND_MVS stored procedure” on page 217.

Conversion mode

Changes that occur when you run this job in CM are:

1. Drop all existing obsolete AMI-based DB2 MQ functions and all DB2 XML MQ routines. You can use the following query if you would like to identify the affected routines:

```
  SELECT SCHEMA
       , NAME
       , SPECIFICNAME
       , ROUTINETYPE
    FROM SYSIBM.SYSROUTINES
   WHERE SCHEMA IN ( 'DB2MQ1C' , 'DB2MQ2C'
                   , 'DB2MQ1N' , 'DB2MQ2N'
                   , 'DMQXML1C', 'DMQXML2C'
                   )
ORDER BY SCHEMA, SPECIFICNAME;
```

   These routines are also identified in one report after running job DSNTIJPM/B as described in “Premigration checkout job DSNTIJPM” on page 335.

2. Bind all packages from DB2 11 DBRMs.

NFM

You must run this job a second time in NFM. This time, DB2 creates:

- Associated created global temporary table SYSIBM.MVS_CMD_OUTPUT

Note: DSNTRIN is a program called by job DSNTIJRT to install and configure DB2-supplied routines. This includes validation and adjustment of various SQL objects that are used by the routines and that have been modified in the service stream or on a product version/release boundary. Current APARs for this program are PM45652 and PM93782.

DSNTIJIC

DSNTIJTC is a sample job that copies DB2 subsystem's catalog and directory. Starting with DB2 11 CM, this job has seven additional table spaces included in the list of objects to copy.

These following table spaces in ENFM replace three DB2 10 table spaces as follows:

- **DSNDB06.SYSTSCPY** replaces **DSNDB06.SYSCOPY**
- **DSNDB06.SYSTSCKS**, **DSNDB06.SYSTSCHX**, **DSNDB06.SYSTSCKD**, and **DSNDB06.SYSTSSRG** replace **DSNDB06.SYSSTR**
- **DSNDB06.SYSTSISS** and **DSNDB06.SYSTSTSS** replace **DSNDB06.SYSRTSTS**

These new table spaces are only created in ENFM, but **DSNTIJIC** is changed starting in CM, so the following message indicates that there is special handling in place for those objects by DB2:

```
DSNU153OI CSECT-NAME - OBSOLETE OR NFM CATALOG OR DIRECTORY OBJECT object-type
DSNDBOn.object-name WILL NOT BE PROCESSED
```

The same message is issued if you run **DSNTIJIC** in NFM and the three replaced table spaces have been removed from the catalog.

Tip: DSNTIJIC is not changed in NFM. You can remove the job steps to remove those three COPY steps from the job manually.

In case you do not use **DSNTIJIC** to copy your catalog and directory, make sure to adjust your own copy job accordingly. You might want to refer to **DSNTIJIC** if you are not sure about the order by which copy the catalog and directory objects.

A second change in **DSNTIJIC** is that the copies are produced using **SHRLEVEL CHANGE** instead of **SHRLEVEL REFERENCE**. You might want to consider changing your own image copy jobs accordingly.

Application compatibility

A new DB2 version typically comes with several changes or enhancements to SQL. These changes are listed in 12.6.1, "Application and SQL release incompatibilities" on page 357.

In the past releases all applications affected by those incompatibilities had to be adjusted to the new behavior of DB2 prior to the migration.

DB2 11 introduces support of SQL application compatibility using a system parameter, special register and bind option. This function is described in 12.7.1, "Example of DB2 10 application compatibility" on page 373.

12.4.3 DB2 11 ENFM and NFM

After you have successfully tested DB2 11 in CM for a reasonable time, you continue the migration to NFM. If you complete the few panels of the installation CLIST in ENFM mode, the CLIST generates the necessary installation and IVP on the library that you specified. Example 12-2 shows the list of generated jobs.

Example 12-2 Install and IVP jobs generated as result of ENFM installation CLIST completion

```
DSNT478I BEGINNING EDITED DATA SET OUTPUT
 DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTIJEN)', ENFM PROCESSING
 DSNT489ICLISTEDITING'DBOBM.NEW.ENFM.SDSNSAMP(DSNTIJNH)',HALTENFMPROCESSING
```

```
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTIJNF)', TURN NEW FUNCTION
MODE ON
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTIJNX)', CREATE NFM-DEPENDENT
OBJECTS
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTIJES)', DISABLE USE OF NEW
FUNCTION (ENFM*)
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTIJCS)', RETURN FROM ENFM OR
ENFM* TO CM*
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTIJCI)', CHECK INDEXES AFTER
ENFM
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTIJCV)', CONVERT CATALOG AND
DIRECTORY FORMAT
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTESC)',  SAMPLE DATA
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTESH)',  SAMPLE DATA
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTESD)',  SAMPLE DATA
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTESA)',  SAMPLE DATA
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTESE)',  SAMPLE DATA
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTEJ0)',  SAMPLE JCL
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTEJ1)',  SAMPLE JCL
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTEJ1L)', SAMPLE JCL
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTEJ1P)', SAMPLE JCL
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTEJ1S)', SAMPLE JCL
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTEJ1T)', SAMPLE JCL
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTEJ1U)', SAMPLE JCL
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTEJ2A)', SAMPLE JCL
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTEJ2C)', SAMPLE JCL
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTEJ2D)', SAMPLE JCL
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTEJ2E)', SAMPLE JCL
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTEJ2F)', SAMPLE JCL
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTEJ2H)', SAMPLE JCL
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTEJ2P)', SAMPLE JCL
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTEJ2U)', SAMPLE JCL
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTEJ3C)', SAMPLE JCL
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTEJ3P)', SAMPLE JCL
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTEJ3M)', SAMPLE JCL
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTEJ4C)', SAMPLE JCL
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTEJ4P)', SAMPLE JCL
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTEJ5A)', SAMPLE JCL
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTEJ5C)', SAMPLE JCL
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTEJ5P)', SAMPLE JCL
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTEJ6O)', SAMPLE JCL
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTEJ6R)', SAMPLE JCL
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTEJ6U)', SAMPLE JCL
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTEJ6V)', SAMPLE JCL
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTEJ6W)', SAMPLE JCL
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTEJ6Z)', SAMPLE JCL
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSN8ES1)', SAMPLE SQL PROCEDURE
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSN8ES2)', SAMPLE SQL PROCEDURE
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTEJ63)', SAMPLE JCL
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTEJ64)', SAMPLE JCL
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTEJ65)', SAMPLE JCL
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTEJ66)', SAMPLE JCL
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTEJ67)', SAMPLE JCL
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTEJ7)', SAMPLE JCL
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTEJ71)', SAMPLE JCL
```

```
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTEJ73)', SAMPLE JCL
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTEJ75)', SAMPLE JCL
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTEJ76)', SAMPLE JCL
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTEJ77)', SAMPLE JCL
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTEJ78)', SAMPLE JCL
DSNT489I CLIST EDITING 'DBOBM.NEW.ENFM.SDSNSAMP(DSNTIJNG)', UPDATE DSNHDECP FOR
ENFM
```

The number of installation jobs, that is the ones that start with DSNTI, are just a handful. You most likely only run few of them because the majority deals with halting processes or going backwards rather than forward.

Refer to Figure 12-10 on page 333 for an overview of the job steps described here:

- **DSNTIJEN**
- **DSNTIJNX**
- **DSNTIJRT**

DSNTIJEN

DSNTIJEN prepares the DB2 catalog and directory for NFM. This job has been around since DB2 8. However, each version requires different changes to the DB2 catalog and therefore contains different steps and its execution time varies accordingly. After you have first started **DSNTIJEN**, the status of your DB2 11 subsystem changes from CM to enabling-NFM (ENFM).

After **DSNTIJEN** has completed, all DB2 11 catalog changes are completed as well. Table 12-4 show the progression of the number of objects of the DB2 catalog and directory across the DB2 versions.

Table 12-4 Number of catalog and directory objects

Version	Table Spaces	Tables	Indexes	Columns	LOB columns	Inline LOB columns
V1	11	25	27	269	0	0
V3	11	43	44	584	0	0
V5	12	54	62	731	0	0
V7	20	84	118	1212	2	0
V8	22	85	132	1265	2	0
V9	28	104	165	1652	6	0
V10	95	134	233	2036	36	4
V11	108	143	250	2202	42	9

The following **SELECT** statement lists the names of the catalog columns defined as inline LOBs and their table names, as shown in Table 12-5:

```
SELECT NAME,TBNAME,TBCREATOR,LENGTH FROM SYSIBM.SYSCOLUMNS WHERE TBCREATOR LIKE
'SYS%' AND COLTYPE IN ('BLOB','CLOB') AND LENGTH > 4;
```

Table 12-5 Tables having inline LOB columns

COLUMN NAME	TBNAME	TBCREATOR	INLINE LENGTH
SPT_DATA	SPTR	SYSIBM	32146

COLUMN NAME	TBNAME	TBCREATOR	INLINE LENGTH
DESCRIPTOR	SYSCONTROLS	SYSIBM	12004
RULETEXT	SYSCONTROLS	SYSIBM	16004
STATEMENT	SYSPACKSTMT	SYSIBM	15364
STMTBLOB	SYSPACKSTMT	SYSIBM	7172
STMTTEXT	SYSQUERY	SYSIBM	2052
DEFAULTTEXT	SYSVARIABLE	SYSIBM	2004
DESCRIPTOR	SYSVARIABLES	SYSIBM	2004
PARSETREE	SYSVIEWS	SYSIBM	27674

Tip: If you are not sure in which mode your DB2 subsystem currently runs, you can use DB2 command **-DIS GROUP** or **-DIS GROUP DETAIL**. The command works in data sharing and non-data sharing.

The output of **-DIS GROUP** to check the mode is shown in Example 12-3.

Example 12-3 -DIS GROUP result from non-data-sharing subsystem

```
DSN7100I  -DB0B DSN7GCMD
*** BEGIN DISPLAY OF GROUP(........) CATALOG LEVEL(111) MODE(CM  )
                  PROTOCOL LEVEL(2)  GROUP ATTACH NAME(....)
-------------------------------------------------------------------
DB2                                     DB2 SYSTEM    IRLM
MEMBER   ID  SUBSYS CMDPREF   STATUS    LVL NAME      SUBSYS IRLMPROC
-------- --- ----   --------  -------- --- --------  ----   --------
........   0 DB0B   -DB0B     ACTIVE    111 SC63      IDOB   DB0BIRLM
-------------------------------------------------------------------
SPT01 INLINE LENGTH:        32138
*** END DISPLAY OF GROUP(........)
DSN9022I  -DB0B DSN7GCMD 'DISPLAY GROUP ' NORMAL COMPLETION
***
```

In general, job DSNTIJEN performs the following functions:

- Saves the current RBA or LRSN in the BSDS.
- Converts SYSCOPY to a new table space, SYSTSCPY.
- Converts SYSRTSTS to two new table spaces, SYSTSISS and SYSTSTSS.
- Converts SYSSTR to four new table spaces, SYSTSCKS, SYSTSCHX, SYSTSCKD, and SYSTSSRG.
- During REORG, converts all table spaces and indexes that are processed by DSNTIJEN to use the RBA and LRSN format that is specified in the DSN6SPRM.UTILITY_OBJECT_CONVERSION setting.
- Resets and re-initializes the SYSUTILX table space in step ENFM0010. Therefore, utilities should not be run during this step.
- Changes types and lengths of existing catalog columns.

DSNTIJEN consists of the following job steps:

ENFM0000	Terminates pending **DSNENFM.*** utilities.
ENFM000A	Gets a list of table spaces that are in **ICOPY** and **COPY** status.
ENFM000B	Image copies table spaces that are identified in job step ENFM000A.
ENFM0001	Updates the catalog for the new release.
ENFM0010	Enabling-NFM for **SYSUTILX**.
ENFM002x	Enabling-NFM steps for **SYSLGRNX**.
ENFM003x	Enabling-NFM steps for **SYSCOPY**.
ENFM004x	Enabling-NFM steps for **SYSRTSTS**.
ENFM005x	Enabling-NFM steps for **SYSTSIXS**.
ENFM006x	Enabling-NFM steps for **SYSTSTAB**.
ENFM007x	Enabling-NFM steps for **SYSSTR**.
ENFM9900	Terminates pending **DSNENFM.*** utilities

When the **DSNTIJEN** job ran to migrate the DB2 subsystem to DB2 11 ENFM, the error message shown in Example 12-4 was encountered.

Example 12-4 DSNU2902I error message

```
DSNU2902I -DBOB 251 04:29:53.96 DSNURMAP - MAPPING DATABASE MAPDB IS INVALID
```

This is a new error message. The autonomic creation of mapping tables for REORG SHRLEVEL CHANGE is described in 11.1.4, “Automated REORG mapping table management” on page 278.

The message text clearly describes the problem as shown in Example 12-5. This example uses the **MAPDB** in **REORG_MAPPING_DATABASE** system parameter but have not created this database in the subsystem. To correct the problem, which caused **DSNTIJEN** to end with RC 8, the database MAPDB needs to be created.

Example 12-5 DSNU2902I message text

```
DSNU2902I
csect-name MAPPING DATABASE database-name IS INVALID

Explanation
The REORG utility statement has detected that the database that is specified for
the REORG TABLESPACE utility MAPPINGDATABASE keyword or the REORG_MAPPING_DATABASE
subsystem parameter does not exist or cannot be used to implicitly create a
mapping table.

csect-name
The name of the control section that issued the message.
database-name
The name of the database.
System action
Utility processing terminates.

Administrator response
Specify a valid database name for the REORG TABLESPACE utility MAPPINGDATABASE
keyword or specify a valid value for the REORG_MAPPING_DATABASE subsystem
parameter.
Severity
8 (error)
```

Because **DSNTIJEN** failed, not all table spaces changed during ENFM catalog processing have been adjusted correctly. Example 12-6 shows the results of **-DIS GROUP DETAIL** DB2 command. Only **SYSUTILX** has been completed at this point. All other table spaces listed here still need to be processed.

Note that the **MODE** this is displayed in the second line of the command output is already set to EN, because the example is now at a point of no return.

Restriction: When you entered mode EN and left CM, you cannot fallback to DB2 10 NFM anymore!

Example 12-6 -DISPLAY GROUP DETAIL output

```
DSN7100I  -DBOB DSN7GCMD
 *** BEGIN DISPLAY OF GROUP(........) CATALOG LEVEL(111) MODE(EN  )
                   PROTOCOL LEVEL(2)  GROUP ATTACH NAME(....)
 ------------------------------------------------------------------------
 DB2                                      DB2 SYSTEM    IRLM
 MEMBER   ID  SUBSYS CMDPREF   STATUS   LVL NAME      SUBSYS IRLMPROC
 -------- --- ----   --------  -------- --- --------  ----   --------
 ........   0 DBOB   -DBOB     ACTIVE   111 SC63      IDOB   DBOBIRLM
 ------------------------------------------------------------------------
 TABLE    ENABLED
 SPACE    NEW FUNCTION
 -------- ------------
 SYSUTILX        YES
 SYSLGRNX         NO
 SYSCOPY          NO
 SYSRTSTS         NO
 SYSTSIXS         NO
 SYSTSTAB         NO
 SYSSTR           NO
 ------------------------------------------------------------------------
 SPT01 INLINE LENGTH:        32138
 *** END DISPLAY OF GROUP(........)
 DSN9022I  -DBOB DSN7GCMD 'DISPLAY GROUP ' NORMAL COMPLETION
 ***
```

After creation of database MAPDB, the **DSNTIJEN** job is run. After completion of **DSNTIJEN**, the **-DISPLAY GROUP DETAIL** command output in Example 12-7shows that all page sets, which are touched during **CATENFM** execution are now ready for NFM.

Example 12-7 -DIS GROUP DETAIL output in ENFM

```
DSN7100I  -DBOB DSN7GCMD
 *** BEGIN DISPLAY OF GROUP(........) CATALOG LEVEL(111) MODE(EN  )
                   PROTOCOL LEVEL(2)  GROUP ATTACH NAME(....)
 ------------------------------------------------------------------------
 DB2                                      DB2 SYSTEM    IRLM
 MEMBER   ID  SUBSYS CMDPREF   STATUS   LVL NAME      SUBSYS IRLMPROC
 -------- --- ----   --------  -------- --- --------  ----   --------
 ........   0 DBOB   -DBOB     ACTIVE   111 SC63      IDOB   DBOBIRLM
 ------------------------------------------------------------------------
 TABLE    ENABLED
 SPACE    NEW FUNCTION
 -------- ------------
```

```
SYSUTILX        YES
SYSLGRNX        YES
SYSCOPY         YES
SYSRTSTS        YES
SYSTSIXS        YES
SYSTSTAB        YES
SYSSTR          YES
---------------------------------------------------------------------
SPT01 INLINE LENGTH:        32138
*** END DISPLAY OF GROUP(........)
DSN9022I  -DBOB DSN7GCMD 'DISPLAY GROUP ' NORMAL COMPLETION
***
```

Tip: If **DSNTIJEN** fails, fix the reason for the failure and rerun the job without modifying it at all. Just re-submit it. Steps that were already completed previously are figured out and the job continues with the correct step.

The subsystem in this example is a small sandbox system. The **DSNTIJEN** job completed in 0.78 minutes elapsed time. Experiences from the early support program show that the elapsed time for the ENFM-NFM migration of other subsystems was also short, that is approximately in the 2 minute interval.

However, the majority of the time is spent reorganizing the following table spaces:

- **DSNDB01.SYSLGRNX**
- **DSNDB06.SYSCOPY**
- **DSNDB06.SYSRTSTS**
- **DSNDB06.SYSTSIXS**
- **DSNDB06.SYSTSTAB**
- **DSNDB06.SYSSTR**

So if you want to determine what the expected elapsed time for your subsystems might be, you can reorg those table spaces and add up the elapsed time.

Tip: In order to reduce the elapsed time for CATENFM, make sure that you use MODIFY RECOVERY to clean up obsolete SYSLGRNX and SYSCOPY entries.

DSNTIJNX

The **DSNTIJNX** job creates installation objects that are dependent on NFM. **DSNTIJNX** creates a new **DSNRLMTxx** table (used by the Resource Limit Facility) in the new format plus statements for altering an existing **DSNRLMTxx** table to the new format after DB2 enters NFM.

DSNTIJNX also rebinds both SPUFI packages and REXX Language Support packages with the bind options supported in DB2 11 NFM:

- **ARCHIVESENSITIVE**

  Determines whether references to archive enabled tables in static SQL statements and dynamic SQL statements are affected by the value of a new **SYSIBMADM.GET_ARCHIVE** global variable. This global variable indicates whether a reference to an archive enabled table in a table-reference should include rows in the associated archive table.

  Refer to 6.2, “Global variables” on page 102 for an explanation about global variables.

  Archive enabled tables are discussed in 7.2, “Transparent archiving of temporal data” on page 130.

- SYSTIMESENSITIVE

  Indicates whether references to system-period temporal tables in static and dynamic SQL statements are affected by the value of the CURRENT TEMPORAL SYSTEM_TIME special register. When a system-periods temporal table is referenced and the value in effect for the **CURRENT TEMPORAL SYSTEM_TIME** special register is not the null value, the following period specification in implicit: FOR SYSTEM_TI;E AS OF CURRENT TEMPORAL SYSTEM_TIME.

- BUSTIMESENSITIVE

  Indicates whether references to system-period temporal tables in static and dynamic SQL statements are affected by the value of the CURRENT TEMPORAL BUSINESS_TIME special register. When a system-period temporal table is referenced and the value in effect for the CURRENT TEMPORAL BUSINESS_TIME special register is not the null value, the following period specification is implicit: FOR SYSTEM_TIME AS OF CURRENT TEMPORAL BUSINESS_TIME.

DSNTIJRT

If you are migrating from DB2 10, you should run this job two times. Refer to “DSNTIJRT” on page 338, which describes the steps necessary to get into DB2 11 conversion mode.

When you execute the job, it creates the new stored procedure **SYSPROC.ADMIN_COMMAND_MVS** (if not already created in DB2 10) and its created global temporary table **SYSIBM.MVS_CMD_OUTPUT**.

12.5 Subsystem parameters

New DB2 releases typically have new and removed system parameters (**DSNZPARMs**) as well as changes in their default values. This section describes the following information:

- New system parameters
- Changed defaults for existing system parameters
- Removed system parameters
- Deprecated system parameters

12.5.1 New system parameters

This section describes new system parameters introduced with DB2 11 and gives a brief explanation about their functionality:

- APPLCOMPAT
- AUTHEXIT_CACHEREFRESH
- AUTHEXIT_CHECK
- INDEX_CLEANUP_THREADS
- LIKE_BLANK_INSIGNIFICANT
- MAXSORT_IN_MEMORY
- OBJECT_CREATE_FORMAT
- PARAMDEG_DPSI
- PARAMDEG_UTIL
- PCTFREE_UPD
- PKGREL_COMMIT
- PREVENT_ALTERTB_LIMITKEY
- REORG_DROP_PBG_PARTS
- REORG_MAPPING_DATABASE

- STATFDBK_SCOPE
- TEMPLATE TIME
- UTILITY_OBJECT_CONVERSION
- WFSTGUSE_AGENT_THRESHOLD
- WFSTGUSE_SYSTEM_THRESHOLD

APPLCOMPAT

The APPL COMPAT LEVEL field on panel DSNTIP41 specifies the default release level value of the APPLCOMPAT BIND and REBIND option.

Acceptable values are V10R1 and V11R1. The default behavior is V10R1 after migration and V11R1 for a new installation. See “Example of DB2 10 application compatibility” on page 373.

AUTHEXIT_CACHEREFRESH

The AUTHEXIT_CACHEREFRESH system parameter specifies whether entries in the cache package authorization cache, the routine authorization cache, the DDF user authentication cache, and the dynamic statement cache are refreshed or whether the dependent packages are invalidated when access control authorization exit is active and the user profile is changed in RACF. Acceptable values are ALL and NONE, with a default of NONE.

ALL	Specifies that DB2 refreshes the cache entries of the package authorization, the routine authorization, and the dynamic statement and invalidates dependent packages when the user profile or resource access is changed in RACF. The cache entries are refreshed only when the access control authorization exit is active.
NONE	Specifies that DB2 does not refresh the cache entries of the package authorization, the routine authorization, and the dynamic statement or invalidate dependent packages when the user profile or resource access is changed in RACF. This is the default value for the field.

When the AUTHEXIT_CACHEREFRESH system parameter is set to ALL and the access control authorization exit is active, DB2 listens to type 62, type 71, and type 79 ENF signals from RACF for user profile or resource access changes and refreshes the DB2 cache entries accordingly. If you define RACF classes for DB2 objects and administrative authorities without using IBM-supplied RACF resource classes, you need to enable the SIGNAL=YES option for these classes in the RACF Class Descriptor Table.

AUTHEXIT_CHECK

The AUTHEXIT_CHECK subsystem parameter specifies whether the owner or the primary authorization ID is used for authorization checks when the access control authorization exit (DSNX@XAC) is active.

Acceptable values are PRIMARY and DB2, the default is PRIMARY.

PRIMARY	Specifies that DB2 provides the ACEE of the primary authorization ID to perform all authorization checks. The primary authorization ID must be permitted access to the resources in RACF. This is the default value for the field.
DB2	Specifies that DB2 provides the ACEE of the package owner to perform authorization checking when processing the **AUTOBIND**, **BIND**, and **REBIND** commands. DB2 provides the ACEE of the authorization ID as determined by the DYNAMICRULES option to perform dynamic SQL authorization checking. The access control authorization exit uses the ACEE for XAPLUCHK for authorization checking.

The XAPLUCHK authorization ID can be a user or a group in RACF. To ensure successful authorization checks with the owner ACEE, the owner authorization ID in XAPLUCHK must be permitted access to the resources in RACF. If the owner is a group in RACF, you need to permit the group access to the resource associated with the connection in the RACF DSNR class. You can issue the PERMIT command to grant a group access to subsystem.BATCH in the DSNR class, as follows:

PERMIT DSN.BATCH CLASS(DSNR) ID(DB2GRP) ACCESS(READ)

INDEX_CLEANUP_THREADS

INDEX_CLEANUP_THREADS specifies the maximum number of threads that can be created to process the cleanup of pseudo-deleted index entries on this subsystem or data sharing member. Pseudo-deleted entries in an index are entries that are logically deleted but still physically present in the index.

Acceptable values are in the range between 0 and 128, with a default of 10.

This parameter works in conjunction with the SYSIBM.SYSINDEXCLEANUP catalog table, which controls cleanup processing of pseudo-deleted index entries.

The default setting is appropriate for most situations. However, a larger setting might be appropriate in cases where large indexed tables or large numbers of indexed tables are constantly modified. A smaller value might be appropriate in cases where thread count is severely constrained.

When INDEX_CLEANUP_THREADS is set to 0, no cleanup is performed by the subsystem or data sharing member regardless of the entries in the SYSIBM.SYSINDEXCLEANUP catalog table.

Refer to 4.10, “Idle thread break-in” on page 82 for a discussion of index cleanup.

LIKE_BLANK_INSIGNIFICANT

The LIKE BLANK INSIGNIFICANT field specifies whether blanks are significant when applying the LIKE predicate to a string. If set, the blank insignificant behavior applies.

Acceptable values are NO and YES. NO is the default setting, which actually represents the DB2 10 behavior.

NO	LIKE treats trailing blanks within fixed-length character strings as significant.
YES	When the LIKE predicate is applied against fixed-length character column data, DB2 strips trailing blanks from the data before performing the comparison. If the data is all blank, DB2 reduces it to a single blank before performing the comparison.

Refer to Example 12-8 for a few simple examples of the result DB2 returns based on the setting for this system parameter.

Example 12-8 LIKE_BLANK_INSIGNIFICANT

```
Given:
CREATE TABLE LIKETEST (C1 CHAR(10));
INSERT INTO LIKETEST VALUES('   AA   ');
INSERT INTO LIKETEST VALUES('A  AA  A');
```

When **LIKE_BLANK_INSIGNIFICANT=NO**

```
for:
SELECT * FROM LIKETEST
WHERE C1 LIKE 'AA%'
=> No rows are returned

for:
SELECT * FROM LIKETEST
WHERE C1 LIKE '%AA'
=> No rows are returned

for:
SELECT * FROM LIKETEST
WHERE C1 LIKE '%AA%'
=> '   AA   ' and 'A  AA  A' are returned
```

When **LIKE_BLANK_INSIGNIFICANT=YES**

```
for:
SELECT * FROM LIKETEST
WHERE C1 LIKE 'AA%'
=> No rows are returned

for:
SELECT * FROM LIKETEST
WHERE C1 LIKE '%AA'
=> '   AA   ' is returned

for:
SELECT * FROM LIKETEST
WHERE C1 LIKE '%AA%'
=> '   AA   ' and 'A  AA  A' are returned
```

MAXSORT_IN_MEMORY

The MAXSORT_IN_MEMORY subsystem parameter specifies the maximum allocation of storage in kilobytes for a query that contains an ORDER BY clause, a GROUP BY clause, or both. The storage is allocated only during the processing of the query. Increasing the value in this field can improve performance of such queries but might require large amounts of real storage when several such queries run simultaneously.

Acceptable values: 1000 to the value specified in SORT POOL SIZE, whichever is larger

The default value: 1000.

This value is used for the final in-memory work file storage for the final sort. So it can use up to this value. Because the default for the SORT POOL size is 10,000 and if this value is set to 1000, then sort will still only use up to 1 MB for the final in-memory sort work file.

You would need to adjust this value to a larger number if you want more in-memory storage for the final work file. Say if you sets it to 128000 and you still have 10,000 for the SORT POOL SIZE, then sort will only use up to the maximum. Sort only allocates the storage needed for this which can be up to the value defined. So if sort only needs 10 KB, then that is all it will allocate.

OBJECT_CREATE_FORMAT

The OBJECT_CREATE_FORMAT subsystem parameter specifies whether DB2 is to create new table spaces and indexes to use a basic or extended log record format. See 3.1, “Extended RBA and LRSN” on page 24.

PARAMDEG_DPSI

The PARAMDEG_DPSI system parameter specifies the maximum degree of parallelism that you can specify for a parallel group in which a data partitioned secondary index (DPSI) is used to drive parallelism.

A DPSI is a non partitioning index that is physically partitioned according to the partitioning scheme of the table. When you specify a value of greater than 0 for this parameter, you limit the degree of parallelism for DPSIs so that DB2 does not create too many parallel tasks that use virtual storage.

Acceptable values are between 0-254, and **DISABLE**. The default is 0.

0	Specifies that DB2 uses the value that is specified for the **PARAMDEG** subsystem parameter, instead of **PARAMDEG_DPSI**, to control the degree of parallelism when DPSI is used to drive parallelism. This is the default value for the field.
1	Specifies that DB2 creates multiple child tasks but works on one task at a time when DPSI is used to drive parallelism.
2-254	Specifies that DB2 creates multiple child tasks and works concurrently on the tasks that are specified. The number of specified tasks can be larger or smaller than the number of tasks as specified in **PARAMDEG**. When **PARAMDEG** is set to 1, the rest of the query does not have any parallelism.
DISABLE	Specifies that DB2 does not use DPSI to drive parallelism. Parallelism might still occur for the query if **PARAMDEG** is greater than 1.

PARAMDEG_UTIL

The **PARAMDEG_UTIL** subsystem parameter specifies the maximum number of parallel subtasks for some utilities.

PARAMDEG_UTIL affects the following utilities:

- **REORG TABLESPACE**
- **REBUILD INDEX**
- **CHECK INDEX**
- **UNLOAD**
- **LOAD**

Acceptable values are positive integers between 0 to 32767, with a default of 0.

0	No additional constraint is placed on the maximum degree of parallelism in a utility.
1 to 32767	Specifies the maximum number of parallel subtasks for all affected utilities.

PCTFREE_UPD

The **PCTFREE_UPD** subsystem parameter specifies the default value to use for the **PCTFREE FOR UPDATE** clause of **CREATE TABLESPACE** or **ALTER TABLESPACE** statements. It specifies the default amount of free space to reserve on each page for use by subsequent **UPDATE** operations when

data is added to the table by **INSERT** operations or utilities. This parameter has no effect on table spaces that have fixed length rows.

Acceptable values are AUTO and 0 to 99, with a default of 0. This value is used as the default **FOR UPDATE** value when no **PCTFREE FOR UPDATE** clause is specified for a **CREATE TABLESPACE** or **ALTER TABLESPACE** statement.

AUTO	DB2 uses real-time statistics values to automatically calculate the percentage of free space that is to be used by update operations. This value is equivalent to specifying **PCTFREE FOR UPDATE -1** in the **CREATE TABLESPACE** or **ALTER TABLESPACE** statement.
0 to 99	DB2 reserves the specified percentage of space for use by update operations.

Without having experience with this function, you would assume that setting a value at page set level is more efficient in most cases than setting a positive integer for the system parameter.

For the use of **PCTFREE_UPD**, see 13.3, “Reduced need for REORG” on page 390.

PKGREL_COMMIT

The **PKGREL_COMMIT** subsystem parameter specifies whether, at **COMMIT** or **ROLLBACK**, a persistent DB2 thread will release a package that is active on that thread if certain DB2 operations are waiting for exclusive access to that package.

The value in this field is meaningful only for packages that are bound with the RELEASE(DEALLOCATE) bind option.

Acceptable values are YES and NO, with a default setting of YES

- YES

  For packages that are bound with the RELEASE(DEALLOCATE) option, the following operations are permitted at COMMIT or ROLLBACK while the package is active and allocated by DB2 for a persistent DB2 thread:

  - BIND REPLACE PACKAGE and REBIND PACKAGE requests, including auto rebind online schema changes for tables and indexes that are statically referenced by the package
  - Online REORG operations that materialize pending definition changes for objects that are statically referenced by the package

- NO

  DB2 will not implicitly release an active package at COMMIT or ROLLBACK for a persistent DB2 thread. See 4.9, “Allow BIND, REBIND, and DDL to break-in persistent threads” on page 81.

