DB2 Information Management Software

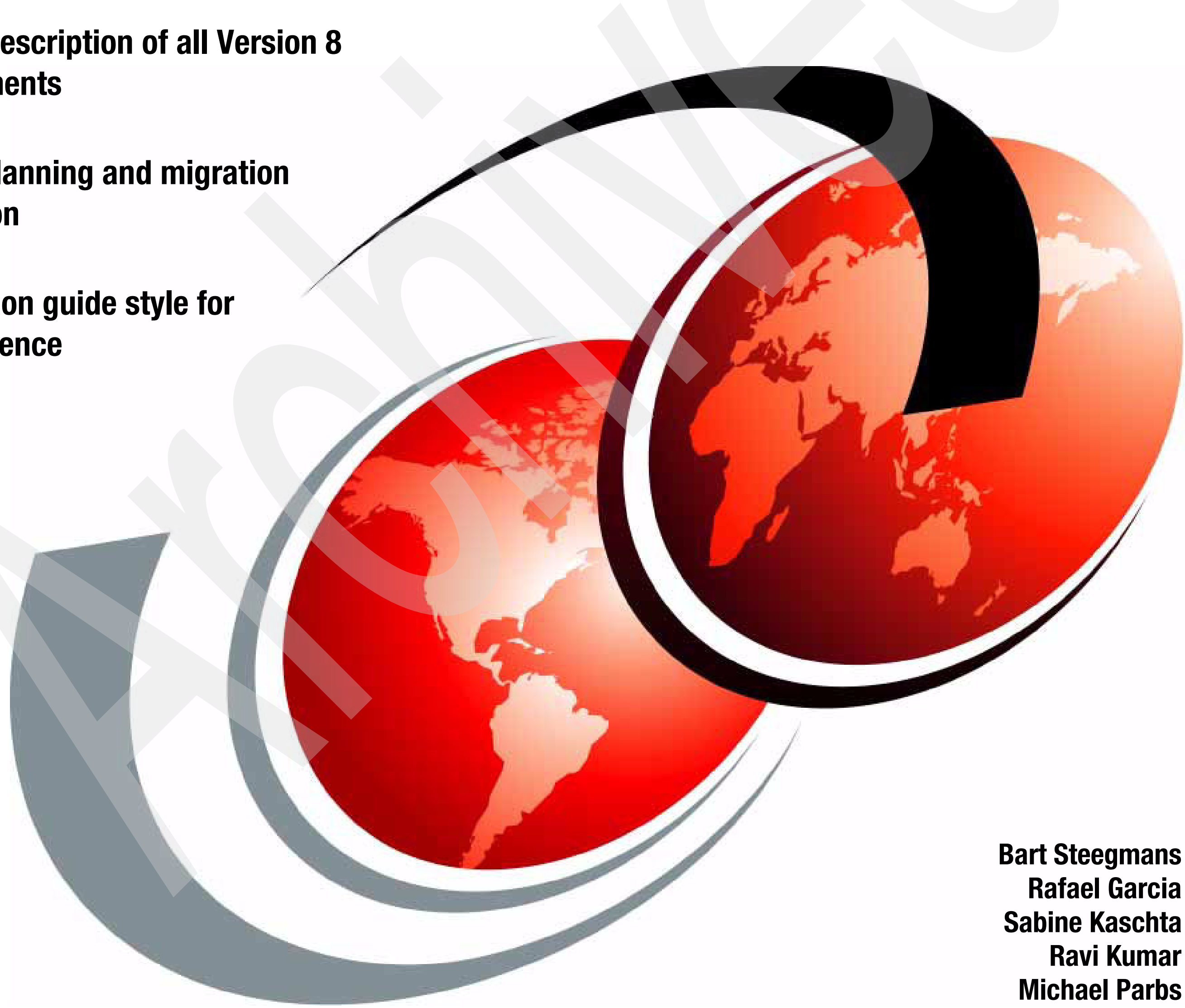
IBM

DB2 UDB for z/OS Version 8: Everything You Ever Wanted to Know, ... and More

Detailed description of all Version 8 enhancements

Release planning and migration information

Presentation guide style for easy reference



Bart Steegmans
Rafael Garcia
Sabine Kaschta
Ravi Kumar
Michael Parbs

Redbooks

ibm.com/redbooks

International Technical Support Organization

DB2 UDB for z/OS Version 8: Everything You Ever Wanted to Know, ... and More

May 2004

SG24-6079-00

Note: Before using this information and the product it supports, read the information in "Notices" on page xix.

First Edition (May 2004)

This edition applies to Version 8 of DB2 UDB for z/OS (product number 5625-DB2).

Note: This book is based on a pre-GA version of DB2 UDB for z/OS Version 8. Some functions described may not apply when the product becomes generally available. We recommend that you consult the product documentation or follow-on versions of this redbook for more current information.

Contents

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 give 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 Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites 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.

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 illustrates 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. 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 IBM's application programming interfaces.

Trademarks

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

AD/Cycle®
AS/400®
BookManager®
C/370™
CICS®
Cloudscape™
DB2 Connect™
DB2 Extenders™
DB2 Universal Database™
DB2®
DFSMSdss™
DFSMShsm™
DFSORT™
Distributed Relational Database Architecture™
DRDA®
eserver ™
eserver™
FlashCopy®
GDDM®
IBM®
ibm.com®
IMS™
iSeries™
Language Environment®
Lotus®
Multiprise®
MVS™
Net.Data®
OS/2®
OS/390®
Parallel Sysplex®
pSeries®
QMF™
RACF®
Redbooks™
Redbooks(logo) ™
RETAIN®
RMF™
SAA®
SQL/DS™
VTAM®
WebSphere®
xSeries®
z/Architecture™
z/OS®
zSeries®
1-2-3®

The following terms are trademarks of other companies:

Intel and Intel Inside (logos) are trademarks of Intel Corporation in the United States, other countries, or both.

Microsoft, Windows, Windows NT, 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 Sun Microsystems, Inc. in the United States, other countries, or both.

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

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

Preface

IBM® DATABASE 2 Universal Database Server for z/OS® Version 8 (DB2® V8 throughout this publication) is the twelfth and largest release of DB2 for MVS™. It brings new synergy with the zSeries® hardware and is the first major subsystem to exploit the z/OS 64-bit virtual addressing capabilities. DB2 V8 offers data support, application development, and query functionality enhancements for e-business, while building upon the traditional characteristics of availability, exceptional scalability, and performance for the enterprise of choice. The DB2 V8 environment is available only for the z/OS platform, either for brand new installations of DB2, or for migrations exclusively from DB2 UDB for OS/390® and z/OS Version 7 subsystems.

DB2 Version 8 has been re-engineered for e-business, with many fundamental changes in architecture and structure. Key improvements enhance scalability, application porting, security architecture, and continuous availability. Management for very large databases is made much easier, while 64-bit virtual storage support makes management simpler and improves scalability and availability. This new version breaks through many old limitations in the definition of DB2 objects, including SQL improvements, schema evolution, longer names for tables and columns, longer SQL statements, enhanced Java™ and Unicode support, enhanced utilities, more log data sets, and many more advantages.

This redbook introduces the many changes and enhancements made available with DB2 V8. It will help you understand the functions offered by DB2 V8, and provides information to help you to evaluate their applicability to your environment, as well as to plan for the installation of DB2 V8 or the migration from DB2 V7.

The team that wrote this redbook

This IBM Redbook™ was produced by a team of specialists from around the world working at the International Technical Support Organization, San Jose Center.

Bart Steegmans is a DB2 Product Support Specialist from IBM Belgium on assignment as a Data Management for z/OS Specialist at the ITSO, San Jose Center. He has over 14 years of experience in DB2. Before joining IBM in 1997, Bart worked as a DB2 system administrator at a banking and insurance group. His areas of expertise include DB2 performance, database administration, and backup and recovery.

Rafael Garcia has been in the IT business for 20 years and has held various positions. He was a COBOL and CICS® developer, an application development manager, and a DB2 applications DBA for one of the top 10 banks in the US. For the last seven years he has been a Field DB2 Technical Specialist working for the IBM Silicon Valley Laboratory supporting DB2 for OS/390 customers across various industries, including migrations to data sharing. He has an associate's degree in Arts and an associate's degree in Science in Business Data Processing from Miami-Dade Community College.

Sabine Kaschta is a DB2 Specialist working for IBM IT Education Services in Germany as an education consultant. She has 13 years of experience working with DB2. Before she joined IBM in 1998, she worked for a third-party vendor providing second-level support for DB2 utility products. She is also experienced in DB2 system programming and client/server implementations within an insurance practice in Germany. She is also a co-author of the IBM Redbooks, *DB2 UDB Server for OS/390 Version 6 and Continuous Availability*, SG24-5486, and *Cross-Platform DB2 Distributed Stored Procedures: Building and Debugging*, SG24-5485-01.

Ravi Kumar is a Senior Instructor and Specialist for DB2 with IBM IT Education Services Australia. He has over 18 years of experience in DB2. He was on assignment at ITSO San Jose Center as a data management specialist from 1994 to 1997. He is currently on virtual assignment as a course developer for DB2 for z/OS in EMEA Development Team, IBM IT Education Services.

Michael Parbs is a DB2 Specialist with IBM Global Services A/NZ and is located in Canberra. He has over 12 years experience with DB2, primarily on the OS/390 platform. Before joining IBM Global Services Australia in 2000 he worked for 11 years for the Health Insurance Commission, where he started using DB2. Michael's main areas of interest are data sharing and distributed processing, but his skills include database administration and DB2 performance and tuning. Michael has co-authored the DB2 Data Sharing Implementation and DB2 Version 7 Presentation Guide redbooks.

I would like to especially thank **Willie Favero**, IBM Houston, for his contributions to this publication, describing the enhancements to identity columns and sequences.

Also, many thanks go to **Yvonne Lyon**, technical editor, ITSO San Jose Center, for undertaking the editing of this large publication under a tight deadline and making it happen.

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

Blake Bauman
Tom Beavin
Jeff Berger
Sigi Bigelis
Bill Bireley
Delmar Blevins
Patrick Bossman
Patrick Buzbee
John Campbell
Chris Crone
Tammie Dang
Janet Figone
Craig Friske
Georgia Fuller
James Guo
Sandy Hartman
Jeff Josten
John Kelly
Laura Kunioka-Weis
Fung Lee
Dave Levish
Irene Liu
Adrian Lobo
John Lyle
Bruce McAlister
Roger Miller
Nadyne Mielke
Chris Munson
Paul Ostler
Jim Pickel
Emily Prakash

Vivek Prasat
Terry Purcell
Sharon Roeder
Jim Ruddy
Michael Shadduck
Jack Shedden
Akira Shibamiya
Kalpana Shyam
Nigel Slinger
Don Smith
Hugh Smith
Derek Tempongko
James Teng
Judy Tobias
John Tobler
Patti Tonello
Steve Turnbaugh
Bituin Vizconde
John Vonau
Ping Wang
Julie Watts
Cathy Wolfenbarger
Jay Yothers
Guogen Zhang
Maureen Zoric
IBM Silicon Valley Lab, San Jose

Emma Jacobs
Sokkieng Wang
International Technical Support Organization, San Jose Center

Richard Conway
Bob Haimowitz
International Technical Support Organization, Poughkeepsie Center

Judy Ruby-Brown
Mark Wilson
IBM ATS Americas, Dallas, USA

Morgan Tong
IBM Austin,USA

Ian Cook
Sarah Ellis
IBM Product Introductory Centre, Hursley, UK

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

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 Redbooks to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

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

  ibm.com/redbooks

- Send your comments in an Internet note to:

  redbook@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. QXXE Building 80-E2
  650 Harry Road
  San Jose, California 95120-6099

Summary of changes

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

Summary of Changes
for SG24-6079-00
for DB2 UDB for z/OS Version 8: Everything You Ever Wanted to Know, ... and More
as created or updated on April 10, 2005.

July 2004

This revision reflects the addition, deletion, or modification of new and changed information described below.

New information

- “DBRMs in Unicode - How to read them?” on page 544
  Added a tip, pointing out that APAR OA07685 will allow browsing of Unicode data in ISPF.
- “Using the RETURN statement for the SQL procedure status” on page 604
  Added some clarification about the use of the SQLERRD flags in different programming languages.
- 12.1.1, “z/OS prerequisites” on page 882
  Added information about the Long Displacement Facility on the z900 as a prerequisite for DB2 for z/OS Version 8.
- 12.1.2, “Other prerequisites” on page 884
  Added information about COBOL and PL/I versions supported with DB2 for z/OS Version 8, and migration information for COBOL.

Changed information

- 2.26, “Sort pool - moving above the bar” on page 47
  Adapted to reflect design change not to use a global sort pool, but to use thread related (agent local) storage above the 2GB bar.
- 4.13.7, “CREATE example” on page 320
  ... the sequence will cycle back to the minimum value of 1 (one) instead of 0 (zero). (After a CYCLE, the sequence starts again with the MINVALUE. The value specified in the START WITH clause is only used for initial startvalue of the sequence.)
- 4.13.8, “ALTER SEQUENCE statement” on page 321
  The figure indicated that you can use the ORDER integer-constant clause. You cannot use an integer constant with the ORDER keyword. You must specify ORDER or NO ORDER.
- 4.13.14, “NEXT VALUE and PREVIOUS VALUE restrictions” on page 329
  The figure has been correct to show a correct SQL statement using a NEXT/PREVIOUS VALUE FOR expression. The previous statement was incomplete.
- “Multiple CCSIDs in a single table - a solution” on page 494
  In the left-hand side of the figure, the last row (ROW3) that is retrieved has been changed to X’D9D6E6F3’ x’**4A**’ instead of X’D9D6E6F3’ x’B0’
- 6.7.5, “Multiple CCSID set SQL statements - example” on page 557
  The UX constant has been corrected. **UX’004100420043’** instead of UX’414243’. (UX constants are always double byte characters in UTF-16.)

- 10.2.10, “MQT exploitation - simple example” on page 727
  The last sentence in the text has been corrected to **COUNT(QTY*PRICE)** instead of AVG(QTY*PRICE).
- 11.1.4, “XES contention before Version 8” on page 849
  The figure has been adjusted to match the text.
- 11.1.7, “LOCKPART considerations” on page 852
  More accurate description of the use of LOCKPART NO in DB2 Version 7.

April 2005

This revision reflects the addition, deletion, or modification of new and changed information described below.

New information

- 3.6.3, “Converting to table-controlled partitioning” on page 102
  Added clarification about the changes to the limit key of the last partition by DB2 during conversion to table-controlled partitioning for partitioned tables that currently do not enforce the limit key value on the last partition.
- 3.18.10, “Logging manageability enhancements” on page 212
  This section describing a number of interesting logging enhancements was also added.
- 4.13.5, “CREATE SEQUENCE statement” on page 314
  Added clarification related to the combined use of the CACHE and ORDER keywords in a data sharing environment.
- 4.20, “EXPLAIN STMTCACHE” on page 368
  Information about the use of EXPLAIN STMTCACHE ALL has been added.
- 4.22.5, “Enhanced Unicode support for UPPER, LOWER and TRANSLATE” on page 379
  Added a short description of the enhancements to these built-in functions related to Unicode strings.
- 4.26, “SELECT INTO with ORDER BY” on page 385
  Added section describing this enhancement.
- 5.5, “Security enhancements” on page 454
  Added information about the support for 1012 secondary authorization IDs
- “Materialized query tables with multilevel security” on page 477
  The section discussing considerations for multilevel security with row granularity in combination with materialized query tables has been enhanced.
- 5.5.14, “DB2 Command authorization enhancements” on page 481
  Section added.
- 10.18.2, “EXPLAIN using a stored procedure” on page 819
  Added information about the functionality provided via the PTFs for APARs PQ90022 and PQ93821.
- 10.19.10, “Statistics Advisor” on page 838
  Added information about this new functionality of Visual Explain.
- 10.14.2, “Lock avoidance for singleton select using ISO(CS) and CD(YES)” on page 811
  Section added.
- “You must now specify valid CCSIDs” on page 891
  Added information about a new SPUFI warning message (DSNE345I) if a mismatch between the CCSID used by the TSO session and the CCSID used to bind the SPUFI plans and packages is detected. This message was introduced by PQ89018.

Changed information

- Since z/OS V1.3 went out of support on March 31 2005, the book was updated to reflect that z/OS 1.4 is now the minimum z/OS level to run DB2 V8.

- 2.28.3, “Up to 65 000 open data sets” on page 51
  The maximum number of open data sets on any DB2 system is limited to 65 000 by APAR/PTF PQ96189/UQ96862, instead of 100 000.
- 5.5.10, “Accessing data in a table defined with row level security” on page 469 and 5.5.11, “DB2 utilities and multilevel security” on page 473
  Incorporating the changes introduced by the PTF for APAR PQ94303 on the use of disjoint seclabels in combination with write-down authority.
- Table 7-2 on page 579
  Corrected error in table name and content to reflect the information from the accompanying figure.
- 7.3, “Rollup of DB2 accounting data for DDF and RRSAF threads” on page 583
  Incorporated the changes introduced by the PTF for APAR PQ90547.
- 8.8.4, “ODBC Unicode support” on page 630
  After the installation of the PTF for APAR PQ86094, the use of the wide API is independent of the setting of the CURRENTAPPENSCH keyword in the .INI file.
- “Using work data sets when using the COLGROUP keyword” on page 767
  Correction to the formula to calculate the size of the sort workfiles required when gathering statistics on non-indexed or non-leading index columns.
- 10.14, “Locking enhancements” on page 810
  Additional qualification that this enhancement is only in effect for row level locking in a V8 data sharing environment.
- 12.4, “Other recommendations / information” on page 889
  Corrected the information about the requirement of allocating the DB2 SDSNLOAD data set as a PDS/E. Although you can allocate this data set as a PDS/E, it is not a requirement. The SDSNLOAD data set can still be a normal PDS.
- 12.1.2, “Other prerequisites” on page 884
  Updated COBOL and PL/I prerequisite release information.
- “Location name required” on page 898
  After the installation of PQ91009/UQ90701, DB2 will enforce the existence of a location name in the BSDS, with or without the use of DDF.
- 12.14.2, “DB2 Management Clients Package” on page 1026
  With DB2 for z/OS Version 8, DB2 Estimator is no longer packaged with DB2. DB2 Estimator is only available as a download from The Web.
- C.1.2, “Enhanced DBM1 and z/OS storage usage reports” on page 1044
  Adapted after the changes introduced by the PTF for APAR PQ91101.

1

DB2 UDB for z/OS Version 8: At a glance

DB2 UDB for z/OS Version 8 is the biggest release DB2 ever shipped in its 20 years of existence. That is why it takes a 1000-page book to try to explain all the great new features that come with this new version of DB2.

With Version 8, DB2 UDB for z/OS takes a giant leap forward in enterprise data management technology. What can you expect from DB2 UDB for z/OS Version 8?

1.1 DB2 UDB for z/OS Version 8 - overview

IBM DB2 UDB for z/OS Version 8 (DB2 V8 throughout this publication) includes dozens of changes in SQL, improving family consistency in many cases, and leading the way in others. Many barriers that had been limiting our customers are now removed: using 64 bit virtual memory addressing, providing consistent table and column name lengths, allowing 2-megabyte SQL statements, 4096 partitions, and three times the log space. These improvements include:

- Virtual storage constraints removal
- Unicode support
- Automated prior point-in-time recovery
- Multi-row fetch, insert
- Multiple DISTINCT clauses
- Lock contention avoidance via volatile tables
- Use of index for backwards searches
- Transparent ROWID
- Create deferred index enhancement
- Longer table names
- Providing DSTATS functionality inside RUNSTATS
- Converting column type
- Altering the CLUSTER option
- Adding columns to an index
- Index-only access path for VARCHAR
- Changing the number of partitions
- Data-partitioned secondary indexes
- Control Center enhancement
- DRDA® enhancements

Key performance enhancements deliver better family consistency and run many times faster. Being able to make database changes without an outage, such as adding a partition, is a breakthrough for availability. Improvements in Java function, consistency, and integration with WebSphere® make z/OS a much better platform for Java. Expansions to security allow for row level granularity, helping with the security issues of Web related applications. Many of these enhancements also help in key vendor applications like PeopleSoft, SAP, and Siebel.

In this section we introduce the main enhancements available to you in New Function Mode. These enhancements are grouped into categories based on the area of impact on your applications. These categories are:

- Architecture
- Usability, availability, and scalability
- Data Warehouse
- Performance
- Tools and administration
- Backup and recovery

1.2 Architecture

In this section we consider some important enhancements to the DB2 architecture.

1.2.1 Unicode support

Architectural changes to DB2 V8 expand the DB2 catalog with enhanced support for the Unicode catalog. This means that you can manage data from around the world. DB2 now converts any SQL statement to Unicode before parsing, and as a result, all characters parse correctly. DB2 also supports hexadecimal string constants.

1.2.2 DB2 Connect and DRDA

DB2 Connect™ and DRDA remain as the cornerstone for DB2 distributed processing, and V8 has many enhancements in this area. Enhancements in Version 8 remove roadblocks to performance and DB2 family compatibility by providing support for a common client and standardizing database connection protocols based on the Open Group Technical Standard DRDA Version 3.

1.2.3 Universal Driver for SQLJ and JDBC

Organizations increasingly require access to data residing in multiple sources in multiple platforms throughout the enterprise. More and more companies are buying applications rather than database management systems, as database selection is being driven by interoperability, price performance, and scalability of the server platform. The Universal Driver for SQLJ and JDBC provides an open and consistent set of database protocols to access data on the UNIX®, Windows®, and z/OS platforms.

Tools and applications can be developed using a consistent set of interfaces regardless of the platform where the data resides. End users can integrate their desktop tools and other applications in a consistent manner with whatever databases (or multiple databases concurrently) are in the enterprise. The objective of this enhancement is to implement Version 3 of the Open Group DRDA Technical Standard. It eliminates the need for gateways, improves desktop performance, and provides a consistent set of database protocols accessing data from a z/OS server as well as UNIX and Windows servers.

1.2.4 Schema evolution

As 24x7 availability becomes more critical for applications, the need grows for allowing changes to database objects while minimizing the impact on availability. Online schema evolution allows for table, index, and table space attribute changes while maximizing application availability. For example, you can change column types and lengths, add columns to an index, add, rotate, or rebalance partitions, and specify which index (the partitioning index or the non-partitioning index) you want to use as the clustering index.

1.2.5 64-bit virtual storage

This enhancement utilizes zSeries 64-bit architecture to support 64-bit virtual storage.

The zSeries 64-bit architecture allows DB2 UDB for z/OS to move various storage areas above the 2-GB bar:

- Buffer pool
- EDM pool (DBDs, OBDs, and dynamic statement cache)
- Sort pool
- RID pool
- Castout buffers
- Compression dictionaries

A single large address space of up to 2^{64} bytes (16 exabytes) replaces hiper spaces and data spaces. As a result, managing virtual storage becomes simpler, and the scalability, availability, and performance improve as your real storage requirements and number of concurrent users increase.

1.3 Usability, availability, and scalability

In this section we consider enhancements related to usability, availability, and scalability.

1.3.1 Partitioning

Here we describe several partitioning enhancements included in DB2 V8 that are useful in almost all real-life environments.

Online partitioning changes

You can add a new partition to an existing partitioned table space and rotate partitions.

More partitions

This enhancement increases the maximum number of partitions in a partitioned table space and index space past the current maximum of 254. The new maximum number of partitions is 4096. The DSSIZE and page size determine the maximum number of partitions that is possible.

Data-partitioned secondary indexes

V8 introduces data-partitioned secondary indexes to improve data availability during partition level utility operations (REORG PART, LOAD PART, RECOVER PART) and facilitate fancier partition level operations (roll on/off part, rotate part) introduced by Online Schema Evolution. The improved availability is accomplished by allowing the secondary indexes on partitioned tables to be partitioned according to the partitioning of the underlying data.

There is no BUILD2 phase component to REORG SHRLEVEL CHANGE when all secondary indexes are partitioned, nor is there contention between LOAD PART jobs executing on different partitions of a table space. Query-wise, a data-partitioned secondary index is most useful when the query has predicates on both the secondary index column(s) and the partitioning index column(s).

Separation of partitioning and clustering

Partitioning and clustering were bundled together in versions prior to V8. Now you can have a partitioned table space without an index and can cluster the data on any index. These changes may be able to eliminate one index (since you are no longer required to have a partitioning index) and reduce random I/O (as you can now define any index as the clustering index).

REORG utility enhancements

The REORG utility is enhanced to allow you to specify that only partitions placed in Reorg Pending state should be reorganized. You do not have to specify the partition number or the partition range. You can also specify that the rows in the table space or the partition ranges being reorganized should be evenly distributed for each partition range when they are reloaded. Thus, you do not have to execute an ALTER INDEX statement before executing the REORG utility. You can specify DISCARD with SHRLEVEL CHANGE. You can avoid the BUILD2 phase during online REORG by using the new data-partitioned secondary indexes.

1.3.2 Create index dynamic statement invalidation

Create index will now invalidate the cached statements associated with the base table contained in the dynamic statement cache without draining active statements.

1.3.3 Minimize impact of creating deferred indexes

Indexes created as deferred will be ignored by the DB2 optimizer.

1.3.4 Column data type change

You can change the data type for columns. In V5 you could increase the size of VARCHAR columns, but the changes in V8 allow you to extend numeric and character columns and to change between CHAR and VARCHAR.

1.3.5 LOB ROWID transparency

This enhancement includes the capability of hiding the ROWID column from DML and DDL. This way, applications running on other platforms that do not have a ROWID data type can avoid the special code to handle ROWID and use the same code path for all platforms.

1.3.6 Longer table and column names

Architectural changes to DB2 V8 expand the DB2 catalog with support for long names. Support for longer string constants (up to 32,704 bytes), longer index keys (up to 2,000 bytes), and longer predicates (up to 32,704 bytes) make DB2 UDB for z/OS compatible with other members of the DB2 family.

1.3.7 SQL statements 2 MB long

Complex SQL coding, SQL procedures, and generated SQL, as well as compatibility with other platform and conversions from other products, have required the extension of the SQL statements in DB2. DB2 V8 extends the limit on the size of an SQL statement to 2 MB.

1.3.8 Multiple DISTINCT clauses in SQL statements

This enhancement allows the DISTINCT keyword to appear in multiple column functions with different expressions. For example, DB2 V8 now allows:

```
SELECT SUM(DISTINCT C1), AVG(DISTINCT C2) FROM T1
```

1.3.9 More open data sets

With DB2 Version 8 and z/OS 1.5, a DB2 subsystem can have up to 100 000 open data sets, instead of 32 767 in Version 7. This enhancement also reduces the amount of virtual storage used by DB2 below the 16 MB line, as MVS control blocks related to dynamically allocated data sets will be requested above the 16 MB line. When those open data sets (or partitions) are compressed, the compression dictionary is loaded above the 2 GB bar.

1.3.10 More log data sets

The maximum number of active log data sets per log copy is increased from 31 to 93. The maximum number of archive log volumes recorded in the BSDS before there is a wrap around, and the first entry that is overwritten is increased from 1,000 to 10,000 per log copy.

1.3.11 CI size larger than 4 KB

DB2 V8 introduces support for CI sizes of 8, 16, and 32 KB. This is valid for user-defined and DB2-defined table spaces. The new CI sizes relieve some restrictions on backup, concurrent copy, and the use of striping, as well as provide the potential for reducing elapsed time for large table space scans.

1.4 Data warehouse

In this section we discuss some enhancements affecting the data warehouse applications.

1.4.1 More tables in joins

In DB2 V7 the number of tables in the FROM clause of a SELECT statement can be 225 for a star join. However, the number of tables that can be joined in other types of join is 15. DB2 V8 allows 225 tables to be joined in all types of joins.

1.4.2 Sparse index and in-memory workfiles for star join

The star join implementation in DB2 UDB for z/OS potentially has to deal with a large number of work files, especially for a highly normalized star schema that can involve many snowflakes, and the cost of the sorting of these workfiles can be very expensive. DB2 V8 extends the use of a sparse index (a dynamically built index pointing to a range of values) to the star join work files and adds a new optional function of data caching on star join workfiles. The decision to use the sparse index is done based on the estimation of the costs of the access paths available.

1.4.3 Common table expression and recursive SQL

DB2 V8 introduces the common table expression and recursive SQL function, which extends the expressiveness of SQL and lets users derive the query result through recursion. It is also a convenient way for users to express complex queries, as using common table expressions instead of views saves both users and the system the work of creating and dropping views.

1.4.4 Materialized query tables

This enhancement provides a set of functions that allow DB2 applications to define, populate, and make use of materialized query tables. Large data warehouses will definitely benefit from MQTs for queries against objects in an operational data store.

1.5 Performance

In this section we discuss locking, RUNSTATS, and a number of other DB2 V8 improvements related to performance.

1.5.1 Locking improvements

Here we consider the improvements in locking.

Reduced lock contention on volatile tables

Volatile (or cluster) tables, used primarily by ERP vendors like SAP, are tables that contain groups (or clusters) of rows which logically belong together. Within each cluster, rows are meant to be accessed in the same sequence every time. Lock contention occurs when DB2 chooses different access paths for different applications operating on the same cluster table. In the absence of support for cluster tables in DB2, users have to either change system-wide parameters that will affect all tables, or change statistics for each such table to ease the lock contention.

Cluster tables are referred to as volatile tables in DB2. Adding a new keyword, VOLATILE, to the CREATE TABLE and ALTER TABLE statements signifies to DB2 which tables should be treated as volatile tables. For volatile tables, index access is chosen whenever possible, regardless of the efficiency of the available index(es). That is, a table scan is not chosen in preference to an inefficient index.

CF lock propagation reduction

In a data sharing environment, this enhancement allows parent L-locks to be granted locally without invoking global contention processing. Thereby, locking overhead due to false contention is reduced. As a result, DB2 data sharing performance is enhanced. Performance benefit varies depending on factors such as commit interval, thread reuse, number of tables accessed in a commit interval, if the SQL processing is read-only or update, etc.

1.5.2 Multi-row INSERT and FETCH

With this SQL enhancement, a single FETCH can be used to retrieve multiple rows of data, and an INSERT can insert one or more rows into a table. This reduces the number of times that the application and database must switch control, as well as reducing the number of network trips required for multiple fetch or insert operations for distributed requests. For some applications, this can help performance dramatically.

1.5.3 RUNSTATS improvements

This section covers the improvements in RUNSTATS.

Distribution statistics

This enhancement adds the new functionality of calculating the frequencies for non-indexed columns to RUNSTATS. The relevant catalog tables are updated with the specified number of highest frequencies and optionally with the specified number of lowest frequencies. The new functionality also optionally collects multi-column cardinality for non-indexed column groups and updates the catalog.

Fast cached SQL statement invalidation

This enhancement adds the new functionality of allowing the UPDATE NONE and REPORT NO keywords to be used on the same RUNSTATS utility execution. This causes the utility to only invalidate statements in the dynamic statement cache without any data access or computation cost.

1.5.4 Host variables impact on access paths

The enhancement allows for a new BIND option, REOPT(ONCE). It allows you to re-optimize an SQL statement based on the host variable value, the first time it is executed. After that, the statement is stored in the dynamic statement cache.

1.5.5 Index only access for VARCHAR

This enhancement removes the key padding associated with VARCHAR index columns. This allows the use of these columns to satisfy the results of queries that can use index only access.

1.5.6 Backward index scan

The backward index chain has been part of Type 2 indexes since day one. However, only in V7 did DB2 start to exploit the backward index chain for MIN and MAX functions. In V8, DB2 extends this functionality with the capability for backward index scans. This allows DB2 to avoid more sorts and allows customers to define fewer indexes.

1.5.7 Local SQL cache issues and short prepare

This enhancement reduces the cost of a short prepare. This will lower the cost of further reductions in MAXKEEPD.

1.5.8 Multiple IN values

This enhancement causes the DB2 optimizer to make a wiser choice when considering index usage on single table SQL queries that involve large numbers of IN list items.

1.5.9 DDF performance

Performance is improved as a result of several changes across the components involved in reducing the path length of the TCP/IP RECEIVE, and several accounting enhancements.

1.5.10 Automatic space management

This feature is considered very valuable for many DB2 installations. It potentially eliminates one of the main causes of failures where growth has not been completely anticipated.

1.5.11 Dynamic statement cache statement ID in EXPLAIN

This enhancement allows the DB2 EXPLAIN statement to report the chosen access path of an SQL statement currently in the dynamic statement cache (DSC).

1.5.12 Instrumentation enhancements

As in previous DB2 versions, V8 comes with numerous enhancements to the Instrumentation Facility Interface, such as these:

- Package level accounting
- Accounting roll-up for DDF and RRSAF threads
- Long-running reader tracking
- Lock escalation trace record
- Full SQL statement text trace record
- PREPARE attributes are also traced
- More high water marks in the statistics record
- Secondary authorization IDs via READS
- Storage 64-bit support as well as READS support
- Temporary space usage
- Auditing for multilevel security
- Option for EBCDIC or Unicode data

1.5.13 Migration changes

Migration is allowed exclusively from DB2 UDB for OS/390 and z/OS Version 7 subsystems. The migration SPE must have been applied and started. The migration process is changed and now consists of three distinct steps or phases:

1. **Compatibility mode (CM):** This is the first phase, during which the user makes all the tests needed to make sure that all the applications run without problems with the new version. Fallback to V7 in case of problems is allowed.
2. **Enabling-new-function mode (ENFM):** During this (possibly short) second phase, the user converts the DB2 catalog and directory to a new format by using on-line REORG executions. No fallback to DB2 V7 is allowed once this phase is entered.
3. **New-function mode (NFM):** This is the target third and final phase, where all new V8 functions are available.

1.6 System level point-in-time backup and recovery

The system level point-in-time recovery enhancement provides the capability to recover the DB2 system to any point in time, irrespective of the presence of uncommitted units of work, in the shortest amount of time. This is accomplished by identifying the minimum number of objects that should be involved in the recovery process, which in turn reduces the time needed to restore the data and minimizes the amount of log data that needs to be applied. For the larger DB2 systems with more than 30,000 tables, this enhancement significantly improves the data recovery time, which in turn results in considerably shorter system downtime.

BACKUP and RESTORE SYSTEM

These two new utilities provide system level backup, and system level point-in-time recovery. They activate new functionality available with the new z/OS V1R5 DFSMShsm™, which allows a much easier and less disruptive way for fast volume-level backup and recovery to be used for disaster recovery and system cloning.

2

Scalability

List of Topics

Redbooks

The problem

The 64-bit architecture support

- 64-bit processor
- The z/OS architecture
  - Real
  - Virtual
- Pre-DB2 V8 real storage exploitation
- DB2 V8's virtual storage expansion
- Moving above the 2GB Bar

More partitions

More log data sets

More tables in a JOIN

Longer SQL statements

Longer index keys and predicates

DB2 for z/OS V8 offers extensive synergy with the zSeries platform and the z/OS operating system, and breaks through many of the limitations previously imposed by the operating system. These enhancements have a positive effect on scalability and availability by delivering large address spaces with the exploitation of the 64-bit virtual addressing provided by the z/Architecture™. With this support, DB2 can guarantee to keep up with the explosive demands of e-business, transaction processing, and business intelligence.

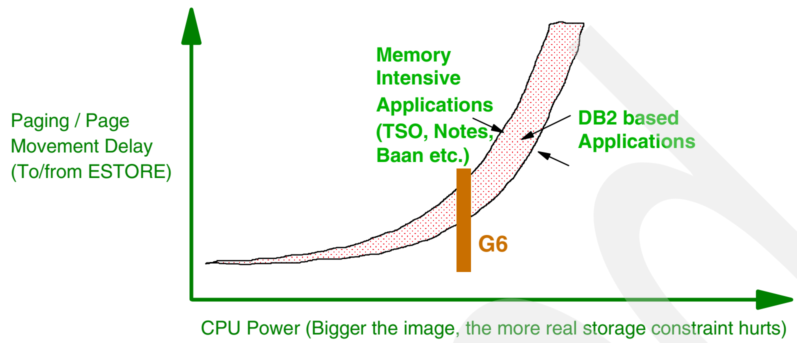
With DB2 V8 you can manage more data with larger buffers in memory, utilize larger control storage areas such as the EDM and RID pools, and gain more capacity for concurrent locks. You can also access your data through more partitions and join more tables in a single SQL statement.

In this chapter we discuss the following topics:

- The 64-bit architecture support
- More partitions
- More log data sets
- More tables in join
- Longer SQL statements

Where Are We Today? (The Problem)

Redbooks



Adding CPU capacity resulted in little or no additional real work being done

- Paging overhead increased due to 2 GB (31 bit) central storage limit
- Path length increased due to high MIPS and N-way system effects
  - Parallel Sysplex multisystem environments

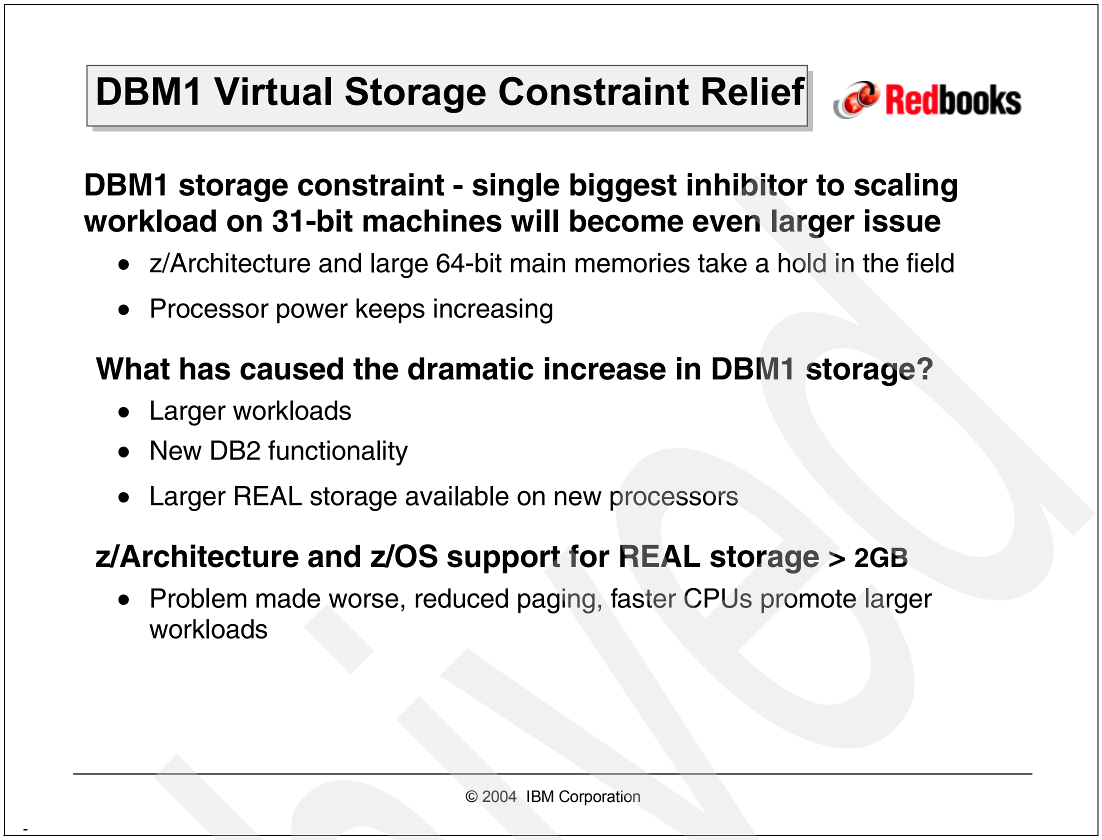
Expanded storage had provided an excellent interim solution

- G5/G6 implemented the enhanced MOVE PAGE instruction

2.1 Where are we today? (The problem)

This visual demonstrates why the new 64-bit architecture was introduced. It shows that, for several storage starved environments, adding CPU capacity does not allow any further growth, and results in little or no additional real work being done. The limiting factor was that the paging overhead increased due to the 2 GB (31-bit) central storage limit.

Expanded storage has provided an excellent interim solution, and the G5/G6 processors brought some relief with the implementation of the enhanced MOVE PAGE instruction, but systems with large and variable workloads need the storage constraint removed.



2.2 DBM1 virtual storage constraint relief needed

Over the years, virtual storage usage has grown dramatically in DB2's DBM1 address space. This storage growth has been fueled by larger workloads, new functions, and larger real storage available on mainframe processors. The latter, in particular, has allowed customers to run workloads that in the past would have been definitely limited by paging overhead.

With the arrival of z/Architecture and z/OS support for real storage larger than 2 GB, we have seen that the problem may become worse, since the reduced paging, faster CPUs, and higher multi-processor levels can promote larger and larger workloads. The DBM1 2 GB virtual storage constraint, already the single biggest inhibitor to scaling DB2 workloads on 31-bit machines, becomes an even larger growth inhibitor as z/Architecture and large 64-bit main memories continue to take hold in the field.

DBM1- Major VSTOR Consumers

Redbooks

DB2 cannot use the entire 2GB addressable virtual storage

- Large chunks are common storage (CSA / ECSA, etc. used by WebSphere, IRLM PC=NO, etc.)

EDM pool partly satisfied with dynamic statement caching in data space

Thread storage

VSAM data set control blocks

Compression dictionaries

More functionality

- Larger code base therefore larger memory requirement

Largest storage consumer usually buffer pools

VSTOR allocation for a typical DB2 V7 installation

Usage	Typical Value
Virtual Buffer Pools	40 to 800MB
EDM Pool	20 to 400MB
User Thread Storage	0.1 to 2MB per active user thread
System Thread Storage	40KB per system thread
Compression Dictionary	64KB per open compressed data set
Local Dynamic Statement Cache	0 to 300MB
Other Areas	200 to 400MB

Recommended White Paper - John Campbell / Mary Petras
"DB2 UDB Server for OS/390 Virtual Storage Management"

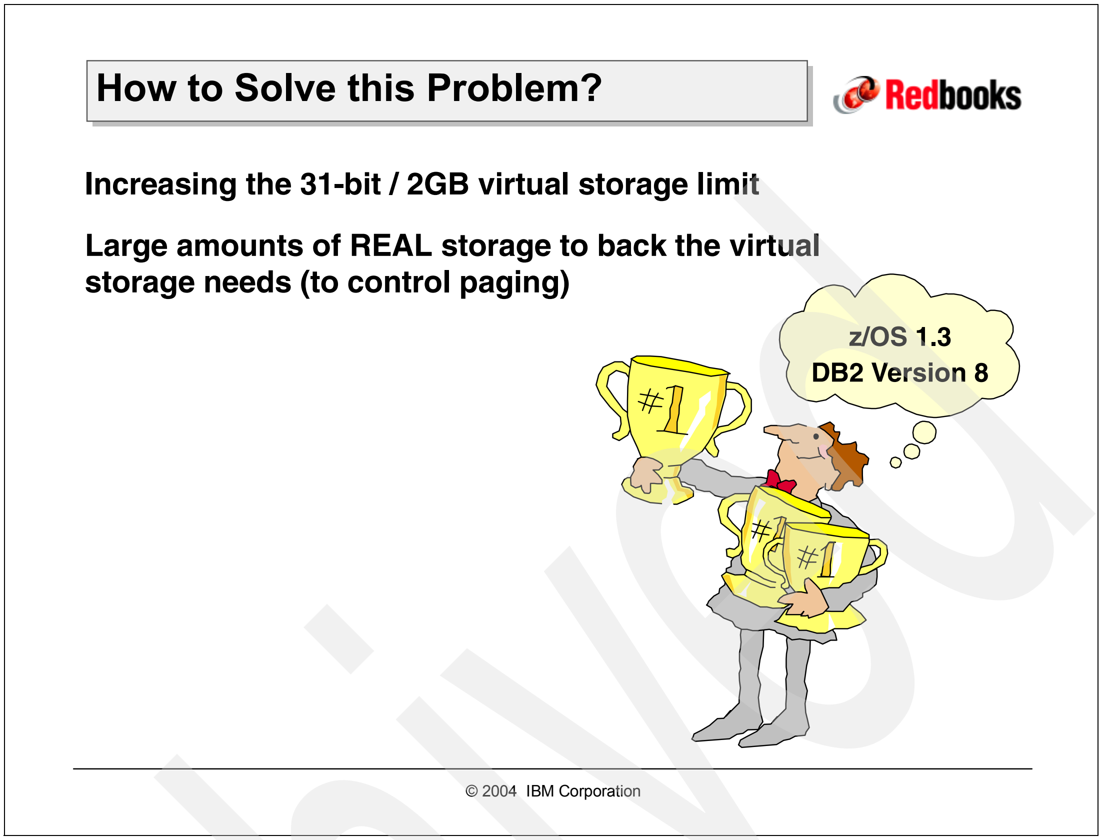
2.3 DBM1 - Major VSTOR consumers

The figure above shows a typical customer case. The biggest storage consumers inside the DBM1 address space are usually the buffer pools, followed by the EDM pool. Remember also that when you use the local dynamic statement cache (KEEPDYNAMIC(YES) in combination with CACHEDYN=YES), that the local dynamic statement cache can take up a large amount of virtual storage. In addition, do not forget that every open compressed data set uses a 64KB compression dictionary that has to be loaded into storage.

And last but not least, it is important to remember that some storage is shared between all users of the system; the largest areas being CSA and ECSA. When you use the IRLM option PC=NO (in V7), your IRLM locks are stored in ECSA. Also, when you have WebSphere running on the same LPAR, WebSphere usually consumes a lot of ECSA storage as well, and all storage allocated to CSA/ECSA cannot be used for addressing the (extended) private areas.