PREVENT_ALTERTB_LIMITKEY

PREVENT_ALTERTB_LIMITKEY determines whether DB2 disallows altering the limit key by using an **ALTER TABLE** statement for index-controlled partitioned table spaces. This alter operation places the table space in **REORG**-pending (REORP) restrictive status, and the data is unavailable until the affected partitions are reorganized. Use **PREVENT_ALTERTB_LIMITKEY** to avoid this data unavailability.

The values are acceptable:

NO	Specifies that you can alter a limit key by using an **ALTER TABLE** statement for index-controlled partitioned table spaces. This is the default.
YES	Specifies that altering a limit key by using an **ALTER TABLE** statement for index-controlled partitioned table spaces is not permitted. An **ALTER TABLE** statement must not attempt to alter the limit key for an index-controlled partitioned table.

Note: The focus here is on index-controlled partitioned table spaces. Altering an index key of a table-controlled partitioned table space is not this much of an issue, because this is considered a pending change, that is the table space is set to advisory state AREOR instead of restrictive state REORP.

Tip: Instead of wanting to make use of this system parameter, you might want to migrate your index-controlled partitioned table spaces to table-controlled partitioning. The easy way to accomplish this is to **ALTER** the clustering Index, which at the same time is the clustering index, to **NOT CLUSTER** and **ALTER** it back to **CLUSTER** right after this.

REC_FASTREPLICATION

The **REC_FASTREPLICATION** parameter specifies whether the **RECOVER** utility uses FlashCopy to recover from a FlashCopy image copy. The following values are acceptable:

NONE	The **RECOVER** utility uses standard I/O to restore a FlashCopy image copy.
PREFERRED	The **RECOVER** utility uses FlashCopy to recover from a FlashCopy image copy if FlashCopy support is available. This is the default.
REQUIRED	The **RECOVER** utility forces the use of FlashCopy to recover from a FlashCopy image copy to ensure that recovery occurs as quickly as possible. This option causes **RECOVERY** to fail if FlashCopy cannot be used.

Note: The last sentence in the description for option **REQUIRED** does not mean that DB2 fails if you only have sequential image copies that are sufficient for the recovery of your page set. The whole **DSNZPARM** applies only to FlashCopy image copies.

If you use **BACKUP SYSTEM** to create system-level backups, note that a recovery from a FlashCopy image copy that uses FlashCopy for the restore can cause **BACKUP SYSTEM** to fail because bidirectional FlashCopy is not supported.

REORG_DROP_PBG_PARTS

The **REORG_DROP_PBG_PARTS** subsystem parameter specifies whether the **REORG** utility removes trailing empty partitions when operating on an entire partition-by-growth table space. An empty trailing partition occurs when the **REORG** utility moves all data records from a partition into lower numbered partitions.

Acceptable values are DISABLE and ENABLE, with a default setting of DISABLE.

This parameter is meaningful only when the REORG utility is run against an entire PBG table space. It is ignored for a REORG of a non-partition-by-growth table space, for a partition-level

REORG of partition-by-growth table spaces, and for a **REORG** of a hash partition-by-growth table space.

ENABLE	Specifies that any trailing empty partitions that are present at the successful completion of the **REORG** are always removed. LOB table spaces and auxiliary indexes that are associated with these empty partition-by-growth partitions are also removed.
DISABLE	Specifies that the number of partition-by-growth partitions at the successful completion of the **REORG** are always equal or greater than the number of partitions before the **REORG** utility was run. Even if the **REORG** is able to relocate all data records into the lowest numbered partitions, trailing empty partition-by-growth partitions are retained.

REORG_MAPPING_DATABASE

The **REORG_MAPPING_DATABASE** subsystem parameter specifies the default database that **REORG TABLESPACE SHRLEVEL CHANGE** uses to implicitly create the mapping table.

An acceptable value is a character string of a maximum of 8 bytes length. The default value are 8 blanks, implying an implicitly defined database will be used

When processing a **REORG TABLESPACE SHRLEVEL CHANGE** request, the REORG utility has the option to create its own mapping table and mapping index, instead of relying on user's input. Specifying this subsystem parameter with a valid database name directs REORG to allocate the mapping table in the database that is specified. By default, REORG uses an implicitly defined database for the mapping table allocation. For details, see 11.1.4, “Automated REORG mapping table management” on page 278.

STATFDBK_SCOPE

The **STATFDBK_SCOPE** subsystem parameter specifies the scope of the SQL statements that DB2 collects statistics recommendations for in the **SYSIBM.SYSSTATFEEDBACK** catalog table.

The following are acceptable values:

NONE	No statistics recommendations are collected in the catalog table.
DYNAMIC	Statistics recommendations are collected in the catalog table for dynamic SQL statements only.
STATIC	Statistics recommendations are collected in the catalog table for static SQL statements only.
ALL	Statistics recommendations are collected in the catalog table for all SQL Statements. This is the default.

Note: Even though all DB2 users can query the SYSIBM.SYSSTATFEEDBACK table, the implemented functionality is primarily meant to be used by certain SQL optimization tools to help you improve access path selection for SQL statements.

Foe details, see 13.5.1, “Identification of critical statistics for improved query performance” on page 402.

TEMPLATE TIME

The TEMPLATE_TIME subsystem parameter specifies the default setting for the TIME option of the TEMPLATE statement.

Acceptable values are UTC and LOCAL, with a default of UTC.

UTC	Coordinated Universal Time.
LOCAL	Local time at the DB2 database manager.

Tip: Set all DB2 data sharing members to the same value.

UTILITY_OBJECT_CONVERSION

The value of the UTILITY_OBJECT_CONVERSION parameter specifies whether DB2 utilities that accept the RBALRSN_CONVERSION option will convert existing table spaces and indexes to 6-byte page format, to a 10-byte page format or prevent conversion of a 10-byte format to a 6-byte page format.

The default behavior normally applies when the utility control statement does not specify the RBALRSN_CONVERSION option. Alternatively, the UTILITY_OBJECT_CONVERSION parameter can also be used to prevent use of RBALRSN_CONVERSION options to convert existing table spaces and indexes to 6-byte page format

For a more detailed description about this system parameter and extended RBAs and LRSNs in general, refer to 3.1, “Extended RBA and LRSN” on page 24.

WFSTGUSE_AGENT_THRESHOLD

The WFSTGUSE_AGENT_THRESHOLD subsystem parameter determines the percentage of available space in the work file database on a DB2 subsystem or data sharing member that can be consumed by a single agent before a warning message is issued.

Refer to 4.4, “Work file database enhancements” on page 65 for the usage and implications of the settings for this system parameter.

WFSTGUSE_SYSTEM_THRESHOLD

The WFSTGUSE_SYSTEM_THRESHOLD subsystem parameter determines the percentage of available space in the work file database on a DB2 subsystem or data sharing member that can be consumed by all agents before a warning message is issued.

Refer to 4.4, “Work file database enhancements” on page 65 if you want to learn more about the usage and implications based on the settings for this system parameter

12.5.2 Changed defaults for existing system parameters

Besides the many new system parameters introduced with DB2 11 for z/OS, there are also a number of changed system parameters. Table 12-6 shows a list of affected system parameters.

Table 12-6 System parameters with changed limits

System parameter	Change	Description
DSMAX	Upper limit increased from 100,000 to 200,000	The maximum number of data sets that can be open at one time.
EDMDBDC	Upper limit increased from 2097152 to 4194304	The minimum size (in KB) of the DBD cache that is to be used by EDM.
EDMSTMTC	Upper limit increased from 1048576 to 4194304	The size (in KB) of the statement cache that is to be used by the EDM.
EDM_SKELETON_POOL	Upper limit increased from 2097152 to 4194304	The minimum size of the EDM skeleton pool in KB.

System parameter	Change	Description
MAXKEEPD	Upper limit increased from 65535 to 204800	The total number of prepared, dynamic SQL statements that can be saved past a commit point by all threads in the system using **KEEPDYNAMIC(YES)** bind option.
REORG_LIST_PROCESSING	Default was changed from SERIAL to PARALLEL	The default setting for the **PARALLEL** option of the **DB2 REORG TABLESPACE** utility. The **PARALLEL** option indicates whether **REORG TABLESPACE** processes all partitions specified in the input **LISTDEF** statement in a single utility execution **(PARALLEL YES)** or process each in a separate utility execution **(PARALLEL NO)**.
REORG_PART_SORT_NPSI	Default was changed from NO to AUTO	Specifies the default method of building a non-partitioned secondary index (NPSI) during **REORG TABLESPACE PART**. This setting will be used when the **SORTNPSI** keyword is not provided in the utility control statement. The **SORTNPSI** keyword specifies whether **REORG TABLESPACE PART** decides to sort all keys of a NPSI and how to make that decision. The setting is ignored for a **REORG** which is not part-level or a **REORG** with no NPSIs.
SUBQ_MIDX	Default was changed from DISABLE to ENABLE	Whether to enable or disable multiple index access for queries having subquery predicates.
PREVENT_NEW_IXCTRL_ PART	Default was changed from NO to YES.	Determines whether DB2 disallows the creation of new index-controlled partitioned tables. This subsystem parameter ensures that new partitioned tables use table-controlled partitioning, which is the preferred partitioning method for non-universal table spaces.

12.5.3 Removed system parameters

The category of system parameters described in this section are those that have been removed. The fact that system parameters have been removed does not necessarily mean that the functions that they influenced is taken out of DB2. Rather there is no way any longer to influence the behavior of DB2 with regards to the function of these system parameters.

Table 12-7 lists the removed system parameters.

Table 12-7 Removed system parameters

System parameter	DB2 11 behavior	Description
MVSGP and MVSGP2		Names, respectively, a group of MSS volumes to be used for archive 1 log data sets and group of MSS values to be used for archive 2 log data sets. DB2 11 does not recognize these devices. There were opaque system parameter, residing in **DSN6ARVP**. They are still there, but ignored if set.
CCORDNTR	NO	Specifies whether this DB2 member can coordinate parallel processing on other members of the group.

System parameter	DB2 11 behavior	Description
ASSIST	NO	Specifies whether this DB2 member can assist a parallelism coordinator with parallel processing. Because there cannot be coordinators anymore, there cannot be any assistants either.
OPTIXIO	ON	OPTIXIO=ON means that DB2 provides stability to I/O costing for queries, with less sensitivity to buffer pool sizes. Use of OPTIXIO=OFF can cause access path selection to be heavily influenced by object size and buffer pool size.
OPTIOWGT	ENABLE	Controls how DB2 balances the I/O cost and CPU estimates when selecting access paths.
OJPERFEH	YES	Specifies whether to enable outer join enhancements.
PTCDIO	OFF	Enables an optimizer fix for inefficient index path for a single-table query.
RETVLCFK	NO	Specifies whether the VARCHAR column is to be retrieved from a padded index.
DISABSL	NO	Specifies whether **SQLWARN1** and **SQLWARN5** are set for non-scrollable cursors on **OPEN** and **ALLOCATE CURSOR**.
SMSDCFL	blank	Specifies the DFSMS data class for indexes
STATCLUS	ENHANCED	Specifies the type of clustering statistics to be collected by the **RUNSTATS** utility. **ENHANCED** means that DB2 uses an improved algorithm for collecting statistics in effect with the drawback that it can change many access paths.
SEQCACH	SEQ	Specifies whether to use the sequential mode to read cached data from a 3990 controller. Prefetch reads will always use sequential access.
SEQPRES	YES	Specifies whether DB2 utilities that do a scan of a non partitioning index followed by an update of a subset of the pages in the index allow data to remain in cache longer when reading data.
DISABSCL	NO	Specifies whether **SQLWARN1** and **SQLWARN5** are set for non-scrollable cursors on **OPEN** and **ALLOCATE CURSOR**.
OPTIOPIN	YES	Specifies whether the DB2 optimizer should use an improved costing formula to estimate the cost of index and data access to the inner table of a join.

PGRNGSCR	YES	Specifies whether to enable a DB2 optimizer change that can improve performance of queries that contain one or more of the following predicates:
		`Timestamp < <host-var or string constant>`
		`Timestamp <= <host-var or string constant>`
		`Timestamp >= <host-var or string constant>`
		`Timestamp > <host-var or string constant>`
		`Timestamp BETWEEN <host-var or string constant>`
		`AND <host-var or string constant>`

Note: All the system parameters that are listed in Table 12-7 were already defined as deprecated in DB2 10.

12.5.4 Deprecated system parameters

The only subsystem parameter deprecated in DB2 11 is **PRIVATE_PROTOCOL**.

12.6 Release incompatibilities

This section describes the following types of incompatibilities:

- Application and SQL release incompatibilities
- Example of DB2 10 application compatibility
- Utility release incompatibilities
- Command release incompatibilities
- Storage release incompatibilities
- Functions that are deprecated
- Functions that are no longer supported

12.6.1 Application and SQL release incompatibilities

There are IBM and industry standards for SQL that DB2 for z/OS must be compliant with. DB2 for z/OS might be out of compliance because of a defect or an incomplete implementation. When detected and prioritized accordingly, try to fix those compliance issues with one of the upcoming new product releases. The fix might lead to an incompatible change. If the applications are not adjusted to the changed behavior before going to the new release, this might break existing applications after the release migration. These changes, which might lead to incompatibilities are saved up and introduced only on DB2 release boundary and not with maintenance stream within one major release.

Change to determination of ASUTIME for dynamic statements

In DB2 11 NFM with application compatibility set to `V11R1`, the dynamic SQL **ASUTIME** limit for each routine is used by the resource limit facility. The **ASUTIME** limit that is specified for the routine determines the limit. If the dynamic SQL statements in a routine use more **ASUTIME** than the limit, then SQLCODE -905 is returned. This SQLCODE occurs even if the value is lower than the ASUTIME limit of a top-level calling package. The **ASUTIME** limit that is specified for the top-level calling package is not considered. In previous versions of DB2, SQLCODE -905 is issued only when the limit of the top-level calling package is encountered.

The possible impact to your DB2 environment might be that because the limit is enforced for each monitored routine, your applications might return more SQLCODE -905 errors.

While in conversion mode with application compatibility for your package set to value `V10R1`, run your applications with IFCID 0366 or IFCID 0376 enabled. Then, review the trace output for incompatible changes with the identifier `1103`. Review and, if necessary, adjust the **ASUTIME** limits on routines and packages that use dynamic SQL.

Automatic rebind of plans and packages created before DB2 Version 9

Plans and packages that were last bound before Version 9 are not supported in DB2 11 CM and later.

- Possible impact to your DB2 environment:

  If you specify YES or COEXIST for the **ABIND** subsystem parameter, DB2 11 automatically rebinds plans and packages that were bound before Version 9. As a result, an execution delay might occur the first time that such a plan or package is loaded. Also, the automatic rebind might change the access path to a potentially more efficient access path.

  If you specify NO for the ABIND subsystem parameter, negative **SQLCODE**s are returned for each attempt to run a package or plan that was bound before Version 9. SQLCODE -908, SQLSTATE 23510 is returned for packages, and SQLCODE -923, SQLSTATE 57015 is returned for plans until they are rebound in DB2 11.

- Actions to take:

  To identify plans and packages that were bound before Version 9, run the DB2 11 premigration job DSNTIJPM on your DB2 10 catalog. If the job output reports some packages, you might want to rebind while still in DB2 10 to prevent those automatic rebinds to occur.

Invalidated plans and packages

During the enabling-NFM processing, plans and packages that reference the DB2 catalog and directory table spaces which are changed by CATENFM, become invalidated.

Refer to “DSNTIJEN” on page 341 for the list of table spaces which are modified during CATENFM:

The packages that are dependent on the following catalog tables are also invalidated:

- SYSIBM.SYSCOPY
- SYSIBM.SYSCHECKS
- SYSIBM.SYSCHECKS2
- SYSIBM.SYSCHECKDEP
- SYSIBM.SYSSTRINGS
- SYSIBM.SYSINDEXSPACESTATS
- SYSIBM.SYSTABLESPACESTATS

Actions to take:

- For SYSLGRNX, existing CHAR(6) columns were changed to CHAR(10). You might need to modify your application before it can run successfully.
- For **SYSUTILX**, the RBA fields were moved to new fields.
- The **SYSCOPY** table space was replaced by a new table space, **SYSTSCPY**.
- The **SYSRTSTS** table space was replaced by two new table spaces, **SYSTSTSS** and **SYSTSISS**. **SYSTSTSS** contains the **SYSIBM.SYSTABLESPACESTATS** catalog table and **SYSTSISS** contains the **SYSIBM.SYSINDEXSPACESTATS** table.
- The **SYSSTR** table space was replaced by the following table spaces:
  - **SYSTSCKS**, which contains **SYSIBM.SYSCHECKS**
  - **SYSTSCHX**, which contains **SYSIBM.SYSCHECKS2**
  - **SYSTSCKD**, which contains **SYSIBM.SYSCHECKDEP**
  - **SYSTSSRG**, which contains the **SYSIBM.SYSSTRINGS** catalog table

Important: For all the new table spaces mentioned previously, you must make sure that you adjust your procedures, such as COPY of the DB2 catalog and directory, as well RECOVER of catalog and directory. When you adjust your procedures, you must make sure that you remember that there is a specific order, which is required for copying and recovering your catalog and directory.

Refer to the description of the **RECOVER** utility in *DB2 11 for z/OS Utility Guide and Reference*, SC19-4067 for more information.

Also, make sure that you carefully thing about any other processes or products, which can rely on the old page set names and adjust those accordingly.

Default for ODBC limited block fetch

The default for the LIMITEDBLOCKFETCH initialization keyword changed.

Limited block fetch guarantees the transfer of a minimum amount of data in response to each request from the requesting system.

With limited block fetch, a single conversation is used to transfer messages and data between the requester and server for multiple cursors. Processing at the requester and server is synchronous. The requester sends a request to the server, which causes the server to send a response back to the requester. The server must then wait for another request to tell it what should be done next.

In DB2 10, ODBC limited block fetch was disabled by default. In DB2 11 NFM, ODBC limited block fetch is enabled by default.

The possible impact to your DB2 environment is that your applications might use limited block fetch, when they did not do so previously.

Tip: If the default is not appropriate for your ODBC applications, you can change it by modifying the value of the LIMITEDBLOCKFETCH initialization keyword.

Views, MQTs, and SQL table functions with period specifications

In DB2 11, views, materialized query tables, and SQL table functions that were created with period specifications in DB2 10 are not supported. Period specifications refer to either system-time or application-time (formerly known as *system-time*) temporal tables. If such views, materialized query tables, or SQL functions are used in DB2 11, incorrect results might occur.

Note: In DB2 11, you can still create a view to reference a System Period Temporal Table (STT), but a temporal predicate (For System_Time....) is not allowed in the view definition. The following option is allowed:

```
CREATE VIEW V_STT1 AS SELECT * from STT1
```

The following option not allowed:

```
CREATE VIEW V_STT1_NO AS SELECT * FROM STT1 FOR SYSTEM_TIME AS OF 2013-09-26
15:25:00.0
```

The temporal predicate is disallowed in a view definition to avoid nesting of temporal predicates, such as:

```
SELECT * FROM V_STT1_NO FOR SYSTEM TIME AS OF 2012-09-27 09:00:00.0.
```

In this case, there is no clearly defined semantic whether to use the temporal predicate from the outmost SELECT or the inner most SELECT.

To prepare for this change, drop all views, materialized query tables, and SQL table functions that contain a **SYSTEM_TIME** or **BUSINESS_TIME** period specification.

To identify such existing views, materialized query tables, and SQL table functions, run the DB2 11 premigration job DSNTIJPM on your DB2 10 catalog. You can also manually issue the following queries.

- To identify views and materialized query tables that were created with a period specification, issue the following query:

  ```
  SELECT * FROM SYSIBM.SYSVIEWDEP WHERE BTYPE IN ('W', 'Z') AND DTYPE IN ('V',
  'M');
  ```

- To identify SQL table functions that were created with a period specification, issue the following query:

  ```
  SELECT * FROM SYSIBM.SYSDEPENDENCIES WHERE BTYPE = 'Z';
  ```

- To identify SQL scalar functions that were created with a period specification or period clause, issue the following query:

  ```
  SELECT * FROM SYSIBM.SYSPACKDEP WHERE BTYPE IN ('W', 'Z') AND DTYPE = 'N';
  ```

Dropping columns named CLONE, ORGANIZATION, or VERSIONING

In DB2 11 NFM, a column that is named **CLONE**, **ORGANIZATION**, or **VERSIONING** should be specified as a delimited identifier to be dropped from a table.

Prior to DB2 11, **CLONE**, **ORGANIZATION**, and **VERSIONING** are reserved keywords that can appear after the **DROP** keyword in an **ALTER TABLE** statement. When **CLONE**, **ORGANIZATION**, or **VERSIONING** is specified as a simple token (that is, not as a delimited identifier), these keywords can only match the **DROP CLONE**, **DROP ORGANIZATION**, or **DROP VERSIONING** clauses on an **ALTER TABLE** statement.

If you intend to drop a column named **CLONE**, **ORGANIZATION**, or **VERSIONING** in DB2 11, and the name is specified as a simple token on the **ALTER TABLE** statement, the DB2 subsystem might interpret the **ALTER TABLE** statement as specifying the **DROP CLONE**, **DROP ORGANIZATION**, or **DROP VERSIONING** clauses instead of the **DROP COLUMN** clause.

To drop a column named **CLONE**, **ORGANIZATION**, or **VERSIONING** in DB2 11, the name must be specified as a delimited identifier (for example, **DROP "ORGANIZATION"** or **DROP "CLONE"**, assuming is the delimiter for a delimited identifier.

Alternatively, you can specify the optional **COLUMN** keyword in the **DROP COLUMN** clause, for example **DROP COLUMN ORGANIZATION** or **DROP COLUMN CLONE**.

See Table 12-8 for a summary of **DROP COLUMN** for **CLONE**.

Table 12-8 DROP example for CLONE

Prerequisite status	SQL statement	Result
CLONE relationship exists	ALTER TABLE ... DROP CLONE	RC 0, Clone table dropped
CLONE relationship exists	ALTER TABLE ... DROP “CLONE”	SQLCODE -148, reason-code 11, which means: The ALTER statement attempted to change a table that has a defined clone, or a table that is a clone.
CLONE relationship exists	ALTER TABLE ... DROP COLUMN CLONE	SQLCODE +610, because this is a schema change which leads to AREOR, that is it is a pending change.
NO CLONE relationship	ALTER TABLE ... DROP CLONE	SQLCODE -650, reason-code 20, which means: ALTER TABLE DROP CLONE cannot be used to drop a clone when the table does not have a defined clone.
NO CLONE relationship	ALTER TABLE ... DROP “CLONE”	SQLCODE -104, because DROP COLUMN needs keyword RESTRICT
NO CLONE relationship	ALTER TABLE .. DROP “CLONE” RESTRICT	SQLCODE +610, because this is a schema change which leads to AREOR, that is it is a pending change.
NO CLONE relationship	ALTER TABLE ... DROP COLUMN CLONE	SQLCODE +610, because this is a schema change which leads to AREOR, that is it is a pending change.

Allow XPath processing to continue with error on filtered results

In DB2 11 NFM with application compatibility set to `V11R1`, XPath processing might return fewer errors on predicate expressions with an explicit cast or an operation with an invalid value.

In previous versions of DB2, even though the invalid result is filtered from the result set, XPath processing returns an error SQLCODE. In DB2 11, examples of XPath expressions that have fewer errors include situations when:

- Data is filtered from the result by the predicate before an invalid operation such as division of a number by zero
- Data is explicitly cast to an incompatible data type

Tip: While in conversion mode with application compatibility for your package set to value `V10R1`, run your applications with IFCID 0366 or IFCID 0376 enabled. Then, review the trace output for incompatible changes with the identifier `1102`.

Here is an example, with the definition of two books. See Example 12-9.

Example 12-9 Books definition

```
<books>
    <book><title>XQuery 3.0</title>
        <publishDate>soon</publishDate>
```

```
        <price>40.00</price>
        <edition>kindle</edition>
   </book>
    <book><title>XQuery 3.0</title>
        <publishDate>2013-09-30</publishDate>
        <price>50.00</price>
        <edition>paper</edition>
   </book>
</books>
```

Here is the query:

```
XMLQUERY('/books/book[title = "XQuery 3.0"][xs:date(publishDate) >
"2013-10-01"][edition="paper"]'  passing xmlcol)
```

You can see that `xs:date(publishDate)` would be an error for the first book because “soon” cannot be cast to date. However, it depends on the order the predicates are evaluated. If `[edition="paper"]` is evaluated first, the first book would be filtered out before the cast on date. If `xs:date(publishDate) > "2013-10-01"` is evaluated first, then the error shows up.

DB2 11 defers the error reporting until the last predicate is evaluated. Thus, the error is not reported. Due to the flexibility of XML, we try to provide more usability and fewer errors.

XML document node implicitly added on insert and update

In DB2 11 NFM with application compatibility set to `V11R1`, if an XML document does not have a document node, then one is added during insert and update operations.

In previous versions of DB2, document nodes are not implicitly added and an SQL insert or update of an XML document returned SQLCODE -20345. To avoid the error, an application needs to invoke the XMLDOCUMENT function before the insert or update.

In DB2 11, an XML document node is added, if one does not exist in the XML document.

The result is, that your applications might return fewer errors on insert and update operations.

Tip: While in conversion mode with application compatibility for your package set to value `V10R1`, run your applications with IFCID 0366 or IFCID 0376 enabled. Then, review the trace output for incompatible changes with the identifier `1101`. In addition, you can review your applications for use of the **`XMLDOCUMENT`** function.

Here is an example:

```
select xmlelement(element "test", 1)  from sysibm.sysdummy1;
```

returns

```
<?xml version="1.0"><test>1</test>
insert into T1(xmlcol) values ('<?xml version="1.0"><test>1</test>');
```

works fine.

However, if you use **`XMLELEMENT`** directly in the insert as shown in the following example, you get -20345 on DB2 10.

```
insert into T1(xmlcol) values (xmlelement(element "test", 1));
```

The reason is that in DB2 that **XMLELEMENT** generates an XML element node but it does not generate a document node which is an invisible root of an XML tree. Insert requires an document node. Thus, you get -20345.

The reason that insert with XML string works is because the **XMLPARSE** function implicitly generates an XML document node. With DB2 10, you have to inject an **XMLDOCUMENT** function as shown in the following example to make the insert work.

```
insert into T1(xmlcol) values (XMLDOCUMENT(xmlelement(element "test", 1)));
```

On DB2 11, you do not have to do that. The original insert would work.

Client information results from ADMIN_COMMAND_DB2

Starting in DB2 11 CM, the **ADMIN_COMMAND_DB2** result set row in the created global temporary table **SYSIBM.DB2_THREAD_STATUS** when `processing-type = "THD"` has changed. The column data type and maximum lengths for **WORKSTATION, USERID, APPLICATION**, and ACCOUNTING have changed.

In DB2 11 the following column data types and lengths change:

- **WORKSTATION** increases from `CHAR(18)` to `VARCHAR(255)`.
- **USERID** increases from `CHAR(16)` to `VARCHAR(128)`.
- **APPLICATION** increases from `CHAR(32)` to `VARCHAR(255)`.
- **ACCOUNTING** increases from `CHAR(247)` to `VARCHAR(255)`.

Your applications now receive a VARCHAR data type and possibly a different length client information value. The length is no longer padded to the supported maximum length.

In DB2 11, the stored procedure SYSPROC.ADMIN_COMMAND_DB2 also allows users to specify PROCESSING_TYPE (formerly PARSE_TYPE) LOB tables spaces (LS), XML table spaces (XS) and unknown spaces (UN) to retrieve information about table spaces when issuing the command **-DISPLAY DATABASE**. Based on the three output messages by type from ADMIN_COMMAND_DB2, you can generate COPY utility jobs to create image copies for the table spaces.

You should review your applications for use of the ADMIN_COMMAND_DB2 stored procedure.

Altering limit keys blocks immediate definition changes

In DB2 11 NFM, if you alter a limit key for certain table space types, you cannot make any immediate definition changes until the limit key changes are materialized.

In previous versions of DB2, altering a limit key was an immediate definition change. In DB2 11, altering a limit key for one of the following types of partitioned table spaces is now a pending definition change:

- Range-partitioned universal table spaces
- Table spaces that are partitioned (non-universal) with table-controlled partitioning

As in DB2 10, you cannot make immediate definition changes before pending definition changes are materialized.

Restriction: Some immediate alter operations that worked in previous versions of DB2 might fail in DB2 11 with SQLCODE -20385 reason code 1 or 2.

The new way for altering limit keys is described in detail in 4.3, “Improved availability when altering limit keys” on page 61.

Removing the SYSPUBLIC schema from the SQL PATH routine option

Starting in DB2 11 conversion mode, SYSPUBLIC is the schema that is used for public aliases. As such, the SQL PATH routine option must not specify the SYSPUBLIC schema.

In previous versions of DB2, you could not define functions, procedures, distinct types, and sequences in the SYSPUBLIC schema, but you were not restricted from specifying SYSPUBLIC as part of the SQL PATH. If you had specified SYSPUBLIC as part of the SQL PATH, it had no effect on their applications. With DB2 11 you will no longer be able to specify SYSPUBLIC as part of the SQL PATH.

Creation or resolution of some objects that worked in previous versions of DB2 might fail in DB2 11 with SQLCODE -713 if SYSPUBLIC is specified as part of the SQL PATH.

Query the catalog to see if any object schemas use SYSPUBLIC as the schema qualifier. This is highly unlikely for any object, but most likely with objects that use the SQL PATH (functions, procedures, distinct types, and sequences).

Change any existing SET PATH statements to not specify SYSPUBLIC as a schema.

SYSIBMADM schema added to the SQL path

In DB2 11 NFM with application compatibility set to `V11R1`, **`SYSIBMADM`** is added to the SQL path as an implicit schema.

If **`SYSIBMADM`** is not specified as an explicit schema in the SQL path, it is included as an implicit schema at the beginning of the path after **`SYSIBM`**, **`SYSFUN`**, and **`SYSPROC`**.

Applications that reference the content of the **`CURRENT PATH`** special register now have the **`SYSIBMADM`** schema returned when implicit schemas are included in the path. For example, the statement **`SELECT CURRENT PATH FROM SYSIBM.SYSDUMMY1`** now returns “SYSIBM”, ”SYSFUN”, ”SYSPROC”, ”SYSIBMADM”, ”authid”, where authid is the authorization ID of the statement, instead of “SYSIBM”, ”SYSFUN”, ”SYSPROC”, ”authid.”

Change in result for CAST(string AS TIMESTAMP)

In DB2 11 NFM with application compatibility set to `V11R1`, the result of `CAST(string AS TIMESTAMP)` is changed in some cases.

Previously, when DB2 executed `CAST(string AS TIMESTAMP)`, DB2 interpreted an 8-byte string as a Store Clock value and a 13-byte string as a GENERATE_UNIQUE value. This interpretation might result in an incorrect result from the CAST specification. Starting with DB2 11, with the application compatibility set to V11R1, when an 8-byte string or a 13-byte string is input to `CAST(string AS TIMESTAMP)`, DB2 interprets the input strings as string representations of TIMESTAMP values.

An invalid representation of an 8-byte or 13-byte string in CAST(string AS TIMESTAMP) results in SQLCODE -180.

Suppose that you execute the SQL statements in DB2 11 NFM listed in Example 12-10 and Example 12-11 which show the DB2 10 and the DB2 11 behavior.

The examples should help you understand the issue.

Example 12-10 sets **APPLCOMPAT** special register to V10R1 to simulate the DB2 10 behavior. The casting character string 01/01/2013 to **TIMESTAMP**, which is supposed to represent January 1st, 2013, results in a completely different timestamp, dated 2034.

In the second part of Example 12-10, you see that if you provide the first 8 bytes of the store clock value, that is X'CAB5060708090100', which represents January 1st, 2013, casting returns the expected date.