There is an introductory article by Mary Petras and John Campbell on this subject. It can be found at:

`http://www.idug.org/idug/member/journal/mar00/storage.cfm`

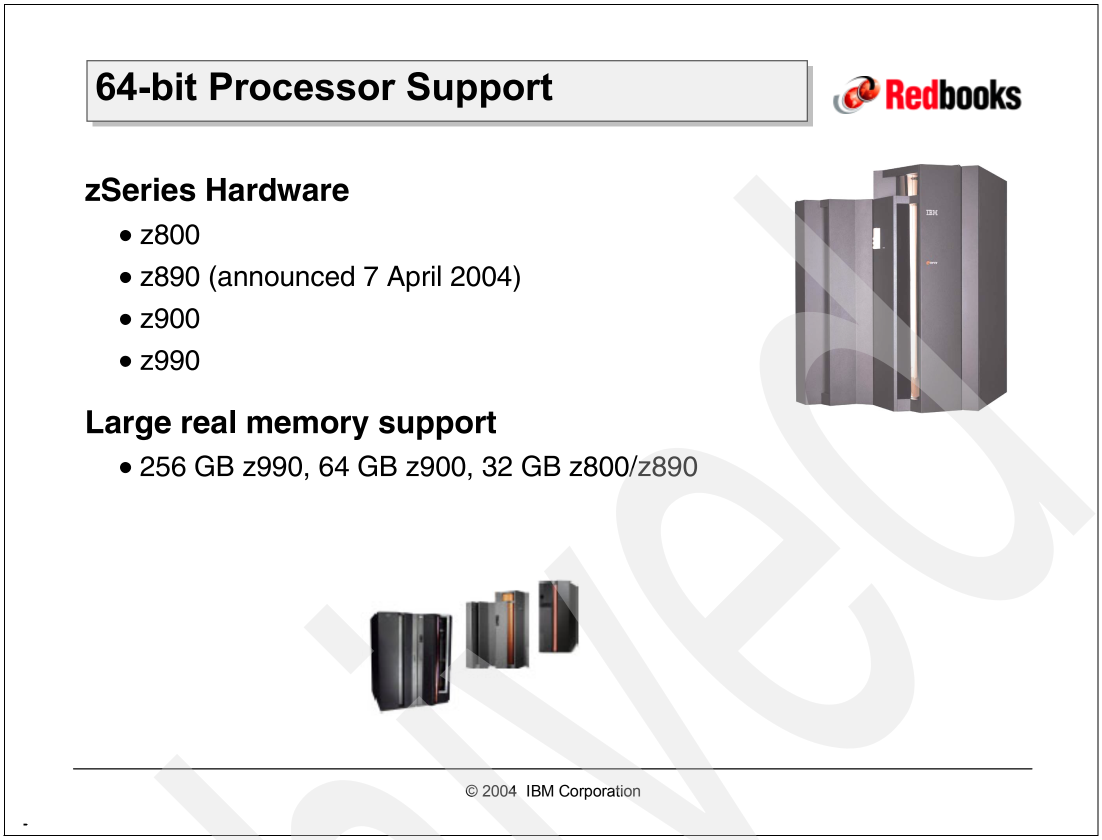


2.4 How to solve this problem?

The solution consists of three parts:

- 64-bit capable hardware (zSeries)
- 64-bit capable operating system:
  - 64-bit real storage support (OS/390 V2R10 and above)
  - 64-bit virtual storage support (z/OS 1.2 and above)
- 64-bit capable subsystems (DB2 for z/OS Version 8)

Before we explore the 64-bit support in DB2 V8, we first look at the hardware and operating system effort of going to 64-bit.



2.5 64-bit processor support

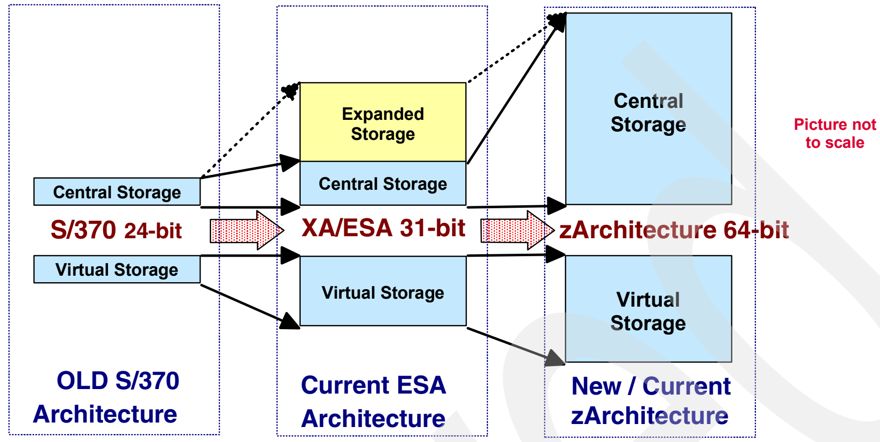
IBM launched the zSeries in the year 2000. This class of servers was designed for high performance data and transaction serving and was optimized to handle the volatile demands of the e-business climate.

zSeries is a family of processors that use the new z/Architecture (formerly known as ESAME Architecture). The z/Architecture is commonly known as "64-bit" architecture, although it provides much more than 64-bit capabilities.

Currently three types of zSeries machines are available, available in different models:

- z990, with up to 256 GB of memory
- z900, with up to 64 GB of memory
- z800/z890, with up to 32 GB of memory

64-bit Memory Architecture (The Solution)



V2R10 provided initial z/Architecture real addressing support:
- 64 bit registers, etc...
- 128GB of central storage (current operating system max. real addressability)

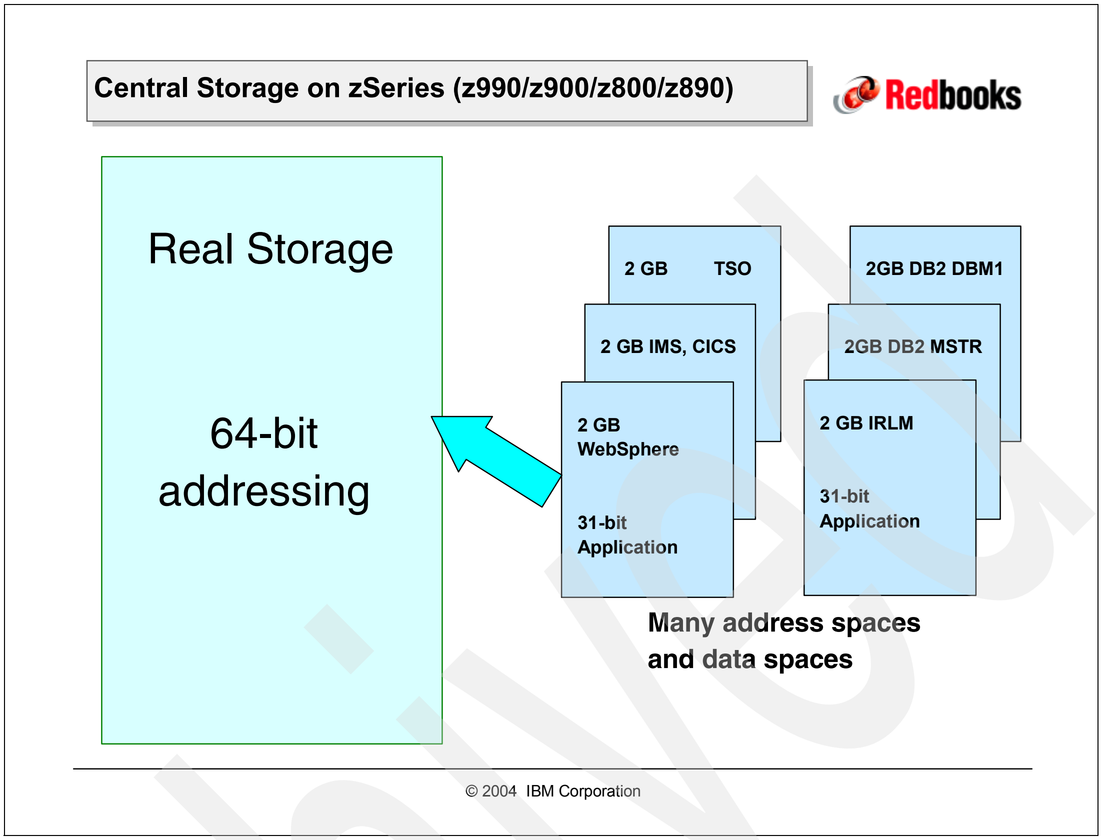
z/OS V1.2 delivered initial 64-bit virtual storage support
- Supporting 24-bit, 31-bit, and 64-bit applications

2.6 64-bit memory architecture (The solution)

This visual gives us a pictorial representation of the evolution of the memory management from the 24-bit to the 31-bit to the 64-bit support.

The ESA/390 architecture limits the amount of central storage that can be configured to a single OS/390 image to 2 GB. OS/390 Version 2 Release 10 removes the 2 GB real storage restriction by utilizing the new 64-bit architecture. OS/390 V2R10 supports up to 128 GB of central storage, when running in z/Architecture mode. The 128 GB limit is a software restriction that was implemented to prevent the Page Frame Table from exceeding 1 MB. (This restriction is still in place in z/OS V1.5, and may be lifted in a future z/OS release.)

In z/OS V1.2, IBM delivered the initial *64-bit virtual storage* management support. With the new z/OS 64-bit operating environment, an application address space can have 2 to the power of 64 (or 2**64) virtual addresses, with backing by real storage as needed. With this new architecture, z/OS delivers the functions to meet the needs of growing e-business application environments that will dominate future commercial data processing while maintaining today's critical applications.



2.7 Central storage on zSeries (z990/z900/z800/z890)

When IPLing the system in 64-bit mode (ARCHLVL = 2), provided you have the right zSeries hardware, and are at least at OS/390 V2R10, you can start exploiting the larger amounts of real memory that are available on the zSeries machines. (Note that starting with z/OS V1.2, the system will enforce the architecture level based on the IPL processor, and you should therefore no longer specify the ARCHLVL in the LOADxx member.)

You can exploit larger real memory with DB2 Version 6 and Version 7, for example, by exploiting data space buffer pools. So even before migrating to Version 8, you can already take advantage of the 64-bit real support and large amounts of real memory. For more information on how DB2 Version 6 and 7 can exploit 64-bit real addressing on the new zSeries hardware, see 2.11, "DB2 (pre-V8) z/Architecture exploitation" on page 23.

REAL Storage Support on zSeries - Provided

Constraint relief

- For workloads limited by 2 GB real storage
- Enabled value from 16-way multiprocessor on a z900, or 32-way on a z990
- Allowed consolidation of LPARs

Removal of expanded storage

- Paging now only to DASD
- Now all z990/z900/z800 memory configured as REAL for z/OS (or OS/390 R10) in z/Architecture (64-bit mode)
  - Previously OS/390 in ESA/390 mode - on 9672 machines, storage beyond 2GB had to be configured as expanded
- Reduced memory management overhead

2.8 REAL storage support on zSeries - provided

OS/390 V2R10 and z/OS have provided the 64-bit *real storage* addressability needed to scale in real memory addressing. OS390 V2R10 has the ability to run in either 31-bit mode or 64-bit mode on a zSeries, while z/OS only runs in a 64-bit mode real storage environment. (For completeness, you can run z/OS in 31-bit mode on a zSeries machine when using the z/OS Bimodal Migration Accommodation offering. This offering is only provided on z/OS V1.2, 1.3, and 1.4.)

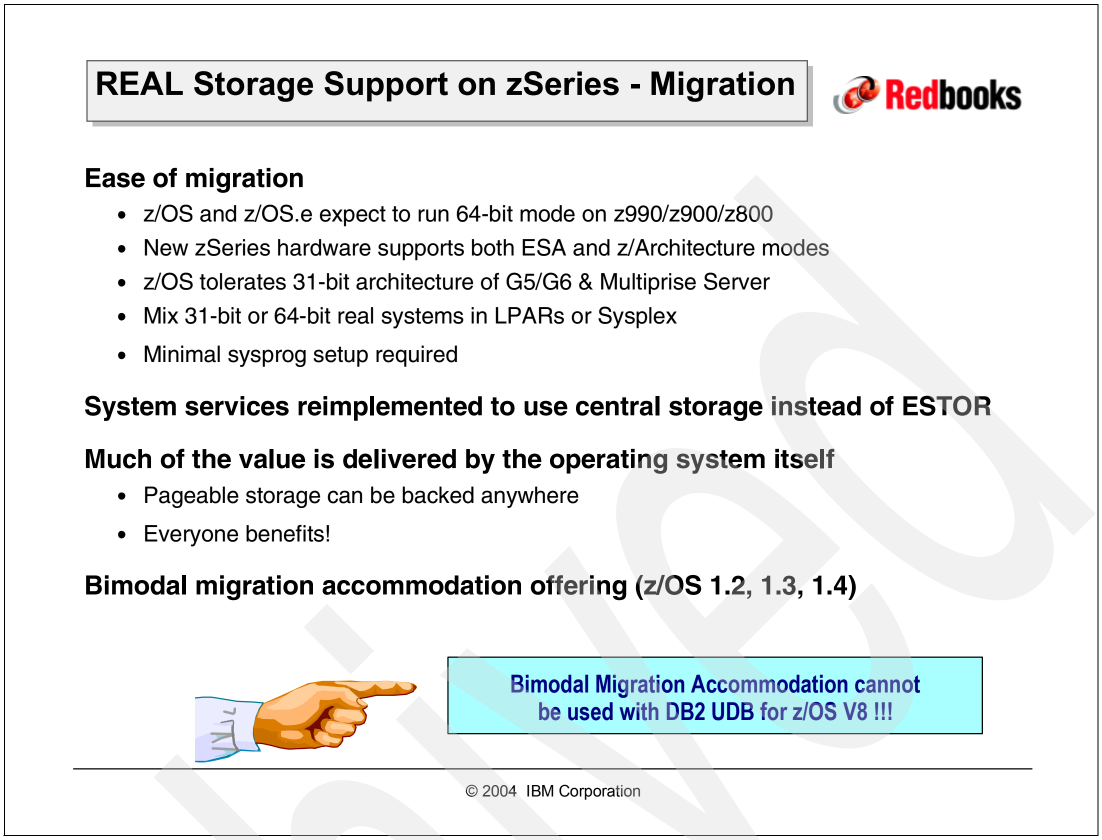
z/OS 1.2 and later releases provide *64-bit virtual storage* exploitation of the addressing range above 2 GB.

Basically, R10 has provided initial z/Architecture real addressing support (up to 128 GB of central storage) and the support for 24-bit, 31-bit, and 64-bit applications.

z/OS 64-bit real storage support has provided significant and transparent reduction of paging overhead, now only to disk, and real storage constraint relief for workload limited by the 2 GB of real storage by configuring all z990, z900, z800 and z890 memory as REAL. The elimination of Expanded Storage support has been handled by z/OS with minimal customer impact while reducing memory management overhead. However, remember that in the absence of expanded storage, paging will now be to disk only. Therefore it is important to have enough real storage to back up your workload in order to avoid paging.

Large real storage memory support has also enabled the exploitation of 16-way multi-processors in z900, and up to 32-way on the z990, and allowed the consolidation of LPARs.For more information on 64-bit real exploitation, see the z/OS migration Web site at:

`http://www.ibm.com/servers/eserver/zseries/zos/installation/`



2.9 REAL storage support on zSeries - migration

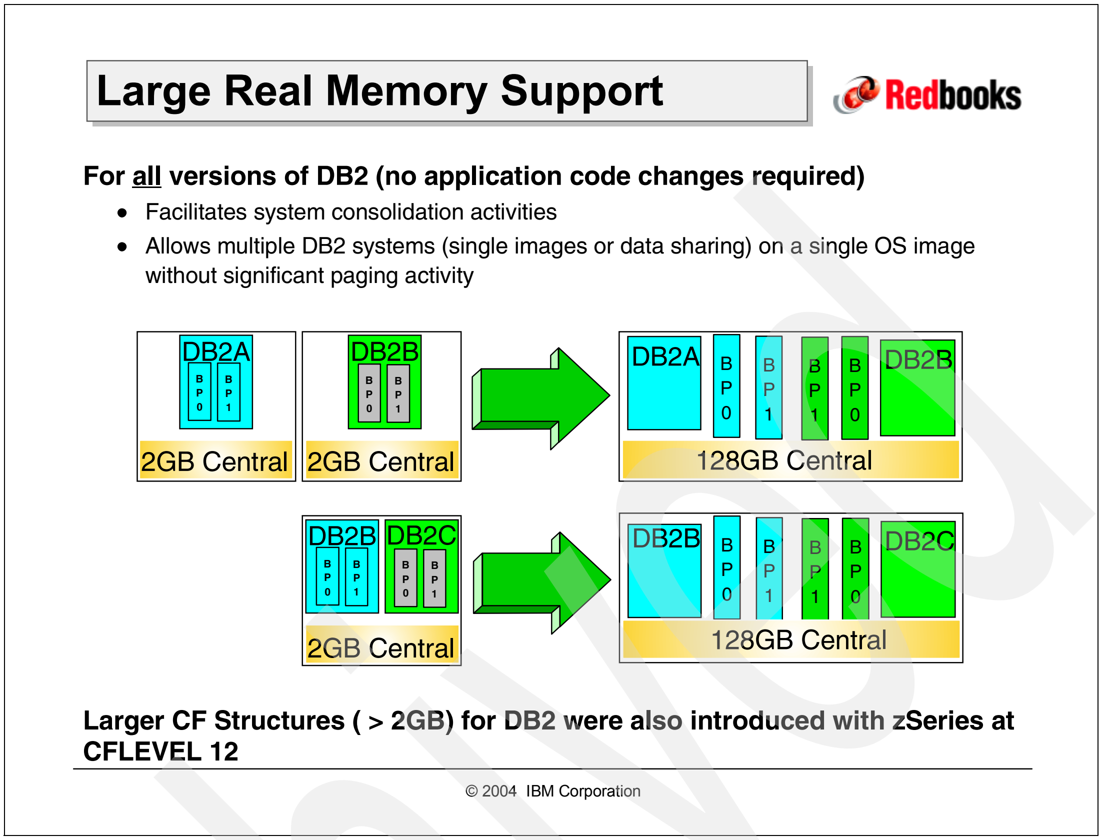
IBM has put a lot of effort into making it easy to migrate to z/OS and zSeries. The zSeries hardware supports both ESA (ARCHLVL 1) and z/Architecture (ARCHLVL 2) modes.

z/OS can be run on the 31-bit architecture hardware of G5/G6 & Multiprise® Servers. (It is worth noting that z/OS V1.5 will be the last release to do so. Starting with z/OS V1.6, a zSeries machine is required.) However, when running z/OS and z/OS.e on the z990/z900/z800/z890 hardware, these operating systems expect to run 64-bit mode. To help out customers that are reluctant to run in 64-bit mode, the day they install a zSeries machine, a z/OS Bimodal Migration Accommodation offering is available. It is available at no charge as a download off the Web. It gives customers the security of knowing they can fallback to 31-bit mode if there are any 64-bit problems during their migration. The 31-bit mode is fully supported during the Accommodation period. The Bimodal Migration Accommodation offering is only available for z/OS 1.2, 1.3, and 1.4.

> **Important:** It should be noted that DB2 Version 8 can only run in 64-bit mode. Bimodal mode is not supported for DB2 V8. When you are using the Bimodal support as an “insurance policy” in case something goes wrong after migrating to z/OS and zSeries, you need to create a stable 64-bit environment, where you no longer need to be able to “fall back” to 31-bit mode, before migrating to DB2 Version 8.

You can mix 31-bit or 64-bit real systems in LPARs or sysplex.

With the 64-bit support, system services have been re-implemented to use central storage instead of expanded storage (ESTOR).



2.10 Large REAL memory support

All DB2 versions can benefit from larger real memory support. With more real storage available on a single LPAR, you may consider consolidating LPARs, or just start taking advantage of using the larger amounts of real storage that is available, by resizing your buffer pools, for example. As always, make sure you that your workload is backed by enough real storage; we do not want to introduce too much paging in the system by overcommitting real storage.

However, you may want to take the opportunity to re-evaluate how DB2 uses its buffer pool storage: virtual pools versus hiperpools versus data space buffer pools.

DB2 (Pre-V8) z/Architecture Exploitation

Redbooks

Increasing processor speed

Real storage exploitation

- Larger buffer pools
- Buffer Pools in data spaces

2.11 DB2 (pre-V8) z/Architecture exploitation

Even though this book is focusing on DB2 V8, you do not have to wait until you get to V8 before you can start taking advantage of the 64-bit capabilities of the hardware and operating system. Pre-V8 systems can already get a serious performance boost from being able to exploit the increased number of CPUs and faster CPUs of the z/Architecture machines, as well as the support for real memory beyond 2 GB. As mentioned earlier, even pre-V8 versions of DB2 can make good use of this extra memory and increase in processing power, as we will demonstrate in the next sections.

In order to exploit real storage above the 2GB bar in releases prior to V8, you need to have the PTFs for the following APARs installed on Version 6. You almost certainly have those installed, as this support was introduced in 1999.

- PQ25914 allows data spaces and virtual buffer pools to be backed by real storage above 2 GB.
- PQ36933 avoids ABEND0D3 when testing whether a page is in memory with real storage above the 2 GB bar.

DB2 V7 has this support in the base code. Also make sure to be current on OS/390 and z/OS maintenance, to avoid real storage manager problems that were discovered at the early stages of 64-bit real addressing support.

For more information about which DB2 versions can run on zSeries hardware, see:

http://www-1.ibm.com/support/docview.wss?rs=64&context=SSEPEK&q1=64+bit+support&uid=swg21009394&loc=en_US&cs=utf-8&lang=en+en

Increasing Processor Speeds

Good news for DB2 customers

- Drives business workloads to greater heights

Problem

- More workload requires more data which subsequently means more I/O

Over the years I/O latency improvements have NOT kept up with CPU speeds

- I/Os become more and more precious

Larger impact on general performance

Solution

- Larger buffer pools

2.12 Increasing processor speeds

Increased processing power is always good news for DB2 users, as it can drive business workloads to greater heights.

However, there are also problems associated with increased processor capacity, especially when it is not balanced with equal improvements in I/O bandwidth.

When the workload increases, it usually requires more data to be processed, which subsequently means more I/O. Over the years I/O latency has improved quite a lot, but it has not been able to keep up with the increases in CPU speed. For that reason, I/Os become more and more precious, and the amount of I/O required can have a big impact on the performance of applications and systems in general. The solution to this problem is to avoid I/O as much as possible, and one of the ways to do so is to cache more data in memory. However, more data caching means we need larger amounts of memory to be able to cache that data.

Larger Buffer Pools

Redbooks

Large and ever cheaper main memories available

Gain performance advantages

Restrictions

- Due to DBM1 VSTOR constraint, DB2 forces the maximum **BPOOL** sizes < machine memory (on a 2GB central storage LPAR):
  - Size of VPOOL limited to 1.6GB (Typically only 1 GB due to other constraints)
  - Frequently in the range 400-800MB

Problem

- Cannot increase virtual pools as needed to avoid I/O for many workloads
  - Specifically those which repeatedly access same data elements and index keys

Solution

- Hiperpools (DB2 Version 3)

2.13 Larger buffer pools

With the very large and ever-cheaper main memory capacity that is available on the current and upcoming z/Architecture machines (currently 10s of GB, into the 100s of GB now), it is becoming feasible for customers to configure very large buffer pools to gain significant performance advantages. However, due to DBM1 virtual storage constraints, DB2 currently enforces maximum buffer pool sizes that are less than the memory capacities of these machines.

The total size of virtual pools is limited to 1.6 GB. However, in actual practice, customers typically cannot configure more than 1.0 GB for their buffer pools, due to DBM1 virtual storage constraints. EDM pool, buffer pool control blocks, VSAM control blocks, and compression dictionaries are other sizeable contributors to the demands on DBM1.

The fact that DBM1 virtual storage was becoming an inhibitor to DB2 scalability became clear in the early nineties. To reduce the size of the virtual pool inside the DBM1 address space, DB2 Version 3 introduced hiperpools. Hiperpools “live” in expanded storage, so they do not compete for real storage, in so-called expanded-storage-only hiperspaces, outside the DBM1 address space, bringing excellent virtual storage constraint relief for many years.

Hiperpools

Buffers in hiperspaces (hiperpools) offer some relief

- Expanded Storage exploitation - Cheaper memory option

Benefits

- High performance data access
  - Larger amount of data in memory
  - I/O elimination - Large buffer pool without OS paging problems

Drawbacks

- Still requires a substantial virtual pool size for effective usage
  - Limits on size (8GB max HP size)
- Contains ONLY clean pages - NOT updated pages not written to disk
- Page addressable, hence buffers must be in virtual pool before use
- No direct I/O in and out of a hiperpool

Solution

- Data space buffer pools

2.14 Hiperpools

Although a very good solution for many years, and exploited by many customers, hiperpools come with a set of limitations of their own:

- DB2 V7 limits the total size of hiperpools to 8 GB. This limit could be raised, however hiperpools have several other drawbacks which make them undesirable as a long term solution:
  - They are only page addressable, not byte addressable, and therefore buffers must be moved into the virtual pool before they can be used.
  - They can contain only clean pages.
  - You cannot do I/O directly into or out of a hiperpool.
  - The hiperpool page control blocks reside in the DBM1 address space and thus contribute to virtual storage constraints.
  - Hiperpools require a fairly substantial virtual pool size for effective use. Typically, the hiperpool to virtual pool size is on the order of 2:1 to 5:1. Therefore, virtual pool size ultimately limits hiperpool size.
  - A separate set of latches is used to manage hiperpool and virtual pool buffers, so as the frequency of page movement between virtual pools and hiperpools increases, the Least Recently Used (LRU) management of these pools increases, and latch contention issues can quickly arise.

Hiperpools were designed over a decade ago to exploit ESA and to make efficient use of large amounts of expanded storage. To overcome some of the hiperpool limitations, DB2 V6 introduced virtual pools in data spaces.

BPs in Data Spaces - Benefits

Data spaces provide a good short term solution by exploiting 64-bit REAL

- Buffer pools and statement caching in data spaces
  - Frees up space for other work in the DBM1 address space
  - Performance penalty when not 100% backed by real storage

Advantages of data spaces over hiperpools

- Read and write cache with direct I/O to data space
- Byte addressability
- Very large buffer pool sizes
  - 32 GB for 4K page size
  - 256 GB for 32K page size

Single buffer pool can span multiple data spaces

Excellent performance experienced with

- z990, z900 and large processor storage
- OS/390 V2R10 and z/OS releases

2.15 BPs in data spaces - benefits

Data spaces provide a good short term solution by exploiting the 64-bit real memory support introduced with OS/390 V2R10. Since DB2 V6 you can place buffer pools (as well as the dynamic statement cache) in data spaces, freeing up storage for other work in the DBM1 address space. Note that there is a performance penalty associated with buffer pools in data spaces, when such data space buffers are not 100% backed by real storage.

The advantages of data spaces over hiperpools are:

- Read and write cache with direct I/O to data space
- Byte addressability
- Large buffer pool sizes (32 GB for 4 KB page size and 256 GB for 32 KB page size)
- Single buffer pool can span multiple data spaces
- Multiple buffer pools in same data space
- Excellent performance experienced with z990 and z900 with large processor storage
- Performance dependent upon being in 64-bit real mode

With the z/Architecture processors running in 64-bit addressing mode and having no expanded storage (all storage is central), hiperpools have no reason to exist.

Hiperpools and supporting ESO hiperspace APIs are emulated when running in 64-bit ESAME mode, as there is no expanded storage on a zSeries machine. So when using hiperpools, we move real storage (hiperpool) to real storage (buffer pool). zSeries is not as efficient relative to G6 with regards to the MVPG instruction, which is issued for page movement; when moving pages between expanded storage and real storage, zSeries took 20 to 30% more CPU time compared to G6.

Therefore, data space buffer pools are definitely recommended over hiperpools when running on zSeries with 64-bit ESAME mode, when backed by real storage.

BPs in Data Spaces - Problems

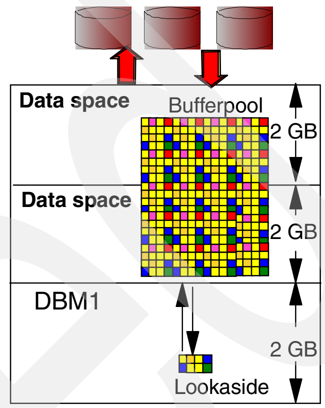
Redbooks

Undesirable as a long term solution due to size limitations

- Still limited to 32GB (for 4K buffer pools)
  - Imposed by control structures in DBM1 that allow up to 8 million pages

Overhead in copying buffers between data spaces and lookaside pool as they are accessed and updated

- Lookaside pool resides in DBM1



2.16 BPs in data spaces - problems

Even though data space buffer pools outperform hiperpools on a zSeries running in 64-bit mode with sufficient amounts of real storage to back up the data space buffers, data space buffer pools are not free of shortcomings either.

The total size of data space virtual pools is limited to 32 GB (4 KB page size). This limit is imposed by the control structures to manage the data space pages, which reside in the DBM1 address space, and allow up to a maximum of 8 million pages. Also, the lookaside pool resides in DBM1 and requires storage.

Although data spaces provide a good short term solution for exploiting 64-bit real memory, they are undesirable as a long term solution, not only because of the size limitations, but also because of the overhead involved with copying buffers between the data spaces and the lookaside pool as they are accessed and updated. Data spaces have scalability issues, and the VSTOR limit of 2 GB for DBM1 address space remains the biggest constraint to achieving linear scalability for DB2 systems.

Other Considerations

Redbooks

Problems with both hiperpools and data space buffer pools

- Still require work space in the DBM1 address space
- Require page movement
  - Elapsed time
  - CPU time
- Some amount of monitoring and tuning still required

Business requirements to manipulate more data in DB2

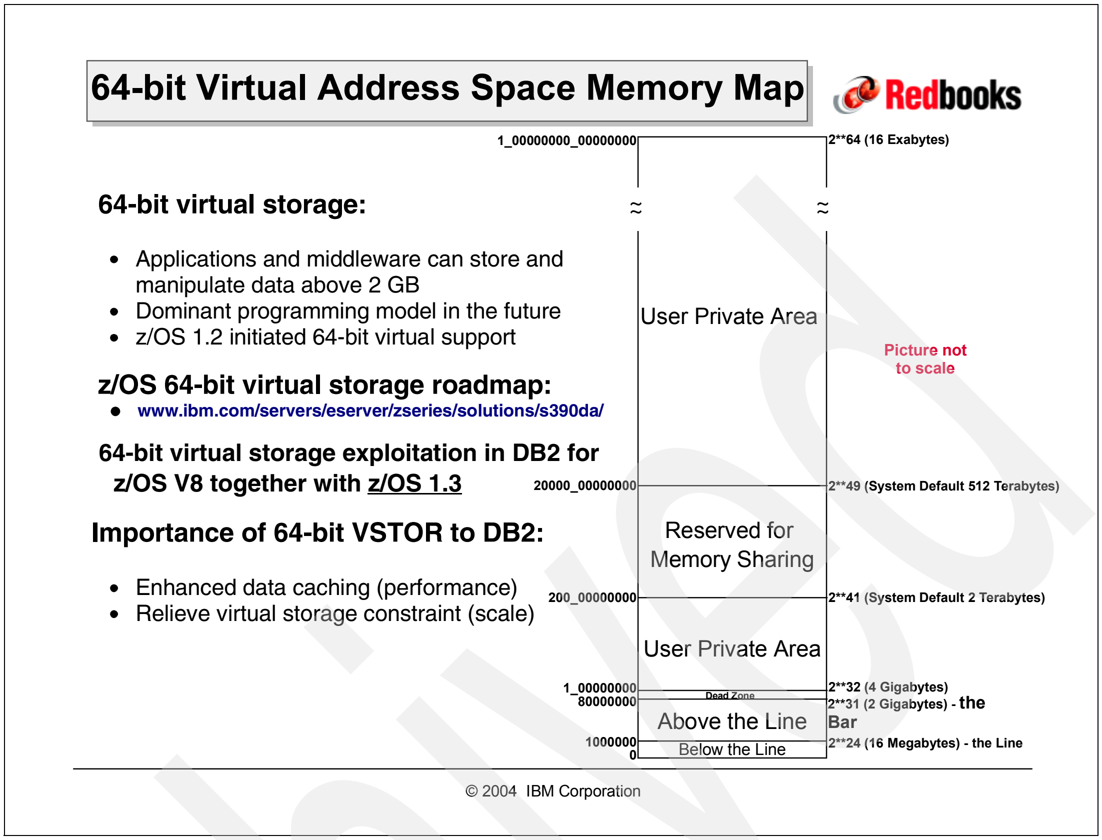
- Larger objects
  - Image, text, video
- XML data

2.17 Other considerations

Some other considerations and problems with both hiperpools and data space buffer pools is that they require page movement, which takes CPU cycles and elapsed time to complete. They both also require a considerable amount of virtual storage inside the DBM1 address space.

In addition, new business requirements demand that more data and new types of data need to be handled by DB2 systems. These requirements include:

- Handling larger objects
- Handling Image, text, video data
- Handling XML data



2.18 64-bit virtual address space memory map

Whereas OS/390 V2R10 introduced 64-bit *real* addressing support, which benefits all DB2 versions, z/OS V1.2 introduced the infrastructure for the 64-bit *virtual* addressing support that DB2 Version 8 will exploit.

In this visual, we show the mapping of an address space using 64-bit addressability. The left-hand column numbers are upper boundary hexadecimal values associated with each area. The picture is obviously not drawn to scale. The area above 2 GB, also called "above the bar", would be dramatically larger if drawn to scale, and should be larger by a factor of 10 to the power of 12. This huge addressing range should be able to accommodate the storage requirements for many years to come.

With z/OS 64-bit virtual storage support, database subsystems like DB2 and other middleware can make use of this large 64-bit virtual storage to increase capacity by supporting a larger number of concurrent users and concurrent transactions. DB2 was one of the first subsystems designed for the MVS and OS/390 31-bit environment and one of the first subsystems to support MVS and OS/390 extended addressability.

DB2 has established itself as the enterprise database manager of choice for OS/390 with its abilities to handle varied large system workloads efficiently, including transaction and large query environments. DB2 is now again one of the first subsystems to take advantage of 64-bit data addressability. With 64-bit virtual storage exploitation, DB2 can relieve virtual storage constraints and provide capacity enhancement to a large number of DB2 applications.

DB2 64-bit virtual storage exploitation is a two-step plan described in the white paper, *IBM eserver zSeries 900 z/OS 64-bit Virtual Storage Roadmap,* available in PDF from the Web site:

http://www.ibm.com/servers/eserver/zseries/library/whitepapers/gm130076.html

In summary, the first step is to take advantage of the basic 64-bit virtual storage system infrastructure and system services to enhance database manager buffer support. With DB2 V8 all existing 31-bit DB2 applications (including those written in Assembler, PL/I, COBOL, FORTRAN, C/C++ and Java), as well as future DB2 applications, can benefit transparently from DB2's 64-bit virtual storage support.

The second step is to exploit the z/OS C/C++ and Java infrastructure to extend DB2's support to 64-bit C/C++ and Java applications. DB2 APIs will be enabled to support 64-bit data, to facilitate 64-bit C/C++, and Java applications, to access existing data or store new data into the database.

DB2 V8 makes use of extra services provided by z/OS V1R3, and therefore you have to be on z/OS V1.3 before you can migrate to DB2 Version 8.

Note: When DB2 V8 became generally available in March 2004, z/OS V1.3 was the minimum z/OS level required for DB2 V8. Since March 31 2005, z/OS V1.3 is out of service. Therefore the actual minimum z/OS level to run DB2 V8 is now z/OS V1.4.

DB2 V8 Exploitation of 64-bit Virtual Support

z/OS 64-bit virtual addressing support
- Why implement 64-bit VSTOR support
- What is moving above the 2G Bar
- Benefits of 64-bit VSTOR support

Moving above 2GB bar
- Buffer pools and related control blocks
  - Configuration changes
  - Migration sizing of buffer pools
  - New error messages
- RIDPOOL
- Compression dictionaries
- Sort pool
- Castout buffers
- EDM pool - DBDs, OBDs, dynamic statement cache

IRLM V2.2 64-bit IRLM for DB2 V8

AMODE considerations

Planning

Other recommendations / information

2.19 DB2 V8 exploitation of 64-bit virtual storage

As mentioned before, DB2 Version 8 will be one of the first IBM subsystems, if not the first, to exploit 64-bit virtual storage support. This is mainly achieved by moving large storage consumers from above the 16 MB line, to above the 2 GB bar.

As discussed in more detail later on in this publication, DB2 V8 uses a multi-step migration process; from V7 to V8 compatibility mode, to V8 enabling-new-function mode, to V8 new-function mode. Most of the new functions in DB2 V8 are only available when you get to new-function mode. The 64-bit exploitation, on the other hand, is available after you migrate your Version 7 system to Version 8 compatibility mode; in other words, it is available from day one in DB2 V8.

The 64-bit enablement of DB2 is completely transparent to all existing applications.

Why Implement 64-bit VSTOR support?

Significant importance to DB2:

- Enhanced data caching in memory will deliver higher performance
- Relieving virtual storage constraint will provide scalability

Increases maximum buffer pool sizes

Eliminates need for hiperpools and data spaces

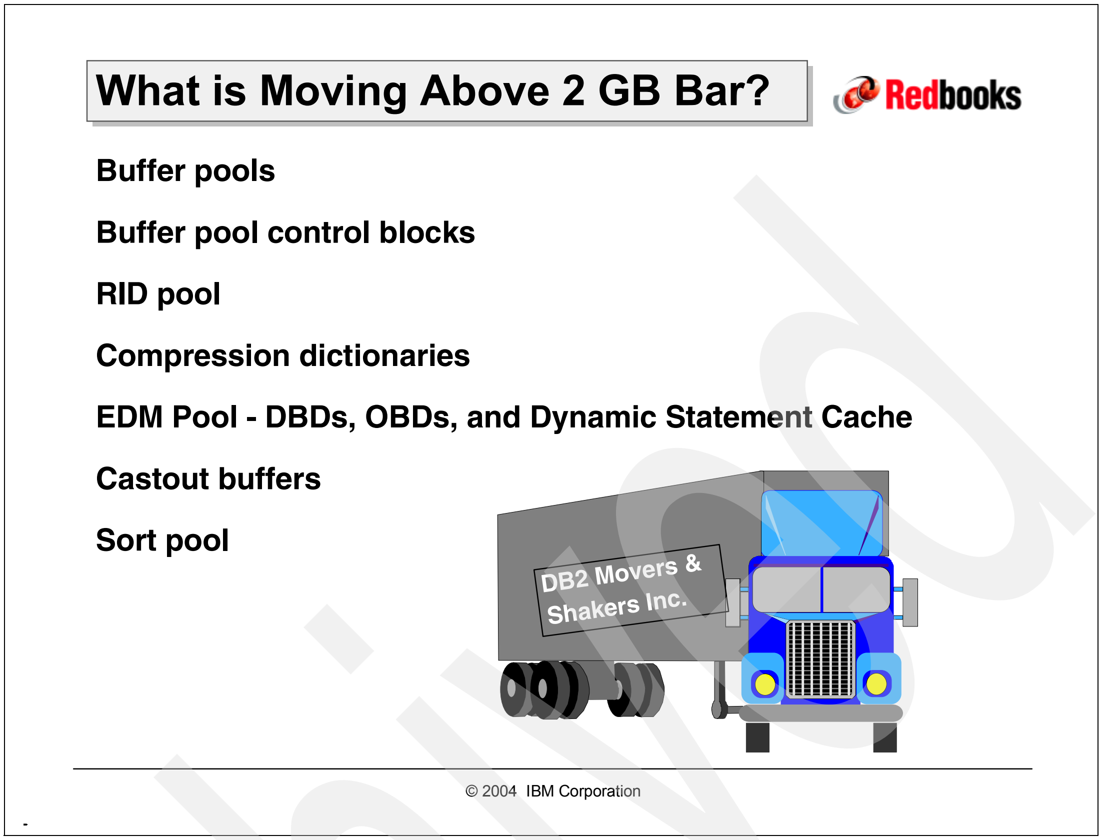
Simplifies DB2 systems management and operational tasks

2.20 Why implement 64-bit VSTOR support?

The implementations of 64-bit virtual support is very important to DB2 for z/OS. It allows DB2 to significantly enhance its in-memory data caching capabilities, and allows DB2 to deliver better performance.

Relieving the currently existing virtual storage constraints will provide much better scalability for DB2 systems in a mainframe environment. Today's virtual storage constraints in the DBM1 address space is the most important scalability inhibitor for DB2 subsystems.

With the advent of 64-bit virtual support in the operating system and DB2's exploitation thereof, it allows DB2 to increase maximum buffer pool sizes, and eliminates the need for hiperpools and data spaces, which will simplify DB2 systems management and operational tasks as well.



2.21 What is moving above the 2 GB Bar?

The following storage areas inside the DBM1 address space move above the 2 GB bar:

- Buffer pools
- Buffer pool control blocks
- RID pool
- Compression dictionaries
- EDM Pool - DBDs, OBDs, and dynamic statement cache
- Castout buffers
- Sort pool

DB2 Benefits of 64-bit VSTOR support Redbooks

VSTOR relief utilizes 64-bit virtual addressing to move data areas above 2GB bar in DBM1 address space

- Especially large storage areas like buffer pools will move

DB2's "data access" modules enhanced to access 64-bit addressable buffers "in place"

NO data movement between the data space and DBM1 as per data space buffer pools today

2.22 DB2 benefits of 64-bit VSTOR support

DB2 V8’s virtual storage constraint relief utilizes the z/OS 64-bit virtual addressing capabilities to move data areas above the 2 GB bar in DBM1 address space. Especially large storage areas like buffer pools, sort and RID pool, compression dictionaries, DBDs, and the dynamic statement cache will move above the 2 GB bar.

DB2's “data access” modules have been enhanced to access 64-bit addressable buffers “in place”. In V8 there is no data movement of pages between the data space and the lookaside buffer in DBM1, as is the case with data space buffer pools today in V7.

General expectations

The performance objective with the 64-bit virtual support is to increase system throughput with virtual storage constraint relief. This allows DB2 to support more concurrent threads.

The new factors affecting performance are:

- The 64-bit address translation
- Increased code size due to 4 byte versus 2 byte instructions
- More expensive linkage between modules

In addition, as with previous version to version migrations, it is to be expected that medium to large DB2 subsystems will see a 1% to 10% real storage increase (with smaller systems we will probably see slightly higher percentages, as there are a number of “fixed” size increases that will weigh more in smaller systems).

64-bit Virtual Buffer Pool Support

Redbooks

DB2 Version 8 is 64-bit exclusive (even in compatibility mode)

- Buffer pools always allocated above 2GB bar
- Eliminates need for hiperpools and data space buffer pools

Terminology

- As of V8, terms buffer pool and virtual pool become "synonymous"
- Data space buffer pools, hiperpools no longer exist
  No longer needed / supported in V8

Sizing and placement

- Buffer pool max size is 1TB
- Total buffer pool max size is 1TB
- **Make sure buffer pools are backed by real storage**
- Buffer pool control blocks also go above the bar
- Castout buffers (data sharing) above the bar

V7 buffer pool information saved for fallback

2.23 64-bit virtual buffer pool support

The main focus of DB2 V8 and virtual storage constraint relief is to utilize 64-bit virtual addressing to move large storage consumers like the DB2 buffer pools and their associated buffer control blocks above the 2 GB bar in the ssnmDBM1 address space.

DB2 V8 requires z/OS V1R3 or above as prerequisite. If an attempt is made to start DB2 V8 on an OS/390 or a z/OS R1 or R2 system, then DB2 issues an error message during startup and terminates.

> **Note:** These prerequisites can have implications for disaster recovery and sysplex cross-system restart scenarios.

The use of 64-bit virtual addressing greatly increases the maximum buffer pool sizes. DB2 V8 is 64-bit exclusive, and always allocates the buffer pools above the 2 GB bar. This effectively eliminates the need for hiperpools and data space buffer pools, thereby simplifying DB2 systems management and operations tasks. Therefore, hiperpools and data space buffer pools are no longer supported in DB2 V8. As of DB2 V8, the terms *buffer pool* and *virtual pool* become synonymous.

Buffer pools can now scale to extremely large sizes, constrained only by the physical memory limits of the machine (64-bit allows for 16 exabytes of addressability). System consolidation by having multiple DB2 images, with or without data sharing, is now possible without significant paging activity.

DB2 V8 increases the (theoretical) maximum buffer pool sizes to the limit of the architecture, 1 TB. This limit is imposed by the real storage available:

- The maximum size for a single buffer pool is 1 TB.
- The maximum size for summation of all active buffer pools is 1 TB.

The buffer pool control blocks, also known as page manipulation blocks (PMBs), and the data sharing castout buffers are moved above the 2 GB bar.

> **Important:** Before you get carried away and start allocating huge buffer pools to eliminate all DB2 I/Os in the system, you have to make sure that you do not overcommit your real storage on your z/OS image. Make sure that by increasing the buffer pool size, you do not introduce excessive paging. This is especially important in a 64-bit real environment where there is no expanded storage, and all paging will be to DASD. Also remember that DB2 is usually not the only application running on the z/OS image that requires real storage.

DB2 V8 maintains the old V7 virtual pool and hiperpool definitions as they were at the time of migration to be used in case of fallback, and it adds new definitions of buffer pools for those catalog page sets which are not defined with a 4K buffer pool.

If you are running with adequate real storage and have many I/Os in some buffer pools, you should look into using the new buffer pool PGFIX(YES) option to improve performance. See 2.23.4, “Additional buffer pool information” on page 42 for more details on the PGFIX(YES) option.

Configuration Changes

Max number of read, write, castout engines are now all 600

DB2 ensures at least 4TB above bar using MEMLIMIT keyword

Additional enhancements:

Parameter	Value
BP0	Minimum 2000 and default (from 56)
BP8K0	Minimum 1000 and default
BP16K0	Minimum 500 and default
BP32K	Minimum 250 and default
CTHREAD	Default is now 200 (from 70)
MAXDBAT	Default is now 200 (from 64)
CONDBAT	Default is now 10 000 (from 64)
IDFORE	Default is now 50 (from 40)
IDBACK	Default is now 50 (from 20)

2.23.1 Configuration changes

DB2 now uses 600 engines for asynchronous reads (already 600 since V6), writes (300 in V7) and castout (300 in V7) processing. This will increase the amount of storage required by system threads. (A system thread is usually around 128 KB per system thread). The good news is that for castout buffers, this storage moves above the bar.

At startup time DB2 specifies a MEMLIM of 4 TB. The 4 TB limit is more to draw the line somewhere, than a real practical limit on today's machines. DB2 uses MEMLIM=4TB to provide some protection for massive storage leaks that could bring down the entire LPAR. Using 0 TB has the same effect as REGION=0M for storage below the bar, indicating that there is no limit, and that is not advisable. But even with a limit of 4 TB, you must have enough paging devices available to store all those page frames; otherwise you can still bring down the entire LPAR, before reaching the 4 TB limit.

Buffer pools can now scale to extremely large sizes, constrained only by the physical memory limits of the machine. The recommendation still stands that buffer pools should not be over-allocated relative to the amount of real storage that is available. DB2 V8 issues the following warning messages:

- **DSNB536I:** This indicates that the total buffer pool virtual storage requirement exceeds the size of real storage of the z/OS image.
- **DSNB610I:** This indicates that a request to increase the size of the buffer pool will exceed twice the amount of real storage.
- **DSNB508I:** This is issued when the total size used by all buffer pools exceeds 1 TB.

For more information about the effect of these messages, see 2.23.3, "Buffer pool sizing" on page 40. Some DSNZPARM defaults have also changed, including CTHREAD, MAXDBAT, CONDBAT, IDFORE, and IDBACK. See also 12.5.5, "Changed defaults" on page 911.