Example 12-10 CAST as TIMESTAMP with APPLCOMPAT set to V10R1

```
SELECT CAST('1/1/2013' AS TIMESTAMP) FROM SYSIBM.SYSDUMMY1;
---------+---------+---------+---------+---------+---------+

---------+---------+---------+---------+---------+---------+
2034-07-25-16.43.41.599503

SELECT CAST(X'CAB5060708090100' AS TIMESTAMP) FROM SYSIBM.SYSDUMMY1;
---------+---------+---------+---------+---------+---------+--------

---------+---------+---------+---------+---------+---------+--------
2013-01-01-20.37.04.246928
```

Example 12-11 uses the exact same SQL statements but sets the **APPLCOMPAT** special register to V11R1. The first **SELECT** statement now returns what you in fact might have expected, that is the date of January 1st, 2013.

The second SELECT statement fails now, because in DB2 11, DB2 interprets the input strings as string representations of TIMESTAMP values, which X'CAB5060708090100' clearly not is.

Example 12-11 CAST as TIMESTAMP with APPLCOMPAT set to V11R1

```
SELECT CAST('1/1/2013' AS TIMESTAMP) FROM SYSIBM.SYSDUMMY1;
---------+---------+---------+---------+---------+---------

---------+---------+---------+---------+---------+---------
2013-01-01-00.00.00.000000


SELECT CAST(X'CAB5060708090100' AS TIMESTAMP) FROM SYSIBM.SYSDUMMY1;    00
---------+---------+---------+---------+---------+---------+---------+----

---------+---------+---------+---------+---------+---------+---------+----
DSNE610I NUMBER OF ROWS DISPLAYED IS 0
DSNT408I SQLCODE = -180, ERROR:  THE DATE, TIME, OR TIMESTAMP VALUE *N IS
         INVALID
```

Tip: While in DB2 11 conversion mode, or in DB2 11 NFM with application compatibility set to V10R1, identify applications with this incompatibility by starting a trace for IFCID 0366 or IFCID 0376, and then running the applications. Review the trace output for incompatible changes with the identifier 1109. If you need to convert Store Clock values to the **TIMESTAMP** data type, use the **TIMESTAMP** built-in function instead of CAST(string AS TIMESTAMP).

Example 12-12 lists an example.

Example 12-12 Invoke scalar function TIMESTAMP with store clock value

```
SELECT TIMESTAMP(X'CAB5060708090100') FROM SYSIBM.SYSDUMMY1;
---------+---------+---------+---------+---------+---------+-

---------+---------+---------+---------+---------+---------+-
2013-01-01-20.37.04.246928
```

New maximum lengths for values that are returned for some built-in functions

In DB2 11 NFM with application compatibility set to `V11R1`, the maximum lengths for values that are returned for the **SPACE** and **VARCHAR** built-in functions are decreased from 32767 to 32764.

If the length of the output string for any of these functions is greater than 32764 bytes, SQLCODE -171 is returned.

Review your applications for use of these functions, and, if necessary, modify the function input so that the result does not exceed 32764 bytes. While in conversion mode with application compatibility for your package set to value `V10R1`, run your applications with IFCID 0366 or IFCID 0376 enabled. Then, review the trace output for incompatible changes with the identifier `1110` or `1111`.

Timestamp string representations

DB2 11 NFM with application compatibility set to `V11R1` strictly enforces valid string representations of timestamp values.

DB2 11 behavior with application compatibility set to V11R1 is equivalent to DB2 10 with subsystem parameter `BIF_COMPATIBILITY = CURRENT`. With application compatibility set to V10R1, the enforcement of valid string representations depends on the **BIF_COMPATIBILITY** value.

Review your setting of the **BIF_COMPATIBILITY** subsystem parameter. If the value is not `CURRENT`, while in conversion mode with application compatibility for your package set to value `V10R1`, run your applications with IFCID 0366 or IFCID 0376. Then, review the trace output with the function identifier `3` to identify SQL with unsupported time stamp values. Make appropriate changes to your SQL.

12.6.2 Utility release incompatibilities

As for some other areas, utilities also have some incompatible changes, which are described in this section.

Parallelism change to the REBUILD INDEX utility

In DB2 11 conversion mode, the degree of parallelism can increase for the **REBUILD INDEX** utility.

The **REBUILD INDEX** utility previously limited the degree of parallelism to 18 subtasks. Now, because of the **PARALLEL** option value or the **PARAMDEG_UTIL** subsystem parameter value, the amount of parallelism might increase.

Attention: Increasing the degree of parallelism could constrain your system resources.

Refer to “PARAMDEG_UTIL” on page 350 if you want to read more about **PARAMDEG_UTIL** system parameter.

Changes to REORG default values

In DB2 11 conversion mode, the following changes are made to the default values for the REORG utilities:

The following changes are made to the default values for the **REORG TABLESPACE** utility:

- The default **DRAIN** value is changed from **WRITERS** to **ALL**.
- The **NOPAD** keyword is now the default value in the **UNLOAD EXTERNAL** clause and the **DISCARD** clause.

For the **REORG INDEX** utility the default for the **DRAIN** value is also changed from **WRITERS** to **ALL**.

Change to DSNU126I return code when running REORG on an LOB table space

DB2 10 deprecated the use of **REORG TABLESPACE SHRLEVEL NONE** for LOB table spaces. When you nevertheless used it, a **REORG SHRLEVEL REFERENCE** was performed and the **DSNU126I** message, which indicates that **SHRLEVEL NONE** is no longer supported, was associated with RC 0. If you try to use **REORG LOG SHRLEVEL NONE**, the job fails with **DSNU126I** and return code 8.

In preparation for this change, review your **REORG** job outputs for instances of **DSNU126I** while you are still running in DB2 10.

Changes to RECOVER utility

The **TOLOGPOINT**, **TORBA**, and **RESTOREBEFORE** keywords can accept basic 6-byte format or extended 10-byte format based on the length of the RBA or LRSN value that is specified. Previously, any length was accepted and then extended or truncated as required.

Operands of 6 bytes or less are interpreted as being in basic 6-byte format. Operands greater than 6 bytes are interpreted as being in ended 10-byte format. In both cases, padding on the left with X'00' occurs to form complete 6-byte or 10-byte operands. Conversion between basic and extended format is performed as required for the recovery operation.

Changes to DSNACCOX stored procedure default values

In DB2 11 NFM, changes are made to the defaults of the **DSNACCOX** stored procedure.

RRTDataSpaceRat input parameter default value is -1. Previously, it was 2.0.

RRTDataSpaceRat is the ratio of the space allocated to the actual space used. Specifies a criterion for recommending that the **REORG** utility is to be run on table space for space reclamation. If the following condition is true, **DSNACCOX** recommends running REORG:

- The object is not using hash organization.
- The **SPACE** allocated is greater than **RRTDataSpaceRat** multiplied by the actual space used. (SPACE > RRTDataSpaceRat × (DATASIZE/1024))

Tip: Review your calls to the DSNACCOX stored procedure. Look for NULL as the value of **RRTDataSpaceRat**. The new default turns off this criterion. Any positive values continue to be processed as in DB2 10.

Changes to DSNACCOX stored procedure result set

In DB2 11 NFM, 'XS' for XML table spaces and 'LS' for LOB table spaces are now possible values of the OBJECTTYPE column results.

Review your processing of results from the DSNACCOX stored procedure. Unexpected values might be handled as invalid by applications processing these result sets.

Changes to DSNACCOX processing for REORG and COPY recommendations

In DB2 11 NFM, more information is evaluated when **REORG** or **COPY** is recommended.

When the input parameter QueryType specifies **REORG** or **COPY** recommendations, DSNACCOX also checks the database exception table (DBET) entry for an exception state.

Review your processing of results from the DSNACCOX stored procedure. Unexpected values might be handled as invalid. Database exception table (DBET) states are added to the OBJECTSTATUS column of the result set.

Changes to DSNACCOX stored procedure processing for ChkLvl 8

In DB2 11 NFM, a new row is not inserted if the result set already has a recommendation for a utility operation.

When the input parameter ChkLvl specifies level 8 processing, DSNACCOX adds the utility operation recommendation to an existing row if one exists for the object. If an existing row does not exist, DSNACCOX continues to add a row.

ChkLvl 8 means: Check for objects that have restricted states. The value of the QueryType option must be ALL or contain RESTRICTED when this value is specified. The OBJECTSTATUS column of the result set indicates the restricted state of the object. A row is added to the result set for each object that has a restricted state.

Review your processing of results from the DSNACCOX stored procedure. Unexpected values might be handled as invalid. New rows for objects are only added to the result set if the object is not already present.

Differences in materializing limit key changes

In DB2 11 NFM, you can no longer materialize limit key changes for certain types of table spaces by using REORG TABLESPACE SHRLEVEL NONE or LOAD REPLACE.

Instead, this alter is a pending definition change, and the data remains accessible before the limit key changes are materialized. However, you cannot use the REORG TABLESPACE utility with the SHRLEVEL NONE option or the LOAD utility with the REPLACE option to materialize these changes. (SHRLEVEL NONE is the default value for REORG TABLESPACE. If you do not specify the SHRLEVEL option for REORG TABLESPACE, SHRLEVEL NONE is in effect.)

To learn more about the new behavior for limit key changes, you should refer to 4.3, "Improved availability when altering limit keys" on page 61.

12.6.3 Command release incompatibilities

In terms of commands, there are just a few incompatible changes that you should be aware of. They are listed within this section.

Change to DISPLAY UTILITY output

The output for the DISPLAY UTILITY command now includes the date and the time when the job was submitted.

A sample output is available in 11.6.1, “DISPLAY UTILITY additional output” on page 306.

Tip: Determine if any of your applications parse output of the DISPLAY UTILITY command and update the applications if needed.

Removing the SYSPUBLIC schema from the PATH bind option

Starting in DB2 11 conversion mode, SYSPUBLIC is the schema that is used for public aliases. As such, the PATH bind option must not specify the SYSPUBLIC schema.

In previous versions of DB2, you were not restricted from specifying SYSPUBLIC as part of the PATH bind option. With DB2 11 you will no longer be able to specify SYSPUBLIC as part of the PATH bind option.

Creation or resolution of some objects that worked in previous versions of DB2 might fail in DB2 11 with SQLCODE -713 if SYSPUBLIC is specified as part of the PATH bind option.

Query the catalog to see if any object schemas use SYSPUBLIC as the schema qualifier. This is highly unlikely for any object, but most likely with objects that use the PATH (functions, procedures, and sequences).

Change any existing PATH bind option to not specify SYSPUBLIC as a schema.

Note: PUBLIC ALIASES can only be defined for SEQUENCEs. This functionality does not apply to tables.

12.6.4 Storage release incompatibilities

When you migrate to DB2 11, be aware of the storage release incompatibilities.

There is a new minimum that your z/OS application programmers have to set for HVSHARE.

In DB2 11, the required amount of contiguous 64-bit shared private storage for each DB2 subsystem is 1 TB. In previous releases, the minimum requirement was 128 GB.

Restriction: If you do not have an adequate amount of contiguous 64-bit shared private storage, DB2 11 will not start.

12.6.5 Functions that are deprecated

During migration, be aware of the functions that are deprecated in DB2 11. Although they are supported in DB2 11, support for these functions might be removed in the future. Avoid creating new dependencies that rely on these functions, and if you have existing dependencies on them, develop plans to remove these dependencies.

The following functions are deprecated in DB2 11.

NEWFUN SQL processing options and DECP values

The SQL processing options NEWFUN(YES) and NEWFUN(NO) are deprecated, and the NEWFUN(V11) option is added in DB2 11. Use NEWFUN(V11) instead of NEWFUN(YES). Use NEWFUN(V10) instead of NEWFUN(NO). The NEWFUN(V8) and NEWFUN(V9) values are supported in DB2 11, but they cause the precompilation process to support only a Version 8 or Version 9 level of function.

The DSNHDECP parameter values NEWFUN=YES and NEWFUN=NO are also deprecated. Although these values are supported in DB2 11, you should use NEWFUN=V11 instead of NEWFUN=YES and use NEWFUN=V10 instead of NEWFUN=NO.

Note: You can only use NEWFUN(V8) or NEWFUN(V9) as a precompiler option. It is not allowed as DSNHDECP parameter option.

Some utility options

The following DB2 utility options are deprecated. Although they are supported in DB2 11, they will be removed in a later release of DB2.

- **REORG TABLESPACE UNLOAD ONLY**

  Use the **UNLOAD** utility instead.

- **REORG TABLESPACE UNLOAD PAUSE**

  Use the **UNLOAD FORMAT INTERNAL** utility instead.

- **REORG TABLESPACE UNLOAD EXTERNAL**

  Use the **UNLOAD** utility instead.

- **REORG TABLESPACE INDREFLIMIT**

  Use the **DSNACCOX** stored procedure to determine if the object needs to be reorganized.

- **REORG TABLESPACE OFFPOSLIMIT**

  Use the **SYSPROC.DSNACCOX** stored procedure to determine if the object needs to be reorganized.

- **REORG TABLESPACE INDREFLIMIT REPORTONLY and REORG TABLESPACE OFFPOSLIMIT REPORTONLY**

  REPORTONLY is valid only when the **INDREFLIMIT** or **OFFPOSLIMIT** option is specified, and these options are deprecated.

- **REORG INDEX UNLOAD ONLY**

  Use the **DIAGNOSE** utility stop the process instead.

- **REORG INDEX UNLOAD PAUSE**

  Use the **DIAGNOSE** utility stop the process instead.

- **REORG INDEX LEAFDISTLIMIT**

  Use the **DSNACCOX** stored procedure to determine if the object needs to be reorganized.

- **REORG INDEX LEAFDISTLIMIT REPORTONLY**

  REPORTONLY is valid only when the **LEAFDISTLIMIT** option is specified, and this option is deprecated.

- **LOAD FORMAT UNLOAD**

  This is what you used when you generated the **SYSREC** using **REORG TABLESPACE UNLOAD ONLY**. A few steps back indicated that this deprecated in DB2 11 and that you should use **UNLOAD FORMAT INTERNAL** if you want to generate the same type of date.

Use the **LOAD FORMAT INTERNAL** option to load data that was unloaded with UNLOAD **FORMAT INTERNAL**.

- **COPY CHANGELIMIT**

  Use the **DSNACCOX** stored procedure to determine if the object needs to be copied.

- **REPAIR VERSIONS**

  Use the **REPAIR CATALOG** utility instead.

12.6.6 Functions that are no longer supported

If you are migrating to DB2 11 from DB2 10, be aware of the functions that are no longer supported.

Password protection for active log and archive log data sets

As of DB2 11, password protection for active log and archive log data sets is no longer supported.

Previous NEWFUN values

As of DB2 11, the DSNH CLIST no longer supports values of NEWFUN=V8 or NEWFUN=V9.

Some DB2-supplied routines

The following DB2-supplied routines are removed in DB2 11 and are unavailable to callers after migration to conversion mode. A report is added to the DSNTIJPM premigration job to detect occurrences of these routines on an existing subsystem or data sharing group, and to specify that these routines are not available in DB2 11.

- SYSPROC.DSNAEXP
- AMI-based DB2 MQ functions[1]
  - DB2MQ1C.GETCOL
  - DB2MQ1C.MQPUBLISH
  - DB2MQ1C.MQREAD
  - DB2MQ1C.MQREADALL
  - DB2MQ1C.MQREADALLCLOB
  - DB2MQ1C.MQREADCLOB
  - DB2MQ1C.MQRECEIVE
  - DB2MQ1C.MQRECEIVEALL
  - DB2MQ1C.MQRECEIVEALLCLOB
  - DB2MQ1C.MQRECEIVECLOB
  - DB2MQ1C.MQSEND
  - DB2MQ1C.MQSUBSCRIBE
  - DB2MQ1C.MQUNSUBSCRIBE
  - DB2MQ2C.GETCOL
  - DB2MQ2C.MQPUBLISH
  - DB2MQ2C.MQREAD
  - DB2MQ2C.MQREADALL
  - DB2MQ2C.MQREADALLCLOB
  - DB2MQ2C.MQREADCLOB
  - DB2MQ2C.MQRECEIVE
  - DB2MQ2C.MQRECEIVEALL
  - DB2MQ2C.MQRECEIVEALLCLOB

[1] They were deprecated in DB2 9. You can convert those applications that use the AMI-based functions to use the MQI-based functions

- DB2MQ2C.MQRECEIVECLOB
- DB2MQ2C.MQSEND
- DB2MQ2C.MQSUBSCRIBE
- DB2MQ2C.MQUNSUBSCRIBE
- DB2MQ1N.GETCOL
- DB2MQ1N.MQPUBLISH
- DB2MQ1N.MQREAD
- DB2MQ1N.MQREADALL
- DB2MQ1N.MQREADALLCLOB
- DB2MQ1N.MQREADCLOB
- DB2MQ1N.MQRECEIVE
- DB2MQ1N.MQRECEIVEALL
- DB2MQ1N.MQRECEIVEALLCLOB
- DB2MQ1N.MQRECEIVECLOB
- DB2MQ1N.MQSEND
- DB2MQ1N.MQSUBSCRIBE
- DB2MQ1N.MQUNSUBSCRIBE
- DB2MQ2N.GETCOL
- DB2MQ2N.MQPUBLISH
- DB2MQ2N.MQREAD
- DB2MQ2N.MQREADALL
- DB2MQ2N.MQREADALLCLOB
- DB2MQ2N.MQREADCLOB
- DB2MQ2N.MQRECEIVE
- DB2MQ2N.MQRECEIVEALL
- DB2MQ2N.MQRECEIVEALLCLOB
- DB2MQ2N.MQRECEIVECLOB
- DB2MQ2N.MQSEND
- DB2MQ2N.MQSUBSCRIBE
- DB2MQ2N.MQUNSUBSCRIBE

An application programming default value

The following application programming default value is removed in DB2 11:

CHARSET

ENABLE and DISABLE (REMOTE) REMOTE (location-name,…,<luname>,…)

In DB2 11, you cannot use the BIND PACKAGE options ENABLE and DISABLE (REMOTE) REMOTE (location-name,…,<luname>,…) to enable or disable specific remote connections. You can use the ENABLE(REMOTE) or DISABLE(REMOTE) options to enable or disable all remote connections.

Sysplex query parallelism

In DB2 11, Sysplex query parallelism is no longer supported. Packages that used Sysplex query parallelism in releases before DB2 11 use CPU parallelism in DB2 11.

DSN1CHKR utility

In DB2 11, the DSN1CHKR utility is no longer supported. The DSN1810I and DSN1816I messages are issued when the DSN1CHKR utility is invoked.

12.7 Controlling application compatibility

Requirements coming from SQL standard compliance and completion of support for new functions produce changes that might impact the compatibility of existing applications.

We look at a pervasive example of incompatibility in DB2 10 and provide an overview of the application compatibility support in DB2 11.

- Example of DB2 10 application compatibility
- Overview of application compatibility in DB2 11

12.7.1 Example of DB2 10 application compatibility

One example for an incompatible change in DB2 10 was the changed results of a CHAR built-in scalar function. V9 result for CHAR was not consistent with the result for VARCHAR and CAST of decimal data types.

The problem that was raised for those functions was that leading zeroes were no longer returned when there is a decimal point. Though the functions were now working as designed to conform to SQL standards, this is an incompatible change if the applications rely on the leading zeros.

Example 12-13 shows the result of the implicit casting of a decimal value using the CHAR built-in scalar function in DB2 9. Notice that the leading zeros are included in the result in column DEC2CHAR but not in DECVARCHAR.

Example 12-13 V9 result of implicit cast of decimal using CHAR function

```
SELECT CHAR ( DECIMAL(00123.45,7,2) ) AS DEC2CHAR
, VARCHAR ( DECIMAL(00123.45,7,2) ) AS DEC2VARCHAR
FROM SYSIBM.SYSDUMMY1 ;
---------+---------+---------+---------+---------+---------+---
DEC2CHAR DEC2VARCHAR
---------+---------+---------+---------+---------+---------+---
00123.45 123.45
DSNE610I NUMBER OF ROWS DISPLAYED IS 1
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
```

The incompatible change in DB2 10 is shown in Example 12-14. The same SELECT in DB2 10 shows that the result of CHAR is consistent with what VARCHAR returns.

Example 12-14 V10 result of implicit cast of decimal using CHAR function

```
SELECT CHAR ( DECIMAL(00123.45,7,2) ) AS DEC2CHAR
, VARCHAR ( DECIMAL(00123.45,7,2) ) AS DEC2VARCHAR
FROM SYSIBM.SYSDUMMY1 ;
---------+---------+---------+---------+---------+---------+-----
DEC2CHAR   DEC2VARCHAR
---------+---------+---------+---------+---------+---------+-----
123.45     123.45
DSNE610I NUMBER OF ROWS DISPLAYED IS 1
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
```

The explicit **CAST** of a 00123.45 decimal value to **CHAR** or **VARCHAR** always returned the 123.45 character string.

The change to be consistent in DB2 10 caused some applications to be incompatible. DB2 10 introduced a **BIF_COMPATIBILITY** system parameter. If the **BIF_COMPATIBILITY** subsystem parameter is set to **V9_DECIMAL_VARCHAR**, it reverts the result to how it looked before migrating to DB2 10. Another way to bring back the DB2 9 behavior was at the **SYSCOMPAT_V9** package-level setting to beginning of **PATH BIND** option or in **CURRENT PATH**.

Because at some point you should have adjusted your applications to the SQL Standard, IFCID 366 was also introduced. You can use IFCID 366 to report applications that use the build-in scalar function **CHAR** with a decimal value. The trace record is written out once per thread for a particular SQL statement. Thus, this trace record can help identify which applications need to be changed to support the new behavior.

Note: If an index expression is created with the **CHAR BIF** in the index key, the trace is written during the execution of the **INSERT SQL** statement that inserts into the index. Likewise, for a materialized query table, the trace record is written on **REFRESH TABLE**.

Start the trace by using the following DB2 command:

```
-START TRACE(P) CLASS(32) IFCID(366)
```

A detailed description about which information you can get from turning on the tracing of IFCID 366 is provided later in this section, after a description of the enhancements that are introduced for similar incompatibility situation in DB2 11.

12.7.2 Overview of application compatibility in DB2 11

Sometimes incompatible changes cannot be avoided when SQL functionality is changed in a new DB2 release. DB2 11 helps you decide when your applications are ready for new SQL functionality.

You can influence the availability of new SQL functions after you are in NFM in the following ways:

- A new system parameter providing the default **BIND** option

  New system parameter **APPLCOMPAT** is introduced to support the concept of V10 Application Compatibility. Acceptable values are `V10R1` and `V11R1`. When you migrate a DB2 subsystem, the setting defaults to `V10R1`. When you install a new DB2 subsystem, it defaults to `V11R1`.

`V10R1`	The default **BIND** option is V10R1 compatibility behavior.
`V11R1`	The default **BIND** option is Vd11R1 compatibility behavior. This value is allowed in only NFM.

Attention: Even if you are still in DB2 11 compatibility mode, the assembly of your **DSNZPARM** works fine if you set the value to V11R1, which you are supposed to use only after you are in NFM.

Also, activating the changed **DSNZPARM** by using the **-SET SYSPARM** command or by restarting your DB2 subsystem works fine. You will not see any indication that the system parameter is set to a “wrong” setting in the sense that you cannot use the V11R1 while in CM.

If you try to **BIND** a package without specifying anything for **APPLCOMPAT** on the **BIND** statement, DB2 uses whatever is set in **APPLCOMPAT** system parameter and this would be V11R1 in this scenario. As a consequence, the **BIND** fails with the following error message:

```
DSNT225I  -DBOB BIND ERROR FOR PACKAGE DBOB.DSNESPCS.DSNESM68 APPLCOMPAT(V11R1)
OPTION IS NOT SUPPORTED
DSNT233I  -DBOB UNSUCCESSFUL BIND FOR PACKAGE =
DBOB.DSNESPCS.DSNESM68.(UK92200)
```

- **BIND/REBIND** options for packages

  The **APPLCOMPAT BIND** option specifies the package compatibility level behavior for static SQL. The acceptable values and meanings are the same as described for the system parameter.

  Use Table 12-9 to determine defaults that apply if you do not specify the **APPLCOMPAT** keyword on the **BIND** or **REBIND** statements.

Table 12-9 APPLCOMPAT defaults for BIND

Process	Default value
BIND PLAN	N/A
BIND PACKAGE	The value of subsystem parameter APPLCOMPAT
REBIND PLAN	N/A
REBIND PACKAGE	Existing value. If there is no existing value, the APPLCOMPAT subsystem parameter is used.
REBIND TRIGGER PACKAGE	Existing value. If there is no existing value, the APPLCOMPAT subsystem parameter is used.

- Special Register for Dynamic **SQL (CURRENT APPLICATION COMPATIBILITY)**

  The **CURRENT APPLICATION COMPATIBILITY** specifies the compatibility level support for dynamic SQL.

  The data type is **VARCHAR(10)**.

  The initial value of **CURRENT APPLICATION COMPATIBILITY** is determined by the value of the **APPLCOMPAT** bind parameter for the package. The initial value of **CURRENT APPLICATION COMPATIBILITY** in a user-defined function or stored procedure is inherited from the value of bind option **APPLCOMPAT** for the user-defined function or stored procedure package

  Set the value with the **SET APPLICATION COMPATIBILITY** statement.

When your DB2 environment is migrated to NFM you can run applications with the features and behavior of either previous versions or the current version. For static SQL, the behavior is determined by application compatibility value of a package. For dynamic SQL, the behavior is determined by the **APPLICATION COMPATIBILITY** special register. If no application compatibility

value is set, then the default value is determined by the **APPLCOMPAT** subsystem parameter. The default **APPLCOMPAT** value for a new installation is set to the current DB2 version. The default **APPLCOMPAT** value for a migrated environment is set to the previous DB2 version.

Attention: If you get an error testing DB2 11 DML in NFM, double check the setting of **CURRENT APPLICATION COMPATIBILITY** for dynamic SQL, and **APPLCOMPAT** bind option for static SQL.

APPLCOMPAT = V10R1

When you set the application compatibility value to V10R1, applications that attempt to use functions and features that are introduced in DB2 11 or later might behave differently or receive an error.

When your DB2 11 environment is migrated to NFM, you can run individual applications with some of the features and behavior of DB2 10. If application compatibility is set to V10R1 and you attempt to use the new functions of a later version, SQL might behave differently or result in a negative SQLCODE, such as SQLCODE -4743.

A migrated DB2 11 environment in conversion mode can have only applications that are bound with V10R1 application compatibility. This behavior ensures that fallback to a previous version of DB2 is successful.

Table 12-10 shows many of the features and functions that are controlled by application compatibility, and the results if you specify V10R1.

You might want to ignore the IFCID information in the third column for now. The contents of the IFCID records are discussed later.

Table 12-10 Behavior of V10R1 application compatibility

Feature or function	Result with V10R1 application compatibility	IFCID 0366 or IFCID 0376 trace function code
Specification of bind option DBPROTOCOL(DRDACBF)	DSNT298I	
A period specification that follows the name of a view in the FROM clause of a query	SQLCODE -4743	
A period clause that follows the name of a target view in an UPDATE or DELETE statement	SQLCODE -4743	
A SET CURRENT TEMPORAL SYSTEM_TIME statement	SQLCODE -4743	
A SET CURRENT TEMPORAL BUSINESS_TIME statement	SQLCODE -4743	
A SET SYSIBMADM.MOVE_TO_ARCHIVE or SET SYSIBMADM.GET_ARCHIVE global variable assignment statement	SQLCODE -4743	
Use of array operations and built-in functions such as: ► Use of the UNNEST collection-derived-table ► Use of the ARRAY_FIRST, ARRAY_LAST, ARRAY_NEXT, ARRAY_PRIOR, ARRAY_AGG, TRIM_ARRAY, CARDINALITY, MAX_CARDINALITY built-in functions A SET assignment-statement of an array element as a target table A CAST specification with a parameter marker as the source and an array as the data type	SQLCODE -4743	

Feature or function	Result with V10R1 application compatibility	IFCID 0366 or IFCID 0376 trace function code
An aggregate function that contains the keyword DISTINCT and references a column that is defined with a column mask	SQLCODE -20478	
A reference to an alias for a sequence object or a public alias for a sequence object	SQLCODE -4743	
Invocation of the SPACE or VARCHAR built-in function when the result is defined as VARCHAR(32765), VARCHAR(32766), or VARCHAR(32767)	No error	1110, 1111
A SELECT with a table function reference that includes a typed correlation clause	SQLCODE -4743	
An implicit insert or update of an XML document node	SQLCODE -20345	1101
A predicate expression with an explicit cast or an operation with an invalid value that does not affect the results of XPath processing	SQLCODE -20345	1102
A CALL statement that specifies an autonomous procedure	SQLCODE -4743	
The lengths of values that are returned from CURRENT CLIENT_USERID, CURRENT CLIENT_WRKSTNNAME, CURRENT CLIENT_APPLNAME, or CURRENT CLIENT_ACCTNG special register are longer than the DB2 10 limits.	The special register values are truncated to the DB2 10 maximum lengths and padded with blanks	1104, 1105, 1106, 1107
How the resource limit facility uses ASUTIME value for nested routines	SQLCODE -905 is issued only when the ASUTIME limit of the top-level calling package is encountered.	1103
A `CAST(string as TIMESTAMP)` specification with an input string of length of 8 or an input string of length 13	An explicit cast specification from string as TIMESTAMP interprets an 8-byte character string as a Store Clock value and a 13-byte string as a GENERATE_UNIQUE value. CAST result might be incorrect.	1109

Attention: APPLCOMPAT(V10R1) is assumed for all static SQL packages bound prior to and in DB2 10.

Important: Static SQL packages, which were last bound prior to V9 are invalidated in conversion mode and go through automatic rebind the first time they are called. If you would like to prevent those automatic rebinds, you can:

- SET system parameter ABIND to NO. If you do this, you must remember that the program would not be able to execute successfully, because it remains invalidated.
- Rebind affected packages while you are still in DB2 10 NFM.

The list of affected packages is one of the reports generated by pre-migration job DSNTIJPM/B as explained in “Premigration checkout job DSNTIJPM” on page 335.

Valid time frame for APPLCOMPAT (V10R1)

As described, APPLCOMPAT(V10R1) is valid in all modes of DB2 11. Which setting is valid in which mode is also summarized in

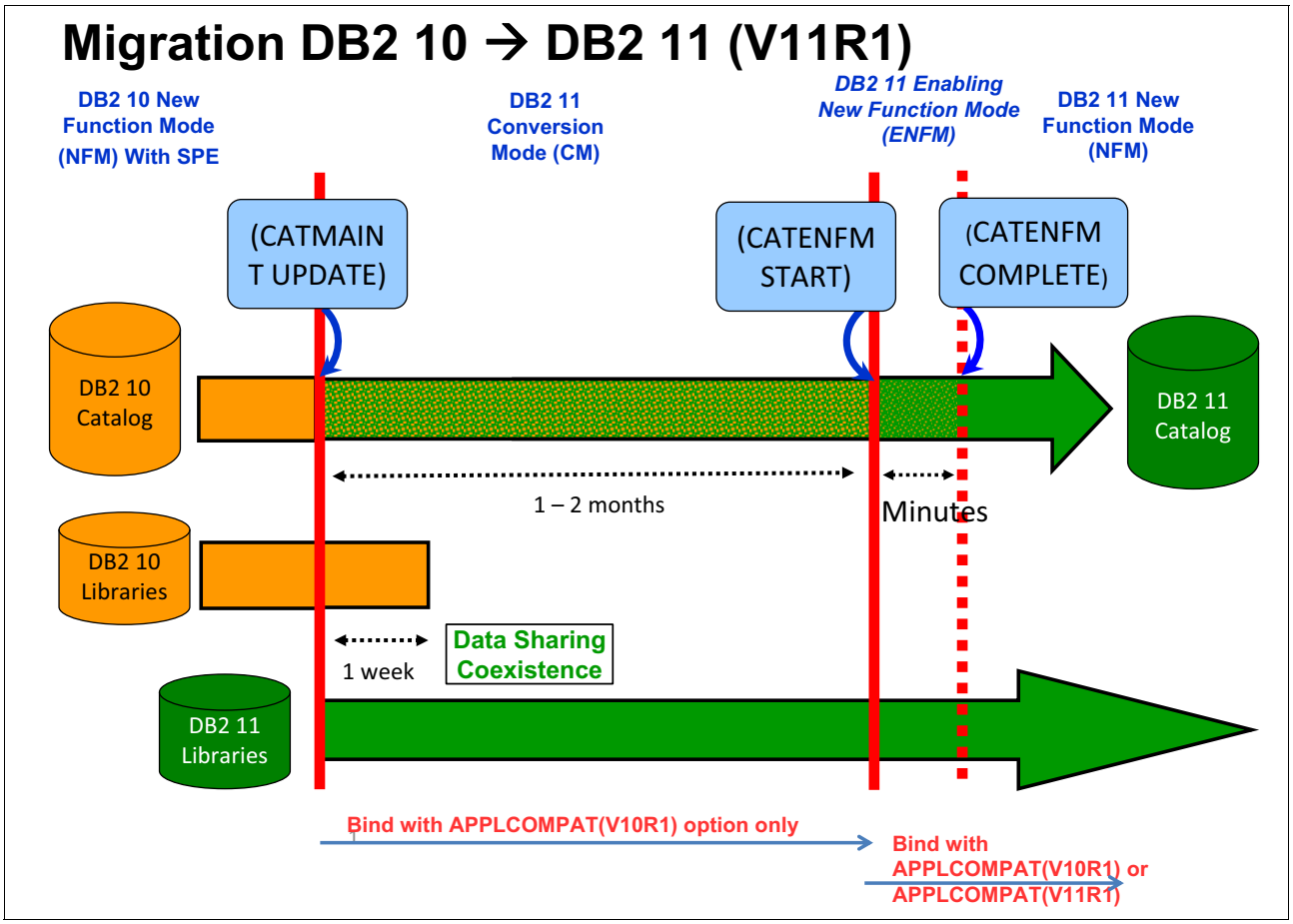


Figure 12-11 V11 modes and APPLCOMPAT(V10R1)

Even though you are just now starting with DB2 11, looking ahead into DB2 11+1, you might ask yourself if V10R1 is still a valid option there. Figure 12-12, shows what you can expect! In DB2 V11 + 1, you are allowed to still stick with APPLCOMPAT(V10R1) in all modes. In addition to that all modes also support APPLCOMPAT (V11R1). Just the new VnnR1 setting is only allowed once you are in DB2 11+1 NFM.

Prior to migrating to DB2 11+1, if you run the pre-migration job **DSNTIJPx**, you can expect to see warnings for all packages, which are at that time bound with APPLCOMPAT(V10R1) and APPLCOMPAT(VnnR1).

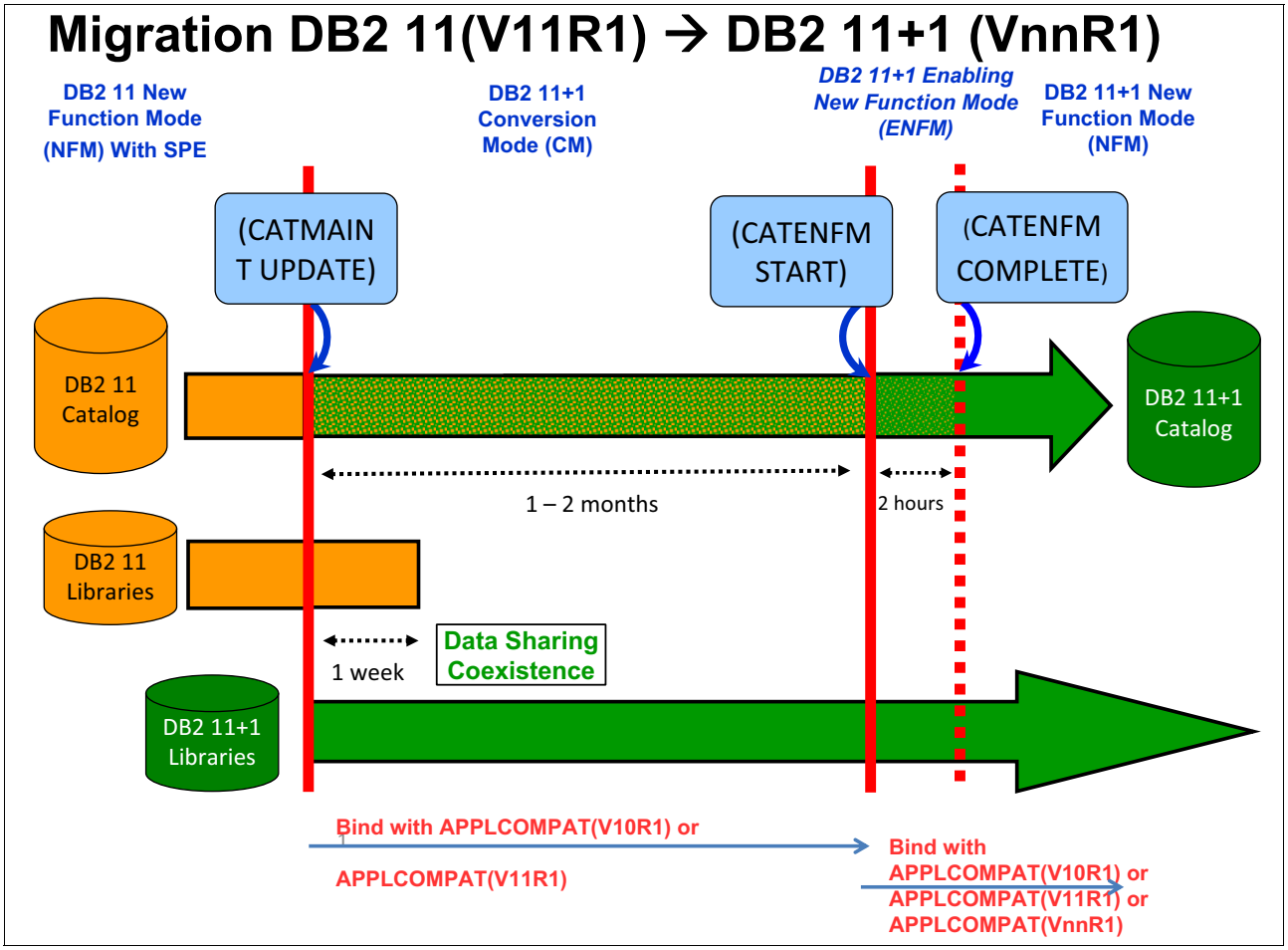


Figure 12-12 V11+1 modes and APPLCOMPAT(V10R1)

Looking ahead to DB2 11 + 2, **DSNTIJPx** pre-migration job acts as listed here:

- Warnings for packages bound with **APPLCOMPAT** for DB2 11 + 1
- Warnings for packages bound with **APPLCOMPAT** for DB2 11
- Errors for packages bound with **APPLCOMPAT(V10R1)**
  - Packages set as Inoperative
  - No **AUTOBIND** allowed
  - SQL *must* be changed to be valid for DB2 11 or DB2 11 +1 or +2

How to find applications that use incompatible SQL statements?

After this extensive description of the Application Compatibility feature in DB2 11, and after you learned that at the second release past DB2 11 you are no longer allowed to use V10R1, you might ask yourself what IBM does for you to help identify the applications, for which you need to take action changing the used SQL to make them compatible with any subsequent release.

The answer is instrumentation!

IFCID 366 has already been introduced with DB2 10, which at that time primarily was intended to help identify applications which use implicit casting of decimal data using the CHAR function. IFCID 366 reports on packages affected in both modes and dynamic SQL

DB2 11 introduces a second IFCID, IFCID 376. IFCID 376 is a roll up of 366. DB2 writes One record for each unique static or dynamic statement

If you want to collect this type of information, you must turn on the tracing those IFCIDs.

Example 12-15 shows the description of the information that you can gain from tracing IFCID 366. Also refer to the description of field QW0366FN. Different finding are categorized in different values. The numbers listed in there also corresponding to the trace function codes listed in Table 12-10 on page 376.

Example 12-15 IFCID 366 record description

```
0366 QW0366               _________________IFCID 0366_________________________
0366 QW0366              IFCID 0366 RECORDS INFORMATION THAT CAN BE USED TO
0366 QW0366              IDENTIFY APPLICATIONS THAT ARE AFFECTED BY INCOMPATIBLE
0366 QW0366              CHANGE
0366 QW0366
0366 QW0366              THIS TRACE RECORD MIGHT CONTAIN INFORMATION ABOUT MULTIPLE
0366 QW0366              INSTANCES OF AN SQL STATEMENT. FOR EXAMPLE, WHEN THE SAME
0366 QW0366              DYNAMIC STATEMENT IS EXECUTED BY SEVERAL THREADS, OR
0366 QW0366              MULTIPLE TIMES BY THE SAME THREAD, MULTIPLE RECORDS ARE
0366 QW0366              WRITTEN.
0366 QW0366
0366 QW0366              THIS RECORD IS FOR SERVICEABILITY ONLY.
0366 QW0366              ------------------------------------------------------
0366 QW0366FN            THIS FIELD CAN HAVE THE FOLLOWING VALUES:
0366 QW0366FN            1: THE DB2 9 FOR Z/OS VERSION OF
0366 QW0366FN               SYSIBM.CHAR(DECIMAL-EXPR) WAS EXECUTED.
0366 QW0366FN            2: THE DB2 9 FOR Z/OS VERSION OF
0366 QW0366FN               SYSIBM.VARCHAR(DECIMAL-EXPR),
0366 QW0366FN               CAST (DECIMAL AS VARCHAR), OR
0366 QW0366FN               CAST (DECIMAL AS CHAR) WAS EXECUTED.
0366 QW0366FN            3: AN UNSUPPORTED CHARACTER STRING REPRESENTATION
0366 QW0366FN               OF A TIMESTAMP WAS USED.
0366 QW0366FN            4: THE DB2 10 FOR Z/OS DEFAULT SQL PATH WAS USED,
0366 QW0366FN               INSTEAD OF THE V11 PATH, WHICH HAS MORE IMPLICIT
0366 QW0366FN               SCHEMAS.
0366 QW0366FN            1101: AN INSERT STATEMENT THAT INSERTS INTO AN XML COLUMN
0366 QW0366FN                  WITHOUT THE XMLDOCUMENT FUNCTION WAS EXECUTED,
0366 QW0366FN                  WHICH GENERATES SQLCODE -20345 ON A DB2 RELEASE
0366 QW0366FN                  PRIOR TO V11, BUT DOES NOT GENERATE AN ERROR
0366 QW0366FN                  STARTING IN V11.
0366 QW0366FN            1102: V10 XPATH EVALUATION BEHAVIOR WAS IN EFFECT, WHICH
0366 QW0366FN                  RESULTED IN AN ERROR. FOR EXAMPLE, A DATA TYPE
0366 QW0366FN                  CONVERSION ERROR OCCURRED FOR A PREDICATE THAT
0366 QW0366FN                  WOULD OTHERWISE BE EVALUATED TO FALSE. STARTING IN
0366 QW0366FN                  V11, SUCH ERRORS MIGHT ARE SUPPRESSED.
0366 QW0366FN            1103: A DYNAMIC SQL STATEMENT USES THE ASUTIME LIMIT THAT
0366 QW0366FN                  WAS SET FOR THE ENTIRE THREAD FOR RLF REACTIVE
0366 QW0366FN                  GOVERNING. FOR EXAMPLE, WHEN A DYNAMIC SQL STATEMENT
0366 QW0366FN                  IS PROCESSED FROM PACKAGE A, IF THE ASUTIME LIMIT
0366 QW0366FN                  WAS ALREADY SET DURING OTHER DYNAMIC SQL PROCESSING
0366 QW0366FN                  FROM PACKAGE B IN THE SAME THREAD, THE SQL FROM
0366 QW0366FN                  PACKAGE A USES THE ASUTIME LIMIT THAT WAS SET DURING
0366 QW0366FN                  THE SQL PROCESSING FROM PACKAGE B. STARTING WITH V11,
0366 QW0366FN                  DYNAMIC SQL FROM MULTIPLE PACKAGES USES THE ASUTIME
```

```
0366 QW0366FN                LIMIT THAT IS SET IN THEIR OWN PACKAGE INFORMATION.
0366 QW0366FN            1104: THE CLIENT_USERID SPECIAL REGISTER WAS SET TO A
0366 QW0366FN                  VALUE THAT IS LONGER THAN THE SUPPORTED LENGTH
0366 QW0366FN                  PRIOR TO V11. THE VALUE WAS TRUNCATED.
0366 QW0366FN            1105: THE CLIENT_WRKSTNNAME SPECIAL REGISTER WAS SET TO
0366 QW0366FN                  A VALUE THAT IS LONGER THAN THE SUPPORTED LENGTH
0366 QW0366FN                  PRIOR TO V11. THE VALUE WAS TRUNCATED.
0366 QW0366FN            1106: THE CLIENT_APPLNAME SPECIAL REGISTER WAS SET TO
0366 QW0366FN                  A VALUE THAT IS LONGER THAN THE SUPPORTED LENGTH
0366 QW0366FN                  PRIOR TO V11. THE VALUE WAS TRUNCATED.
0366 QW0366FN            1107: THE CLIENT_ACCTNG SPECIAL REGISTER WAS SET TO
0366 QW0366FN                  A VALUE THAT IS LONGER THAN THE SUPPORTED LENGTH
0366 QW0366FN                  PRIOR TO V11. THE VALUE WAS TRUNCATED.
0366 QW0366FN            1108: THE CLIENT_USERID, CLIENT_WRKSTNNAME,
0366 QW0366FN                  CLIENT_APPLNAME, OR CLIENT_ACCTG SPECIAL REGISTER
0366 QW0366FN                  WAS SET TO A VALUE THAT IS LONGER THAN THE
0366 QW0366FN                  SUPPORTED LENGTH PRIOR TO V11. THE TRUNCATED VALUE
0366 QW0366FN                  WAS USED FOR A RESOURCE LIMIT FACILITY SEARCH.
0366 QW0366FN            1109: CAST(STRING AS TIMESTAMP) WAS EXECUTED WITH ONE
0366 QW0366FN                  OF THE FOLLOWING TYPES OF INPUT STRINGS:
0366 QW0366FN                  - A STRING OF LENGTH 8, WHICH DB2 TREATED AS A
0366 QW0366FN                    STORE CLOCK VALUE.
0366 QW0366FN                  - A STRING OF LENGTH 13, WHICH DB2 TREATED AS A
0366 QW0366FN                    GENERATE_UNIQUE VALUE.
0366 QW0366FN                  PRIOR TO V11, THIS BEHAVIOR IS INVALID FOR A CAST.
0366 QW0366FN                  IT IS VALID FOR THE TIMESTAMP BUILT-IN FUNCTION
0366 QW0366FN                  ONLY. STARTING IN V11, INPUT TO CAST IS NOT
0366 QW0366FN                  TREATED AS A STORE CLOCK VALUE OR A
0366 QW0366FN                  GENERATE_UNIQUE VALUE.
0366 QW0366FN            1110: THE VALUE OF THE ARGUMENT OF THE SPACE BUILT-IN
0366 QW0366FN                  FUNCTION WAS GREATER THAN 32764.
0366 QW0366FN            1111: THE VALUE OF THE OPTIONAL INTEGER ARGUMENT OF THE
0366 QW0366FN                  VARCHAR BUILT-IN FUNCTION WAS GREATER THAN 32764.
0366 QW0366SN            STATEMENT NUMBER FOR THE QUERY.
0366 QW0366PL            PLAN NAME FOR THE QUERY.
0366 QW0366TS            TIMESTAMP FOR THE QUERY.
0366 QW0366SI            STATEMENT IDENTIFIER.
0366 QW0366TY            STATEMENT INFORMATION:
0366 QW0366DY             X'8000': STATEMENT IS DYNAMIC.
0366 QW0366SC             X'4000': STATEMENT IS STATIC.
0366 QW0366PC_OFF        OFFSET FROM QW0366 TO QW0366PC_LEN.
0366 QW0366PN_OFF        OFFSET FROM QW0366 TO QW0366PN_LEN.
0366 QW0366VL DS         VERSION LENGTH.
0366 QW0366VN DS         VERSION.
0366 QW0366PC_LEN        LENGTH OF THE FOLLOWING FIELD.
0366 QW0366PC_VAR        %U PACKAGE COLLECTION ID.
0366 QW0366PN_LEN        LENGTH OF THE FOLLOWING FIELD.
```

Example 12-16 lists the description of IFCID record 376.

Example 12-16 IFCID 376 record description

```
0376 QW0376              _________________IFCID 0376_____________________________
0376 QW0376              IFCID 0376 RECORDS INFORMATION ABOUT SQL STATEMENTS
0376 QW0376              THAT HAVE POTENTIAL INCOMPATIBLE CHANGES WHEN YOU SWITCH
0376 QW0376              TO NEW APPLICATION BEHAVIOR.
0376 QW0376
0376 QW0376              THIS TRACE RECORD IS SIMILAR TO THE IFCID 0366 RECORD,
0376 QW0376              EXCEPT THAT THIS TRACE RECORD CONTAINS INFORMATION FOR
0376 QW0376              UNIQUE INSTANCES OF SQL STATEMENTS. THIS TRACE
```

```
0376 QW0376              RECORD IS WRITTEN ONCE FOR EACH UNIQUE INSTANCE
0376 QW0376              OF THE FOLLOWING TYPES OF SQL STATEMENTS:
0376 QW0376              - DYNAMIC STATEMENTS IN THE DYNAMIC STATEMENT CACHE
0376 QW0376              - STATIC STATEMENTS THAT WERE BOUND IN VERSION 10
0376 QW0376                NEW-FUNCTION MODE OR LATER
0376 QW0376              FOR STATIC SQL STATEMENTS THAT WERE BOUND BEFORE VERSION
0376 QW0376              10 NEW-FUNCTION MODE, THIS RECORD IS WRITTEN ONCE FOR
0376 QW0376              UNIQUE COMBINATION OF PLAN, PACKAGE ID, AND STATEMENT
0376 QW0376              NUMBER. ON RARE OCCASIONS, MORE THAN ONE TRACE RECORD
0376 QW0376              MIGHT BE WRITTEN.
0376 QW0376
0376 QW0376              THIS RECORD IS FOR SERVICEABILITY ONLY.
0376 QW0376              ------------------------------------------------------
0376 QW0376FN            THIS FIELD HAS THE SAME VALUES AS QW0366.
0376 QW0376SN            STATEMENT NUMBER FOR THE QUERY.
0376 QW0376PL            PLAN NAME FOR THE QUERY.
0376 QW0376TS            TIMESTAMP FOR THE QUERY.
0376 QW0376SI            STATEMENT IDENTIFIER.
0376 QW0376TY            STATEMENT INFORMATION:
0376 QW0376TY             X'8000': STATEMENT IS DYNAMIC.
0376 QW0376TY             X'4000': STATEMENT IS STATIC.
0376 QW0376SE            SECTION NUMBER.
0376 QW0376PC_OFF        OFFSET FROM QW0376 TO QW0376PC_LEN.
0376 QW0376PN_OFF        OFFSET FROM QW0376 TO QW0376PN_LEN.
0376 QW0376VL DS         VERSION LENGTH.
0376 QW0376VN DS         VERSION.
0376 QW0376PC_LEN        LENGTH OF THE FOLLOWING FIELD.
0376 QW0376PC_VAR        %U PACKAGE COLLECTION ID.
0376 QW0376PN_LEN        LENGTH OF THE FOLLOWING FIELD.
0376 QW0376PN_VAR        %U PROGRAM NAME.
```

DB2 catalog support for APPLCOMPAT

A new column APPLCOMPAT has been added to DB2 catalog tables SYSIBM.SYSPACKAGE and SYSIBM.SYSPACKCOPY. Possible values are:

V10R1	SQL statements in the package have V10R1 compatibility behavior.
V11R1	SQL statements in the package have V11R1 compatibility behavior.

13

Performance

DB2 11 focuses on a number of performance benefits, especially in the area of CPU cost reduction, scalability enhancements and user pain points such as providing consistent system and application performance with less need to reorganize objects and with less need for performance tuning.

This chapter describes performance enhancements in DB2 11. Many of these improvements are available by migrating to DB2 11 and rebinding.

This chapter includes the following topics:

- Performance expectations
- System level performance
- Reduced need for REORG
- More opportunities for RELEASE(DEALLOCATE)
- Optimizer enhancements

13.1 Performance expectations

DB2 11 provides many performance improvements. This section discusses the results of IBM's early observations and the feedback from the ESP program.

When reading this section, keep in mind that results can vary, depending on environment conditions. Nevertheless, it is important to realize that most of the storage and CPU improvements available in DB2 11 for z/OS can be achieved in conversion mode (CM) and only after **REBIND**.

Important: Although **REBIND** might not be needed to migrate to DB2 11, **REBIND** is often required to obtain the performance benefits of DB2 11

The following observations are expected to be reported by users when comparing DB2 10 to DB2 11 workloads, after **REBIND**, and under the same working conditions, including equivalent **BIND/REBIND** options, such as **RELEASE**.

OLTP workloads can show 0% to 10% CPU reduction in CM mode after **REBIND**. Results might be better for write intensive workloads. Statements processing large number of columns might show even further CPU reduction. Further improvements are executed for workloads accessing a single or a few table space partitions out of 500 or more partitions and using the **RELEASE(COMMMIT) BIND/REBIND** option.

Data warehousing queries are expected to show from 5% to 40% CPU reduction. Higher improvement can be seen for queries that take advantage of access path improvements in DB2 11 after **REBIND** or **PREPARE**. Better results are expected if the tables being accessed are compressed. Queries with table space scan can show better results. Higher improvement are expected for processes with sort intensive workloads.

Update Intensive Batch are expected to report from 5% to15% CPU reduction, with better results in data sharing environments, especially in New Function Mode (NFM) with **EXTENDED** LRSN format.

Figure 13-1 illustrates the DB2 11 performance expectations per workload type.

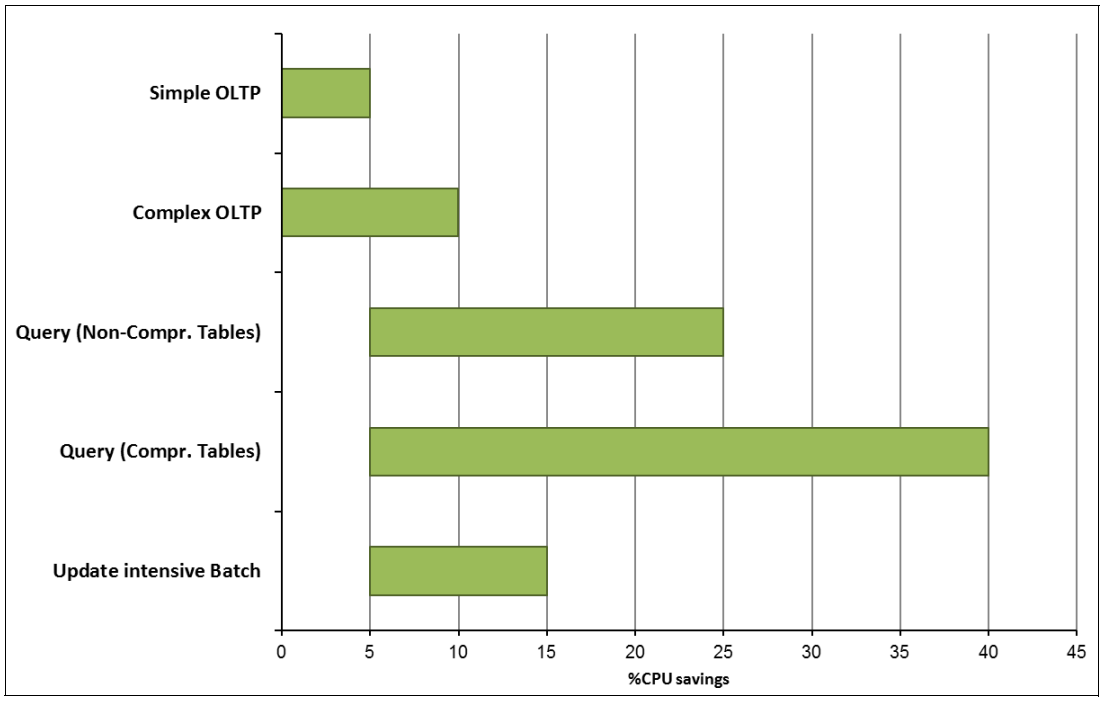


Figure 13-1 DB2 11 performance: CPU changes per workload type

Important: Additional CPU savings might been seen by taking advantage of other DB2 11 capabilities.

As a reference, and for comparison purposes, Figure 13-2 shows the performance expectations published for DB2 10 for z/OS at the equivalent moment in the lifecycle of the database product.

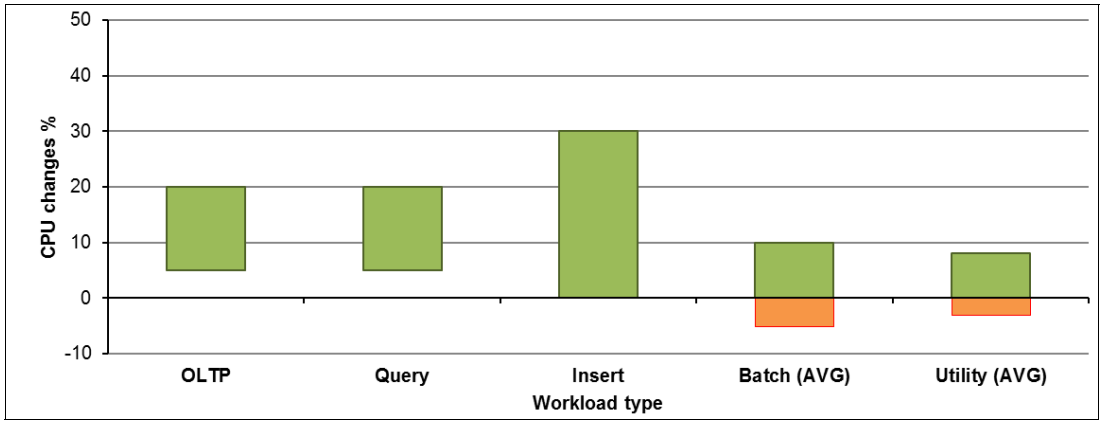


Figure 13-2 DB2 10 performance expectations

Figure 13-3 rearranges the DB2 11 expected values in the same format and scale.

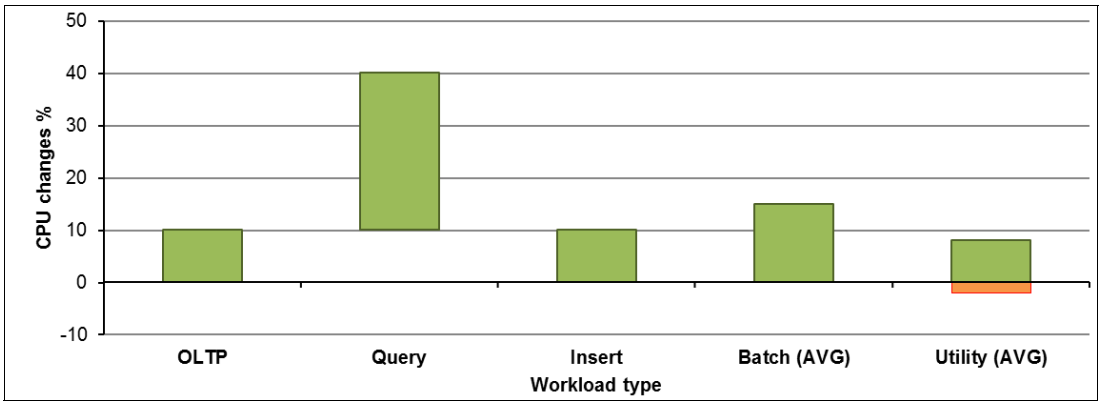


Figure 13-3 DB2 11 performance expectations

These figures allow you to compare the expected performance changes between DB2 10 and DB2 11 for z/OS.

At a glance, DB2 11 for z/OS continues the CPU reduction trend introduced by DB2 10. Users obtain more functionality with less CPU. This fact has the potential to lead to a financial Total Cost of Ownership reduction by means of less CPU associated costs.

The fundamentals for the DB1 CPU reductions are described in the following sections.

13.2 System level performance

There are a number of system level performance enhancements in DB2 11. This section describes the following topics:

- Internal optimization
- Logging
- Synergy with System z
- Buffer management
- Data sharing

13.2.1 Internal optimization

DB2 11 provides performance benefits through some internal optimizations of the DB2 code. These optimizations fall into the following categories:

Customized machine code generation for repeated operations

The customized machine code can provide improved performance for SQL column processing and for RDS sort operations.

Scalability Improvement

z/OS V1.13 supports 64-bit code execution. DB2 takes advantage of this feature by using a 64-bit XProc that is above the bar and some code optimization, which results in a further reduction of DBM1 virtual storage consumption below the bar.

New decompression routine

DB2 11 provides a new decompression routine. This new routine provides a significant CPU reduction to speed up the expansion operation when compressed rows are read.

The new decompression routine is compatible with the existing compression routine. You do not need to take any action to take advantage of this performance feature.

Scalability improvement with large number of partitions

This internal optimization enhancement will provide performance benefits for packages bound with **RELEASE(COMMIT)** and that are accessing partitioned table spaces. This enhancement works with all types of partitioned table spaces: classic partitioned; Universal Table Space (UTS) partition by range; and UTS partition by growth.

The extreme case for biggest performance improvement is found for applications that issue a single **SELECT** statement that touches one partition of a UTS that has 4096 partitions.

13.2.2 Logging

This section describes the two key performance enhancements related to logging in DB2 11.

Large RBA/LRSN support

DB2 11 extends RBA and LRSN values from 6 to 10 bytes. You can find more details about the implementation of this feature in 3.1, “Extended RBA and LRSN” on page 24.

DB2 11 uses the extended RBA and LRSN values internally and converts the logs to basic format in both CM and NFM. You need to run stand-alone utility DSNJCNVT to convert the BSDS to the extended format. Conversion to the new BSDS format is required to write new format log records and remove the 6-byte RBA and LRSN limits.

Your application objects (tables and indexes) also eventually need to be converted to a new page format to accommodate the larger value. Until the BSDS and your application objects are converted to **EXTENDED** format, you can expect some conversion overhead associated with the extended log RBA and LRSN values.

If you are running a data sharing environment, after you have completed the conversion to the extended LRSN values, there will be no more overhead associated with LRSN spin, which can provide a significant CPU reduction in batch write operations in data sharing. See 5.9, “Log record sequence number spin avoidance” on page 95 for more details.

Log buffers in 64 bit common

DB2 11 provides a reduction in CPU cost by removing cross address space operations for logging activity. There is an optional 1 MB of storage for log buffers, if the LPAR is configured with a large frame area (LFAREA). You can find more details about the **LFAREA** feature of the zEC12 in 2.1.3, “zEC12 hardware features” on page 8. This enhancement provides a significant CPU reduction for update intensive batch jobs.

13.2.3 Synergy with System z

There are a number of enhancements in DB2 10 and DB2 11 that take advantage of features in the System z hardware and operating system. These features are discussed in more detail in Chapter 2, “Synergy with System z” on page 7.

This section describes performance benefits of the synergy between DB2 and System z.