Migration Sizing of Buffer Pools

Redbooks

Data spaces and virtual pools (NO Hiperpool) - VPSIZE is used

Virtual pools with a corresponding hiperpool - VPSIZE + HPSIZE is used

VPSEQT, VPPSEQT and VPXSEQT keep their values

- Even if buffer pool size is determined by VPSIZE + HPSIZE

DB2 V8 maintains original virtual pool and hiperpool sizes for fallback

No change for new installations - Values taken from install process

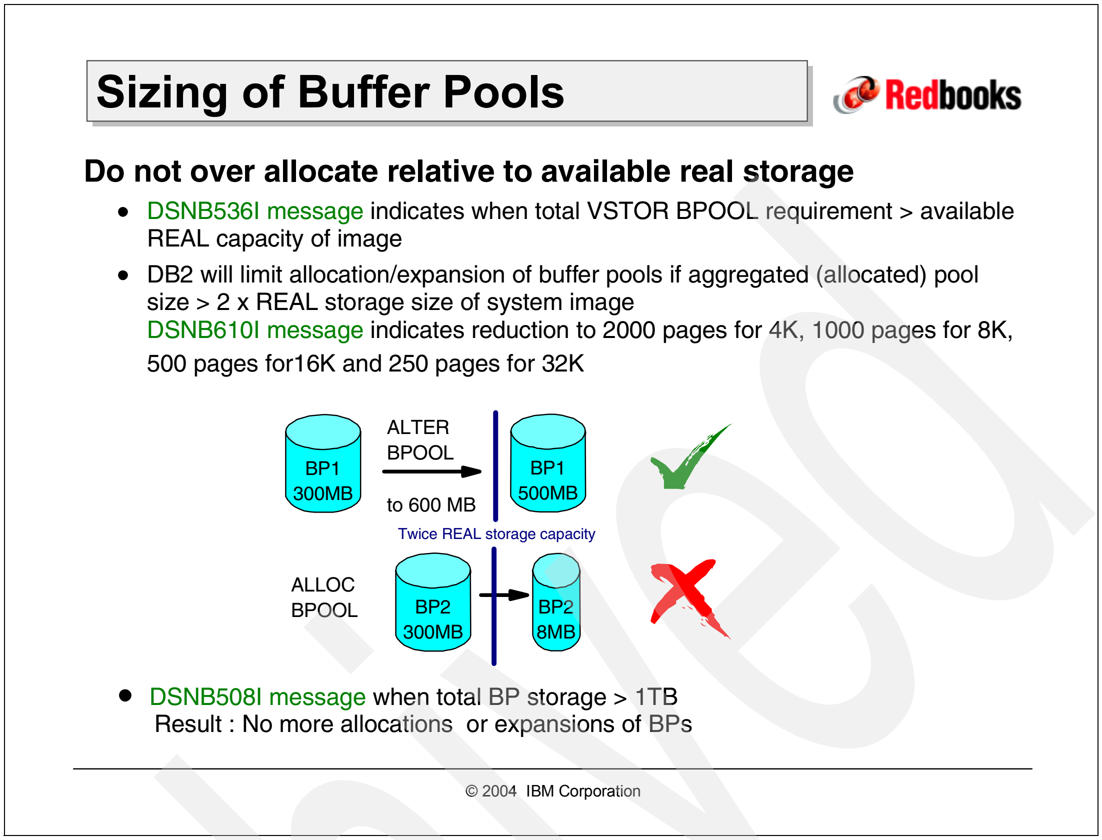
2.23.2 Migration sizing of buffer pools

When first migrating to V8, DB2 uses the following parameters to determine the size of the buffer pool:

- For data space pools and virtual pools with no corresponding hiperpool, the VPSIZE is used.
- For virtual pools with a corresponding hiperpool, VPSIZE + HPSIZE is used.
- VPSEQT, VPPSEQT and VPXSEQT keep their previous values, even if the buffer pool size is determined by VPSIZE + HPSIZE.

DB2 V8 maintains the old V7 virtual pool and hiperpool definitions as they were at the time of migration to be used in case of fallback, and it adds new definitions of buffer pools for the catalog.

For newly installed V8 subsystems, as in prior releases, DB2 initially uses the buffer pool sizes that were specified during the installation process. Thereafter, the buffer pool attributes can be changed via the ALTER BUFFERPOOL command, and they are stored in the BSDS.



2.23.3 Buffer pool sizing

Now that there is plenty of virtual addressing available, there is a concern that greatly over-allocated virtual buffer pools can cause an auxiliary storage shortage, and lead to a system wait state. To draw your attention to this, DB2 will put out warning messages:

- **DSNB536I**: This message is issued when the total size of the virtual pools exceeds the total amount of real storage on the z/OS image. This is only a warning message, but it should be a sign that you are probably overcommitting your real storage, because the real storage is also needed for other storage users like the EDM and sort pool in DBM1, as well as all other users in the system. DB2 puts out a message DSNB538I when you are below the real storage size again. (These messages are not new in V8.)
- **DSNB610I:** This message is issued when the total buffer pool size reaches twice the amount of real storage on the z/OS image, but the total amount of buffer pool space is still less than 1 TB. When this limit is reached, DB2 starts taking action.

  From this point on, allocations of new buffer pools or expansions of existing buffer pools are limited to 8 MB per ALTER command, as follows:

  – 2000 pages for a 4 KB buffer pool
  – 1000 pages for a 8 KB buffer pool
  – 500 pages for a 16 KB buffer pool
  – 250 pages for a 32 KB buffer pool

In the example in the visual, let us assume that we are 200 MB below the limit of twice the amount of real storage capacity. Changing a 300 MB pool (for example, BP1) to 600 MB will only increase the size to 500 MB because that is where we hit the threshold. Now that we have exceeded the threshold, allocating another 300 MB pool (for example, BP2) only results in a new pool (BP2) of 8 MB.

It is worth noting that this “constricted” size is not saved. If a DB2 restart occurs and BP2 is allocated before we hit the threshold again, it will be 300 MB - the intended size. However, because we are now over the threshold again, when BP1 needs to be allocated, it will only be 8 MB.

- **DSNB508I** is issued when the total size used by all buffer pools exceeds 1TB. At that time, no new buffer pools will be created. Existing buffer pools cannot be expanded.

Additional Buffer Pool Information

Redbooks

ALTER BUFFERPOOL command parameters NO LONGER supported:

- VPTYPE, HPSIZE, HPSEQT, CASTOUT
  - If the above are specified, ONLY a warning message is issued

Buffer pool long-term page fixing

- New PGFIX(YES/NO) option
  - Fixes buffer page in real storage
  - **ALT BPOOL(BPxx) VPSIZE(0) followed by an**
  - **ALT BPOOL(BPxx) VPSIZE(yyyy) PGFIX(YES)**
  - Alter effective at the next BP allocation
  - Can be performance boost for I/O intensive workloads (up to 10% CPU reduction)

Parameters remaining unchanged:

- VPSEQT, VPPSEQT, VPXPSEQT, DWQT, VDWQT, and PGSTEAL

LSTATS report removes references to hiperpool related counters

2.23.4 Additional buffer pool information

The following ALTER BUFFERPOOL command parameters are no longer supported: VPTYPE, HPSIZE, HPSEQT, CASTOUT. If they are specified, a warning message DSNB539I is issued. The other parameters remaining unchanged are VPSEQT, VPPSEQT, VPXPSEQT, DWQT, VDWQT, and PGSTEAL, although the defaults for DWQT, VDWQT for local buffer pools, and CLASST and GBPOOLT for group buffer pools are changing when installing a new V8 (not when migrating an existing system). See 12.5.5, “Changed defaults” on page 911 for details.

Buffer pool long-term page fixing

In addition, DB2 Version 8 introduces a new ALTER BUFFERPOOL parameter called PGFIX. It allows customers to fix the buffer pages once and keep them fixed in real storage. This avoids the processing time that DB2 needs to fix and free pages each time there is an I/O (remember that you always have to page fix a page before doing an I/O operation in order to prevent the operating system from stealing it when you are reading data into it).

PGFIX should be used for subsystems' buffer pools which read or write frequently. The recommendation is to alter your DB2 Version 8 buffer pools which have frequent page reads or writes to use PGFIX(YES) if you have sufficient real storage available for these buffer pools.

Important: Again notice the importance of having sufficient real storage to back up your buffer pool 100%. Having 99.9% is not good enough, as the LRU buffer steal algorithm will introduce paging if there is insufficient real storage to back up your buffer pool completely. Therefore, make sure your buffer pool is fully backed by real storage before you start using PGFIX(YES).

In some cases, this processing time (CPU time) reduction can be as much as 10% for I/O intensive workloads with fixed buffer pool pages. To use this option, you can use the PGFIX(YES) option on the ALTER BUFFERPOOL command. The ALTER takes effect at the next BP allocation.

For user data, you can issue the following commands:

```
-ALT BPOOL(bpname) VPSIZE(0)
-ALT BPOOL(bpname) VPSIZE(yyyy) PGFIX(YES)
```

Here, `bpname` is the name of the buffer pool and `yyyy` is the current size of your buffer pool.

For the catalog and directory you can use:

```
-ALT BPOOL(bpname) PGFIX(YES)
-STOP DATABASE or STOP DB2
-START DATABASE or START DB2
```

Notice that the page fixing is at the buffer pool level. This option is available in V8 compatibility mode and beyond.

The LSTATS report removes the references to hiperpool related counters.

New Error Messages

Conditions preventing DB2 (re)start:

- DSNY011 - z/Architecture required
  - Startup abends with reason code 00E8005A
- DSNY012 - z/OS 1.3 (or later) required
  - Startup abends with reason code 00E80058

ALTER BUFFERPOOL command parameters no longer supported:

- VPTYPE
- HPSIZE
- HPSEQT
- CASTOUT
  - DSNB539I - Specified value is ignored, continues processing other parameters

Revised message information:

- References to "virtual buffer pool" changed to "buffer pool"
- References to "data space buffer pool" or "hiperpool" deleted
- All messages changed to reflect buffer pool and value changes

2.23.5 New error messages

As mentioned before, you need to be running on a zSeries machine in 64-bit mode, as well as being on z/OS v1.3 or above. In order to prevent any problems, DB2 verifies these prerequisites at DB2 (re)start.

In case you are not running on a z/Architecture, startup puts out a message DSNY011 - z/Architecture required and abends with reason code 00E8005A.

In case you are not on z/OS V1.3 or above, restart puts out a message DSNY012 - z/OS 1.3 (or later) required and abends with reason code 00E80058.

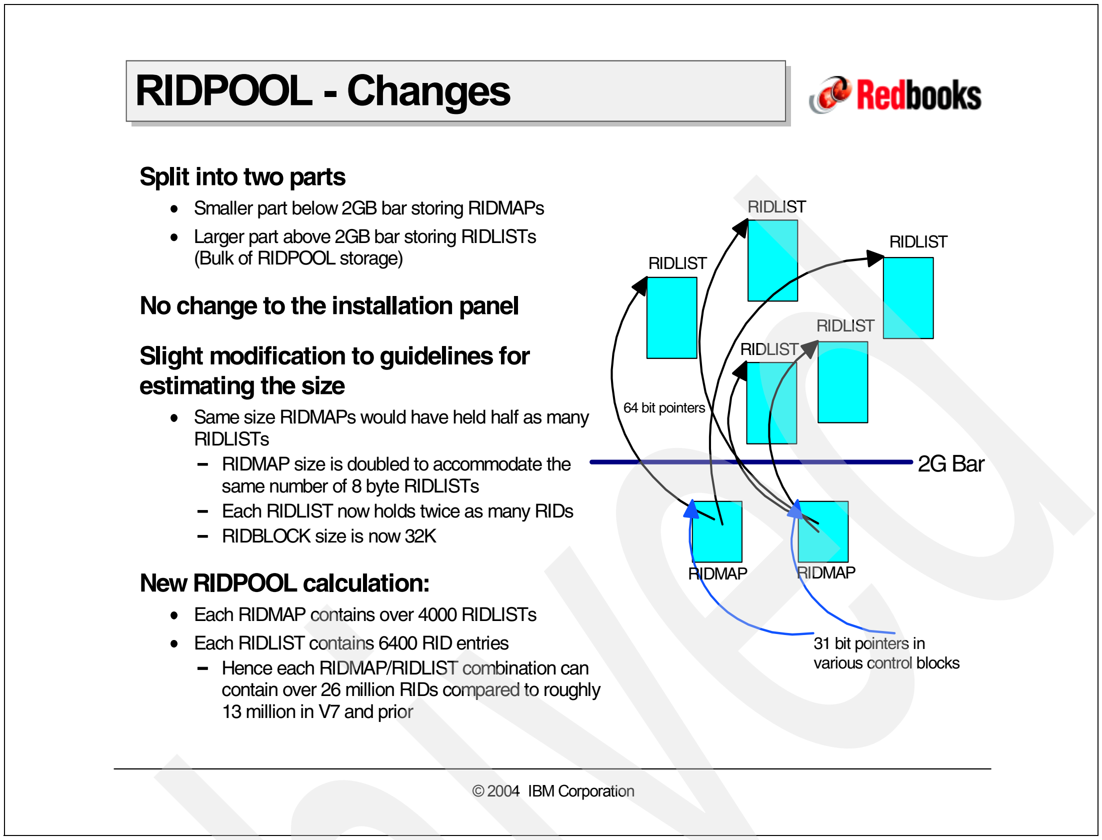
As mentioned before, the following ALTER BUFFERPOOL command parameters are no longer supported:

- VPTYPE
- HPSIZE
- HPSEQT
- CASTOUT

In case you forget to remove them, or specify them by accident, you receive a DSNB539I warning message indicating that the specified value is ignored, and processing continues for the other parameters.

All messages related to buffer pools have been revised:

- References to “virtual buffer pool” have been changed to “buffer pool”
- References to “data space buffer pool” or “hiperpool” have been deleted



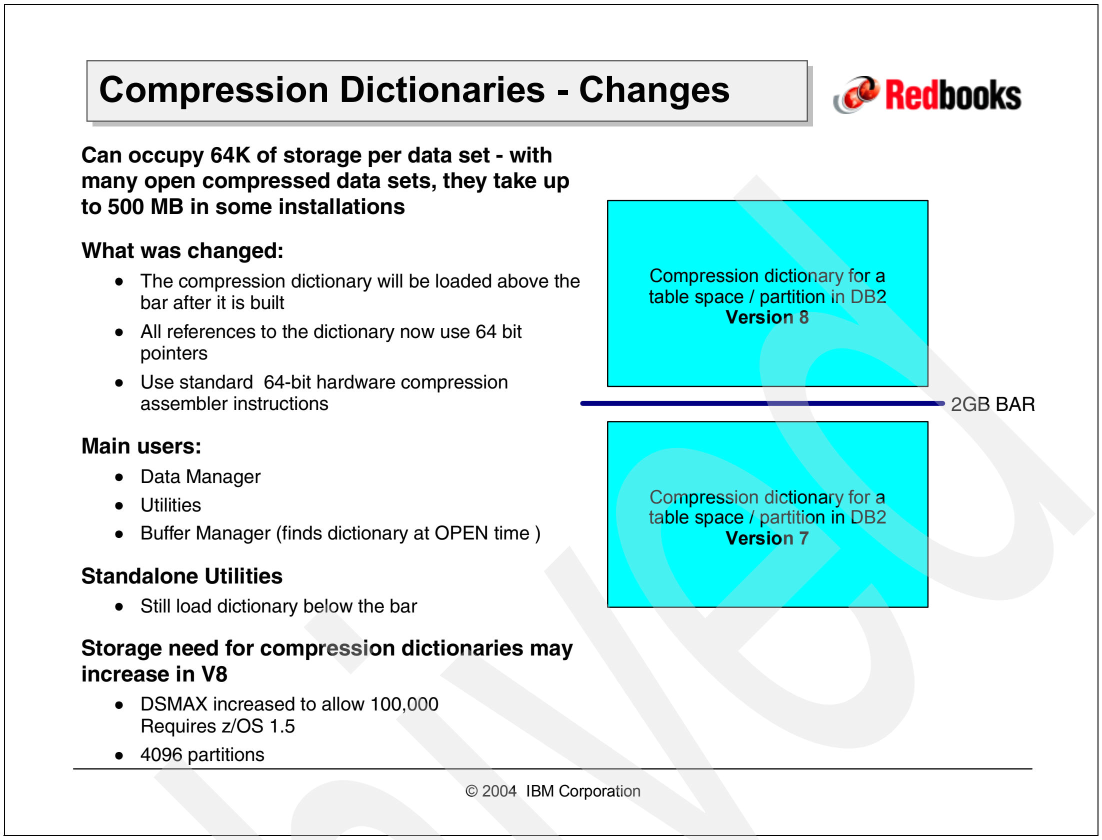
2.24 RIDPOOL - moving above the bar

The RID pool is split into two parts. The RID pool part below the 2 GB bar stores the RID maps which are usually small in number, and the RID pool part above the 2 GB bar contains the RID lists which normally comprise the bulk of the RID pool storage. (For typical DB2 subsystem with heavy RID pool usage, the great majority of RID pool is for RID lists.)

Because of the changes, there are some slight modifications in estimating the size for the RID pool. The same size RIDMAPs would have held half as many RIDLISTs. The RIDMAP size is doubled to accommodate the same number of 8 byte RIDLISTs, and each RIDLIST now holds twice as many RIDs. Each RIDBLOCK is now 32 KB in size.

Here is the new RIDPOOL calculation:

- Each RIDMAP contains over 4000 RIDLISTs.
- Each RIDLIST contains 6400 RID entries.
- Each RIDMAP/RIDLIST combination can then contain over 26 million RIDs, versus roughly 13 million in previous DB2 versions.

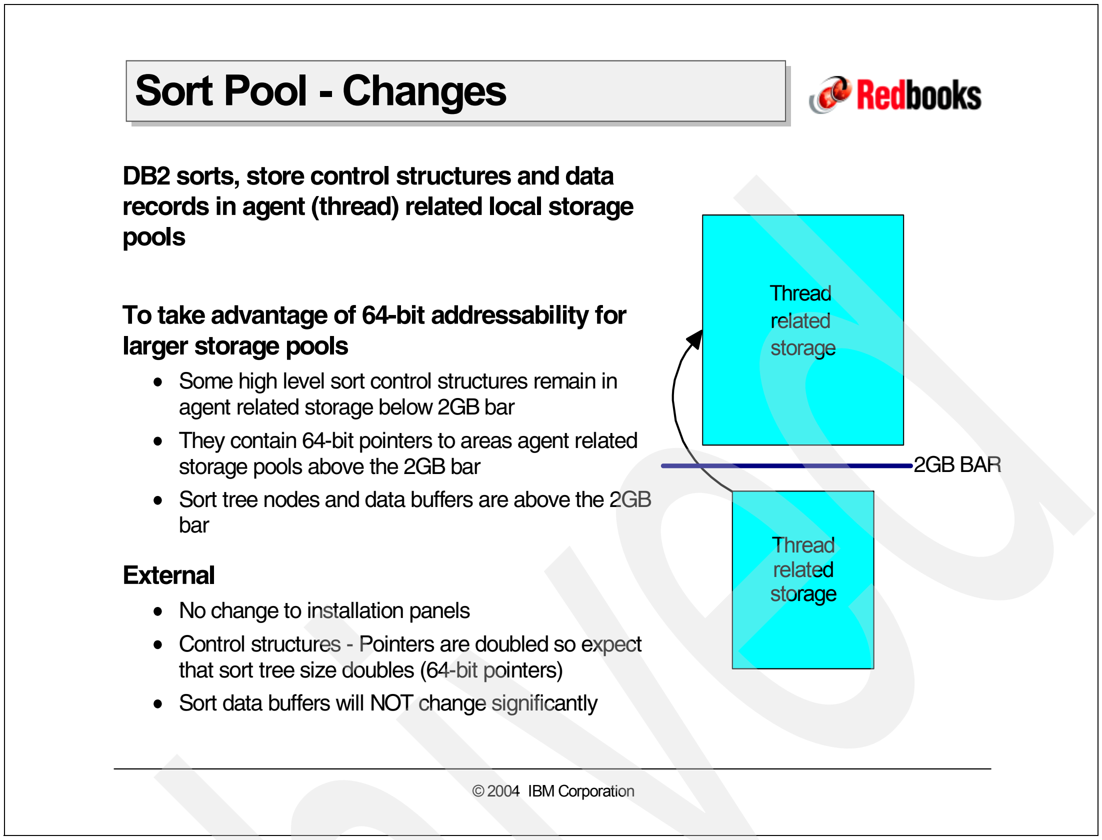


2.25 Compression dictionaries - moving above the bar

The compression dictionary for a compressed table space is loaded into virtual storage for each compressed table space or partition as it is opened. Even though it is not accessed frequently, it occupies a good chunk of storage while the data set is open. A compression dictionary can occupy up to 64 KB bytes of storage per data set (sixteen 4 KB pages). For some customers, those who have a large number of compressed table spaces, the compression dictionaries can use up as much as 500 megabytes. Therefore, moving the dictionary above the 2 GB bar provides significant storage relief for many customers.

DB2 V8 can further increase the compression dictionary storage requirement for some systems, as V8 also implements support for 4096 partitions for a single table; if they are all compressed and open, you have 4096 compression dictionaries in memory. This was another driver for moving compression dictionaries above the 2 GB bar.

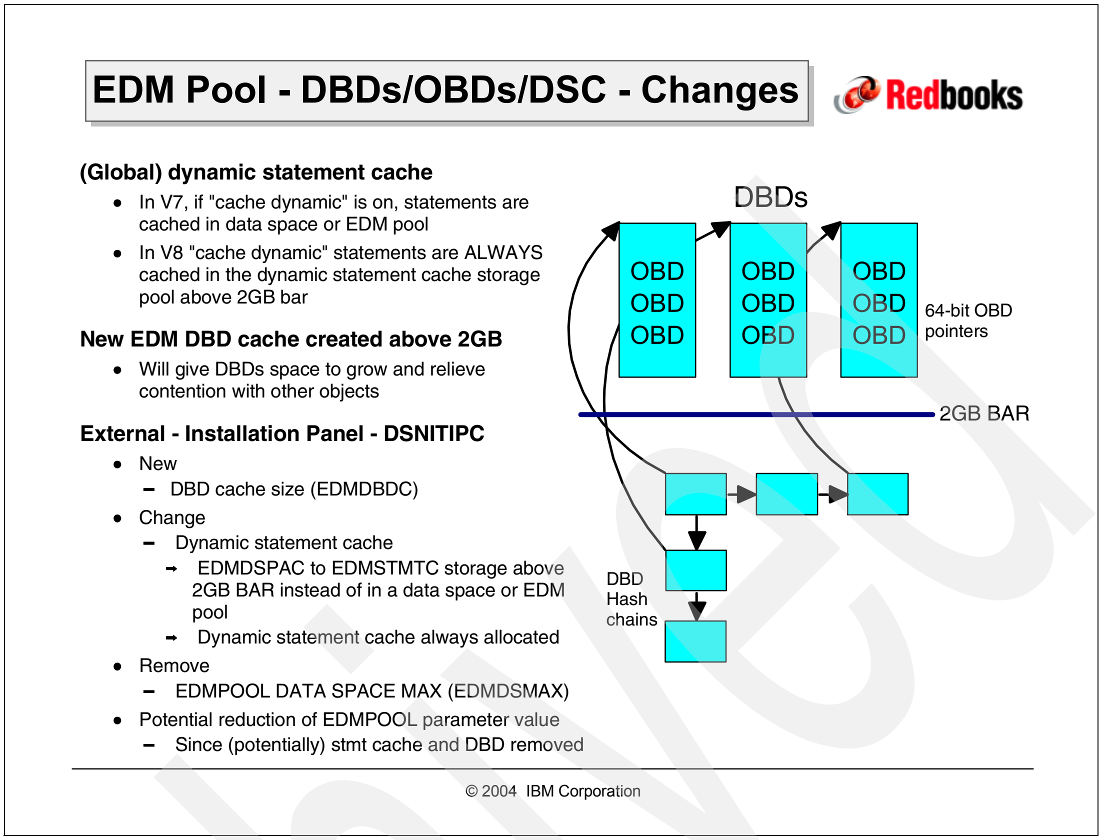
The compression dictionary is loaded above the bar after it is built. All references to the dictionary now use 64-bit pointers. Compression uses standard 64-bit hardware compression instructions. Standalone utilities still load the dictionary below the bar.