More usage of large page frames

The large frame area (LFAREA) of storage on the zEC12 hardware is used for fixed 1 MB large page frames and fixed 2 GB large page frames. Log buffers can now take advantage of 1 MB fixed page frames. These changes make more frame sizes available for DB2 buffer pools. With DB2 11 and zEC12 hardware, buffer pools can utilize 2 GB fixed page frames for additional CPU reduction.

Figure 13-4 shows the different combinations of frame size and page size that are supported in DB2 11 and prior versions and on the level of hardware. You can see that with 1 MB page fixed frames on DB2 10 and 11 on z10 and later hardware, or with 2 GB page fixed frames on DB2 11 with zEC12 hardware, you can benefit from CPU reductions during I/O processing and from an improved hit rate on the translation look-aside buffer (TLB), which is used to translate a virtual address to a physical address.

DB2 Buffer Pool - Frame size

Frame size	Page fix	Supported DB2	H/W Requirement	Benefit
4K	NO	All	N/A	Most flexible configuration
4K	YES	All	N/A	CPU reduction during I/O
1M	NO	DB2 11	zEC12 and Flash Express	CPU reduction from TLB hit
1M	YES	DB2 10 above	z10 above LFAREA 1M=xx	CPU reduction during I/O, CPU reduction from TLB hit
2G	YES	DB2 11	zEC12 LFAREA 2G=xx	CPU reduction during I/O, CPU reduction from TLB hit

Figure 13-4 DB2 buffer pool frame size options

Flash Express

The zEC12 supports an optional hardware feature called Flash Express memory cards. You can use this feature to improve the performance when accessing buffer pool control blocks and the performance of executing the DB2 code. You can find more details about Flash Express in 2.1.3, “zEC12 hardware features” on page 8.

More zIIP Exploitation

DB2 11 will further use the zIIP specialty processors by making additional processes available for zIIP redirect. Those processes are described in 2.3, “Using zIIP speciality processors” on page 14.

13.2.4 Buffer management

DB2 11 provides the following performance enhancements for buffer pool processing.

Faster buffer pool allocation

In DB2 11 it is significantly faster to allocate large buffer pools, such as ones that are 5 to 10 GB or larger. In DB2 10, buffer pool storage was allocated as it was needed. In DB2 11, there is virtual allocation of the buffer pool with the defined size, but real storage allocation is done as needed.

Improved buffer pool metrics

DB2 classifies Getpages as either random or sequential, and DB2 uses the VPSEQT buffer pool parameter to protect random pages from being overrun by sequential pages. DB2 11 enforces a more rigorous alignment between how the Getpages are classified and whether or not DB2 has prefetched the pages. For example, if dynamic prefetch was used, the Getpages will now be classified as sequential. Also, when DB2 is using list prefetch to read a disorganized index or to read pages in a RID list, the Getpages will not be classified as sequential. Utilities that use format writes will also classify the pages as sequential. The first consequence of this change is that the random buffer hit is a more accurate measure of buffer pool performance. A second consequence is that sequential synchronous I/Os can be used to identify the fact that either DB2 failed to prefetch those pages, or the pages were prefetched and then stolen prior to the getpages, which was a problem that was difficult to detect with prior DB2 versions. Buffer tuning is never easy, but DB2 11 makes it easier.

In addition, DB2 now reports the length of the sequential LRU chain. This support was retrofitted to DB2 10 in PM70981. Using this statistic, you can more easily judge the degree to which prefetch activity is affecting the buffer pool. You can judge from this statistic whether lowering VPSEQT will help to increase the buffer pool hit ratio. (It will not be as long as the number of sequential buffers is less than `VPSEQTxVPSIZE`). Conversely, just because the length of the sequential LRU chain is less than `VPSEQTxVPSIZE` does not mean that the prefetch activity is not affecting the random buffer hit ratio. As always, remember that lowering VPSEQT might introduce synchronous sequential I/Os if you do not have enough sequential buffers to support the prefetch activity in your system.

More MRU usage for utilities

DB2 9 and DB2 10 provided reductions in CPU for utility processing due to changes in buffering from Least Recently Used (LRU) to Most Recently Used (MRU) for the **`COPY`** utility. DB2 11 further improves performance by expanding the MRU buffering to the **`UNLOAD`** utility and to the **`RUNSTATS`** utility for table spaces and indexes. In addition, the MRU processing will also be used for the **`UNLOAD`** phase of the following utilities:

- **`REORG TABLESPACE`**
- **`REBUILD INDEX`**
- **`CHECK INDEX`** and **`DATA`**

13.2.5 Data sharing

DB2 11 provides the following performance enhancements for data sharing environments.

Reduction of log force write during tree structure modification

DB2 provides a throughput improvement for **`INSERT`** and **`DELETE`** workloads by reducing the number of log force writes per index modification event. This results in a reduction in elapsed time and a minor CPU time reduction. This enhancement also provides log disk I/O relief.

Data sharing availability and performance improvements

DB2 11 provides the following availability and performance improvements for data sharing:

- CASTOUT performance improvement
- GBP write around
- CF DELETE NAME enhancement
- Internal resource lock manager (IRLM) enhancements

All of these enhancements are described in detail in Chapter 5, “Data sharing” on page 85.

13.3 Reduced need for REORG

The hardware enhancements that provide a foundation for reducing the need for **REORG**s are described in 2.4, “Reduced need for REORG” on page 15. However, DB2 10 for z/OS also decreased the need for **REORG**s with the following additional enhancements:

- List prefetch to perform disorganized index scan

  DB2 9 RID list scans can benefit from the new hardware features, DB2 10 can also benefit from these hardware features when it scans a disorganized index. See *GPFS in the Cloud: Storage Virtualization with NPIV on IBM System p and IBM System Storage DS5300*, REDP-4682. The I/O time to read a disorganized index is still greater than the I/O time to read an organized index, but remember that the I/O is asynchronous. If the index scan is CPU intensive, then organizing the index will not reduce the elapsed time to scan the index at all.

- Row level sequential detection (RLSD)

  RLSD makes sequential detection more robust as the cluster ratio drops below 100%, ensuring that DB2 uses dynamic prefetch for clustered pages and limiting the synchronous I/O to unclustered pages.

As DB2 continues to move in the direction towards reduced **REORG**s, keep in mind that the goal is not to completely eliminate all **REORG**s. For example, the requirements for materializing pending **ALTER**s are not going away. However, the performance gap between organized and disorganized data should shrink and the tendency to run unnecessary **REORG** should be reduced.

Some misconceptions abound about the value of redistributing or re-establishing free space. If you never reorganize an index and randomly insert keys into it, it will tend to have about 25% free space. If you reorganize the index and use PCTFREE 10, you will shrink the index and increase the likelihood of more index splits. Thus, do not try to use **REORG** for the purpose of avoiding index splits.

The effect of clustering is also often misunderstood. The benefit of clustering is normally associated with the performance of a range scan, where the cluster index is used to determine a range of pages to read. If **REORG** can shrink the number of **GETPAGE**s, range scan performance might improve, which is often the case. However, when your query uses a screening predicate, it is often true that **REORG** does not reduce the number of **GETPAGE**s for such queries. If **REORG** does not reduce the number of **GETPAGE**s, it probably is not improving the performance. Thus, the need for **REORG** depends a lot on the types of queries that you run.

DB2 11 is the next step in the evolution towards meeting the goal of reducing the need for Reorgs. The following features of DB2 11 move in this direction and provide a more consistent performance:

- Asynchronous removal of pseudo-deleted indexes
- Indirect reference avoidance

In addition to reducing the need for **REORG**s, DB2 11 also improves the performance of the switch phase of **REORG**, reducing the amount of time during the switch phase that the objects are unavailable to the application. More about the switch phase is discussed in 11.1.2, “SWITCH phase impact reduction” on page 273.

13.3.1 Asynchronous removal of pseudo-deleted indexes

This enhancement can reduce the size of some indexes, which can improve SQL performance and reduce the need to run the **REORG INDEX** utility.

Prior to DB2 11, when rows are deleted, index entries are not physically deleted unless the delete operation has exclusive control over the index page set. Instead, these index entries that correspond to deleted rows are marked as pseudo-deleted. These index entries are called pseudo-deleted index entries.

Pseudo-empty index pages are pages that contain only pseudo-deleted index entries. DB2 attempts to clean up pseudo-empty index pages as part of the SQL **DELETE** processing. However, if some of the pseudo-deleted entries in the page are not committed during the SQL **DELETE** processing, cleanup cannot be performed. Therefore, some pseudo-empty pages are likely not cleaned up. Index entries are only marked pseudo-deleted to handle a combination of other processes using index access and the potential roll back of deleted rows.

Subsequent searches continue to access these pseudo-deleted entries, which can gradually degrade performance as more rows are deleted. The pseudo-deleted index entries can also result in time-outs and deadlocks for applications that insert data into tables with unique indexes.

A large amount of update activity over a period of time can provide for inconsistent performance and the need to **REORG** your tables and indexes regularly to restore desired performance. The average transaction response time increases throughout the week until a **REORG** is done.

Figure 13-5 shows an example of the pseudo-delete process. The index entries for rows 2 and 4 both have a value of *DBA* for the RESP column and are marked as pseudo-deleted, as denoted by the *PD* in the figure.

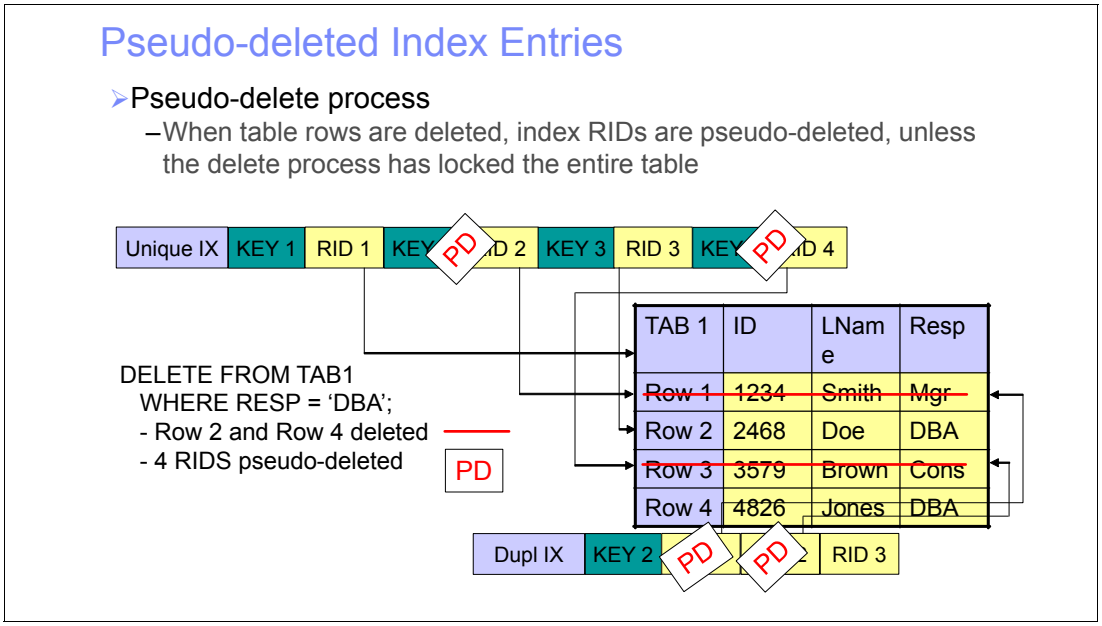


Figure 13-5 The pseudo-delete process

There is a performance impact for maintaining index pseudo delete entries. SQL operations such as **SELECT**, **FETCH**, **UPDATE**, or **DELETE** that require an index search can result in more getpages and more lock requests to access the required data. **INSERT**, **UPDATE**, and **DELETE** operations might see concurrency issues. There can be collisions with committed pseudo-deleted index entries. Also, RID reuse by an **INSERT** statement following a **DELETE** statement can cause a deadlock. Frequent execution of the **REORG INDEX** utility is required to reduce the impact of the pseudo-deleted index entries.

In DB2 11, in addition to the cleanup that was previously done, DB2 autonomically deletes pseudo-empty index pages and pseudo deleted index entries independently of the SQL **DELETE** transaction.

Note: When the system has been configured with one or multiple zIIP processors, this automated cleanup function runs under enclave service request blocks (SRBs) that are zIIP-eligible.

Index cleanup is performed only on the indexes that have been opened for **INSERT/DELETE/UPDATE** by other DB2 processes. The presence of the pseudo deleted entries can be detected by SQL queries or **INSERT/DELETE/UPDATE** processes. There can be large number of pseudo deleted entries in an index, but if this index is not already opened for **INSERT/DELETE/UPDATE**, the cleanup does not happen. The cleanup rate depends on several factors such as the rate that the pseudo deleted entries are generated, the number of threads allowed to run cleanup concurrently, and the commit frequency of the unit of work which generates the pseudo deleted index entries.

This function is designed to remove committed pseudo-deleted entries from the indexes with minimal or no disruption to other concurrent DB2 work in the system.

INDEXCLEANUP_THREADS subsystem parameter

DB2 11 provides an automated cleanup function that is completed under system tasks running as enclave SRBs. The new DB2 system parameter **INDEX_CLEANUP_THREADS** determines the number of threads that are allowed to work on the cleanup of pseudo deleted index entries. You can specify any value between 0 and 128. If you set this subsystem parameter to 0, this means that you do not want any additional index cleanup to occur.

If system parameter **INDEX_CLEANUP_THREADS** has a value greater than zero, DB2 checks Real Time Statistics (RTS) information to identify the indexes with a large number of pseudo-deleted entries or pseudo empty pages. If the identified indexes have already been opened for update, then daemon code schedules a cleanup on these indexes. There is a parent daemon thread per DB2 member, which checks the RTS by looping through RTS blocks for all objects in the system, and identifies the candidate indexes for cleanup. Then the parent daemon thread dispatches child daemon threads (up to the number defined in **INDEX_CLEANUP_THREADS**) to perform the cleanup function. Each child thread works on one index at a time.

The RTS information is checked periodically to identify the indexes with the most pseudo-deletes. There is a limited number of threads doing cleanup (the default is 10, the maximum is 128). The index can only be cleaned up when a thread is freed up, and the index candidates are sorted based on the number of pseudo-deletes, so the ones with the most pseudo-deletes get cleaned up first.

Figure 13-6 shows the DB2 11 pseudo-delete cleanup process.

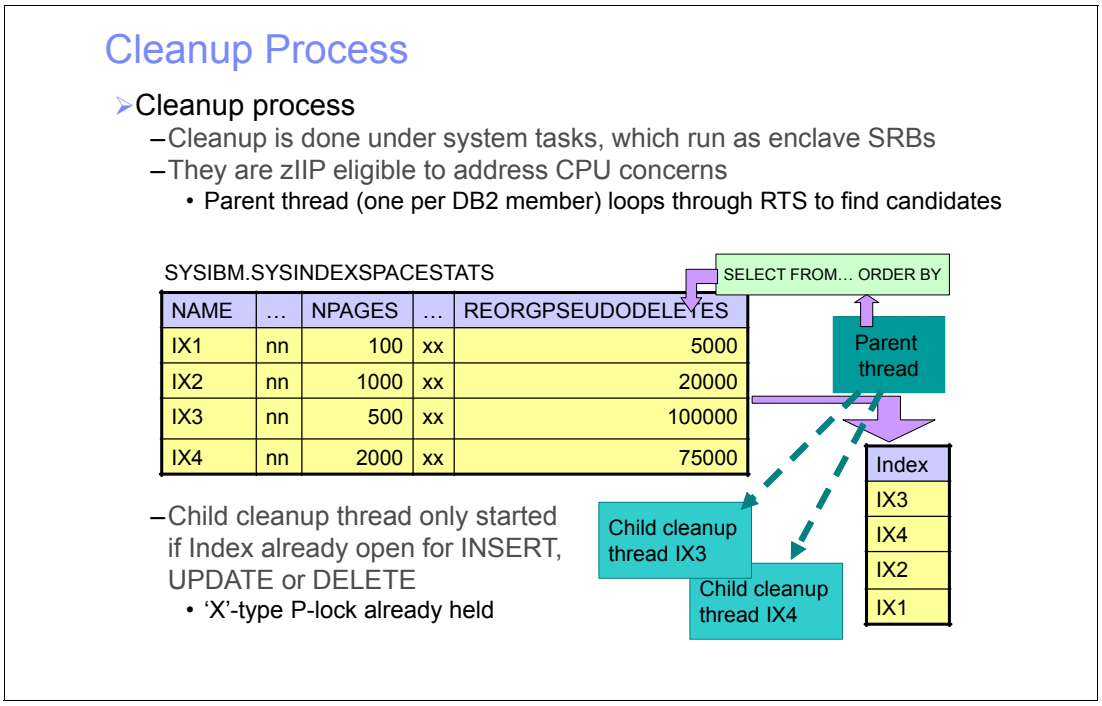


Figure 13-6 Automated pseudo-delete cleanup process

The automated cleanup of pseudo deleted entries in DB2 11 cleans up both pseudo empty index pages and pseudo deleted index entries. The benefits of this process are that it reduces the impact of pseudo delete entries and it reduces the need to run the **REORG INDEX** utility. The potential concerns about the automated clean up are possible CPU overhead, disruption to other concurrent threads and an increase in log volume introduced by the cleanup process.

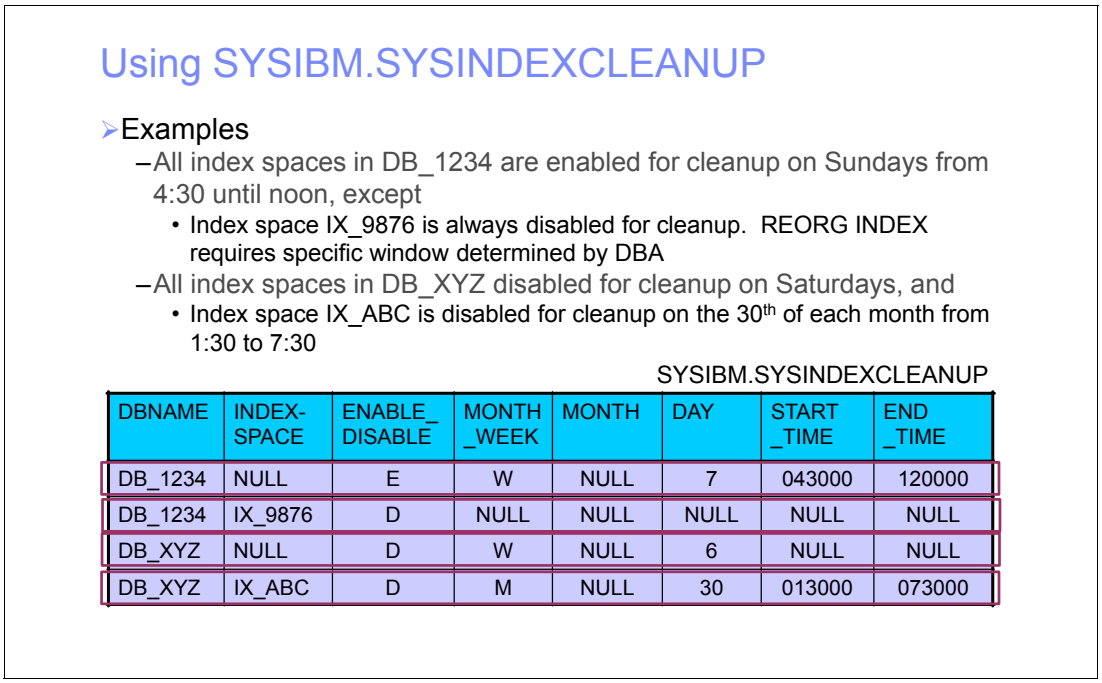
Potential disruption introduced by these concerns can be minimized by managing the number of cleanup threads through the value you choose for system parameter **INDEX_CLEANUP_THREADS**. In data sharing, each member of the group can use a different setting for **INDEX_CLEANUP_THREADS**.

New catalog table SYSIBM.SYSINDEXCLEANUP

You can also control the cleanup function on the object level by inserting rows into the new **SYSIBM.SYSINDEXCLEANUP** catalog table. You can use this table to specify the time when indexes are subject to cleanup. It indicates when and which indexes are enabled or disabled for cleanup. The catalog table includes the following information for use in the cleanup process:

- Name of databases and indexes
- Cleanup enabled or disabled
- Day of week or day of month
- Start time and end time

Figure 13-7 shows an example of using the **SYSIBM.SYSINDEXCLEANUP** catalog table to control the cleanup of pseudo-deleted index entries for two databases. If the **SYSIBM.SYSINDEXCLEANUP** table is not accessible, index cleanup is disabled. Because the data is stored in a catalog table, a single set of values exists in each row for all members in a data sharing group, as opposed to the **INDEX_CLEANUP_THREADS** system parameter, which can have a separate value for each member.



DBNAME	INDEX-SPACE	ENABLE_DISABLE	MONTH_WEEK	MONTH	DAY	START_TIME	END_TIME
DB_1234	NULL	E	W	NULL	7	043000	120000
DB_1234	IX_9876	D	NULL	NULL	NULL	NULL	NULL
DB_XYZ	NULL	D	W	NULL	6	NULL	NULL
DB_XYZ	IX_ABC	D	M	NULL	30	013000	073000

Figure 13-7 Example of using SYSINDEXCLEANUP for cleanup of pseudo-deleted entries

Use the catalog table as an exception only, for those cases when you know there is a disruption.

DB2 provides instrumentation for the cleanup by introducing IFCID 377, which is written once per index page being cleaned up.

Table 13-1 shows the layout of the new catalog table with a short description of its columns.

Table 13-1 SYSIBM.SYSINDEXCLEANUP

Column name	Description
DBNAME	The name of the database that contains the index space.
INDEXSPACE	The name of the index space
ENABLE_DISABLE	Specifies whether the row enables or disables cleanup for the specified index space. 'E' Enabled 'D' Disabled
MONTH_WEEK	Indicates the meaning of the value of the DAY column: 'M' The value indicates the day of the month. 'W' The value indicates a day of the week.
MONTH	The month in which the time window applies. For example a 1 value indicates January and a 12 value indicates December. If this column contains NULL, the time window applies to all months. If the value of the MONTH_WEEK column is 'W', this value must be NULL.
WEEK	The day of the month or the day of the week for which the time window applies, as specified by the value of the MONTH_WEEK column. For example, if MONTH_WEEK='W', a 1 value indicates Monday and 7 indicates Sunday. If the value of this column is NULL, the time window applies to every day of the month or every day of the week.
START_TIME	The local time at the beginning of the time window specified by the row. When this column contains a null value, the row applies at all times on the specified days. This column must contain NULL if the END_TIME column contains NULL.
END_TIME	The local time at the end of the time window specified by the row. When this column contains a null value, the row applies at all times on the specified days. This column must contain NULL if the START_TIME column contains NULL.

When there is an index that needs to be cleaned up, DB2 checks the **SYSIBM.SYSINDEXCLEANUP** catalog table to see if entries in this table allow this index to be cleaned up at the current time.

If the **SYSIBM.SYSINDEXCLEANUP** catalog table is not accessible, index cleanup is disabled, no index can be cleaned up in the system.

Each row in the **SYSIBM.SYSINDEXCLEANUP** catalog table has database name (**DBNAME**) and index space name (**INDEXSPACE**) information. **DBNAME** and **INDEXSPACE** columns are nullable columns.

There is also time window information specified in the **SYSIBM.SYSINDEXCLEANUP** catalog table.

The value of the **ENABLE_DISABLE** column indicates whether the cleanup is enabled (value E) or disabled (value D) for the specified index space during the time window. In data sharing, the rows in the **SYSIBM.SYSINDEXCLEANUP** catalog table apply to all DB2 members.

For the DB2 members with **INDEXCLEANUP_THREADS** set to a non zero value, if the **SYSIBM.SYSINDEXCLEANUP** table is empty, index cleanup is enabled for all indexes on the system. In order to disable the cleanup for certain indexes during certain time period, you can insert rows into the **SYSIBM.SYSINDEXCLEANUP** catalog table to control the cleanup at the object level.

There are three levels of control that can be achieved with different settings on **DBNAME** and **INDEXSPACE** columns. When **DBNAME** and **INDEXSPACE** columns are both NULL, the row applies to all indexes on the system, it is defined on system level. When **DBNAME** is not NULL but **INDEXSPACE** is NULL, the row applies to all the indexes in the specified database, it is defined on database level. When the **DBNAME** and **INDEXSPACE** names are both not NULL, the row applies to a single index, it is defined on index level. If the **DBNAME** column has a NULL value, but **INDEXSPACE** column has a not NULL value, the row is not valid.

If there are multiple rows applicable to the same index a nd these rows cover overlapping time window, but with conflicting information in the **ENABLE_DISABLE** column, the rows defined on index level override the rows defined on database level, which in turn override the rows defined on system level. If these rows are defined on same level, the index cleanup function is disabled during the overlapping time window for the specified indexes.

To minimize the performance impact, the checking of the **SYSIBM.SYSINDEXCLEANUP** catalog table is no real-time. Instead there is up to 10 minutes delay between the time a row is inserted into the **SYSIBM.SYSINDEXCLEANUP** catalog table and the time that DB2 checks the newly inserted row. As a consequence, plan ahead of time when using this table to control the index cleanup.

Tip: If you need to turn off the index cleanup immediately, you can set the **INDEXCLEANUP_THREADS** system parameter to zero and activate the new setting using the **-SET SYSPARM** DB2 command.

Use **SYSIBM.SYSINDEXCLEANUP** catalog table only as an exception table when the default behavior is not desired. Make sure that you keep this table at a reasonable size.

The following examples show the use of the **SYSIBM.SYSINDEXCLEANUP** catalog table.

Example 13-1 shows how to enable the cleanup on all indexes.

Example 13-1 Enable the cleanup on all indexes

```
INSERT INTO SYSIBM.SYSINDEXCLEANUP(DBNAME, INDEXSPACE,
ENABLE_DISABLE, MONTH_WEEK, MONTH, DAY, START_TIME, END_TIME)
values(NULL,NULL,'E', 'W', NULL, NULL, NULL , NULL );
or
keep the table empty
```

Example 13-2 shows how to disable the cleanup on all indexes.

Example 13-2 Disable the cleanup on all indexes

```
INSERT INTO SYSIBM.SYSINDEXCLEANUP(DBNAME, INDEXSPACE,
ENABLE_DISABLE, MONTH_WEEK, MONTH, DAY, START_TIME, END_TIME)
values(NULL,NULL,'D', 'W', NULL, NULL, NULL , NULL );
OR
set subsystem parameter INDEXCLEANUP_THREADS to be zero.
```

Example 13-3 shows how to disable the cleanup on all indexes except on every Saturday and Sunday.

Example 13-3 Disable cleanup on all indexes except on every Saturday and Sunday

```
INSERT INTO SYSIBM.SYSINDEXCLEANUP(DBNAME, INDEXSPACE,
ENABLE_DISABLE, MONTH_WEEK, MONTH, DAY, START_TIME, END_TIME)
values(NULL,NULL,'D', 'W', NULL, 1, NULL , NULL );
```

```
INSERT INTO SYSIBM.SYSINDEXCLEANUP(DBNAME, INDEXSPACE,
ENABLE_DISABLE, MONTH_WEEK, MONTH, DAY, START_TIME, END_TIME)
values(NULL,NULL,'D', 'W', NULL, 2, NULL , NULL );
INSERT INTO SYSIBM.SYSINDEXCLEANUP(DBNAME, INDEXSPACE,
ENABLE_DISABLE, MONTH_WEEK, MONTH, DAY, START_TIME, END_TIME)
values(NULL,NULL,'D', 'W', NULL, 3, NULL , NULL );
INSERT INTO SYSIBM.SYSINDEXCLEANUP(DBNAME, INDEXSPACE,
ENABLE_DISABLE, MONTH_WEEK, MONTH, DAY, START_TIME, END_TIME)
values(NULL,NULL,'D', 'W', NULL, 4, NULL , NULL );
INSERT INTO SYSIBM.SYSINDEXCLEANUP(DBNAME, INDEXSPACE,
ENABLE_DISABLE, MONTH_WEEK, MONTH, DAY, START_TIME, END_TIME)
values(NULL,NULL,'D', 'W', NULL, 5, NULL , NULL );
```

Example 13-4 shows how to disable cleanup on all indexes every day from 8 am to 6 pm local time.

Example 13-4 Disable cleanup on all indexes every day from 8am to 6pm local time

```
INSERT INTO SYSIBM.SYSINDEXCLEANUP(DBNAME, INDEXSPACE,
ENABLE_DISABLE, MONTH_WEEK, MONTH, DAY, START_TIME, END_TIME)
values(NULL,NULL,'D', 'W', NULL, NULL,'08:00:00' , '18:00:00' );
Disable cleanup on index IX1 in database RMCDB00 on June 1st.
INSERT INTO SYSIBM.SYSINDEXCLEANUP(DBNAME, INDEXSPACE,
ENABLE_DISABLE, MONTH_WEEK, MONTH, DAY, START_TIME, END_TIME)
values('RMCDB00','IX1','D', 'M', 6, 1,NULL,NULL);
```

Example 13-5 shows how to disable cleanup on all indexes in database RMCDB00 on every Monday from 8 am to 5 pm.

Example 13-5 Disable cleanup on all indexes in database RMCDB00

```
INSERT INTO SYSIBM.SYSINDEXCLEANUP(DBNAME, INDEXSPACE,
ENABLE_DISABLE, MONTH_WEEK, MONTH, DAY, START_TIME, END_TIME)
values('RMCDB00',NULL,'D', 'W', NULL, 1,'08:00:00','17:00:00');
Disable cleanup on all indexes in database RMCDB00 but enable cleanup on index IX1
in
the same database.
INSERT INTO SYSIBM.SYSINDEXCLEANUP(DBNAME, INDEXSPACE,
ENABLE_DISABLE, MONTH_WEEK, MONTH, DAY, START_TIME, END_TIME)
values('RMCDB00',NULL,'D', 'W', NULL, NULL,NULL,NULL);
INSERT INTO SYSIBM.SYSINDEXCLEANUP(DBNAME, INDEXSPACE,
ENABLE_DISABLE, MONTH_WEEK, MONTH, DAY, START_TIME, END_TIME)
values('RMCDB00','IX1','E', 'W', NULL, NULL,NULL,NULL);
```

Example 13-6 shows two rows on the same level with conflicting information about Monday, cleanup is disabled on Monday.

Example 13-6 Two rows on the same level with conflicting information about Monday

```
INSERT INTO SYSIBM.SYSINDEXCLEANUP(DBNAME, INDEXSPACE,
ENABLE_DISABLE, MONTH_WEEK, MONTH, DAY, START_TIME, END_TIME)
values('RMCDB00',NULL,'D', 'W', NULL, 1,NULL,NULL);
INSERT INTO SYSIBM.SYSINDEXCLEANUP(DBNAME, INDEXSPACE,
ENABLE_DISABLE, MONTH_WEEK, MONTH, DAY, START_TIME, END_TIME)
values('RMCDB00',NULL,'E', 'W', NULL, 1,NULL,NULL);
```

The cleanup of pseudo-deleted index entries starts with DB2 11 CM.

Note: You can activate IFCID 0377 to monitor the cleanup processing for pseudo-empty index pages and pseudo-deleted index entries.

13.3.2 Indirect reference avoidance

When using variable length rows, or when using data compression, if a row is updated and the row size increases but can no longer fit on the original page, DB2 finds another page to store the row. It then modifies the original RID (Row IDentifier) to point at the overflow RID, thus creating an indirect reference because every access to the row requires an extra level of indirection to find the row. An indirect reference requires an extra DB2 Getpage, which often requires extra synchronous I/O.

How can you tell if your data base contains indirect references? You can tell by monitoring **NEARINDREF** and **FARINDREF** in **SYSIBM.SYSTABLEPART**. A “near” overflow is one that is likely to be prefetched by dynamic prefetch. However, random row access is more or less equally affected by both “near” and “far” indirect reference.

Let’s consider now the type of applications that are most likely to suffer a lot of indirect references. Nullable **VARCHAR** columns are indicative of the worst case, because some applications insert null values and later update the null values. The greater the update size quantity is as a percentage of the original row size, the more likely it is that indirect references will occur.

Indirect references are also possible when compression is used because a row might not compress as well after an update. Alternatively, a non-null **VARCHAR** column might grow. But, these situations do not necessarily cause indirect references in a systematic fashion the way nullable **VARCHAR** columns do.

REORG cleans up existing indirect references and also re-establishes more free space through **PCTFREE**. **PCTFREE** is the percentage of space on each page that **REORG** reserves. **LOAD REPLACE** also establish such free space. That reserved space is used by both inserts and updates. The inserts use it to maintain clustering. There is nothing in DB2 10 to prevent the inserts from consuming all of the free space, leaving no reserved space for the updates to increase the row size. Consequently, cluster ratios and indirect references tend to be correlated. If the cluster ratio is high, there will not be a lot of indirect references. When the reserved space becomes exhausted, the cluster ratio starts to degrade and updates that increase the row size start to cause indirect references.

When the reserved space is used up, the inserts start to become sequential. Thus, new rows are appended to the end of the table. If those newly inserted rows are also updated in the same order that the rows were inserted, then the overflows are sequential too, although the new rows and the overflows can be interspersed among each other. This function becomes important when you consider dynamic prefetch and sequential prefetch, but it is not important when you consider random fetch or list prefetch.

MAXROWS is the only tuning feature in DB2 10 that enables customers to avoid indirect references. If **MAXROWS** is based on the maximum row size, there will never be any indirect references. Alternatively, if **MAXROWS** is based on the average row size, indirect references will usually be avoided. However, the success of **MAXROWS** depends on the row size distribution being somewhat static. If the new rows that are created on Tuesday are of a different size than the rows that were created on Monday, it is hard to choose an optimal MAXROWS value that can apply to both days. Such dynamically changing distributions are unlikely, but

nevertheless using **MAXROWS** requires you to do some performance monitoring. is desirable. DB2 11 provides an autonomic solution.

Figure 13-8 shows an example of how an update to a **VARCHAR** column or to a compressed row that results in a larger row can cause the row to no longer fit on the same page. These rows need to be relocated to a new page, and a pointer to the new page is placed on the original page.

These indirect references cause the following negative impacts:

- Additional getpages and potentially additional I/Os to the overflow pages
- Lower clustering
- REORG TS is necessary to remove indirect references

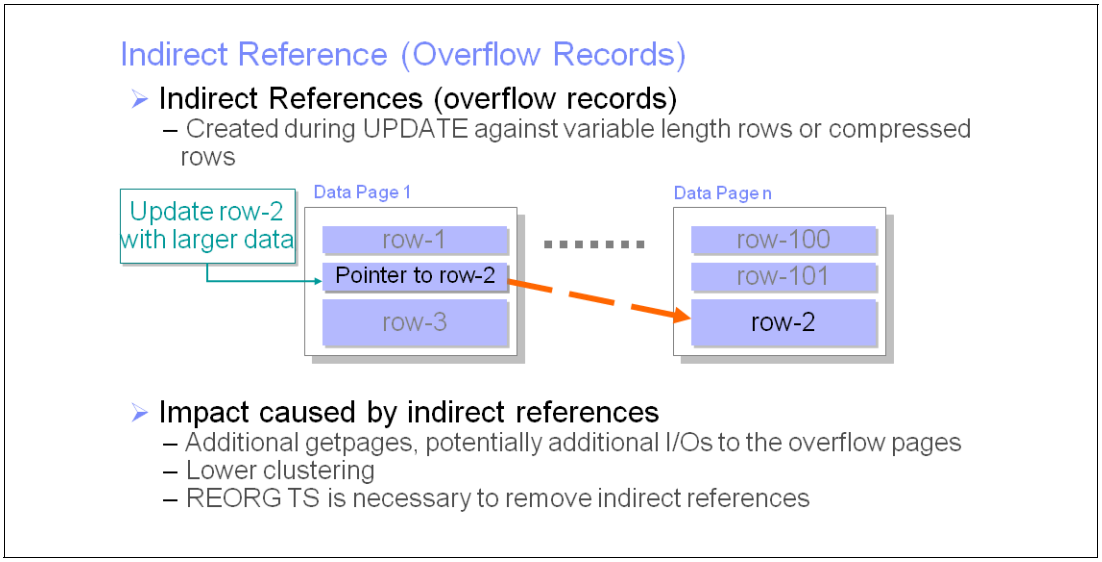


Figure 13-8 Indirect reference - Overflow records

DB2 11 provides the capability to reduce the number of indirect references by allowing the insert process to reserve the space for subsequent updates. This is accomplished through the new **FOR UPDATE** option of the **CREATE TABLESPACE** statement:

```
CREATE/ALTER  TABLESPACE   PCTFREE x  FOR UPDATE y
```

The explanation of the syntax is as follows:

- x = % of free space to leave in each data page by **LOAD** or **REORG**
- y = % of free space to leave in each data page by **INSERT**, **LOAD**, or **REORG**

An **INSERT** statement preserves the value provided by y% while **REORG** preserves (x+y) %.

The **PCTFREE_UPD** (**PERCENT FREE FOR UPDATE**) system parameter provides the system default for the **FOR UPDATE** value. If the system parameter value is not specified, then the behavior is the same as in DB2 10.

There is also an autonomic option available by specifying **PERCENT FOR UPDATE** on the **CREATE TABLESPACE** statement to override the default system parameter.

Example 13-7 shows the use of the new **FOR UPDATE** option on the **CREATE TABLESPACE** statement.

Example 13-7 Sample use of the new FOR UPDATE option of PCTFREE

```
CREATE TABLESPACE  TS1
FREEPAGE  0
PCTFREE  20 FOR UPDATE  10
```

In either of these cases, DB2 determines the value to use by using the history of **UPDATE** behavior based on Real Time Statistics (RTS). Use **FOR UPDATE -1** unless you know better due to consistent behavior of certain table spaces.

PCTFREE FOR UPDATE 0 indicates that DB2 will not reserve any space for updates, unless the **PCTFREE_UPD** system parameter is set to **AUTO**, in which case the behavior is the same as **PCTFREE FOR UPDATE -1**. If you really want to force DB2 to honor **PCTFREE FOR UPDATE 0** for some table spaces, then you cannot use **PCTFREE_UPD AUTO**. However, you can also minimize the effect of **PCTFREE FOR UPDATE** by setting it to 1%.

The autonomic behavior (**FOR UPDATE -1**) is a learning process based on RTS values for update rate and updated row size. If there are no **UPDATE**s or an infrequent number of **UPDATE**s, then either no space or less space is reserved for update. If there is a significant UPDATE rate, then the **INSERT** process will calculate the row size and reserve the appropriate space for subsequent **UPDATE**s.

The **REORG** and **LOAD** utilities calculate an estimated **PCTFREE FOR UPDATE** for **INSERT** statements to use. This value is stored in the **PCTFREE_UPD_CALC** column of catalog table **SYSIBM.SYSTABLEPART**.

INSERT processing continues to adjust the value based on RTS values.

Migration considerations

For DB2 to begin to make intelligent autonomic decisions about free space management, RTS in DB2 11 collects **UPDATE** information about the growth (or reduction) in the update row sizes. This RTS information is stored in **REORGUPDATESIZE** in **SYSIBM.SYSTABLESPACESTATS** as soon as you migrate to NFM, no matter what you set for **PCTFREE FOR UPDATE** or **PCTFREE_UPD**.

When you alter **PCTFREE FOR UPDATE** to -1 or modify **PCTFREE_UPD** to **AUTO**, DB2 uses **REORGUPDATESIZE**. However, unless the old RTS statistics for the number of inserts, updates and deletes are consistent, DB2 might not reserve much space, because the statistics are not consistent with each other.

To make them consistent, you can manually update **REORGINSERTS**, **REORGUPDATES**, and **REORGDELETES** in **SYSIBM.SYSTABLESPACESTATS** to 0. Alternatively, you can run **REORG**. Subsequently, you might still see more indirect references initially, but because the RTS statistics are consistent, after there have been a sufficient number of updates after, DB2 can derive a proper amount of space to reserve for updates.

When using the autonomic option, DB2 recalculates a new value after each RTS interval. Thus, the shorter the RTS interval is, the quicker DB2 reacts.

For tables spaces with heavy update activity (and especially for compressed data), specify a **PCTFREE FOR UPDATE** value. The **FOR UPDATE** value specifies the percentage of each page that is reserved to be used only by future update operations. When you specify FOR UDPATE -1, DB2 uses real-time statistics to automatically calculate how much free space to reserve for updates.

When you specify both **PCTFREE** and **FOR UPDATE** values, the percentage of free space reserved by a **REORG** or **LOAD REPLACE** operation is the sum of the two values.

13.4 More opportunities for RELEASE(DEALLOCATE)

In many cases, you specify **RELEASE(DEALLOCATE)** as a **BIND** option for applications that have critical performance needs, due to the CPU costs incurred to free resources at **COMMIT** points. You then reacquire those resources when needed if you specified **RELEASE(COMMIT)** instead. However, **RELEASE(DEALLOCATE)** needed to be used with caution, because in the case of persistent threads, the thread might not be deallocated for a long period of time.

As a DBA, you might need to break into these persistent threads to take one of the following actions:

- Perform a **BIND REPLACE** or **REBIND PACKAGE** for an application bound with **RELEASE(DEALLOCATE)**
- Perform online schema changes to tables or indexes accessed by an application bound with **RELEASE(DEALLOCATE)**
- Run an online **REORG** utility to materialize pending **ALTER**s that affect applications bound with **RELEASE(DEALLOCATE)**

The problem in each of these scenarios is that you needed to identify and stop/cancel any active persistent DB2 threads running packages bound with **RELEASE(DEALLOCATE)** before you can take any of the actions listed previously.

DB2 11 introduces the **PKGREL_COMMIT** system parameter, which you can use to handle those scenarios where you need to break into a persistent thread to accomplish one of these listed tasks. **PKGREL_COMMIT** is an online-changeable DB2 11 installation system parameter that, when set to **YES**, allows DB2 to break into persistent threads at **COMMIT** or **ROLLBACK** points. If the parameter is set to **YES** and a package is bound with **RELEASE(DEALLOCATE)** and if DB2 detects a **BIND REPLACE** or **REBIND PACKAGE** command, a DDL statement or a utility operation that needs to quiesce or invalidate the application's DB2 package, then DB2 will implicitly de-allocate/release the package at a **COMMIT** or **ROLLBACK**.

With the **PKGREL_COMMIT** system parameter set to **YES**, you no longer need to identify in advance and stop or cancel any active persistent DB2 threads running packages bound with **RELEASE(DEALLOCATE)** before attempting a **BIND REPLACE/REBIND PACKAGE** command, schema change or utility associated with those packages. Instead, the behavior is the same as though the package was bound with **RELEASE(COMMIT)**.

This new behavior is not supported for any of the following situations:

- Packages that have **OPEN** and **HELD** cursors at the time of the **COMMIT** or **ROLLBACK**
- Packages that are bound with **KEEPDYNAMIC(YES)**
- When the **COMMIT** or **ROLLBACK** occurs within a DB2 stored procedure

The DB2 11 default for parameter **PKGREL_COMMIT** is **YES**.

13.5 Optimizer enhancements

The following optimizer enhancements are provided in DB2 11 to improve application performance:

13.5.1 Identification of critical statistics for improved query performance

You might often find it challenging to know what statistics to collect to obtain the best possible access path for your SQL statements. At an individual query level, identification of important statistics is difficult. At an application or subsystem level, identification of important statistics requires that you have knowledge of each SQL statement. If you have ad-hoc dynamic SQL in your environment, then the closest representation of the workload is the contents of your dynamic statement cache, which by nature of realistic size limitations can only contain a portion of the dynamic SQL that is actually executed.

If you collect insufficient statistics, you might end up with an inefficient access path and poor query performance. In general, collecting more complete and accurate statistics results in more accurately estimated selectivity, which results in improved access path choices. There are still scenarios where cost estimation is difficult and performance regression can occur. However, deciding to collect less information and depending on more inaccuracy to get a better access path by chance is not a viable long term strategy.

Figure 13-9 illustrates the classic way of collecting and exploiting DB2 statistics, prior to DB2 11. The DB2 optimizer exploits the statistics in the DB2 catalog, does not takes advantage of the Real Time Statistics, and does not provide feedback about the value of the existing statistics.

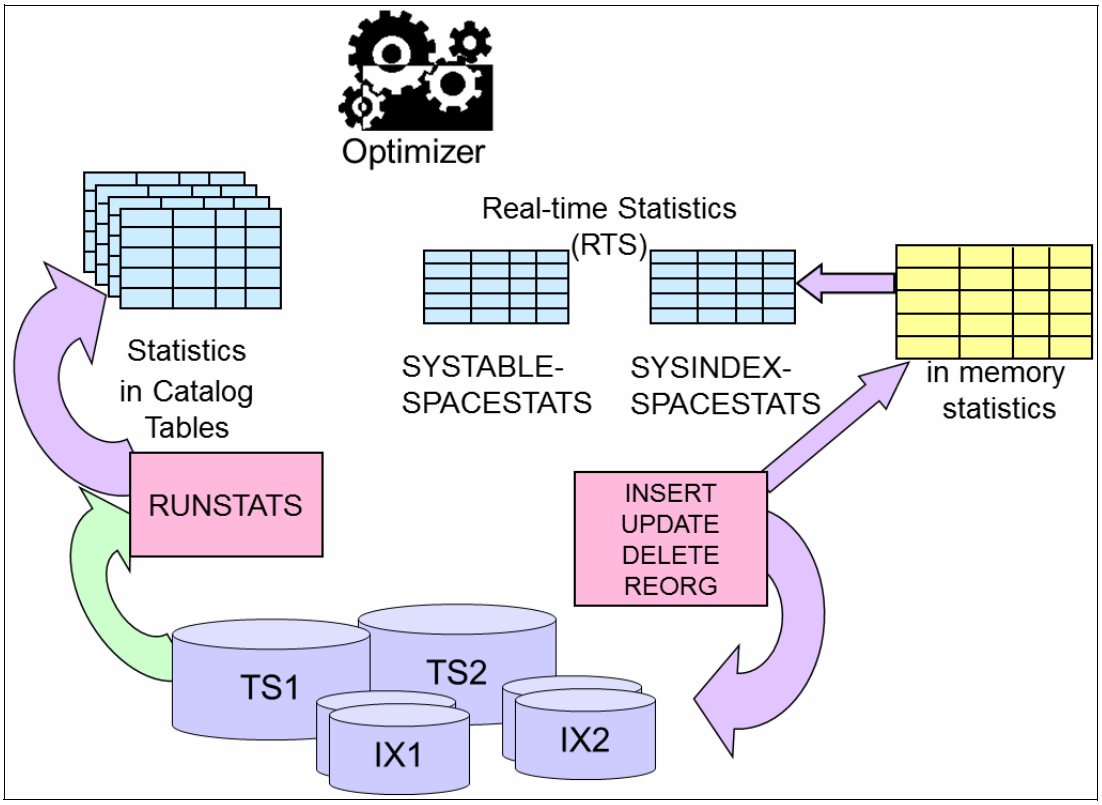


Figure 13-9 DB2 statistics and the optimizer, previous to DB2 11

DB2 provides an enhancement to externalize missing statistics information during query optimization. Statistics collection utilities can then use this information as input to collect the missing statistics at the next execution.

This enhancement externalizes statistics recommendations for missing or conflicting statistics encountered during query optimization. The statistics recommendations can then be used to

drive **RUNSTATS** such that DB2 has more accurate and complete statistics during query optimization and, as a result, can choose more efficient access paths.

Figure 13-10 illustrates how the DB2 11 optimizer provides feedback about the DB2 statistics at **BIND**, **REBIND**, and **PREPARE**.

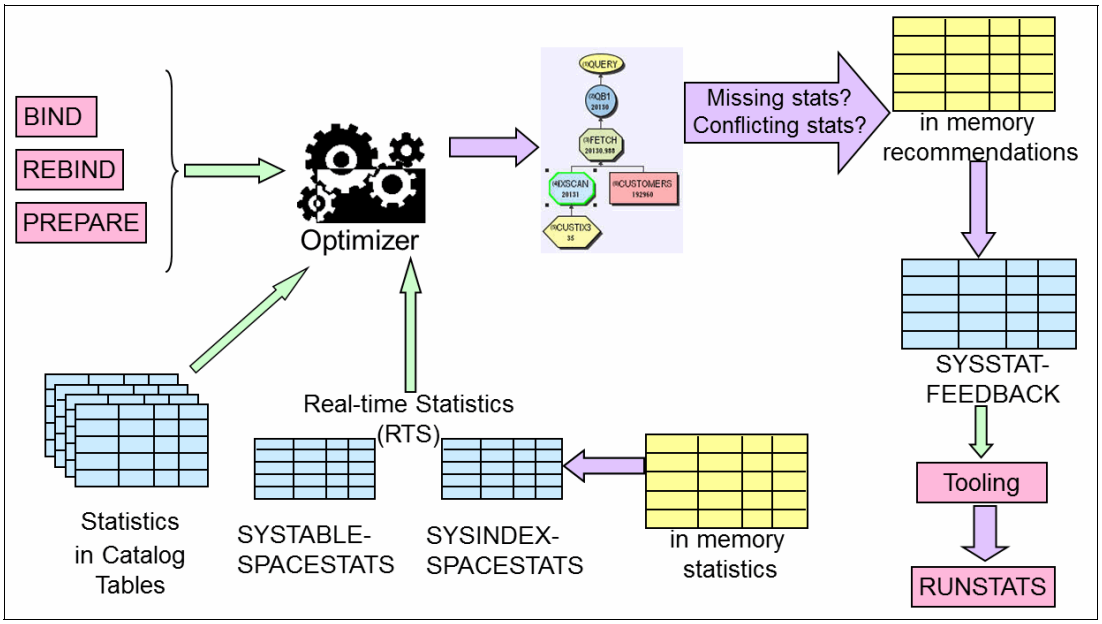


Figure 13-10 The DB2 11 optimizer and BIND, REBIND, and PREPARE: statistics feedback

On every **BIND** and **PREPARE** DB2 identifies missing or conflicting statistics, which are then externalized to a **SYSIBM.SYSSTATFEEDBACK** catalog table. This catalog table is populated asynchronously to avoid any performance impact to the PREPARE process. The frequency with which the statistics recommendations are externalized to the **SYSIBM.SYSSTATFEEDBACK** table is controlled by existing **STATSINT** subsystem parameter.

DB2 also externalizes statistics recommendations during **EXPLAIN** processing. A new **DSN_STAT_FEEDBACK** explain table is populated synchronously with the statistics recommendations during **EXPLAIN** processing.

Figure 13-11 shows the relationship between the DB2 statistics, **PREPARE**, and the optimizer feedback.

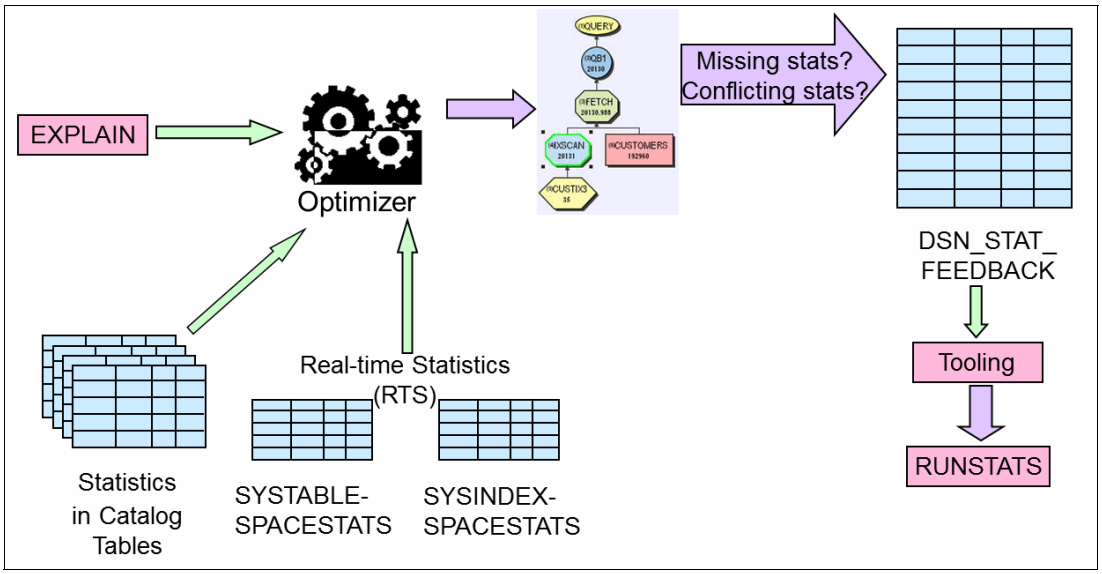


Figure 13-11 he DB2 11 optimizer and EXPLAIN: statistics feedback

The contents of the **SYSSTATFEEDBACK** or **DSN_STAT_FEEDBACK** tables can be used to generate input to the **RUNSTATS** utility to allow more complete statistics to be collected. DB2 will not convert the output of the **SYSIBM.SYSSTATFEEDBACK** to a format directly consumable by **RUNSTATS**. However, you can use capabilities built into the Optim Query Workload Tuner tool to identify what statistics to collect and to generate **RUNSTATS** control statements to collect those statistics.

Note: Statistics recommendations are not made for volatile tables, declared global temporary tables (DGTTs), or created global temporary tables (CGTTs).

To maintain an accurate picture of currently missing statistics in catalog table **SYSIBM.SYSSTATFEEDBACK**, the **RUNSTATS** utility ensures that recommendations for statistics that have subsequently been collected do not remain in the **SYSSTATFEEDBACK** catalog table. The **DSN_STAT_FEEDBACK** explain table maintains the set of missing statistics as of the **EXPLAIN** time and is not affected by the execution of the **RUNSTATS** utility.

Interpreting the statistics recommendations

In addition to using Optim Query Workload Tuner to generate statistics, you can manually create your own **RUNSTATS** jobs based on the information in the **SYSSTATFEEDBACK** table. Here are some guidelines on how to interpret the statistics recommendations and what to focus on.

The statistics recommendations can be at the table, index or column level. Therefore, the **SYSSTATFEEDBACK** table includes columns that can represent any of these identifiers, as shown in Figure 13-12.

Statistic could be recommended at table, index or column level

TBCREATOR	VARCHAR(128) NOT NULL	The creator of the table.
TBNAME	VARCHAR(128) NOT NULL	The name of the table.
IXCREATOR	VARCHAR(128) NOT NULL	The creator of the index.
IXNAME	VARCHAR(128) NOT NULL	The name of the index.
COLNAME	VARCHAR(128) NOT NULL	The name of the column.
NUMCOLUMNS	INTEGER NOT NULL	The number of columns in the column group.
COLGROUPCOLNO	VARCHAR(254) NOT NULL FOR BIT DATA	A hex representation that identifies the set of columns associated with the statistics. If the statistics are only associated with a single column, the field contains a zero length. Otherwise, the field is an array of SMALLINT column numbers with a dimension equal to the value in NUMCOLUMNS.

Figure 13-12 Statistics granularity in SYSIBM.SYSSTATFEEDBACK table

In addition to the identifying information, the following additional columns in the table contain information that you can use to determine what statistics to collect:

- **TYPE**
- **REASON**

The **TYPE** column specifies the statistics to collect, and the **REASON** column identifies why the type of statistics were recommended. You can use the information in both of these columns to make decisions about what statistics to collect.

The **TYPE** column is defined as `CHAR(1)`. Table 13-2 lists the possible values for the **TYPE** column.

Table 13-2 TYPE of statistics recommendation

TYPE value	Type of statistic to collect
C	Cardinality
F	Frequency
H	Histogram
I	Index
T	Table

The **REASON** column is defined as CHAR(8). Table 13-3 lists the possible values for the **REASON** column.

Table 13-3 REASON why statistics are recommended

REASON values	Description
BASIC	A basic statistic value for a column, table or index is missing.
KEYCARD	The cardinalities of index key columns are missing.
LOWCARD	The cardinality of the column is a low value, which indicates that data skew is likely.
NULLABLE	Distribution statistics are not available for a nullable column.
DEFAULT	A predicate references a value that is probably a default value.
RANGEPRD	Histogram statistics are not available for a range predicate.
PARALLEL	Parallelism can be improved by uniform partitioning of key ranges.
CONFLICT	Another statistic conflicts with this statistic.
COMPFFIX	Multi-column cardinality statistics are needed for an index compound filter factor.

Interpreting the TYPE column

The **TYPE** column identifies one of the following possible types of statistics that are recommended, depending on the value for **TYPE**:

- If TYPE = T, then collect basic table statistics. The **RUNSTATS** format to use is:

```
RUNSTATS TABLESPACE ...
   TABLE(table-name)
```

- If TYPE = I, then collect basic index statistics. The **RUNSTATS** format to use is:

```
RUNSTATS INDEX
```

- If TYPE = C, then collect cardinality statistics. The **RUNSTATS** format to use depends on whether the recommendation is for single column cardinality statistics or multi-column cardinality statistics, which is indicated by the **NUMCOLUMNS** column.

  For single column cardinality statistics, use the **COLUMN** option of **RUNSTATS**:

```
RUNSTATS TABLESPACE ...
   TABLE(table-name)
   COLUMN(column-name)
```

  For multi-column cardinality statistics, use the **COLGROUP** option of **RUNSTATS**:

```
RUNSTATS TABLESPACE ...
   TABLE(table-name)
   COLGROUP(column-name1,column-name2 ...)
```

- If TYPE = F, then collect frequency statistics. Use the **FREQVAL** option of **RUNSTATS**:

```
RUNSTATS TABLESPACE ...
   TABLE(table-name
   COLGROUP(column-name) FREQVAL COUNT integer
```

- If TYPE = H, then collect histogram statistics. Use the **HISTOGRAM** option of **RUNSTATS**:

```
RUNSTATS TABLESPACE ...
   TABLE(table-name)
   COLGROUP(column-name) HISTOGRAM
```

Interpreting the REASON column

The **REASON** column identifies one of nine possible reasons why the statistics are being recommended. Each **REASON** value is described in Table 13-3 on page 406. Here are some recommendations for interpreting the **REASON** values and focusing on those that will provide the most benefit.

Your first priority should be to focus on any statistics recommendations with a **REASON** of BASIC. This reason indicates that basic table or index statistics are missing. The optimizer can only use default values if basic statistics are missing, and default statistics will not provide you with optimal access paths.

Your second priority should be any statistics recommendations with a **REASON** of CONFLICT. This reason indicates that there is a conflict between table and index statistics or between frequency and cardinality statistics. The existence of a conflict implies that statistics were run on different objects at different times.

After addressing any recommendations for BASIC and CONFLICT, focus on **LOWCARD**, **NULLABLE**, and **DEFAULT** recommendations. Any other reasons are targeted towards a more specific recommendation and might require further investigation.

Additional notes on interpreting the recommendations

The recommendations provided by this enhancement are only recommendations for a statistic that can be used if it is collected. The recommendation is not a guarantee that the statistic is needed.

There is still a benefit to making an attempt to determine whether collecting the recommended statistics will add value, meaning whether it will provide information that will aid the optimizer in determining the least cost access path. For example, if the **TYPE** is F, for frequency, you might want to investigate whether the data is really skewed before collecting the frequency statistics. In addition, you need to decide what is a good value to use for the **COUNT** option. Typically, 10 is a good default, but if the value of **COLCARDF** column in **SYSIBM.SYSCOLUMNS** is less than or equal to 10, then use a COUNT value of one less than the **COLCARDF** value.

In addition, you need to look at the **REASON** value when making your decision. For example. if the TYPE = F for frequency statistics, but the **REASON** is NULLABLE, and if NULL is the most frequently occurring value, then you only need a **COUNT** value of 1, not 10.

Controlling externalization of statistics recommendations

DB2 provides two mechanisms to control when statistics recommendations are externalized to the **SYSIBM.SYSSTATFEEDBACK** catalog table.

The first mechanism is new subsystem parameter **STATFDBK_SCOPE**, which takes one of the following values:

NONE	Disable collection of recommended **RUNSTATS**
STATIC	Collect recommended **RUNSTATS** for static queries only
DYNAMIC	Collect recommended **RUNSTATS** for dynamic queries only
ALL	Collect recommended **RUNSTATS** for all SQL statements (default)

The second mechanism is a new column in **SYSTABLES** named **STATS_FEEDBACK**, which provides control of statistics recommendations at the table level. If **STATS_FEEDBACK** is updated to a value of N for a given table, no statistics recommendations will be made for that table or its associated columns and indexes. The default value for this column is Y for **YES**.

Independent of the settings of the **STATFDBK_SCOPE** parameter and **STATS_FEEDBACK** column in **SYSTABLES**, recommended statistics are written to the **DSN_STAT_FEEDBACK** explain table. These two mechanisms govern population of recommendations to the **SYSSTATFEEDBACK** catalog table only.

Part 4

Appendixes

This part of the book includes the following appendixes:

A

Information about IFCID changes

This appendix includes the details of new or changed IFCIDs previously discussed in the chapters of this book. For more information about IFCIDs, refer to *DB2 11 for z/OS What's New?*, GC19-4068. For collecting accounting and statistics, see *Subsystem and Transaction Monitoring and Tuning with DB2 11 for z/OS*, SG24-8182.

DB2 for z/OS has system limits, object and SQL limits, length limits for identifiers and strings, and limits for certain data type values. Restrictions exist on the use of certain names that are used by DB2. In some cases, names are reserved and cannot be used by application programs. In other cases, certain names are not recommended for use by application programs though not prevented by the database manager.

For information about limits and name restrictions, refer to *DB2 11 for z/OS SQL Reference*, SC19-4066.

You can find up-to-date mapping in the SDSNMACS data set that is delivered with DB2.

This appendix includes the following topics:

- New IFCIDs
- Aggregate accounting overview and purpose
- IFCID 53 and 58 enhancements overview
- Accounting trace enhancements overview
- IRLM Storage Accounting enhancement
- Stored procedure monitoring overview and purpose
- Other accounting changes

A.1 New IFCIDs

DB2 11 includes the following instrumentation facility component identifiers (IFCIDs):

- IFCID 377: Pseudo-deleted index entries are automatically cleaned up
- IFCID 27: Monitor sparse index usage
- IFCID 382 and 383: Records suspend operations for parallel task

A.1.1 IFCID 377: Pseudo-deleted index entries are automatically cleaned up

IFCID 377 is introduced to monitor the index daemon activity when it cleans up committed pseudo deleted entries from an index. It includes the DBID, PSID of the index being cleaned up, the partition number of the index and the page number being cleaned up. It has an indicator to show if the cleanup is a pseudo empty page cleanup, in which case the pseudo empty index page is deleted from the index tree, or if the cleanup is pseudo deleted entry cleanup, in which case the index page remains in the index tree. Only committed pseudo deleted entries are removed from the index page. It also has a field to show the number of pseudo deleted entries removed from each index page.

The IFCID 377 record is written once per each index page being cleaned up. It is not included in any trace class because its volume can be large.

Example A-1 maps the new IFCID 377.