2.26 Sort pool - moving above the bar

Sorting requires a large amount of virtual storage, as there can be multiple copies of the data being sorted at a given time. Two kinds of storage pools are used for DB2 V8 sort (also known as RDS sort) to store various control structures and data records. One is a thread-related local storage pool below the bar, and the other is thread-related storage pool sort pool above the bar.

To take advantage of the 64-bit addressability, some high level sort control structures remain in thread-related storage below the 2 GB bar. These structures contain 64-bit pointers to areas in the thread-related storage pool above the 2 GB bar. The sort pool above the 2 GB bar contains sort tree nodes and data buffers, which take up the most space when sorting.



2.27 EDM pool - DBDs/OBDs/DSC - moving above the bar

In V8 the EDM pool is always split into three parts:

- Storage for the global dynamic statement cache
- Storage for DBDs
- Storage for plans and packages (SKCT, CT, SKPT, and PTs)

2.27.1 Dynamic statement cache

Today (V7), if global dynamic statement caching is active (DSNZPARM CACHEDYN=YES), the statements can be cached in a data space (EDMDSPAC>0), or in the normal EDM pool, if a data space is not defined. In V8, cached, dynamic SQL statements are always cached in the dynamic statement cache pool above the 2 GB bar (DSNZPARM EDMSTMTC>0). This value is used at DB2 startup time as the minimum value. This value cannot be decreased below the value that is specified at DB2 startup.

It is worth noting that in V8, storage for the dynamic statement cache is always allocated (at least 5 MB), and is no longer related to whether CACHEDYN is YES or NO. This is because the CACHEDYN DSNZPARM is online changeable in V8, and DB2 needs an initial size to be allocated at startup time to allow the size to be changed online.

2.27.2 Storing the DBDs

In V8, almost all of the storage related to managing DBDs is allocated above the 2 GB bar. This gives the DBDs the needed space to grow and relieves contention with other objects in the EDM pool. The size of the DBD cache is governed by the EDMDBDC DSNZPARM, with a size between 5 MB and 2 GB). This value is used at DB2 startup time as the minimum value. This value cannot be decreased below the value that is specified at DB2 startup.

2.27.3 Storage for plans and packages

Plans and packages (SKCT, CT, SKPT, and PT) are the only things that are stored in the EDM pool that we know from previous releases. The DSNZPARM is EDMPOOL as before.

The installation clist calculates values for all three parameters based on the input provided on the installation panels. The installation is aware of the fact that instead of a single storage structure (or potentially 2 storage areas if using statement caching in data spaces), V8 uses three different storage areas, governed by three DSNZPARM.

> **Important:** Using the current V7 size of the EDMPOOL DSNZPARM in V8, results in over allocation, as the DBDs (and potentially also the dynamic statement cache) no long reside in that storage area in V8.
>
> Therefore, it is important to provide accurate values as input to the installation clist (for example, for the number of databases in the system, the number of tables in a database, etc. on panel DSNTIPD). This will allow the clist to calculate accurate values for the EDMPOOL, EDMDBDC, and EDMSTMTC.
>
> If you are using a DB2 monitor product, you can also find out how much space each of those take up on your current V7 system. (Note that in V7, DB2 statistics do not distinguish between storage for dynamic statements and storage for SKPTs.)

Other Virtual Storage Enhancements

LOBs

- When LOB materialization is required, V8 uses storage above the bar instead of data space
- LOBVALA (per user) and LOBVALS (total system) remain valid

IPCS DB2 and IRLM dump formatting support 64-bit addressing

Number of open data sets has increased to 65 000 with V8 and z/OS V1.5

- Max 65 000 open data sets instead of 32K
- DB2 uses user-generated DDNAMES (instead of system-generated in V7 and prior)
- Close MIN(3% of DSMAX, 300) when DSMAX is reached

2.28 Other virtual storage related enhancements

It is worth noting that other storage areas moved above the bar as well, such as accounting blocks, lock trace, buffer trace, and so on, some of which are at the thread level, and can therefore still represent a considerable amount of storage in systems with a lot of concurrent thread activity.

In addition, LOBs are no longer materialized in a data space in V8, but above the 2 GB bar inside the DBM1 address space, as described next.

2.28.1 LOB data

When LOBs need to be materialized in V7, DB2 does so using data spaces. In V8, storage above the 2 GB bar inside the DBM1 address space is used instead. Storage allocation is limited by the system parameters previously used for materializing LOBs in data spaces:

- LOBVALA (the size per user):

  The default is 10240 KB, the limit value is 2097152 KB.

- LOBVALS (the size per system):

  The default is 2048 MB, the limit value is 51200 MB.

2.28.2 IPCS IRLM and DB2 dump formatting enhancements

Other changes have been made to complement the 64-bit virtual support. There are 64-bit serviceability enhancements such as the 64-bit dump formatter and continued IPCS support for both DB2 and IRLM.

2.28.3 Up to 65 000 open data sets

DB2 Version 8 allows for up to 65 000 open data sets. The previous limit was 32K. This enhancement is only available in z/OS V1.5 and above. The reason for keeping the current limit at 32K is because open data sets require storage for MVS control blocks below the 16 MB line. Since z/OS 1.2, dynamic allocations can request these control blocks to be allocated above the 16 MB line. With z/OS V1.5, media manager, the interface that DB2 uses, can fully exploit this feature. This reduces the amount of storage that DB2 requires below the 16 MB line.

Theoretically DB2 allows you to specify a DSMAX value up to 100 000. However, since z/OS V1.6 and below only support up to 65,041 open VSAM data sets, DB2 returns an SQLCODE -904 with a RC00C20113 when the concurrent open data sets exceeds 65,000. This limit is introduced by the APAR/PTF PQ96189/UQ96862.

This enhancement does not only provide virtual storage relief. It also removes a potential scalability problem with growing systems and growing machine capacity, especially for people that consolidate multiple applications into a single DB2 subsystem.

In addition, DB2 V8 will no longer use system-generated DDNAMEs, as the current limit there is also 32K. DB2 will generate its own DDNAMEs for dynamic allocation. This way DB2 can have better open data set performance as well, and also avoids the problem that the task which generates the system DDNAMEs becomes a bottleneck for parallel data set open.

When DSMAX is reached, DB2 currently attempts to close up to 3% of the open data sets. However, 3% of 100 000 is a large number, and this burst of data set close activity could be disruptive. Therefore DB2 will now close MIN(3%,300) data sets when DSMAX is reached.

IRLM V2.2 64-bit IRLM for DB2 V8

IRLM V2.2 is 64-bit application

- Ships both 64-bit and 31-bit version
  (64-bit version requires z/OS 1.3 or above)

All locks are above the bar

- PC=YES enforced, MAXCSA ignored (can still specify for compatibility)
- MEMLIM EXEC= parm manages storage size allowed above the bar
- Can manage up to 100 million locks
  - NUMLKTS (number of lock per table space) and NUMLKUS (number of locks per user) can be up to 100 million
- MAX STORAGE FOR LOCKS - passed to z/OS as MEMLIMIT value
- PAGE PROTECT startup option
  - Specifies whether IRLM loads its common storage modules into page-protected storage or not

2.29 IRLM V2.2 64-bit IRLM for DB2 V8

IRLM V2.2 is a 64-bit application. Actually, IRLM V2.2 ships with both a 31-bit version of the modules, as well as a 64-bit capable version. If the operating system is able to handle 64-bit, the 64-bit version are loaded. The 64-bit version requires the z/Architecture as well as z/OS 1.3 or above. DB2 V8 requires IRLM V2.2 in 64-bit mode.

With IRLM V2.2, locks always reside above the 2 GB bar. This allows IRLM to manage many more locks than was the case in previous releases. IRLM V2.2 can manage up to 100 million locks, around 16 times more than V2.1. Therefore, you can increase NUMLKTS (maximum number of locks per table space before lock escalation occurs) or NUMLKUS (number of lock per user before a resource unavailable occurs) up to 100 million. Here the same rules apply as for the storage areas that move above the 2 GB bar in DBM1. Make sure there is enough real storage to back this up.

Locks can no longer reside in ECSA. Therefore, the PC=NO parameter in the IRLM procedure is no longer honored. PC=YES is always used. This also means that the MAXCSA parameter no longer applies, as it was related to ECSA storage usage. You can still specify both PC=NO and MAXCSA= for compatibility reasons, but they are ignored by IRLM.

The fact that IRLM locks no longer reside in ECSA (when you were using PC=NO in V7) also means that a considerable amount of ECSA storage, which used to contain the IRLM locks, is now freed up in V8, and can be used by other subsystems (such as WebSphere) and other applications.

IRLM V2.2 has two additional parameters that can be specified in the IRLM startup procedure. These parameters are:

- MLMT - Max storage for locks:

  This specifies, in megabytes, gigabytes, terabytes, or petabytes, the maximum amount of private storage available above the 2 GB bar that the IRLM for this DB2 uses for its lock control block structures. The IRLM address space procedure uses this parameter (which you specify on the IRLM startup proc EXEC statement) to set the z/OS MEMLIMIT value for the address space. Ensure that you set this value high enough so that IRLM does not reach the limit. Note that the value you choose should take into account the amount of space for possible retained locks as well. IRLM only gets storage as it needs it, so you can choose a large value without any immediate effects. IRLM monitors the amount of private storage used for locks. If the specified limit is reached, new lock requests will be rejected unless they are “must complete”.

  After the PTF for APAR PQ87611 is applied (still open at the time of writing of this publication), you can dynamically change the amount of storage used for locks above the bar by using the z/OS command:

  ```
  MODIFY irlmproc,SET,PVT=nnnn
  ```

  Here, *nnnn* specifies the upper limit of private storage that is used for locks. You can specify the number in *nnnn*M (MB) or *nnnn*G (GB). If neither 'M' nor 'G' is specified, the default is 'M' (as in IRLM 2.1). The PVT= parameter will control private storage above the bar, and is tied to MEMLIMIT. A SET,PVT= command will cause the MEMLIMIT to be updated; never lower than the default 2G, and never lower than the amount of storage in use, plus the 10% of reserved space.

  Before this PTF, you can use the MLMT startup to control the maximum amount of storage used above the bar (MEMLIM) by IRLM.

  (Even before PQ87611, you can use the SET,PVT= command. However, it does not control the storage above the bar, but only the extended private area below the bar, and above the line, as with IRLM V2.1.)

- PGPROT - Page protect:

  Acceptable values are NO and YES (default). The page protect IRLM startup procedure parameter specifies whether IRLM loads its modules that reside in common storage into page-protected storage.

  YES indicates that modules located in common storage are to be loaded into page-protected storage to prevent programs from overlaying the instructions. YES is recommended because it requires no additional overhead after the modules are loaded, and the protection can prevent code-overlay failures.

  NO indicates that common storage modules are to be loaded into CSA or ECSA without first page protecting that memory.

Immediate Benefits

Simplified buffer pool monitoring and tuning

- Only ONE type of buffer pool
- Buffer pool size limits increased
  - Eases the worry over monitoring once the system stabilizes

DBM1 VSTOR constraint relief

- Increase CTHREAD - ECSA allocation may need increasing if CTHREAD raised
- Expect a single DB2 subsystem to run larger workloads
  - Could defer going to data sharing
  - Consolidating members in a group

Expected increases in workload

- Maximum number of prefetch, deferred write, and castout engines is now 600 in order to decrease "engine not available" conditions

IFCID's 217 and 225 reflect DBM1 VSTOR usage above and below 2GB bar

2.30 Immediate benefits

With DB2 V8's exploitation of 64-bit virtual storage, you can benefit from the following capabilities (even in V8 compatibility mode):

- Buffer pool monitoring and tuning becomes simpler:
  - Hiperpools and data space buffer pools are eliminated, thus reducing complexity. There is now only one type of buffer pool. Dynamic statement cache and LOB data spaces have also been eliminated.
  - Buffer pool size limits are increased, therefore buffer pool storage does not need to be as tightly monitored and controlled, especially in cases where there is a large amount of real memory available on the machine.
  - The ssnmDBM1 virtual storage constraints are no longer a key consideration in determining the optimum sizes for buffer pools.
- This may allow installations to increase the number of current active threads (CTHREAD). ECSA allocation may need to be increased if CTHREAD is raised.
- A single DB2 subsystem is able to run larger workloads. This may cause some installations to defer going to a data sharing environment for capacity reasons (since data sharing is still required for the highest scalability and availability), or to consolidate the data sharing group to fewer members.
- To handle the expected increases in workload, the maximum number of deferred write and castout engines are increased from 300 to 600 in order to decrease *engine not available* conditions.

Note: DB2 for z/OS V8 always uses 64-bit addressing, independent of the mode it is running in (compatibility mode, enabling-new-function mode, or new-function mode).

You can use IFCIDs 0217 and 0225 to monitor ssnmDBM1 virtual storage usage above and below 2 GB.

z/OS provides a new MEMLIMIT JCL keyword which controls how much virtual storage above the 2 GB bar is available in each address space.

Virtual storage information is collected in SMF by RMF™ in record type 78-2. RMF can produce:

- Common storage summary and detail reports
- Private area summary and detail reports

The reports are requested as follows:

- Specify S, either explicitly or by default, and RMF produces summary reports
- Specify D, and RMF produces both summary reports and detail reports

These are the available options:

- REPORTS(VSTOR(D)):

  This produces a summary and detail report for common storage.
- REPORTS(VSTOR(D,xxxxDBM1)):

  This produces a summary and detail report for common storage and a summary and detail report for the private area of the xxxxDBM1 address space.
- REPORTS(VSTOR(MYJOB)):

  This produces a summary report for common storage and a summary report for the private area of the MYJOB address space.

More information on setting up and monitoring 64-bit is contained in the technical bulletin, *z/OS Performance: Managing Processor Storage in an all "Real" Environment*, available from:

http://www.ibm.com/support/techdocs

AMODE Considerations

Redbooks

DB2 and IRLM have been enhanced to run with AMODE(64) where appropriate

- DBM1 address space now uses AMODE(64) to access the data above the bar
- IRLM 2.2 uses AMODE(64) to access the lock structures

Stored procedures support

- AMODE(64) stored procedures not supported

AMODE(64) callers (your applications) not supported

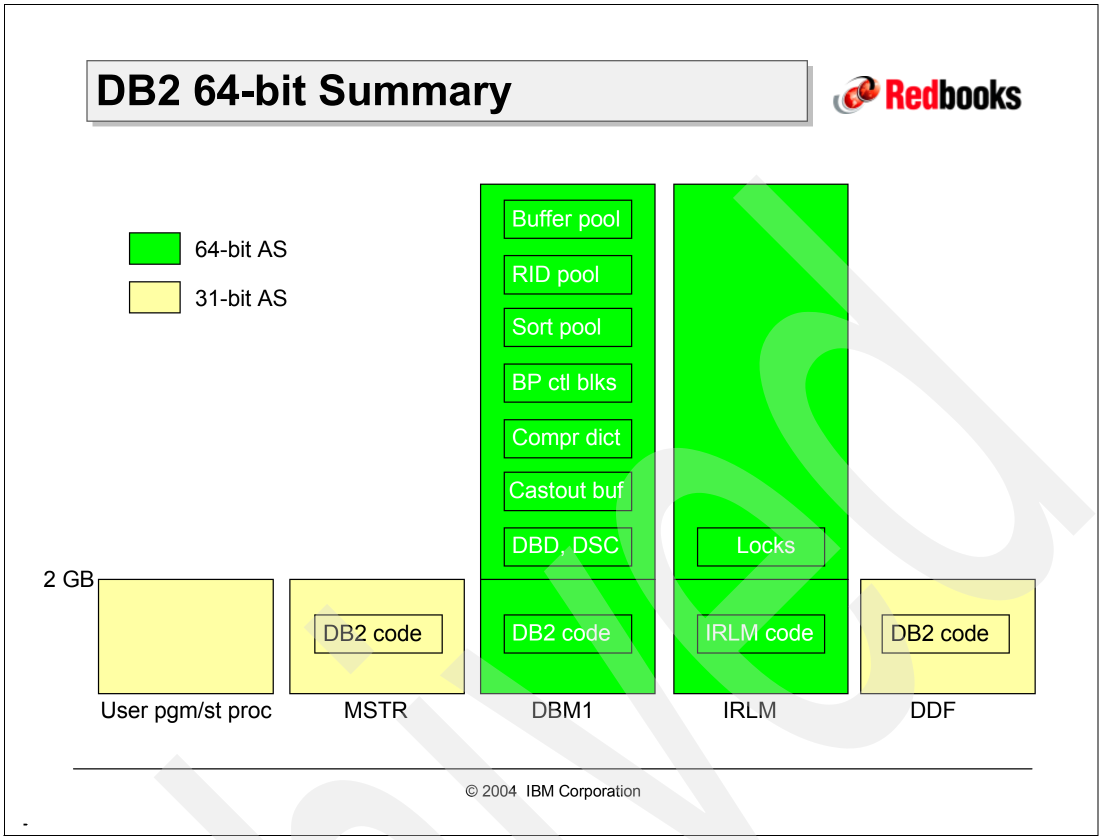
- This also means there is **NO** need to change your existing applications to benefit from DB2's effort to go 64-bit

Exits called in 31-bit mode

2.31 DB2 code considerations

DB2 has been enhanced to allow the DBM1 address space to use AMODE(64) to access the data above the bar. The same is true for IRLM 2.2. IRLM now uses AMODE(64) to access the lock structures that are now allocated above the 2 GB bar.

Note, however, that your applications as well as stored procedures are not allowed to run in AMODE(64). This is actually good news as it means that there is NO need to change your existing applications to benefit from DB2's effort to go 64-bit. In addition, exits are still called in 31-bit mode, so there is no need to change those to be able to co-exist with DB2 running in 64-bit mode.

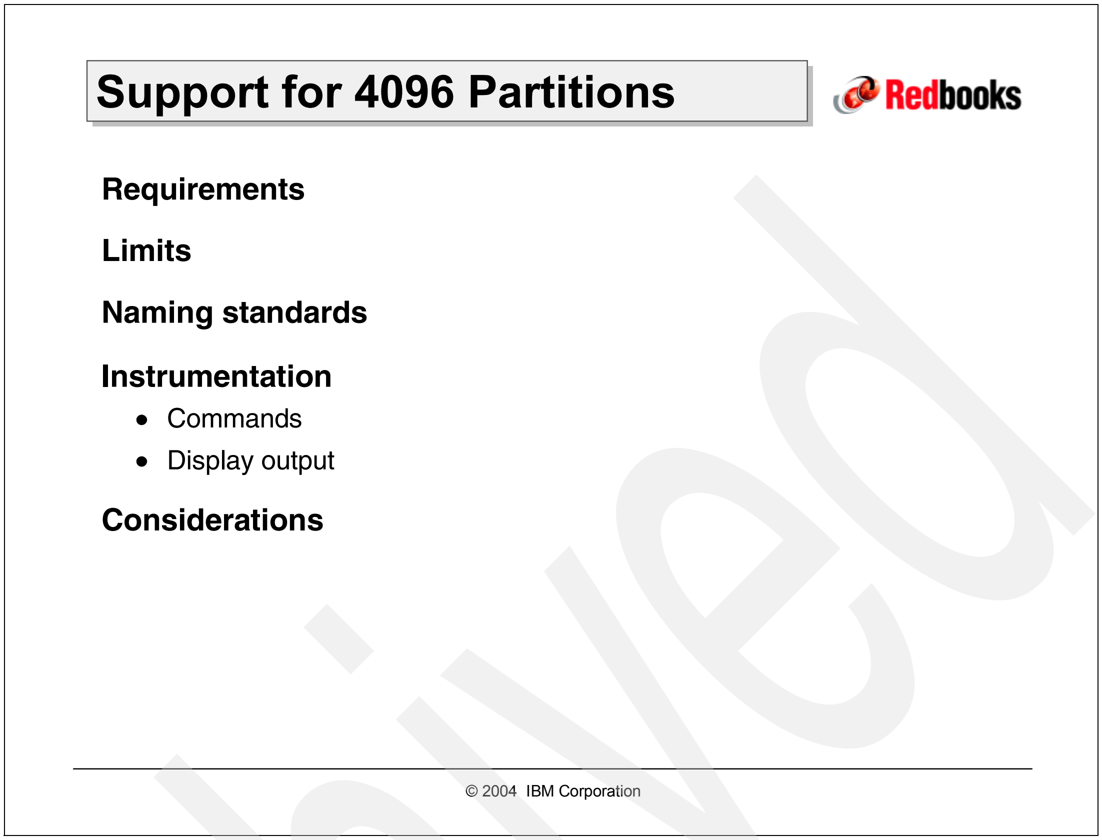


2.32 Summarizing 64-bit changes in DB2

In summary, the figure above shows that only the DBM1 address space and IRLM address space exploit the 64-bit virtual storage capabilities, as these are the two address spaces that suffer the most from virtual storage constraints in DB2 Version 7.

Other DB2 address spaces, such as the master address space (MSTR) and the distributed data facility (DDF) or DIST address space, remain in 31-bit.

User programs as well as stored procedures have to be in 31-bit mode. As mentioned before, this is not a problem. On the contrary, it means that no application changes are required to exploit DB2 V8 functionality.



2.33 Support for 4096 partitions

DB2 Version 8 allows you to create a partitioned table space with up to 4096 partitions, a drastic increase from today's 254.

In this section we explore the reasons for doing so, the impact on data set names, DB2 command syntax, and the output of those DB2 commands.

2.33.1 Requirements for 4096 partitions

Customers have a need for more partitions in their partitioned table spaces for certain types of applications. One example of these applications is the collection and retrieval of daily data, which, of course, means 365 partitions (or 366 for a leap year). If the customer wants to keep 11 years worth of daily data in separate daily partitions, that would be 4026 partitions. Another example is keeping weekly data in partitions; if a customer wants to keep 10 or 20 years worth of weekly data, that would be 520 or 1040 partitions.

Customers also want to have their large table spaces spread out over many partitions to reduce the size of their partition data sets. For a 16 terabyte table, the maximum data set size allowed for a 4 KB page size is 64 gigabytes and that may be too unwieldy to manage. More partitions allow us to reduce the data set size, by making the data more granular and manageable (for example, having daily partitions instead of weekly partitions.)

With DB2 V7, the maximum number of partitions in a partitioned table space and index space is 254. With DB2 V8 the new maximum number of partitions is 4096 for partitioned table spaces. The actual maximum number of partitions that you can specify is dependent on the page size and the DSSIZE (preferred) or LARGE (not recommended) parameter (see 2.33.3, “Table space size and number of partitions” on page 62 for details). 4096 partitions will help the applications mentioned above and eliminate the need for work-around solutions. You can define new partitioned table spaces with the new value of partitions, or you can use online schema changes (see 3.13, “Partition management” on page 166) to apply the new limits to existing partitioned table spaces.

The CREATE TABLESPACE statement now allows you to specify up to 4096 partitions in the NUMPARTS clause for a partitioned table space.

Maximum Number of Partitions

Maximum number of PARTITIONs depends on the DSSIZE and the page size in the CREATE TABLESPACE statement

As in DB2 V7, use DSSIZE when creating a table space for partitions of 4GB and larger

- LARGE clause is for compatibility of previous release to identify each partition of a partitioned table space that has a maximum size of 4 GB
- Use DSSIZE clause as a preferred method instead

DB2 creates default DSSIZE if LARGE or DSSIZE keyword is omitted and NUMPARTS > 254
(governed by the page size value)

Page Size	Default DSSIZE
4 KB	4 GB
8 KB	8 GB
16 KB	16 GB
32 KB	32 GB

2.33.2 Maximum number of partitions

The maximum number allowed in the NUMPARTS keyword is dependent on the page size and DSSIZE specified for the table space; see 2.33.3, “Table space size and number of partitions” on page 62 for details. If the DSSIZE (or LARGE) keywords are not specified, and NUMPARTS is greater than 254, a default data set size is given depending on the page size value; for 4 KB page size, DSSIZE default is 4 GB; for 8 KB page size, DSSIZE default is 8 GB; for 16 KB page size, DSSIZE default is 16 GB; and for 32 KB page size, DSSIZE default is 32 GB.