Example A-1 New IFCID 377 to record index pseudo delete daemon cleanup

```
************************************************************************
*   IFCID 0377 to record index pseudo delete daemon cleanup            *
************************************************************************
*
QW0377   DSECT               IFCID(QWHS0377)
QW0377DB DS    CL2           DATA BASE ID
QW0377OB DS    CL2           INDEX PAGE SET ID
QW0377PT DS    CL2           PARTITION NUMBER
QW0377FL DS    CL1           FLAGS
         DS    CL1           RESERVED
QW0377PG DS    CL4           PAGE NUMBER of the index page cleaned up
QW0377NU DS    CL4           NUMBER OF PSEUDO DELETED ENTRIES REMOVED
QW0377DL EQU   X'80'         PSEUDO EMPTY PAGE IS DELETED
QW0377CL EQU   X'40'         PSEUDO DELETED ENTRIES CLEANED UP
*
```

A.1.2 IFCID 106

The **QWP4IXCU** field is added to trace the internal settings of the new **INDEXCLEANUP_THREADS** subsystem parameter. The IFCID 106 formatter stored procedures, **SYSPROC.DSNWZP** and **SYSPROC.ADMIN_INFO_SYSPARM** are updated to report the **INDEXCLEANUP_THREADS** setting.

Example A-2 maps the new IFCID 106.

Example A-2 Changed IFCID 106 to record INDEXCLEANUP_THREADS

```
QWP4DM1636 DS  CL8       (s)                                  DM1636
QWP4MIMTS DS   F         MAXSORT_IN_MEMORY                   N4504r5
QWP4MUSE DS    XL2       (s)                                 N4504r5
QWP4IXCU DS    H         INDEXCLEANUP_THREADS                n0010r5
QWP4DEGD DS    F         PARAMDEG_DPSI                        n231r5
```

```
       DS    CL132     UNUSED                                    n0010r5
*********************************************************************
*     ASSEMBLY DATE                                                 *
*********************************************************************
QWP4DATE DS    CL8       (S)
```

A.1.3 IFCID 27: Monitor sparse index usage

New IFCID 27 records are added to let the user know which type of sparse index is used for probing, the amount of storage used, and many other characteristics about the current sparse index. Also global trace records are added to IFCID 2 and 3 to let the user know if the sparse index cannot be optimized because not enough storage is available or if it had to be placed into a physical work file, which might hurt query performance.

With this new instrumentation, you can adjust the **MXDTCACH** value higher or lower, depending on the storage available on your system and to make the query perform at its most optimal performance, as shown in Example A-3.

Example A-3 New IFCID 27

```
*********************************************************************
* IFC ID 0027 FOR RMID 20 RECORDS DETAILED SORT INFORMATION. *
* NUMBER OF SEQUENTIAL RECORDS IN THIS WORKFILE. *
*********************************************************************
*
QW0027 DSECT IFCID(QWHS0027)
QW0027NR DS D NUMBER OF RECORDS IN THE NEW WORKFILE
QW0027SP DS CL1 TYPE OF QW0027 RECORD:
* ****************** CONSTANTS FOR QW0027SP *************************
QW0027CB EQU C'B' INPUT PHASE OR MERGE PASS END
*                   INDICATES SPARSE INDEX COMBINATION
*                   OF HASH AND WORKFILE USED
*                   (BOTH IN-MEMORY AND PHYS. WORKFILE)
QW0027CH EQU C'H' INPUT PHASE OR MERGE PASS END
*                   INDICATES SPARSE INDEX HASH USED
*                   (IN-MEMORY WORKFILE ONLY)
QW0027CO EQU C'O' INPUT PHASE OR MERGE PASS END
*                   INDICATES SPARSE INDEX BINARY USED
*                   (IN-MEMORY WORKFILE ONLY)
QW0027CS EQU C'S' INPUT PHASE OR MERGE PASS END
*                   INDICATES NO SPARSE INDEX WAS USED
*                   BECAUSE OF STORAGE CONSTRAINTS
QW0027CT EQU C'T' INPUT PHASE OR MERGE PASS END
*                   INDICATES SPARSE INDEX WORKFILE USED
QW0027CW EQU C'W' INPUT PHASE OR MERGE PASS END
*                   INDICATES NO SPARSE INDEX USED
* ********************************************************************
DS CL3 RESERVED
*********************************************************************
* THE FOLLOWING INFORMATION FOR IFC ID 27 WILL ONLY BE SET IF THE *
* CURRENT SORT IS PROCESSING A SPARSE INDEX, OTHERWISE SET TO 0. *
*********************************************************************
QW0027SF DS F SKIP FACTOR IF SPARSE INDEX RECORDS
*                   IN WORKFILE. 1 IF IN-MEMORY
QW0027OZ DS D SIZE OF SPARSE INDEX SPACE USED
```

```
*                    (IN KB)
QW0027IE DS F NUMBER OF RECORDS IN IN-MEMORY
*                 PART OF SPARSE INDEX
QW0027WE DS F NUMBER OF RECORDS IN WORKFILE
*                 PART OF SPARSE INDEX
QW0027DS DS F DATA AREA SIZE FOR SPARSE INDEX
*                 (IN BYTES)
QW0027KS DS F KEY SIZE FOR SPARSE INDEX (IN BYTES)
QW0027TS DS F TOTAL NUMBER OF SPARSE INDEXES IN QUERY
QW0027SC DS F CURRENT SPARSE INDEX BEING PROCESSED
QW0027TZ DS D APS ESTIMATED SIZE OF ALL SPARSE
*                    INDEXES IN QUERY IF ALL IN-MEMORY (IN KB)
QW0027IR DS D APS ESTIMATED NUMBER OF RECORDS IN CURRENT SPARSE INDEX
```

The IFCID 2 trace record is for the system statistics records and IFCID 3 is for accounting records. Sparse index adds two additional records to the system statistics and accounting records to let the user know the number of times that sparse index used a physical work file and the number of times that sparse index ran into storage problems where it had to disable sparse index. This information is tracked per transaction. Below are the instrumentation changes for IFCID 2 and IFCID 3. See Example A-4 on page 414

Example A-4 IFCID 2 and IFCID 3

```
...
QXSISTOR DS D THE NUMBER OF TIMES THAT SPARSE INDEX
*                    WAS DISABLED BECAUSE OF INSUFFICIENT
*                    STORAGE.
QXSIWF DS D THE NUMBER OF TIMES THAT SPARSE INDEX
*                 BUILT A PHYSICAL WORK FILE FOR PROBING.
...
```

A.1.4 IFCID 382 and 383: Records suspend operations for parallel task

Two new IFCIDs are introduced that are part of the following classes:

- Accounting Class 3
- Accounting Class 8
- Monitor Class 3
- Monitor Class 8

If IFCID382 and IFCID383 are started to an external destination. Example A-5 lists the new IFCID 382/383 to records suspend operations for parallel task.

Example A-5 New IFCID 382 and 383 to record suspend operations for parallel task

```
************************************************************************
* 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
*
************************************************************************
* END Suspend for parallel task synchronization                        *
************************************************************************
*QW0383       DSECT             IFCID(QWHS0383)
```

```
*QWO383RT     DS  CL1           Type of task resumed.
*QWO383PT     EQU C'P'          Task resumed is a parent
*QWO383CT     EQU C'C'          Task resumed is a child
*
```

A.2 Aggregate accounting overview and purpose

Users need to externalize accounting values at the statistics intervals. Analysis of various performance problems suffers greatly from the need to identify the correct time frame. In addition, to do so, analysis of all accounting records is required. DB2 10 added IFCID369 to **STATISTICS CLASS 9** and externalize them every minute. IFCID369 externalizes the summation of all agents that completed processing during this 1 minute interval. See Example A-6 for details.

Example A-6 IFCID369 - Aggregate accounting interface details

```
************************************************************************
* IFCID 369 is a statistics record containing wait and CPU             *
* aggregated by connection type. It is written at statistics           *
* intervals and is available via the IFI READS interface.              *
*                                                                      *
* This record contains 4 sections mapped as follows:                   *
*                                                                      *
*   Data Section 1 is mapped by QWO369_1                               *
*   Data Section 2 is a repeating group, each mapped by QWO369_2       *
*   Data Section 3 is a repeating group, each mapped by DSNDQWAC       *
*   Data Section 4 is a repeating group, each mapped by DSNDQWAX       *
*                                                                      *
* Notes:                                                               *
*  1.Statistics collection will become enabled when both IFCID 369     *
*    and IFCID 3 is enabled on the system.                             *
*  2.All counters will be reset to zeroes when DB2 is restarted        *
*  3.Statistics are aggregated by connection type. If no agents        *
*    for that connection type have executed since the 369              *
*    collection in enabled, no data will be externalized for that      *
*    connection type.                                                  *
*  4.QWACPCNT indicates the number of transactions aggregated for      *
*    a given connection type.                                          *
*                                                                      *
************************************************************************
QWO369_1 DSECT
QWO369ST DS    CL8          Timestamp when 369 statistics collection
*                           was enabled
QWO369SP DS    CL8          Timestamp when 369 statistics collection
*                           was disabled
*
QWO369_2 DSECT
QWO369CN DS    CL8          Connection name
```

A.3 IFCID 53 and 58 enhancements overview

IFCIDs 53 and 58 are the end SQL statement IFCIDs for a number of SQL statement types. To understand which SQL statement type a given IFCID 53/58 closes, it is necessary to correlate to a specific begin SQL statement (IFCID 66). DB2 11 adds an identifier to IFCID 53/58 to make the beginning SQL statement IFCID unnecessary. The **QW0058TOS** and **QW0053TOS** fields are introduced.

Example A-7 shows the details for IFCID0053.

Example A-7 IFCID0053

```
************************************************************************
* IFC ID 0053 FOR RMID 22                                              *
*   CHANGED 8-17-87 - THIS RECORD CHANGED TO DEFINE END OF DESCRIBE*
*   SQL COMMIT, SQL ROLLBACK, OR AN ERROR CONDITION OCCURRED BEFORE*
*   SQL STATEMENT ANALYZED.  THE BEGINNING STATEMENT IS NOT        *
*   RECORDED.  IT IS AN UNPAIRED EVENT.                            *
************************************************************************
*
QW0053TOS DS    X          Type of SQL Request
*                          Constants for QW0053TOS field are defined in
*                          QW0058 mapping.
*
```

Example A-8 shows the details for IFCID0058.

Example A-8 IFCID0058

```
*
*/********************************************************************/
*/* IFC ID 0058 FOR RMID 22 RECORDS END SQL STATEMENT EXECUTION      */
*/********************************************************************/
*
QW0058TOS DS    X          Type of SQL Request
*
*/********************************************************************/
```

Note: IFCID0058 is identical to IFCID0053. If both IFCI0058 and IFCID0053 are started, IFCID0058s without an **BEGIN** SQL statement are written as IFCID0053.

A.4 Accounting trace enhancements overview

Accounting trace contains thread execution information, such as CPU consumed, waiting times, SQL executions, number of lock events, commits, buffer pool requests, RLF numbers, DDF process, RID pool, and start and stop times. This section includes information about the following trace enhancements:

- New field QWHCAACE
- QWACZIIP_ELIGIBLE field

A.4.1 New field QWHCAACE

Correlating some performance records that are written by system agents on behalf of an accounting interval is difficult. A **QWHCAACE** field is introduced to the correlation header (QWHC) to make correlation easier. **QHWCAACE** can be correlated to QWHSACE. See Example A-9. For child tasks, the **ACE** of the parent is stored in **QWHCAACE**.

Example A-9 Accounting trace enhancements filed QHWCAACE

```
QWHCAACE     DS    CL8             /* This field is 0 if this IFCID  */
*                                  /* written outside an accounting  */
*                                  /* interval. Otherwise it is the  */
*                                  /* ACE of the agent that initiated*/
*                                  /* the accounting interval. This  */
*                                  /* can be used to correlate to    */
*                                  /* QWHSACE for the non-rollup     */
*                                  /* IFCID3's. For DDF/RRSAF rollup */
*                                  /* accounting, it can be          */
*                                  /* used to correlate to QWARACE.  */
```

A.4.2 QWACZIIP_ELIGIBLE field

To show on a DB2 accounting report whether a user's workload running on a general purpose engine is eligible to run on a zIIP specialty engine if one is installed, DB2 10 APAR PM57206 reintroduces the **QWACZIIP_ELIGIBLE** field in IFCID3 as a serviceability field. The field captures the IBM specialty engine eligible time that is run on a general purpose CP for a subset of IBM specialty engine eligible processing. More specifically, the time reflects the eligible time for the following functions:

- Distributed DBATs
- Parallel query parent threads
- zIIP eligible utilities

All other cases of IBM specialty engine offload are *not* reflected in this serviceability field. DB2 11 removes the restrictions and captures all zIIP and zAAP time for any kind of transaction.

Example A-10 shows the **QWACZIIP_ELIGIBLE** field in DB2 10 and DB2 11.

Example A-10 Accounting trace enhancements QWACZIIP_ELIGIBLE

```
DB2 11
QWACZIIP_ELIGIBLE DS  CL8 /* (S) Accumulated CPU executed on a      */
*                         /* standard CP for IBM specialty engine   */
*                         /* eligible work.                         */
*                         /* For parallel query parent records the  */
*                         /* value will reflect zIIP eligible time  */
*                         /* for the parent and the child tasks.    */
                      /* Child task records will have a 0 value.*/

DB2 10
QWACZIIP_ELIGIBLE DS  CL8 /* (S) Accumulated CPU executed on a      */
*                         /* standard CP for zIIP-eligible work.    */
*                         /* This field will reflect zIIP eligible  */
*                         /* time for accounting records written    */
*                         /* for:                                   */
```

```
*                        /*   1) distributed DBATs                */
*                        /*   2) parallel query parents           */
*                        /*   3) zIIP eligible utilities          */
*                        /* For parallel query parent records the  */
*                        /* value will reflect zIIP eligible time  */
*                        /* for the parent and the child tasks.    */
*                        /* Child task records will have a 0 value.*/
*                        /* All other cases of specialty engine    */
*                        /* offload are NOT reflected in this      */
*                        /* field.                                 */
```

A.5 IRLM Storage Accounting enhancement

Enhanced monitoring of IRLM common and private storage usage. IRLM added additional information to the existing DXR100I message to provide details of IRLM storage usage in ECSA and Private, including tracking details like high water marks, compressions and expansions counts when **F,IRLMxx, STATUS,STOR** command is issued, See Example A-11.

Example A-11 DXR100I message

```
DXR100I IR21021 STOR STATS
      PC: YES  LTEW:n/a LTE:      M RLE:          RLEUSE:
      BB PVT:  1495M  AB PVT (MEMLIMIT):  16383P
      CSA USE: ACNT:     0K  AHWM:     0K  CUR:   309K  HWM:   309K
               ABOVE 16M:    16    309K     BELOW 16M:     0      0K
               AB CUR:              25M     AB HWM:             150M
      PVT USE:   BB CUR:  4377K        AB CUR:     5M
                 BB HWM:   1.5M        AB HWM:    12M
      CLASS   TYPE  SEGS     MEM   TYPE  SEGS     MEM   TYPE  SEGS     MEM
      ACCNT    T-1     2      4M    T-2     1      1M    T-3     1      4K
      PROC     WRK     4     20K    SRB     1      1K    OTH     1      1K
      MISC     VAR     8   4310K    N-V    12    323K    FIX     1     24K

      ********************************************************************
      *       IRLM Serviceability only:  Storage subpool statistics      *
      ********************************************************************
      Pool Name    Ptype     Storage  #Segments  #Elem/S  #EXPN    #CMPR
      DESP         LCVBN    --------    ------   ----     ----     ----
      IB           LCFBN    --------    ------   ----     ----     ----
      ISL          GCFBN    --------    ------   ----     ----     ----
      NCB          GCFBN    --------    ------   ----     ----     ----
      NPL          GCFBN    --------    ------   ----     ----     ----
      QEFX         LNFBN    --------    ------   ----     ----     ----
      QE28         GNFBN    --------    ------   ----     ----     ----
      RHBL         LNFAA    --------    ------   ----     ----     ----
      RHLB         LCFAA    --------    ------   ----     ----     ----
      RHWK         GCFBN    --------    ------   ----     ----     ----
      RHWL         LCFBN    --------    ------   ----     ----     ----
      RLB          LCFAA    --------    ------   ----     ----     ----
      RLBI         LNFBA    --------    ------   ----     ----     ----
      SIDB         GCFBN    --------    ------   ----     ----     ----
      SPL          GCFBN    --------    ------   ----     ----     ----
      SRB          GNFBN    --------    ------   ----     ----     ----
      STKS         GNFBN    --------    ------   ----     ----     ----
```

```
      TRCE        GNVBN    --------    ------   ----     ----     ----
      VARG        GCVBN    --------    ------   ----     ----     ----
      VARL        LCVBN    --------    ------   ----     ----     ----
      VGFX        GCVBN    --------    ------   ----     ----     ----
      VMSG        LCVBN    --------    ------   ----     ----     ----
      WHB         LCFBA    --------    ------   ----     ----     ----
          DXR100I End of display
```

DB2 also issue a **STAT** request to get the IRLM system statistics and capture into IFCID225, IFCID217 and IFCID106 traces. See Example A-12.

Example A-12 IFCID217, IFCID225 and IFCID106

```
*****************************************************************
*   IFCID 0217 for storage manager pool statistics.             *
*****************************************************************
*
* Section QWT02R10 is mapped by QW0217.
*
* Section QWT02R20 is mapped by QW02172. There will be
* a repeating group entry for each:
*   1. DBM1 private pool (31 or 64-bit)
*   2. Common pool (31 or 64-bit)
*   3. Shared pool
*   Agent local pools will not be reported in this section.
*
* Section QWT02R30 is mapped by QW02173. There will be a
* repeating group entry for each agent local storage pool.
* Agent local storage pools are in 31-bit DBM1 private or in
* 64-bit shared storage.
*
* Section QWT02R40 is mapped by QW02174. There will be a
* repeating group entry for each IRLM storage pool. IRLM pools
* can be in ECSA, 31-bit private, 64-bit common, or 64-bit private.
*
* The maximum number of QW02172 or QW02173 sections in a single
* record is 200. If there are more than 200 QW02172 or QW02173
* sections to be reported then multiple IFCID217 records are
* generated in a sequence.
*
* DB2 will generate 1 or more IFCID217 records per statistics
* interval. The last IFCID 0217 in the sequence will contain
* a QW02174 member with QW02174S = 0.
*
* When activated, IFCID217 is recorded at 1 minute intervals.
......


QW0217QA DS    CL24      Authorization ID %U
QW02173N DS    0C        End of QW02173 mapping
*
* IRLM Storage Pools
QW02174  DSECT           Pointed to by QWT02R40
QW02174_PNM  DS  CL8     Pool Name
QW02174_CSEG DS  F       Current number of segments
QW02174_HSEG DS  F       High number of segments
```

```
QWO2174_PEX  DS  F       Number of pool expansions
QWO2174_PCM  DS  F       Number of pool compressions
             DS  CL8     Reserved
QWO2174_FLG1 DS  X       Flags
QWO2174S     EQU X'8O'   1 = More QWO2174 data will follow in one or
*                            more IFCID217
QWO2174E     EQU X'4O'   1 = Internal error occurred while gathering
*                            stats data. Data in this section is
*                            invalid.
QWO2174N     DS  OC      End of QWO2174 mapping
*                                                                    *
*********************************************************************
.............

* ! IFCID225 summarizes system storage usage
* ! The record is divided into data sections described as follows:
* !
* ! Data Section 1: Address Space Summary (QWO225)
* !                 This data section can be a repeating group.
* !                 It will report data for DBM1 and DIST
* !                 address spaces. Use QWO225AN to identify
* !                 the address space being described.
* ! Data Section 2: Thread information (QWO2252)
* ! Data Section 3: Shared and Common Storage Summary (QWO2253)
* ! Data Section 4: Statement Cache and xPROC Detail (QWO2254)
* ! Data Section 5: Pool Details (QWO2255)
* ! Data Section 6: IRLM Pool Details (QWO2256)
......

QWO225RP DS  D          ! Total RID pool storage
*                       !   (64-bit shared fixed pool)
QWO225CD DS  D          ! Total compression dictionary storage
*                       !   (64-bit DBM1 private GETMAINed)
*
* Data Section 6: IRLM Storage Information
QWO2256 DSECT
QWO225I_ABCSA   DS D      ! Total of all currently used 64-bit common
*                         ! storage in all IRLM 64-bit common pools
QWO225I_ABCSH   DS D      ! High water mark for 64-bit common storage
*                         ! requests of all 64-bit common IRLM pools
QWO225I_BBPVT   DS D      ! Total of all currently used 31-bit private
*                         ! storage in all IRLM 31-bit private pools
QWO225I_BBPVH   DS D      ! High water mark for 31-bit private storage
*                         ! requests of all 31-bit private IRLM pools
QWO225I_ABPVT   DS D      ! Total of all currently used 64-bit private
*                         ! storage in all IRLM 64-bit private pools
QWO225I_ABPVH   DS D      ! High water mark for 64-bit private storage
*                         ! requests of all 64-bit private IRLM pools
QWO225I_BBESCA  DS D      ! Total of all currently used ESCA storage

*                         ! in all IRLM ECSA pools
QWO225I_BBESCAH DS D      ! High water mark for ESCA storage requests
*                         ! of all ESCA IRLM Pools
*
*********************************************************************
```

```
DSNDQWPZ.copy will be changed as following to include IRLM private
storage limits.

*
********************************************************************
* IRLM processing parms.                                           *
********************************************************************
QWP5     DSECT
QWP5FLG  DS    X                 /* Process flags                 */
QWP5PCY  EQU   X'80'              /* 1=PC yes specified            */
         DS    XL3               /* Reserved                      */
QWP5DLOK DS    H                 /* Wait time for local deadlock  */
QWP5DCYC DS    H                 /* # of local cycles/global cycle */
QWP5TVAL DS    F                 /* Timeout interval              */
QWP5MCSA DS    F                 /* IRLM maximum CSA usage allowed */
QWP5HASH DS    F                 /* MVS lock table hash entries   */
QWP5PHSH DS    F                 /* Pending # Hash entries        */
QWP5RLE  DS    F                 /* MVS lock table list entries   */
         DS    F                 /* Reserved                      */
QWP5BPM  DS    D                 /* Maximum amount of 31-Bit IRLM
*                                   private storage available (out
*                                   of total 2G virtual storage
*                                   limit) for normal operations in
*                                   IRLM. IRLM reserves an
*                                   additional 10% of the total 2G
*                                   virtual storage, for use by
*                                   requests in IRLM.             */
QWP5APM DS     D                 /* Maximum amount of 64-Bit IRLM
*                                   private storage available (out
*                                   of total storage set as the
*                                   MEMLIMIT) for normal operations
*                                   in IRLM. IRLM reserves an
*                                   additional 10% of the total
*                                   MEMLIMIT storage, for use by
*                                   'must complete' requests in
*                                   IRLM.                        */
*
********************************************************************
```

A.6 Stored procedure monitoring overview and purpose

Stored procedure (SP) and user-defined function (UDF) performance and tuning analysis is typically performed by using a combination of IFCID3 and IFCID239. IFCID3 provides plan-level information and aggregates all executions of store procedures or UDFs into common fields. This method can create difficulty when tuning multiple procedures or functions that are executed in a given transaction.

IFCID239 is also used for performance and tuning analysis at the package level. This method provides better granularity than IFCID3 but still might not be sufficient for all transactions, because multiple package executions reported together, CPU, elapsed, and suspend time reflect averages across many stored procedure packages.

If a procedure or function is executed multiple times, the variation between executions cannot be identified. Instrumentation enhancements are needed. DB2 implements multiple IFCID enhancements to provide more effective performance and tuning analysis of stored procedures and UDF. they are:

- IFCID233 is written at the beginning and end of a stored procedure or UDF invocation. This record is enhanced with the invoking statement ID, the invoking statement type, the version ID (applies only to versioned procedures), and the routine ID.

  Note: The routine ID can be zero if a **REBIND** is not performed for packages containing CALL statements where the stored procedure name is a literal. See DSNDQW02 for mapping details

- New IFCIDs 380 and 381 are created for stored procedure and UDF detail respectively. These records have the following data sections:
  - Data section 1 is mapped by QW0233.
  - Data section 2 is mapped by QW0380, which includes CP, specialty engine, and elapsed time details for nested activity.

  You can use a series of 380 or 381 records to determine the amount of class 1 and class 2 CP, specialty engine, and elapsed time relative to the execution of a given stored procedure or user-defined function. See **DSNDQW05** for mapping details.
- New IFCIDs 497, 498, and 499 are created for statement level detail. These records track dynamic and static DML statements executed by a transaction, including those executed within a stored procedure or user-defined function. A series of IFCID 497, 498, or 499 records can be used to determine the statements executed for a given transaction. Note: Any packages containing static SQL statements that existed prior to DB2 10 must be rebound in DB2 10 New Function Mode (NFM), not necessarily with this APAR applied, to obtain a valid statement ID.
- A new performance class 24 is created to encapsulate IFCID380 and IFCID499 for stored procedure detail analysis (see Figure A-1).

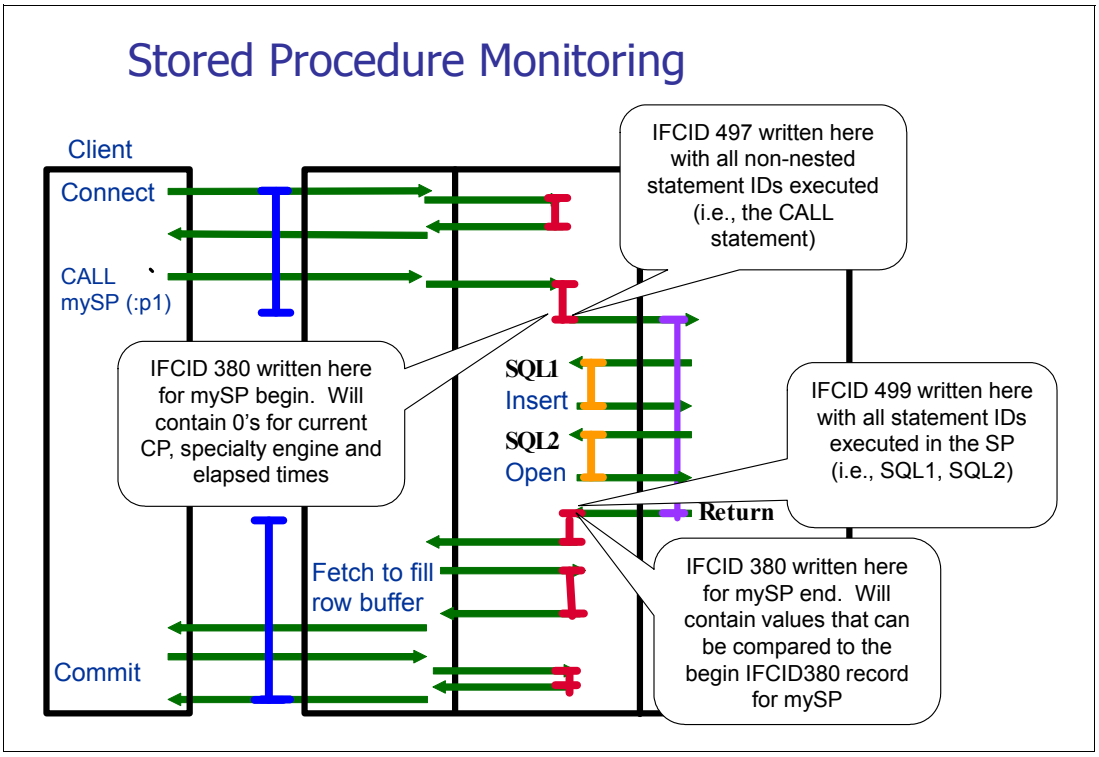


Figure A-1 Stored procedure monitoring

If you are interested in using the functions provided, consider the following actions:

- For a **CALL** statement to a DB2 for z/OS stored procedure, the stored procedure name can be identified by using a literal or by using a host variable or parameter marker. When using a literal for the stored procedure name and to benefit from the enhancement that provides a valid routine ID in various IFCID records, the packages that contain the **CALL** statement must be rebound.
- For an SQL statement that invokes a DB2 for z/OS UDF, and to benefit from the enhancement to provide a valid routine ID in various IFCID records, the packages that contain the SQL statement must be rebound.
- The mapping of IFCID233 remains compatible with prior versions and no immediate change is required. However, you need to change applications that parse this record to take advantage of the new fields.

See the following IFCIDs definitions for further details:

- QW0380
- QW0381
- QW0497
- QW0498
- QW0499

Example A-13 shows the details for QW0380.

Example A-13 IFCID0380

```
**************************************************************************
*   IFCID 380 is a stored procedure detail record.  It is written at     *
*   the beginning and the end of a CALL statement for both external      *
*   and native stored procedures.                                        *
*                                                                        *
```

```
*   The record contains 2 data sections mapped as follows:           *
*                                                                     *
*   Data section 1 will be mapped by qw0233 in DSNDQW03               *
*   Data section 2 will be mapped by qw0380 below                     *
*                                                                     *
*   Notes:                                                            *
*     1. The mapping of QW0380 is also used by IFCID381.              *
*     2. All times are in clock units.                                *
*     3. Times for IFCID380 reflect the total time at the time of     *
*        record write for all SP invocations. This includes SQLPL and *
*        WLM stored procedures.                                       *
*     4. Times for IFCID381 reflect the total time at the time of     *
*        record write for all UDF invocations.  This includes UDFs    *
*        executed on the main application execution unit and WLM      *
*        UDFs.                                                        *
*     5. Class 1 accounting must be enabled for this                  *
*        section to be written.                                       *
*     6. If class 2 accounting is not enabled, the class 2 counters   *
*        will be zero.                                                *
*     7. If class 1 and class 2 accounting are enabled during SP      *
*        or UDF execution, the below values may be inconsistent       *
*        (e.g., class 2 time may exceed class 1)                      *
*********************************************************************
QW0380  DSECT
QW0380_CLS1CP DS   CL8       Current total nested
*                            class 1 CP time. This does
*                            not include time spent
*                            executing on an IBM
*                            specialty engine.
QW0380_CLS1se DS   CL8       Current total nested
*                            class 1 specialty engine
*                            time.
QW0380_CLS2CP DS   CL8       Current total nested
*                            class 2 CP time. This is
*                            time in DB2 processing
*                            SQL statements. This time also
*                            includes in DB2 time needed to
*                            connect and disconnect the SP
*                            task for non-SQLP stored
*                            procedures. This does not
*                            include time spent executing
*                            on an IBM specialty engine.
QW0380_CLS2se DS   CL8       Current total nested
*                            class 2 specialty engine time.
*                            This is time in DB2 processing
*                            SQL statements.
QW0380_CLS2elap DS CL8       Current total nested
*                            elapsed class 2 time. This is
*                            time in DB2 processing
*                            SQL statements. This time also
*                            includes in DB2 time needed to
*                            connect and disconnect the SP
*                            task for non-SQLP stored
*                            procedures.
*
```

```
*********************************************************************
```

Example A-14 shows the details for QW0381.

Example A-14 QW0381 details

```
*********************************************************************
*   IFCID 381 is a UDF detail record.  It is written at             *
*   the beginning and the end of a UDF invocation.                  *
*   The record contains 2 data sections mapped as follows:          *
*                                                                   *
*   Data section 1 will be mapped by qw0233 in DSNDQW03             *
*   Data section 2 will be mapped by qw0380 above                   *
*********************************************************************
```

For QW0497 details, see Example A-15.

Example A-15 QW0497 details

```
*********************************************************************
*   IFCID 497 is the statement ID detail record for statements      *
*   executed outside of a stored procedure or UDF environment.      *
*   This typically would be referred to as a non-nested environment *
*   with the exception that statements executed by triggers on the  *
*   main application execution unit will be recorded in this record.*
*                                                                   *
*   It is mapped identically to IFCID499 and may be written for     *
*   reasons:                                                        *
*     QW0499OV                                                      *
*     QW0499AC                                                      *
*     QW0499SB                                                      *
*     QW0499UB                                                      *
*                                                                   *
*   Notes:                                                          *
*     1. Parallel child tasks will not externalize this IFCID.      *
*     2. Autonomous procedures will not externalize this IFCID.     *
*     3. For dynamic statements, only statements qualifying for the *
*        dynamic statement cache will be returned.                  *
*     4. For static statements, some statement IDs may not          *
*        correlate to IFCID401 static statement cache IDs.          *
*                                                                   *
*********************************************************************
```

Example A-16 shows the details for QW0498.