Another V8 enhancement is that you can add partitions to an existing partitioned table space. This way you do not have to create your partitioned table space with 4096 partitions from day one. You can define the object with the number of partitions that you need today, and add new ones when required. For more information, see 3.13.3, “Add a partition to a partitioned table space” on page 169.

When adding partitions to an existing table space, the maximum number of partitions allowed depends on how the table space was originally created. If DSSIZE was specified when the table space was created, it is non-zero in the catalog. The maximum number of partitions allowed is shown in Table 2-1.

Table 2-1 Maximum number of partitions allowed when DSSIZE >0

DSSIZE	Page size			
	4 KB	**8 KB**	**16 KB**	**32 KB**
1-4 GB	4096	4096	4096	4096
8 GB	2048	4096	4096	4096
16 GB	1024	2048	4096	4096
32 GB	512	1024	2048	4096
64 GB	256	512	1024	2048

If DSSIZE = 0, the maximum numbers of partitions allowed is shown in Table 2-2. If LARGE was specified when the table space was created, the maximum number of partitions is shown in the fourth row of Table 2-2. For more than 254 partitions when LARGE or DSSIZE is not specified, the maximum number of partitions is determined by the page size of the table space.

Table 2-2 Maximum number of partitions allowed when DSSIZE =0

Type of table space	Existing number of partitions	Maximum number of partitions
Non-large	1 - 16	16
Non-large	17 - 32	32
Non-large	33 - 64	64
Large	N/A	4094

Table Space Size and Number of Partitions

Redbooks

Type of Rid	Max # of partitions	Page Size	DSSIZE	Total TS size
5-byte(non-EA) large	4096	4KB	(4GB)	16TB
5-byte EA	4096	4KB	1GB	4TB
5-byte EA	4096	4KB	4GB	16TB
5-byte EA	2048	4KB	8GB	16TB
5-byte EA	1024	4KB	16GB	16TB
5-byte EA	512	4KB	32GB	16TB
5-byte EA	256	4KB	64GB	16TB
5-byte EA	4096	8KB	1GB	4TB
5-byte EA	4096	8KB	8GB	32TB
5-byte EA	2048	8KB	16GB	32TB
5-byte EA	4096	16KB	16GB	64TB
5-byte EA	2048	16KB	32GB	64TB
5-byte EA	4096	32KB	32GB	128TB
5-byte EA	2048	32KB	64GB	128TB

Total table space size depends on DSSIZE and number of partitions

2.33.3 Table space size and number of partitions

The figure above shows the maximum size of a partitioned table space in DB2 Version 8. Note that the table space size is dependent on the number of partitions, the page size, and the DSSIZE.

The maximum size of a DB2 table space in Version 8 is 128 TB. However, to be able to reach this maximum, you must use a 32K page size (either with 4096 partitions and 32 GB VSAM data sets (DSSIZE), or 2048 partitions and 64 GB VSAM data sets).

The reason for not being able to have a 128 TB table space with a 4K page size is that the number of pages you can address is limited by the 5-byte RID (4 byte page number, and 1 byte ID map entry).

If you want to use 4096 partitions, you need 12 bits to represents that number of partitions, which leaves you with 20 bits to address pages (4 bytes * 8 bits/byte - 12). The 20 bits allow up to 1048576 pages (1 MB).

With a 4KB page size and 4096 partitions, the result is:

```
4096 (4K page size) * 4094 (partitions) * 1048576 (pages) = 16 TB
```

This is the maximum you can address with a 4K page size in a single partitioned table space.

New Data Set Naming Convention

Existing data set naming convention only allows up to 999 partitions

VCAT.DSNDBC.DBNAME.TSNAME.I0001.A001 (A001 - A999)

Expand the convention by using other letters B, C, D, and E

- A001-A999 for partitions 1 through 999
- B000-B999 for partitions 1000 through 1999
- C000-C999 for partitions 2000 through 2999
- D000-D999 for partitions 3000 through 3999
- E000-E096 for partitions 4000 through 4096

2.33.4 New data set naming convention

Currently, DB2 names the DB2 data sets with the convention of 'Axxx' as the last qualifier, where xxx is the partition number. This naming convention allows for the definition of no more than 999 partitions. With DB2 V8, a new data set naming convention allows data sets with partition numbers greater than 999.

The naming scheme for a data set with more than 999 partitions is shown in Example 2-1.

Example 2-1 New data set naming convention

```
catname.DSNDBx.dbname.psname.p0001.lnnn
where
p is I or J
lnnn is A001-A999 for partitions 1 through 999
lnnn is B000-B999 for partitions 1000 through 1999
lnnn is C000-C999 for partitions 2000 through 2999
lnnn is D000-D999 for partitions 3000 through 3999
lnnn is E000-E096 for partitions 4000 through 4096
```

Note: If you use jobs that specify the full DB2 data set names, such as DSN1COPY, make sure that they take these new data set naming conventions into account when using more than 999 partitions.

Database Commands Support Enhancements

Redbooks

DISPLAY DATABASE command output changed to allow for 5 characters for logical partitions up to 4096 (for example, L4096)

PART keyword supports combination of partition list and range

- DISPLAY DATABASE
- START DATABASE
- STOP DATABASE

```
STOP DATABASE(DBONE) SPACE(TSONE)
          PART(3,5,6,9:12,15,19:4096)
```


2.33.5 Database commands support enhancement

The output under column PART in message DSNT397I is the partition number. It is blank for a simple table space or simple index space. For non-partitioned indexes on a partitioned table space, it is the logical partition number preceded by the character L, for example L4096.

For data-partitioned secondary indexes, the prefix for this value is the character D, for example, D0001.

As you can see from the example above, you can use a combination of partition list and ranges as argument of the PART keyword for DISPLAY, START, or STOP DATABASE DB2 command.

The STOP DATABASE command on the bottom of the visual above refers to parts 3, 5, 6, 9 to 12, 15, and 19 to 4096.

Display Database Command Support

Uses partition range output if partitions have same status and attributes. For example, 6 parts of TS486X are stopped. The display output shows 0001 through 0006 with STOP state.

Also reduces output size when 4094 partitions are involved

DISPLAY DATABASE(DB486B) SPACE(*)

```
NAME     TYPE PART  STATUS            PHYERRLO PHYERRHI CATALOG  PIECE
-------- ---- ----- ----------------- -------- -------- -------- -----
TS486X   TS    0001 STOP
    - THRU     0006
TS486X   TS    0007 UT,COPY
TS486X   TS    0008 STOP
TS486X   TS    0009 RW
    - THRU     0010
IX486X   IX   L0001 RW
IX486X   IX   L0002 LSTOP
    - THRU    L0003
IX486X   IX   L0004 RW
    - THRU    L0010
IX486Y   IX    0001 RW
    - THRU     0010
IX486Z   IX         RW
******* DISPLAY OF DATABASE DB486A   ENDED      **********************
```


2.33.6 Display database command support

The DISPLAY DATABASE command has been enhanced to support this large number of partitions. In order not to flood you with output (4096 data partitions, 4096 parts for each partitioned index, and 1 or more entries for each non-partitioned index), the DISPLAY command uses "partition range output". This means that it only shows the start and end partition, index name, when all the partitions in that range have the same status and attributes. For example in the visual above, 6 parts of TS486X are stopped. The display output shows 0001 through 0006 with STOP state, only two lines instead of six.

DISPLAY Command - *OVERVIEW Keyword*

Redbooks

OVERVIEW keyword - Displays objects and object type (for example, IX or TS and number of parts) with just one line of output. Does not show exception status.

DISPLAY DATABASE(DB486A) SPACE(*) OVERVIEW

```
DSNT360I . ***********************************
DSNT361I . *  DISPLAY DATABASE SUMMARY 483
           *    GLOBAL OVERVIEW
DSNT360I . ***********************************
DSNT362I .     DATABASE = DB486A  STATUS = RW 485
               DBD LENGTH = 4028
DSNT397I . 486
NAME     TYPE PARTS
-------- ---- -----
TS486A   TS    0004
IX486A   IX   L0004
IX486B   IX    0004
TS486C   TS
IX486C   IX
******* DISPLAY OF DATABASE DB486A   ENDED      ****
```


2.33.7 Database command support - OVERVIEW keyword

The OVERVIEW keyword of DISPLAY DATABASE displays each object in the database on its own line, providing an easy way to see all objects in the database.

OVERVIEW limits the display to only the space names and space types that exist in the specified databases. The number of parts is displayed for any partitioned table space. This keyword is very helpful in case of a large amount of partitions for one or more page sets in your database.

The OVERVIEW keyword cannot be specified with any other keywords except SPACENAM, LIMIT, and AFTER.

In the example shown on the visual above, you can easily see that table space TS486A consists of four partitions. Page set IX486B is a partitioned index, whereas IX486A is a secondary index with four logical partitions.

Table space TS486C is a simple or segmented table space.

Some Considerations

You can alter past 254 partitions with ALTER TABLE and ADD PARTITION clause, up to the maximum allowed based on table space parameters

More partitions =>

- More open data sets
- More compression dictionaries
- More virtual and real storage
- More space in catalog and directory
- Larger DBDs

Maximum number of parallel tasks for I/O queries

- Remains at 254 even with more partitions

A 4096 partitioned table space can have a maximum of 5 LOB columns

2.33.8 Some considerations

The size of catalog objects SYSDBASE, SYSCOPY, SYSSTATS, SYSTABLEPART_HIST, SYSINDEXPART_HIST, SYSTABSTATS_HIST, and SYSINDEXSTATS_HIST is greatly increased, as are directory objects DBD01, SYSUTILX, and SYSLGRNX if a large number of partitions are created or added to table spaces. These tables should be sized correctly so that there is no need to resize them too often in the future.

A good solution for customers who want daily or weekly granular segments is to start with a minimum number of partitions when creating the table space. The customer can then add more partitions using the ALTER ADD PARTITION statement provided with Online Schema Evolution (see 3.13, “Partition management” on page 166.)

For example, if a customer wants daily data segments, the table space can be created with 365 partitions and then partitions can be added later for subsequent years.

Customers that have existing partitioned table spaces with 254 partitions can use online schema evolution to add more partitions to their table spaces.

Be aware that with LOBs, there is one LOB table space, one auxiliary table, and one auxiliary index per partition per LOB column. A single database can hold a maximum of 65,535 objects. Therefore, if a table with 4096 partitions has a LOB column, there is a need to create 12,288 objects (4096 LOB table spaces, 4096 auxiliary tables, 4096 auxiliary indexes), so only 5 LOB columns could be defined on a 4096 partition table space.

More Active Log Data Sets

Redbooks

Maximum size of a single DB2 active log data set is 4 GB

- V8 increases the number of active log data sets from 31 to 93 per LOGCOPY

Requires BSDS conversion

- Must be in NFM
- Convert BSDS with new DSNJCNVB job

```
  DSNTIPL INSTALL DB2 -ACTIVE LOG DATA SET PARAMETERS
===>_
Enter data below:
1 NUMBER OF LOGS    ===>3 Number data sets per active log copy (2-93)
2 OUTPUT BUFFER     ===>4000K Size in bytes (40K-400000K)
3 ARCHIVE LOG FREQ ===>24 Hours per archive run
.......
```


2.34 More active log data sets

Even with the maximum size of each active log data set size of 4 GB minus 1 CI (this increase was made available via the PTF for APAR PQ48126 to DB2 V6 and V7, and is now in the base code of V8), large DB2 systems need more log data available in the active log data sets, because it reduces the chances of DB2 requiring archive log data sets for an extended rollback or for media recovery. Since active log read is generally faster than archive log read, queueing for archive log tape volumes would be virtually eliminated.

DB2 V8 increases the maximum number of active log data sets from 31 to 93 per log copy.

Increasing the maximum number of active log data sets (and archive log volumes, discussed in the next topic) requires a conversion of the BSDS to allow more data set entries. To do the conversion, the user runs a new BSDS conversion utility job DSNJCNVB. In order to minimize the fallback and data sharing co-existence impact of this change, your current DB2 system must be in V8 new-function node (NFM) before you can convert your BSDS to support the new maximum values. BSDS conversion is optional in DB2 V8, but recommended.

DB2 install job DSNTIJIN automatically provides a larger BSDS definition (space allocation) during a new installation; however, you must still convert the BSDS by running the new conversion utility job DSNJCNVB once DB2 operates in NFM. When migrating to V8, you should follow the documented pre-conversion procedure to manually redefine a larger BSDS before converting it with DSNJCNVB.

Increased Max Archive Log Data Sets

Redbooks

```
DSNTIPA INSTALL DB2 -ARCHIVE LOG DATA SET PARAMETERS
===>_
Enter data below:
1 ALLOCATION UNITS ===>      BLK Blk,Trk,or Cyl
2 PRIMARY QUANTITY ===>      Primary space allocation
3 SECONDARY QTY.   ===>      Secondary space allocation
4 CATALOG DATA     ===>NO    YES or NO to catalog archive data sets
5 DEVICE TYPE 1    ===>TAPE  Unit name for COPY1 archive logs
6 DEVICE TYPE 2    ===>      Unit name for COPY2 archive logs
7 BLOCK SIZE       ===>28672 Rounded up to 4096 multiple
8 READ TAPE UNITS  ===>2     Maximum allocated read tape units
9 DEALLOC PERIOD   ===>0     Time interval to deallocate tape units
10 RECORDING MAX   ===>1000  Number of data sets recorded in BSDS
...
```

Specifies the number of archive log volumes recorded in the BSDS

- The upper limit is raised from 1000 to 10000 per LOGCOPY
- Run the new job DSNJCNVB to convert your BSDS(s) to the larger size

2.35 Increased maximum number of archive log data sets

With the explosion of e-business and the extremely high transaction volumes that large customers are processing today, customers are finding that the current maximum of 1,000 archive log volumes (per log copy) recorded in the BSDS is no longer sufficient to remain recoverable without having to take frequent image copies. Even with the maximum size of each log data set, active or archive, now increased to 4 GB minus 1 CI, large DB2 systems are creating so many archive log data sets that the 1,000 archive log data set maximum only allows them to register a few days of log data in the BSDS.

DB2 V8 increases the maximum number of archive log data sets recorded in the BSDS from 1,000 data sets per log copy to 10,000 data sets.

Prior to running the conversion utility, you need to do the following steps to allocate a larger BSDS:

1. Rename existing BSDS data sets to save the original in case the conversion fails.
2. Allocate larger BSDS using the original BSDS name. You can use the VSAM DEFINE statements in installation job DSNTIJIN for this task.
3. Copy the original data set to a new, larger data set; VSAM REPRO is recommended.
4. Repeat for the second copy dual BSDSs.

See also 2.34, “More active log data sets” on page 68 and 2.35, “Increased maximum number of archive log data sets” on page 69.

Complex Joins - up to 225 Tables

Redbooks

What is it? .....

- Maximum number of tables that can be joined in the FROM clause in a query increases from 15 in Version 7 to 225 in Version 8

Benefits .....

- Enhances usability and power of SQL
- Important to ERP and CRM applications
  - ERP and CRM applications are typically designed with hundreds of tables thus requiring large joins

2.36 Complex joins - up to 225 tables

Up until V7 you can only have 15 tables in the FROM clause of your SQL statements (that are non-star join queries). This restriction has been in place since DB2 V1. In V8 this limit is increased to 225.

This enhancement increases the usability and power of SQL. Lifting this restriction is very important to ERP and CRM applications. These applications typically use a highly normalized design with hundreds of tables. Therefore, even fairly simple queries require a lot of tables that need to be joined, easily more than 15, the current restriction. With V8, you can now specify up to 225 tables in the FROM clause of your SQL statements.

Complex Joins - up to 225 Tables - 2

Before DB2 Version 8, the number of tables that can be joined in the FROM clause is limited to 15:

- The limit is set to prevent the possibility that a large query causes the DB2 Optimizer to consume huge amounts of resources (storage, CPU, ET) resulting in critical storage shortage and possible system crash
- APS uses dynamic programming. The number of join combinations, when determining the best access path, grows exponentially
- Can use 'hidden' ZPARM (SPRMMXT) to increase the limit (at your own risk)
- Limit does not apply to star join queries

With DB2 Version 8:

- Optimization enhancements enable support for joins of up to 225 tables
- Code added to recognize common query patterns (that is, Star Schema), which enables more efficient optimization of very large joins
- Queries that do not fit the Star Schema pattern, but join a large number of tables, could still run into problems
  - DB2 has put in place thresholds to force the optimization process to complete quickly

2.36.1 Complex joins - up to 225 tables - 2

The number of tables that can be specified in the FROM clause of an SQL statement is limited to 15, for DB2 versions prior to V8. Many customers need to run queries that join more than 15 tables in their ERP or CRM applications, typically designed with large numbers of tables to be joined. When you exceed this limit, you receive an SQLCODE -129.

Queries that qualify for star join processing already allow up to 225 tables in the FROM clause. To get around the 15 table limit, customers can evaluate the use of a “hidden” DSNZPARM (SPRMMXT - MXTBJOIN) so that their queries can run. This parameter is hidden because, in general, there is a need for extra storage and processor time when dealing with these complex queries.

The default limit on the number of tables joined has stayed at 15 for a long time because a large query could cause DB2 to consume a lot of resources (storage and CPU) when evaluating the cost of each possible join sequence. This in turn can cause critical storage shortages and have a negative impact on the DB2 subsystem. See APARs PQ31326, PQ28813, and PQ57516 for more details.

Note that this limitation applies to the number of tables in the FROM clause. In V7, you can already use up to 225 tables throughout the entire set of SQL statements (for example, including subselects).

In DB2 V8, the default limit is changed from 15 to 225 tables to be joined. This means that users can more easily join more than 15 tables. It also means that DB2 can join this many tables without restriction.

A number of enhancements have been implemented in DB2 V8 to reduce the amount of resources needed for the optimization process. This allows you to join more tables using less resources. A new functionality can recognize common query patterns (like star schema) and optimize large joins very efficiently.

These improvements, while reducing the risk of running into resource shortages, do not by themselves eliminate the risk. Queries that do not fit the star schema pattern, but join a large number of tables, could still run into problems, even in DB2 V8.

Considerations for Optimization Thresholds

How does it work?

- When the optimization process gets within a percentage of any threshold (that is, 80%), curbs are activated to speed up optimization
- If the threshold value is exceeded, curbs force optimization to complete very quickly

Thresholds expressed in terms of storage (number of MB), CPU (number of seconds), and ET (elapsed time)

- To avoid regressing existing queries, thresholds only applied if more than 15 tables joined
- 'Hidden' zparm provided to allow thresholds to apply to queries with 15 or less tables (TABLES_JOINED_THRESHOLD)

2.36.2 Considerations for optimization thresholds

To address this problem, DB2 V8 has enhanced the monitoring of how much storage and CPU is being consumed by the optimization process. If it exceeds certain thresholds, then curbs are put in place to force the optimization process to complete quickly. When excessive resources have been consumed by the optimization process, the goal changes — from selecting the “optimal” plan, to selecting a “reasonable” plan, in a minimal amount of time.

The resource threshold used is expressed in terms of storage (number of megabytes), CPU (number of seconds), and elapsed time (also in number of seconds). The thresholds are large enough so that most existing queries are not impacted, but small enough so that they prevent severe resource shortages.

To guard against regressing existing queries, the threshold is only applied when the number of tables joined is greater than 15 (the limit prior to DB2 V8). This way, only customers that were using the “hidden” ZPARM to run queries with >15 tables may see any change to their existing workload.

Affected Interfaces

New 'hidden' keyword zparms -- all changeable online

- MAX_OPT_STOR
  - Maximum amount of storage to be consumed by Optimizer
  - Default is 20MB
  - Values range from 0-100MB (if 0, then default is used)
- MAX_OPT_CPU
  - Maximum amount of CPU time to be consumed by Optimizer
  - Default is 100 sec
  - Values range from 0 to 500 sec (if 0, then default is used)
- MAX_OPT_ELAP
  - Maximum amount of elapsed time to be consumed by Optimizer
  - Default is 100 sec
  - Values range from 0 to 1000 sec (if 0, then default is used)
- TABLES_JOINED_THRESHOLD
  - Number of tables joined to cause DB2 to limit amount of resources consumed by Optimizer
  - Default is 16
  - Values range from 0 to 225
(if 0, threshold never applied - if 1, threshold always applies)\

2.36.3 Affected interfaces

These parameters change the behavior of the optimizer when it is evaluating SQL statements with more than 15 tables (TABLES_JOINED_THRESHOLD DSNZPARM) in the SQL statement.

They can be overridden by coding these values in the DSN6SPRM macro specification and re-assembling and re-linking DSNZPARM. Please do not change these default values unless directed to do so by IBM.

Longer SQL Statements — Redbooks

What is it? .....

- Limit on size of embedded SQL statement extended to 2 MB (Was 32 KB in previous versions)
- Dynamic SQL statements passed to DB2 as
  - CLOBs and
  - DBCLOBs

Benefits .....

- DB2 Family compatibility
- Required to support long names, 4096 partitions and UNICODE
- Important for SQL Procedure Language applications and SQL statements created by generators
  - SQL Procedure must be completely stated in a single SQL statement -- limit prior to Version 8 was 32K

2.37 Longer SQL statements

Ever since DB2 V1, an SQL statement has been limited to 32 KB. This is normally not a problem, but with the support of longer names for most DB2 objects, 4096 partitions, and especially SQL procedures, the maximum of 32 KB can become a problem. In DB2 Version 8, this restriction is lifted, as is the case with so many other V7 limitations. In DB2 Version 8, your SQL statements can be up to 2 MB.

When using embedded static SQL, this means that you can now code a 2 MB statement between your EXEC SQL and END-EXEC, in your COBOL program, for example.

On the other hand, when using dynamic SQL, these long statements are passed to DB2 as a CLOB or DBCLOB, because a “normal” character string can only be up to 32 KB.

Dynamic SQL Statements

SQL statements passed to DB2 on PREPARE/EXECUTE IMMEDIATE statements now passed in CLOBs and DBCLOBs

Changes to PREPARE/EXECUTE IMMEDIATE

- Host variable can now be specified as CLOB and DBCLOB
- Maximum length of SQL statement contained in CLOB or DBCLOB
  - 2,097,152 bytes for CLOB
  - 1,048,574 double-byte characters for DBCLOB
- Maximum length of contained statement If host variable VARCHAR and VARGRAPHIC
  - 32,767 bytes for VARCHAR
  - 16,383 double-byte characters for VARGRAPHIC

NOTE: Since SQL statements are broken up into pieces and stored in succeeding records with a sequence number to keep them in order, there is no change in the catalog to store a 2 MB statement

2.37.1 Using 2 MB statements with dynamic SQL

As mentioned in the previous section, SQL statements passed to DB2 via PREPARE/EXECUTE IMMEDIATE statements have to be passed in CLOBs and DBCLOBs, when you want to be able to use statements that are longer than 32 KB.

The maximum length of an SQL statement when the host variable is a VARCHAR is 32,767 bytes, and 16,383 double-byte characters for a VARGRAPHIC.

The interface to PREPARE/EXECUTE IMMEDIATE has been enhanced in V8. With this enhancement, PREPARE/EXECUTE IMMEDIATE will accept host variables that are specified as CLOB and DBCLOB.

The maximum length of an SQL statement contained in a CLOB is 2,097,152 bytes, and 1,048,574 double-byte characters for a DBCLOB.