Example A-16 QW0498 details

```
*********************************************************************
*   IFCID 498 is the statement ID detail record for statements      *
*   executed inside of UDF environment. This includes WLM UDFs and  *
*   non-inline scalar functions.                                    *
*                                                                   *
*   It is mapped identically to IFCID499 and may be written for     *
*   reasons:                                                        *
*     QW0499OV                                                      *
*     QW0499SB                                                      *
```

```
*     QW0499UB                                                  *
*     QW0499UE                                                  *
*                                                               *
*   Notes:                                                      *
*     1. Parallel child tasks will not externalize this IFCID.  *
*     2. Autonomous procedures will not externalize this IFCID. *
*     3. For dynamic statements, only statements qualifying for the  *
*        dynamic statement cache will be returned.              *
*     4. For static statements, some statement IDs may not      *
*        correlate to IFCID401 static statement cache IDs.      *
*                                                               *
*****************************************************************
```

Example A-17 shows the details for QW0499.

Example A-17 QW0499 details

```
*****************************************************************
*   IFCID 499 is the statement ID detail record for statements  *
*   executed inside of stored procedure environment. This includes  *
*   WLM SPs and native stored procedures.                       *
*                                                               *
*   It is mapped below and may be written for reasons:          *
*     QW0499OV                                                  *
*     QW0499SB                                                  *
*     QW0499UB                                                  *
*     QW0499SE                                                  *
*                                                               *
*   Notes:                                                      *
*     1. Parallel child tasks will not externalize this IFCID.  *
*     2. Autonomous procedures will not externalize this IFCID. *
*     3. For dynamic statements, only statements qualifying for the  *
*        dynamic statement cache will be returned.              *
*     4. For static statements, some statement IDs may not      *
*        correlate to IFCID401 static statement cache IDs.      *
*                                                               *
*****************************************************************
*
* Data section 1
QW0499   DSECT              IFCID(QWHS0499)
QW0499RS DS  XL4            Reason IFCID was externalized
QW0499OV EQU X'00000001'    The maximum number of unique
*                           statement ID's were collected
QW0499AC EQU X'00000002'    The transaction/accounting
*                           interval is ending
QW0499SB EQU X'00000003'    A stored procedure is beginning
QW0499SE EQU X'00000004'    A stored procedure is ending
QW0499UB EQU X'00000005'    A UDF is beginning
QW0499UE EQU X'00000006'    A UDF is ending
*
* Data section 2 is a repeating group of each individual unique
* statement ID's
QW04992   DSECT
QW0499SID DS   CL8          Statement ID
QW0499NEC DS   D            Number of Executions
QW0499STY DS   CL2          Statement type
```

```
QWO499DY  EQU   X'8000'     Statement is dynamic
QWO499SC  EQU   X'4000'     Statement is static
QWO499CL  EQU   X'2000'     Statement is a CALL statement
         MEND                                                    */
*
```

A.7 Other accounting changes

The section describes the following accounting changes:

- Reduced NOT ACCOUNTED FOR time
- Specialty engine time in the CPU header
- Larger RBA and LRSN

A.7.1 Reduced NOT ACCOUNTED FOR time

DB2 further reduces the amount of **NOT ACCOUNTED FOR** times present in the following accounting records:

- Buffer Manger force write, which is the time is added to the existing Buffer Manager Class three buckets
- Parallel Query Parent/Child Synchronization, which is a New Class 3 bucket (qwac_pqs_wait/qwac_pqs_count) is added.
- Log Manager read, which is the time is added to the existing Log Manager Class 3 buckets

A.7.2 Specialty engine time in the CPU header

DB2 adds a field to the CPU Header (**DSNDQWHU**) to quickly determine CPU time that was spent running on a Specialty Engine. See Example A-18.

Example A-18 CPU time that was spent running on a Specialty Engine

```
*     /*INSTRUMENTATION CPU HEADER DATA  */
QWHULEN  DS     XL2    /* LENGTH OF HEADER                      */
QWHUTYP  DS     XL1    /* TYPE OF HEADER - CPU MAPPED QWHSHU08  */
         DS     XL1    /* RESERVED                              */
QWHUCPU  DS     XL8    /* CPU time of the currently dispatched  */
*                      /* execution unit (TCB or SRB).  This    */
*                      /* time includes CPU consumed on an IBM  */
*                      /* specialty engine. Binary zero         */
*                      /* indicates CPU time was not available. */

QWHUCNT  DS     XL2    /* (S) COUNT FIELD RESERVED              */
QWHUSE   DS     XL8    /* CPU time of the currently dispatched  */
*                      /* execution unit (TCB or SRB) consumed  */
*                      /* on an IBM speciailty engine.          */
*                      /* Note: A given ACE token may           */
*                      /* run under one or more MVS dispatchable*/
*                      /* execution units. Thus the CPU time for*/
*                      /* a given ACE may decrease between      */
*                      /* events. This is true for both QWHUCPU */
*                      /* and QWHUSE.                           */
```

```
QWHUEND  DS     0C
```

A.7.3 Larger RBA and LRSN

Numerous instrumentation changes are required to support a 10-byte RBA/LRSN. Any application parsing the following records needs to be modified to fully take advantage of the new RBA size in these records. An attempt was made to maintain the offsets of other record fields when possible. The following records or mappings in Example A-19 have incompatible changes with prior releases:

- DSNDWQAL (while offsets are not changed for existing fields, applications using the pre-existing WQALLRBA offset ceases to function)
- DSNDIFCA for IFCID129 and IFCID306 requests
- IFCID32
- IFCID34
- IFCID36
- IFCID38
- IFCID39
- IFCID43
- IFCID114
- IFCID119
- IFCID126
- IFCID129
- IFCID185
- IFCID203
- IFCID261
- IFCID306
- IFCID335

Other record changes result in parsing applications reading in zeroes for RBA/LRSN fields using the old offsets.

Example A-19 Incompatible changes for new RBA/LRSN

```
WQALLN6  EQU   264       /* VERSION 6           LENGTH -            */
*                        /*  INCLUDES EUID, EUTX, EUWN              */
*                        /*  FIELDS                                 */
WQALLN9  EQU   920       /* Version 9 length                        */
WQALLN11 EQU   1024      /* Version 11 length                       */
...
WQALCDCD DS    CL1       /* DB2 CHANGED DATA CAPTURE REQUEST FLAG
*                        /*   Y - DATA DESCRIPTION RETURNED FOR EACH */
*                        /*       READS 185 (OTHER THAN REDRIVE      */
*                        /*       REQUESTS) FOR A NEW TABLE          */
*                        /*   N - NO DATA DESCRIPTIONS WILL BE       */
*                        /*       RETURNED.                          */
*                        /*   A - A DATA DESCTIPTION WILL ONLY BE    */
*                        /*       RETURNED THE FIRST TIME OR WHEN    */
*                        /*       IT IS CHANGED FOR A GIVEN TABLE.   */
*                        /*       THIS IS THE DEFAULT.               */
*                        /* SEE IFCID 185 IN DSNDQW02.  DEFINITION   */
*                        /* OF DATA DESCRIPTION BEGINS WITH FIELD    */
*                        /* QW0185DD                                 */
         DS    CL9       /* Reserved                                */
...
```

```
WQALFVAL64 DS  D         /* 64-bit threshold value for IFCID 316 and */
*                        /* 401 requests. If this is non-zero and    */
*                        /* the target system is v10 or higher, this */
*                        /* value will be used in place of WQALFVAL. */
WQALLRBA   DS   0CL12
WQALRBAM   DS   CL2      /* For IFCID 306, this is the member ID     */
*                        /*  for a WQALMODD call                     */
WQALRBA10  DS   CL10     /* For IFCID 129, this is the starting      */
*                        /*  log RBA of the CI(s) to be returned     */
*                        /* For IFCID 306, this is the log RBA or    */
*                        /*  LRSN to be used in a WQALMODF or        */
*                        /*  WQALMODD call                           */
           DS   CL100    /* Reserved                                 */
WQALEND  DS    0F        /* END OF BLOCK                             */
```

A.7.4 Buffer manager force write

Buffer manager force write Class 3 accounting times are added to the existing QWACAWTW and QWACARNW fields. In addition, Example A-20 shows the details of these IFCIDs.

Example A-20 IFCID127 and IFCID 128

```
QW0127F     DS      C        FLAG FOR TYPE OF I/O
QW0127FR    EQU     C'R'     READ I/O WAIT
QW0127FW    EQU     C'W'     WRITE I/O WAIT
QW0127FF    EQU     C'F'     BUFFER MANAGER FORCE WRITE I/O WAIT
QW0127BP    DS      F        BUFFER POOL INTERNAL ID (049
QW0128F     DS      C        FLAG FOR TYPE OF I/O
QW0128FR    EQU     C'R'     READ I/O WAIT
QW0128FW    EQU     C'W'     WRITE I/O WAIT
QW0128FF    EQU     C'F'     BUFFER MANAGER FORCE WRITE I/O WAIT
QW0128AC    DS      F        ACE TOKEN OF ACTUAL REQUESTOR.
```

A.7.5 Parallelism performance enhancement

The IFCIDs listed in the following tables were changed to track the effect of changes to the degree of parallelism after parallel system negotiation that occurs because of resource constraints.

A.7.5.1 Accounting and statistics

Table A-1 lists the accounting and statistics IFCIDs.

Table A-1 The QXST control block size is enlarged

Name	Description
QXSTOREDGRP	Number of parallel group degree be reduced due to system negotiation result of system stress level
QXSTODGNGRP	Number of parallel group is degenerated to sequential due to system negotiation result of system stress level

Name	Description
QXMAXESTIDG	Maximum parallel group estimated degree. It is the bind time estimated degree based on the cost formula. If the parallel group contains a host variable or parameter marker, then bind time estimates the parallel group degree based on a valid assumption value.
QXMAXPLANDG	Maximum parallel group plan degree. It is the ideal parallel group degree obtained at execution time after the host variable or parameter marker value is plug-in and before buffer pool negotiation and system negotiation are performed.
QXPAROPT	Serviceability

A.7.5.2 IFCID 221

Table A-2 lists IFCID 221 details.

Table A-2 IFCID221

Name	Description
QW0221STOLEV	Serviceability
QW0221STOMAP	Serviceability

A.7.5.3 IFCID 225

Table A-3 lists IFCIS 225 details.

Table A-3 IFCID225

Name	Description
QW0225_RS	Serviceability

A.7.5.4 IFCID 316

Table A-4 lists IFCID 316 details.

Table A-4 IFCID316, The QW0316 control block size is enlarged.

Name	Description
QW0316AVGESTI	Average of parallel group estimated degree. Estimated degree is the bind time estimated parallel group degree based on the cost formula. If the parallel group contains a host variable or parameter marker, then bind time estimates the parallel group degree based on a valid assumption value.
QX0316AVGPLAN	Average of parallel group plan degree. Plan degree is the ideal parallel group degree obtained at execution time after the host variable or parameter marker value is plug-in and before buffer pool negotiation and system negotiation are performed.
QW0316AVGACT	Average of parallel group actual degree. The actual degree is obtained at execution time after consider the buffer pool negotiation and system negotiation.

A.7.5.5 IFCID 401

Table A-5 lists IFCID 401 details.

Table A-5 IFCID401, The QW0401 control block size is enlarged.

Name	Description
QW0401AVGESTI	Average of parallel group estimated degree. Estimated degree is the bind time estimated parallel group degree based on the cost formula. If the parallel group contains a host variable or parameter marker, then bind time estimates the parallel group degree based on a valid assumption value.
QW0401AVGPLAN	Average of parallel group plan degree. Plan degree is the ideal parallel group degree obtained at execution time after the host variable or parameter marker value is plug-in and before buffer pool negotiation and system negotiation are performed.
QW0401AVGACT	Average of parallel group actual degree. The actual degree is obtained at execution time after consider the buffer pool negotiation and system negotiation.

A.7.6 Temporal support

The following IFCIDs were changed to indicate the impacts of the **CURRENT TEMPORAL BUSINESS_TIME** special register, the **CURRENT TEMPORAL SYSTEM_TIME** special register, and the **SYSIBMADM.GET_ARCHIVE** built-in global variable:

- 0053
- 0058
- 0059
- 0060
- 0061
- 0064
- 0065
- 0066
- 0401

See Table A-6.

Table A-6 QW00xxER

Name	Description
QW00xxER	**SB** The query contains implicit query transformation driven by the CURRENT TEMPORAL SYSTEM_TIME special register and the CURRENT TEMPORAL BUSINESS_TIME special register. **Blank** The query does not contain implicit query transformation driven by the SYSIBMADM.GET_ARCHIVE built-in global variable, the CURRENT TEMPORAL BUSINESS_TIME special register, or the CURRENT TEMPORAL SYSTEM_TIME special register.

A.7.7 IFCID 002/225: Arrays support

Example A-21 lists the changes to IFCID 002 and 225 to support arrays.

Example A-21 Changes IFCID 002/225 to record arrays support

```
IFC ID 0002 IS RESERVED FOR DATA BASE STATISTICS RECORDS AND IS
MAPPED BY MACRO DSNDQWST SUPTYPE=1

! IFCID225 summarizes system storage usage
```

```
QW0225AR DS  D          ! Total array variable storage *
```

A.7.8 IFCID 003/239: Autonomous transaction support

Example A-22 lists the changes to IFCID 003 and 239 to support autonomous transactions.

Example A-22 Changes IFCID 003/239 to record autonomous transactions

```
IFC ID 0003 IS RESERVED FOR ACCOUNTING  RECORDS AND IS
MAPPED BY MACRO DSNDQWAS

************************************************************************
*  IFCID 0239 FOR RMID 26                                              *
*    THIS RECORD IS WRITTEN WHEN A PACKAGE/DBRM ACCOUNTING INFORMATION*
*    IS AVAILABLE FOR MORE THAN 10 PACKAGES/DBRMs.                     *
*  SEE DSNDQWAS FOR THE MAPPING OF IFCID 239.                          *
************************************************************************
```

A.7.9 IFCID 366: Application incompatibility

To reference the new longer lengths values, the application (package) bind option, **APPLCOMPAT** must specify the value, V11R1. If the application **APPLCOMPAT** bind option does not specify V11R1 (for example, V10R1) the application continues to reference the shorter (truncated) client information values.

If the application **APPLCOMPAT** bind option is not set, the value of the **APPLCOMPAT** bind option is defaulted to the DB2 subsystem parameter **APPLCOMPAT** (**DSN6PRM**) value.

Note: Applications that wants to retrieve the new longer lengths client information special register values (for example, using the **SET** host-variable SQL statement) need to ensure the target (receiving) host variable is defined to be large enough to receive the maximum length of the new longer length values. If this is not performed, your application receives a warning message. An IFCID366 trace record is also recorded to indicate such incompatibility has been detected by DB2.

Example A-23 lists the changes to IFCID 366 to record application incompatibility.

Example A-23 Changes IFCID 366 to record application incompatibility

```
************************************************************************
**  IFCID 0366 is a serviceability trace. It can be used to identify  *
**  applications that are affected by incompatible changes.           *
**  The QW0366FN field indicates the type of incompatible change:     *
**                                                                    *
**  QW0366FN = 1                                                      *
**
**  QW0366FN = 2                                                      *
**                                                                    *
** QW0366FN = 3
**                                                                    *
**  QW0366FN = 1101                                                   *
**    Indicates that the INSERT statement that inserts into an XML    *
**    column without XMLDOCUMENT function has been processed (which   *
**    should result in SQLCODE -20345 when run on DB2 release prior   *
```

```
**    to DB2 11). Starting with DB2 11, SQL error will no longer be   *
**    issued.                                                         *
**    Application will no longer recieve SQLCODE for this statement.  *
**                                                                    *
**  QWO366FN = 1102                                                   *
**    Indicates that V10 XPath evaluation behavior was in effect which*
**    resulted in an error. For instance, a data type conversion error*
**    could have occured for a predicate that would otherwise be      *
**    evaluated to false. tarting from DB2 11,such "irrelevant" errors*
**    might be suppressed so an application might no longer recieve   *
**    the SQLCODE for this statement.                                 *
**                                                                    *
**  QWO366FN = 1103                                                   *
**    Indicates that a dynamic SQL uses the ASUTime limit that has    *
**    been set for the entire thread for RLF reactive governing.      *
**    For instance, when a dynamic SQL is processed from package A,   *
**    if the ASUTime limit is already set during other dynamic SQL    *
**    processing from package B in the same thread, the SQL from      *
**    package A will use the ASUTime limit set during the SQL         *
**    processing from package B.Stating with DB2 11, dynamic SQLs from*
**    multiple packages will use the ASUTime limit that is set        *
**    considering its own package information.                        *
**                                                                    *
**  QWO366FN = 1104, 1105, 1106, 1107                                 *
**    Indicates that CLIENT special register (CLIENT_USERID,          *
**    CLIENT_WRKSTNNAME, CLIENT_APPLNAME, CLIENT_ACCTNG) has been set *
**    to a value that is longer than what is supported prior to DB2 11*
**    A shorter value has been used instead.                          *
**                                                                    *
**  QWO366FN = 1108                                                   *
**    Indicates that CLIENT special register (CLIENT_USERID,          *
**    CLIENT_WRKSTNNAME, CLIENT_APPLNAME, CLIENT_ACCTNG) has been set *
**    to a value that is longer than what is supported prior to DB2 11*
**    Truncated values upto the supported lengths prior to DB2 11 have*
**        been used for RLF table search instead.                     *
**                                                                    *
**  QWO366FN = 1109                                                   *
**    Indicates that CAST(string AS TIMESTAMP) was processed for the  *
**    input string of length 8 and input was treated as a store clock *
**    value (or input string was of length 13 and was treated as a    *
**    GENERATE_UNIQUE value). This behavior is incorrect for a CAST   *
**    and is valid for TIMESTAMP built-in function only. This behavior*
**    is being corrected in DB2 11 so that input to CAST is not       *
**    treated as a store clock value nor GENERATE_UNIQUE.             *
**                                                                    *
**  QWO366FN = 1110                                                   *
**    Indicates the integer argument of SPACE function is greater     *
**    than 32764.                                                     *
**                                                                    *
**  QWO366FN = 1111                                                   *
**    Indicates the optional integer argument of VARCHAR function     *
**    has a value greater than 32764.                                 *
**********************************************************************
**********************************************************************
QWO366       DSECT
```

```
QW0366FN     DS F            Incompatible change indicator
*.........................QW0366FN CONSTANTS.......................
C_QW0366_CHAR     EQU 0001   V9 SYSIBM.CHAR(decimal-expr) function
C_QW0366_VCHAR    EQU 0002   V9 SYSIBM.VARCHAR(decimal-expr) function
*                            CAST (decimal as VARCHAR or CHAR)
C_QW0366_TMS      EQU 0003   Unsupported character string
*                            rpresentation of a timestamp
C_QW0366_XMLINS   EQU 1101   Insert into an XML column without
*                            XMLDOCUMENT function
C_QW0366_XPATHERR EQU 1102   XPath evaluation error
C_QW0366_RLF      EQU 1103   RLF governing
C_QW0366_CLIENTAC EQU 1104   Long CLIENT_ACCTNG Special Reg value
C_QW0366_CLIENTAP EQU 1105   Long CLIENT_APPLNAME Special Reg value
C_QW0366_CLIENTUS EQU 1106   Long CLIENT_USERID Special Reg value
C_QW0366_CLIENTWK EQU 1107   Long CLIENT_WRKSTNNAME Special Reg value
C_QW0366_CLIENTSR EQU 1108   Long client Special Reg value for RLF
C_QW0366_TMSCAST  EQU 1109   CAST(string AS TIMESTAMP)
C_QW0366_SPACEINT EQU 1110   SPACE integer argument greater than 32764
C_QW0366_VCHARINT EQU 1111   VARCHAR int argument greater than 32764
*.....................................................................
QW0366SN     DS F            Statement number of the query
QW0366PL     DS CL8          Plan name for this query
QW0366TS     DS CL8          Timestamp for this query
QW0366SI     DS CL8          Statement Identifier
QW0366TY     DS XL2          Statement information
*.........................QW0366TY CONSTANTS.......................
C_QW0366DYN  EQU X'8000'       Statement is dynamic
C_QW0366STC  EQU X'4000'       Statement is static
*.....................................................................
QW0366SE     DS H            Section number
QW0366PC_Off DS H            Offset from QW0366 to Package
*                            Collection ID
QW0366PN_Off DS H            Offset from QW0366 to
*                            Program name
QW0366VER    DS 0C           Package Version
QW0366VL     DS H            Version length
QW0366VN     DS CL64         Version name
*
QW0366PC_D   DSECT
QW0366PC_Len DS H            Length of Package Collection ID
QW0366PC_Var DS 0CL128       %U Package Collection ID
*
QW0366PN_D   DSECT
QW0366PN_Len DS H            Length of Program Name
QW0366PN_Var DS 0CL128       %U Program Name
*
```

A.7.10 IFCID 230/256: Castout enhancements

Example A-24 lists the changes to IFCID 230 and 256 to record class castout queue threshold values, based on the number of pages.

Example A-24 Changes IFCID 230/256 to record castout queue threshold

```
**********************************************************************
```

```
*   IFCID 0256 FOR RMID 10 TO RECORD THE EFFECTS OF AN ISSUED
*        DB2 -ALTER GROUPBUFFERPOOL COMMAND.
************************************************************************
*
QW0256   DSECT               IFCID(QWHS0256)
QW0256GB DS    CL8           GROUP BUFFER POOL NAME
QW0256OR DS    CL6           OLD DIRECTORY TO DATA RATIO VALUE
QW0256OC DS    XL1           OLD CLASST VALUE
QW0256OG DS    XL1           OLD GBPOOLT VALUE
QW0256OK DS    XL4           OLD GBPCHKPT VALUE
QW0256NR DS    CL6           NEW DIRECTORY TO DATA RATIO VALUE
QW0256NC DS    XL1           NEW CLASST VALUE
QW0256NG DS    XL1           NEW GBPOOLT VALUE
QW0256ON DS    XL2           OLD CLASST (BUF-NUM BASED)
QW0256NN DS    XL2           NEW CLASST (BUF-NUM BASED)
QW0256NK DS    XL4           NEW GBPCHKPT VALUE
QW0256OA DS    CL1           OLD AUTOREC SETTING rev code a
QW0256NA DS    CL1           NEW AUTOREC Setting rev code a
QW0256OB DS    CL1           Old GBPCACHE setting
QW0256NB DS    CL1           New GBPCACHE setting
QW0256AY EQU   C'Y'          AUTOREC or GBPCACHE (YES)
QW0256AN EQU   C'N'          AUTOREC or GBPCACHE (NO)
QW0256EN DS    0C            END OF QW0256
```

B

Summary of relevant maintenance

With a new version of DB2 reaching general availability, the maintenance stream becomes extremely important. Feedback from early users and development of additional functions cause a flux of APARs that enrich and improve the product code.

This appendix describes the following recent maintenance for DB2 11 for z/OS:

- DB2 APARs
- z/OS APARs
- OMEGAMON/PE APARs

These APARs represent a snapshot of the current maintenance at the time of writing. For an up-to-date list, ensure sure that you contact your IBM Service Representative for the most current maintenance at the time of your installation. Also check on IBM RETAIN for the applicability of these APARs to your environment and to verify prerequisites and post-requisites.

Use the Consolidated Service Test (CST) as the base for service as described at:

http://www.ibm.com/systems/z/os/zos/support/servicetest/

DB2 11 is now included in the RSU.

The most recent planned quarterly RSU is CST1Q14 (RSU1403), dated April 4 2014 for DB2 10 and DB2 11. This addendum is based on all service through the end of December 2013 not already marked RSU, PE resolution and HIPER/Security/Integrity/Pervasive PTFs and their associated requisites and supersedes through the end of February 2014.) as described at.

http://www.ibm.com/systems/resources/RSU1312.pdf

B.1 DB2 APARs

Table B-1 lists the APARs that provide functional and performance enhancements to DB2 11 for z/OS. This list is not and cannot be exhaustive; check **RETAIN** and the DB2 website for a complete list.

Table B-1 DB2 10 current function and performance related APARs

APAR #	Area	Text	PTF and notes
II10817	Storage	Info APAR for storage usage error	
II11334	TCP/IP	Info APAR for Communication Server	
II14219	zIIP	zIIP exploitation *support use* information	
II14334	LOBs	Info APAR to link together all the LOB support delivery APARs	
II14426	XML	Info APAR to link together all the XML support delivery APARs	
II14441	Incorrout PTFs	Preferred DB2 9 SQL INCORROUT PTFs	
II14587	Workfile	DB2 9 and 10 work file recommendations	
II14619	Migration	Info APAR for DB2 10 DDF migration	
II14660	V11 migration	Info APAR to link together all the migration APARs to V11	
PM31841	V11 Migration	Toleration of fall back to V10	UK96357 V10
PM45652	Migration	prefix.SDSNLINK lib	UK74535 V10
PM80004	DDF	Synchronous Receive	UK92097
PM84765	IRLM	New option (QWAITER) to QUERYFST request used by DB2	UK92494
PM85053	IRLM	IRLM enhancement for DB2 V11 to suppress unnecessary child lock propagation to the CF lock structure	UK92783
PM89117	V11 Migration	New functions	UK95677 V10
PM89655	DSNZPARM	Restrictions for IXcontrolled TS PREVENT_NEW_IXCTRL_PART and PREVENT_ALTERTB_LIMITKEY	UK98189 also V10
PM91565	Premigration DSNTIJPM	SQLCODE -104	UK95419
PM92730	DSNTIJMV	Corrections to job for migration	UK98196 also V9, V10
PM93577	Query in DSNESQ	DSNTESQ insert INS32 needs to be updated.	UK98216 also V9, V10
PM94681	ADMIN_INFO_SQL	Collection features, enhancements, and service	OPEN

APAR #	Area	Text	PTF and notes
PM94715	ENFM	Improve step ENFM001. Systems with a large number of rows in SYSIBM.SYSCOLUMNS the DSNTIJEN step ENFM0001 can take longer to complete.	UK97335
PM95294	ALTER	Reduce sync getpageS against DSNTPX01 index of SYSCOLDISTSTATS table.	UK97912 also V10
PM95929	Thread break in	The need to break in for BIND/DDL activity - Early code	UI13368 also V10
PM96001	Thread break in	Toleration code for all V11 members	UI12985 also V10
PM96004	Thread break in	Enabling code	UI12985 also V10

B.2 z/OS APARs

Table B-2 lists the APARs that provide additional enhancements for z/OS. This list is not and cannot be exhaustive; check **RETAIN** and the DB2 website for a complete list.

Table B-2 z/OS DB2-related APARs

APAR #	Area	Text	PTF and notes
OA37550	Coupling Facility	Performance improvements are needed for coupling facility cache structures to avoid flooding the coupling facility cache with changed data and avoid excessive delays and backlogs for cast-out processing.	UA66419
OA39392	CALLRTM TYPE=SRBTERM	Terminate a pre-emptable SRB in the -CANCEL THREAD with FORCE option	UA66823
OA40967	RSM Enablement Offering	2 GB frame support	UA68169
OA41617	IGX00031/IGX00032 modules	DFSMS control block accessing support for `NON_VSAM_XTIOT = YES` in DEVSUPxx	UA69320

B.3 OMEGAMON PE APARs

Table B-3 lists the APARs that provide additional enhancements for IBM Tivoli OMEGAMON XE for DB2 PE on z/OS V5.2.0, PID 5655-W37. This list is not and cannot be exhaustive; check **RETAIN** and the DB2 tools website for a complete list.

Table B-3 OMEGAMON PE GA and DB2 related APARs

APAR #	Area	Text	PTF and notes
II14438		Info APAR for known issues causing high CPU utilization.	

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks publications

The following IBM Redbooks publications provide additional information about the topic in this document. Note that some publications referenced in this list might be available in softcopy only.

- *DB2 10 for z/OS Technical Overview*, SG24-7892
- *Extremely pureXML in DB2 10 for z/OS*, SG24-7915
- *DB2 for z/OS and List Prefetch Optimizer*, REDP-4862
- *Subsystem and Transaction Monitoring and Tuning with DB2 11 for z/OS*, SG24-8182

You can search for, view, download or order these documents and other Redbooks, Redpapers, Web Docs, draft and additional materials, at the following website:

ibm.com/redbooks

Other publications

These publications are also relevant as further information sources:

- *DB2 11 for z/OS Administration Guide*, SC19-4050
- *DB2 11 for z/OS Application Programming and SQL Guide*, SC19-4051
- *DB2 11 for z/OS Application Programming Guide and Reference for Java*, SC19-4052
- *DB2 11 for z/OS Codes*, GC19-4053
- *DB2 11 for z/OS Command Reference*, SC19-4054
- *DB2 11 for z/OS Data Sharing: Planning and Administration*, SC19-4055
- *DB2 11 for z/OS Installation and Migration*, SC19-4056
- *DB2 11 for z/OS Internationalization Guide*, SC19-4057
- *Introduction to DB2 for z/OS*, SC19-4058
- *DB2 11 for z/OS DB2 11 for z/OS IRLM Messages and Codes for IMS and DB2 for z/OS,* GC19-2666
- *DB2 11 for z/OS Managing Performance*, SC19-4060
- *DB2 11 for z/OS Managing Security*, SC19-4061
- *DB2 11 for z/OS Messages*, GC19-4062
- *DB2 11 for z/OS ODBC Guide and Reference*, SC19-4063
- *DB2 11 for z/OS pureXML Guide*, SC19-4064
- *DB2 11 for z/OS RACF Access Control Module Guide*, SC19-4065
- *DB2 11 for z/OS SQL Reference*, SC19-4066

- *DB2 11 for z/OS Utility Guide and Reference*, SC19-4067
- *DB2 11 for z/OS What's New?*, GC19-4068
- *DB2 11 for z/OS Diagnosis Guide and Reference,* LY37-3222

Online resources

These websites are also relevant as further information sources:

- DD2 11 for z/OS

  http://www-01.ibm.com/software/data/db2/zos/family/db211/
- DB2 Information Management Tools and DB2 11 for z/OS Compatibility

  http://www-01.ibm.com/support/docview.wss?uid=swg21609691

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

Redbooks

IBM DB2 11 for z/OS Technical Overview

(1.0” spine)
0.875”<->1.498”
460 <-> 788 pages

IBM DB2 11 for z/OS Technical Overview

Understand the synergy with System z platform

Enhance applications and avoid incompatibilities

Run business analytics and scoring adapter

IBM DB2 Version 11.1 for z/OS (DB2 11 for z/OS or just DB2 11 throughout this book) is the fifteenth release of DB2 for MVS. It brings performance and synergy with the new System z hardware and new opportunities to drive business value in the following areas:

- Unmatched reliability, availability, and scalability
  - Improved data sharing performance and efficiency
  - Even less downtime by removing growth limitations
  - Simplified management, improved autonomics, and reduce planned outages with more online schema changes and utilities improvements
- Save money, save time
  - Aggressive CPU reduction goals
  - Additional utilities performance and CPU improvements
  - Save time and resources with new autonomic and application development capabilities
- Simpler, faster migration
  - SQL compatibility, divorce system migration from application migration
  - Access path stability improvements
  - Better application performance with SQL and XML enhancements
- Enhanced business analytics
  - Faster, more efficient performance for query workloads
  - Accelerator enhancements
  - More efficient inline database scoring enables predictive analytics

The DB2 11 environment is available either for brand new installations of DB2, or for migrations from DB2 10 for z/OS subsystems only.

This IBM Redbooks publication introduces the enhancements made available with DB2 11 for z/OS. The contents help database administrators understand the new functions and performance enhancements, start planning for exploiting the key new capabilities, and justify the investment in installing or migrating to DB2 11.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-8180-00 ISBN 0738439053