When the SQL statements are stored in the DB2 catalog for static SQL, they are broken into pieces, and stored in succeeding records (with a sequence number, to allow you to retrieve them in the correct order). Therefore, no catalog changes are required to store SQL statements up to 2 MB in the DB2 catalog.

EXECUTE IMMEDIATE Example

Example illustrates in C how to use a CLOB host variable in the EXECUTE IMMEDIATE statement

```
main() {

EXEC SQL BEGIN DECLARE SECTION;
         .........

   SQL TYPE IS CLOB(100K) string1;
         .........

EXEC SQL END DECLARE SECTION;
         .........

strcpy(string1.data, "UPDATE DSN8610.EMP SET SALARY = SALARY *
1.1");
string1.length = 44;
         .........

EXEC SQL EXECUTE IMMEDIATE :string1;
         .........
```


Example of EXECUTE IMMEDIATE using a CLOB

The example in the figure above shows how to execute a “long” UPDATE statement. In the example, the statement does not really need a CLOB, as it is only 44 bytes, but it is just given to show you how to use a CLOB. The CLOB in our example can be up to 100K (the maximum being 2M). Therefore the statement string :string1 that we pass to the EXECUTE IMMEDIATE can be up to 100 KB long.

Note that the EXECUTE IMMEDIATE is the same as in previous versions of DB2. The only difference is that it also accepts a CLOB or BLOB in V8.

PREPARE Example

Redbooks

Example illustrates in COBOL how to use a DBCLOB host variable in the PREPARE statement

```
IDENTIFICATION DIVISION.
DATA DIVISION.
      ...............
WORKING-STORAGE SECTION.
01 USTRING SQL TYPE IS DBCLOB(400K).
01 ATTRSTG.
    49  ATTR-LEN    PIC S9(4) USAGE BINARY.
    49  ATTR-DATA PIC X(1000).
      .................
PROCEDURE DIVISION.
      .................
MOVE " long sql statement in UNICODE(UTF-16) " TO USTRING-DATA.
MOVE <length of sql statement> TO USTRING-LENGTH.
MOVE "INSENSITIVE SCROLL WITH CS " TO ATTR-DATA.
MOVE 27 TO ATTR-LEN.
      ...................
EXEC SQL EXECUTE PREPARE ATTRIBUTES :ATTRSTG FROM :USTRING.
```

NOTE: CLOB OR DBCLOB cannot be used for ATTRIBUTES host-variable

Example of PREPARE using a DBCLOB

When preparing SQL statements that are bigger than 32 KB, you must use CLOBs or DBCLOBs. The visual above shows how to PREPARE an SQL statement using a DBCLOB in a COBOL program.

Note that the precompiler generates a structure containing two elements, four byte length field, and data field of specified length, for each CLOB or DBCLOB variable declared. The names of these fields vary depending on the host language used:

- For COBOL, they are variable-LENGTH and variable-DATA.
- For C, they are variable.length and variable.data.
- For PL/I, Assembler and FORTRAN, they are variable_LENGTH and variable_DATA.

See the “Declaring LOB host variables and LOB locators” section in *DB2 Application Programming and SQL Guide,* SC18-7415 for more details. An example of a COBOL DBCLOB declaration is shown in Example 2-2.

Example 2-2 Sample DBCLOB declaration

```
01 USTRING  SQL TYPE IS DBCLOB(400K).

is translated into the following by the DB2 precompiler:

01  USTRING.
 02  USTRING-LENGTH                   PIC S9(9) COMP.
 02  USTRING-DATA.
      49 FILLER   PIC G(32767) USAGE DISPLAY-1.
      49 FILLER   PIC G(32767) USAGE DISPLAY-1.
      49 FILLER   PIC G(32767) USAGE DISPLAY-1.
      49 FILLER   PIC G(32767) USAGE DISPLAY-1.
      49 FILLER   PIC G(32767) USAGE DISPLAY-1.
      49 FILLER   PIC G(32767) USAGE DISPLAY-1.
      49 FILLER   PIC G(32767) USAGE DISPLAY-1.
      49 FILLER   PIC G(32767) USAGE DISPLAY-1.
      49 FILLER   PIC G(32767) USAGE DISPLAY-1.
      49 FILLER   PIC G(32767) USAGE DISPLAY-1.
      49 FILLER   PIC G(32767) USAGE DISPLAY-1.
      49 FILLER   PIC G(32767) USAGE DISPLAY-1.
      49 FILLER   PIC G(16396) USAGE DISPLAY-1.
```

Because the COBOL language allows graphic declarations of no more than 32767 double-byte characters, for DBCLOB host variables that are greater than 32767 double-byte characters in length, DB2 creates multiple host language declarations of 32767 or fewer double-byte characters.

Also note that you cannot use a CLOB or DBCLOB for the attributes string. The attribute string definitions remain the same as in V7.

Other Considerations

Redbooks

Distributed data

- Remote support for large SQL statements requires requester and server support for DRDA V3
  - DB2 and DB2 Connect V8 adding support for this protocol and can flow large SQL statements
  - DB2 for iSeries V5R2 does not provide support for DRDA V3 but does support long SQL statements

Trace records that contain entire or partial SQL statement

- IFCID 0063, 0140, 0141, 0142, 0145, 0168, 0316
  - Contain complete or partial SQL statement text
- New IFCID 350 provides the full SQL statement text
  - Use IFCID 317 for statements in the statement cache

2.37.2 Long SQL statement considerations

Here we describe some considerations applying to long SQL statements.

Distributed data

Remote support for large SQL statements requires DRDA requester and server support for The Open Group DRDA V3 Technical Standard. DB2 for z/OS and DB2 Connect V8 added support for this protocol and can flow large SQL statements. DB2 for iSeries™ V5R2 does not provide support for DRDA V3, but does support long SQL statements. DB2 for Linux, UNIX, and Windows are adding support for 2 MB SQL statements in their upcoming release (called Stinger). For more information on Stinger, see:

http://www.ibm.com/software/data/db2/stinger/

Trace records that contain entire or partial SQL statement

As before, IFCID 0063, 0140, 0141, 0142, 0145, 0168, and 0316 can contain all or part of the SQL statement text, depending on the length of the statement.

In case you want to see the full SQL statement, you have two options:

- If the statement is in the dynamic statement cache, you can use IFCID 317 to retrieve the full SQL statement through the READS interface of the IFI.
- If the statement is not in the dynamic statement cache, you can use the new IFCID 350. Unlike IFCID 63, it contains the full SQL statement text.

Long Predicates

Column length for predicate operands is extended to 32704 bytes from 255 bytes (254 for graphic strings)

- DB2 Family compatibility

Change:

- Increase the maximum length for predicates to 32704 bytes
- Matches the maximum defined size of a VARCHAR/VARGRAPHIC column

Max length for LIKE pattern expression remains 4000 bytes

The DB2 maximum sort key size has been increased to 16000 bytes (from 4000 in V7)

2.38 Long predicates

Prior to V8, the maximum length for predicate operands is 255 bytes and for graphic strings 254 bytes. This is incompatible with the rest of the DB2 family. In DB2 V8 the maximum length for predicates is increased to 32704 bytes, matching the maximum defined size of a VARCHAR column.

This support requires no SQL changes. Predicates are supported for both indexable and non-indexable columns. The maximum length for the pattern expression for LIKE predicates remains 4000 bytes.

The maximum sort key size has also been increased in DB2 V8 to 16000 bytes. The limit was 4000 bytes in previous DB2 versions.

Long Index Keys

Requirement for key lengths longer than 255 bytes

- Index keys may be longer if existing data is converted to UNICODE
- Compatibility with the DB2 Family
  - DB2 UDB for Linux, UNIX, and Windows platforms already have long key support (1024 bytes)

Change:

- Increase the maximum key size from 255 bytes to 2000 bytes

Not affected by this change:

- Partitioning key still limited to 255 bytes
- No SQL changes

2.39 Long index keys

Prior to V8, the maximum key length is 255 bytes. This sometimes presents us with some design challenges. With the implementation of Unicode support, multiple bytes may be required to represent a single character. Existing data that is converted to Unicode or new data in Unicode can result in index keys longer than 255 bytes.

Since long index keys are supported by DB2 for UNIX, Linux, and Windows, applications using keys longer than 255 bytes cannot easily be ported to the z/OS platform.

In DB2 V8 the maximum key length is extended from 255 bytes to 2000 bytes.

This support requires no SQL change. The increased key limit of 2000 bytes is only available in new-function mode. The partitioning key limit of 255 bytes does not change with this enhancement.

3

Availability

List of Topics

Redbooks

Partitioning and clustering enhancements

Online schema changes

System level point-in-time recovery

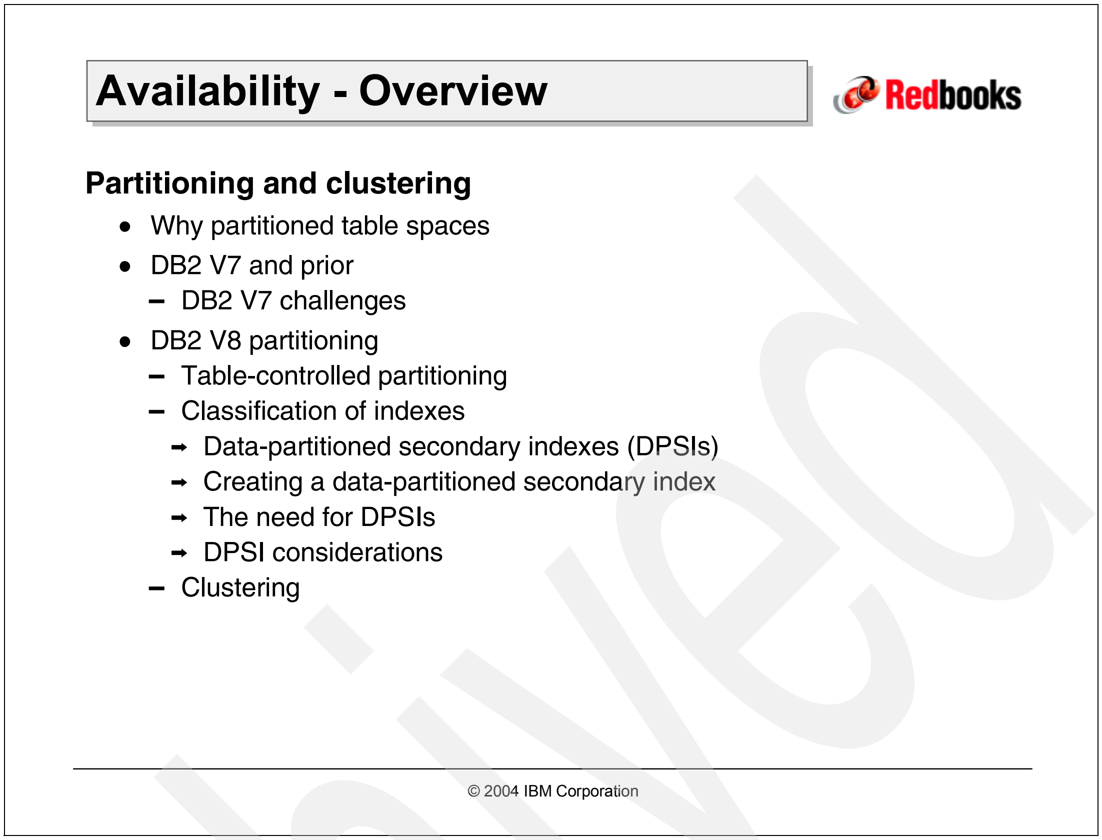
More online changeable DSNZPARMs

Other availability enhancements

DB2 for z/OS Version 8 brings a lot of changes that affect availability, keeping up with the explosive demands of e-business, transaction processing, and business intelligence. DB2 V8 delivers increased application availability with schema evolution support, which permits schema changes without stopping data access while the changes are implemented. You can gain greater availability and management through data partitioned secondary indexes, and minimize partition management for historical data with support for rolling partitions and adding partitions.

Version 8 also introduces a new technique to back up and recover an entire DB2 subsystem or data sharing group.

DB2 Version 7 allows you to change a number on DSNZPARMs online. In V8, more DSNZPARMs are online changeable.



3.1 Availability - overview

In the first part of this chapter we look at the many enhancements to partitioning and clustering.

First we review why and where you want to use partitioned table spaces, and discuss the challenges in partitioning in DB2 V7 and earlier versions.

Then we introduce the first partitioning enhancement in V8, table-controlled partitioning.

With the changes in partitioning, it is also important that we do a better job of using correct terminology when talking about indexes on partitioned table spaces. The new index terminology is introduced in that section.

We also discuss a brand-new type of index in DB2, the data-partitioned secondary index, how to define it, the problems it solves, and some design considerations.

Lastly, we describe the clustering enhancements in V8.

Availability - 2

Redbooks

Online schema changes

- Online schema changes overview
- Table data type changes
- Index changes
- Versioning
- Dynamic partitions (partition management)
- Other schema changes
- New DBET states for online schema changes

3.2 Availability - 2

The second part of this chapter is dedicated to the enhancements in DB2 V8 that allow you to make schema changes without having to drop and recreate objects, so-called *online schema evolution*.

First we discuss the type of online table and index changes that are supported in Version 8. We also discuss the underlying infrastructure that allows this to work, called versioning, and the impact of online changes on the status of your objects (DBET states).

Then we describe the online changes you can make to partitioned table spaces; adding partitions on the fly, and rotating partitions.

Availability - 3

Redbooks

System level point-in-time recovery

- Backing up the system
- Restoring the system

Online ZPARMs

Other availability enhancements

- Control interval larger than 4KB
- Monitoring system checkpoints and log offload activity
- Log monitor long running UR backout
- Detect long readers (IFCID 313)
- Locking enhancements
- Improved LPL recovery
- SMART DB2 extent sizes for DB2 managed object
- Logging manageability enhancements

3.3 Availability - 3

The last parts of this chapter are dedicated to these topics:

- System level point-in-time recovery is a new feature that allows you to take a non-disruptive backup of an entire DB2 subsystem or data sharing group, as well as restoring that entire subsystem or data sharing group to a previous point-in-time.
- DB2 V7 introduced online changeable DSNZPARMs. V8 continues to work its way down the list to make more and more DSNZPARMs online changeable.
- In the final sections we describe a number of miscellaneous enhancements that can have a positive effect on availability, such as:
  - VSAM control intervals larger than 4 KB
  - Monitoring system checkpoints and logging offload activity
  - Log monitoring long running UR backout
  - Detecting long readers (IFCID 313)
  - Locking enhancements
  - Improved LPL recovery
  - SMART DB2 extent sizes for DB2 managed object
  - Logging manageability enhancements

Why Partitioned Table Spaces

Size

- Number of rows
- Size of data sets

Size -- history

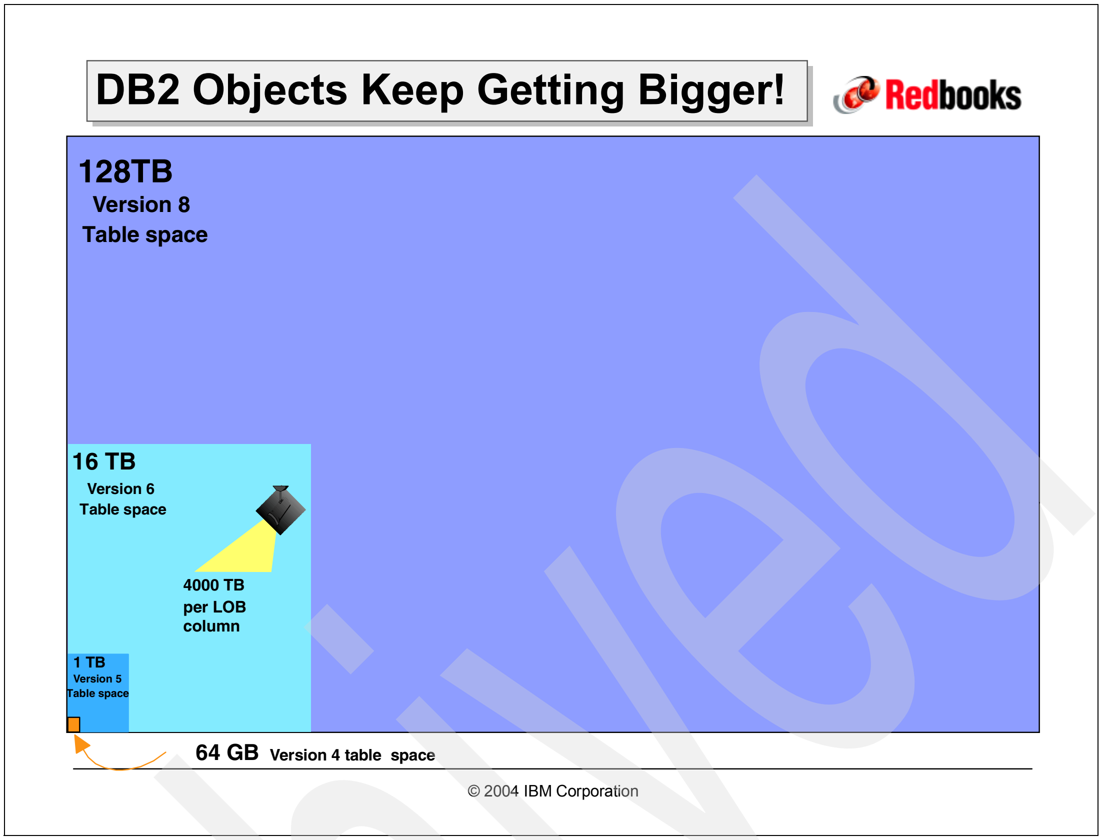
Performance and availability -- reduced elapsed time for:

- Utilities
  - COPY / RECOVER
  - REORG
  - LOAD
- Batch
- Query / BI / Data Warehouse
- Manageability - handle a partition at a time

3.4 Why partitioned table spaces

Partitioned table spaces are usually recommended for storing tables of large size. Two of the reasons for the recommendation deal with availability issues:

- Positive recovery characteristics:
  - If the data set backing a physical partition becomes damaged, the data outage is limited to that partition's data, and only that fraction of the data needs to be recovered.
  - Furthermore, if partitioning is performed along application-meaningful lines, logical damage (by a wayward application, for example) can be isolated to certain partitions — again limiting the data outage and recovery scope.
- The potential to divide and conquer:
  - The elapsed time to perform certain utilities, or the storage requirement to perform online REORG against a large table space, may be prohibitively high. Because utility jobs can be run at the partition level, operations on a table space can be broken along partition boundaries into jobs of more manageable size.
  - The jobs may be run in parallel to accomplish the task in reduced elapsed time, or serially to limit resource consumed by the task at any one point-in-time.



3.4.1 VLDB - DB2 objects keep getting bigger!

This foil is to scale! As we can see, the size of a table space has dramatically increased over the past several versions of DB2. *From DB2 V4 to DB2 V8, the size of a table space has increased 2048 times!* This rapid growth has made it even more desirable to be able to partition a table space to make the object more manageable and to make utility work against such a large object more granular. Note that to get to a 128 TB table space, you must use a page size of 32 KB.

V7 Partitioned Tables

Creating

Index-controlled partitioning

Logical and physical partitions

NPI challenges

- Recovery of NPI is always entire NPI
- NPI contention
  - REORG PART BUILD2 phase
  - LOAD PART
- System affinity in a data sharing environment

3.5 V7 Partitioned tables

In this and the following sections we review how to use partitioned tables in DB2 V7 and before. We have a quick refresh of:

- How to create a partitioned table, using so-called index-controlled partitioning
- The differences between logical and physical partitions
- The challenges of V7's non-partitioning indexes, such as non-partitioned index (NPI) contention

Version 7 and Prior

```
CREATE TABLESPACE  tsname  NUMPARTS  n;

CREATE TABLE CUSTOMER (
        ACCOUNT_NUM        INTEGER,
        CUST_LAST_NM       CHAR(30),
        . . .
        LAST_ACTIVITY_DT      DATE,
        STATE_CD              CHAR(2)  )
        IN ...
```

The table space definition is incomplete at this point!

3.5.1 Creating a partitioned table in Version 7 and prior

In DB2 V7 and prior, in order to create a partitioned table, you created a table space specifying the `NUMPARTS` keyword and created the table to be placed in this table space. The definition of this table space is incomplete at this point and the table is marked as unavailable until the required partitioning index is defined.

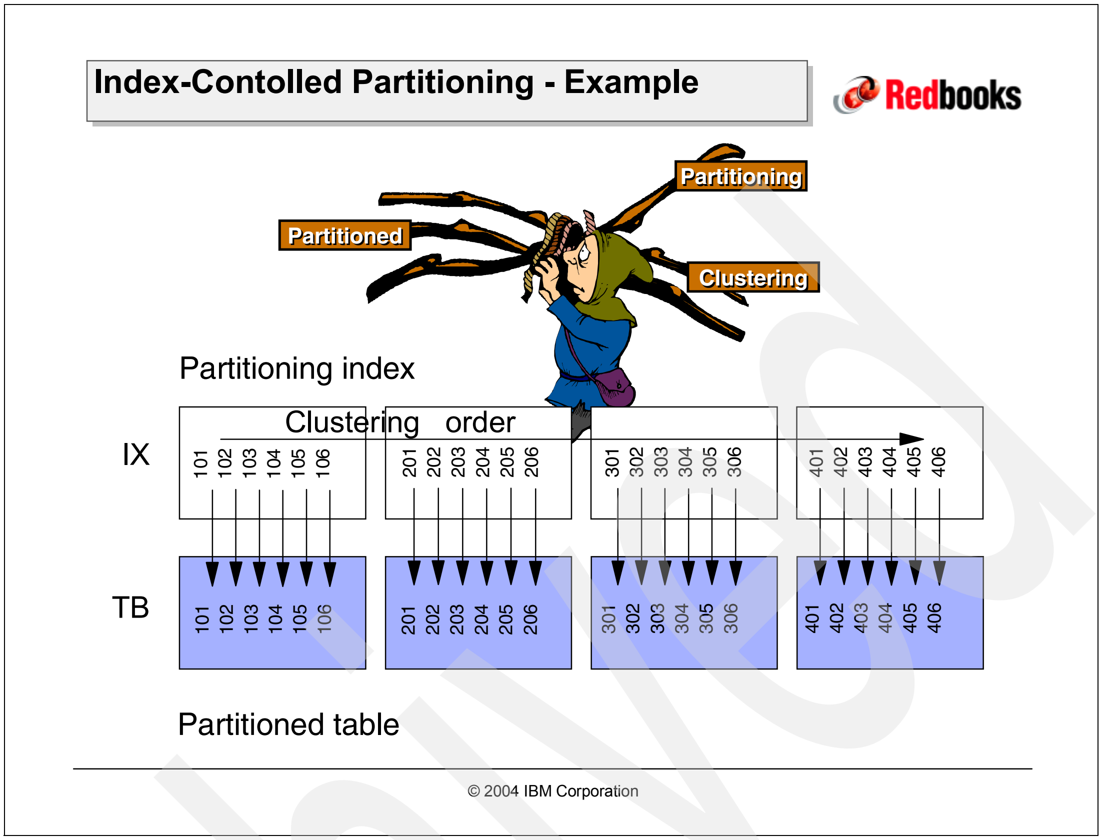
Creating the Partitioning Index

```
CREATE INDEX  index-name  ON CUSTOMER
     (ACCOUNT_NUM   ASC)  . . .
     CLUSTER                        <= required!!
     (
     PART 1  VALUES ( 199 )        <= presence of one or
     PART 2  VALUES ( 299 )           more PART n VALUES
     . . .                            clauses indicates a
     PART 4  VALUES ( 499 )           partitioning index
     )
     . . .
```

The partitioning index is required to complete the definition of the partitioned table space!!

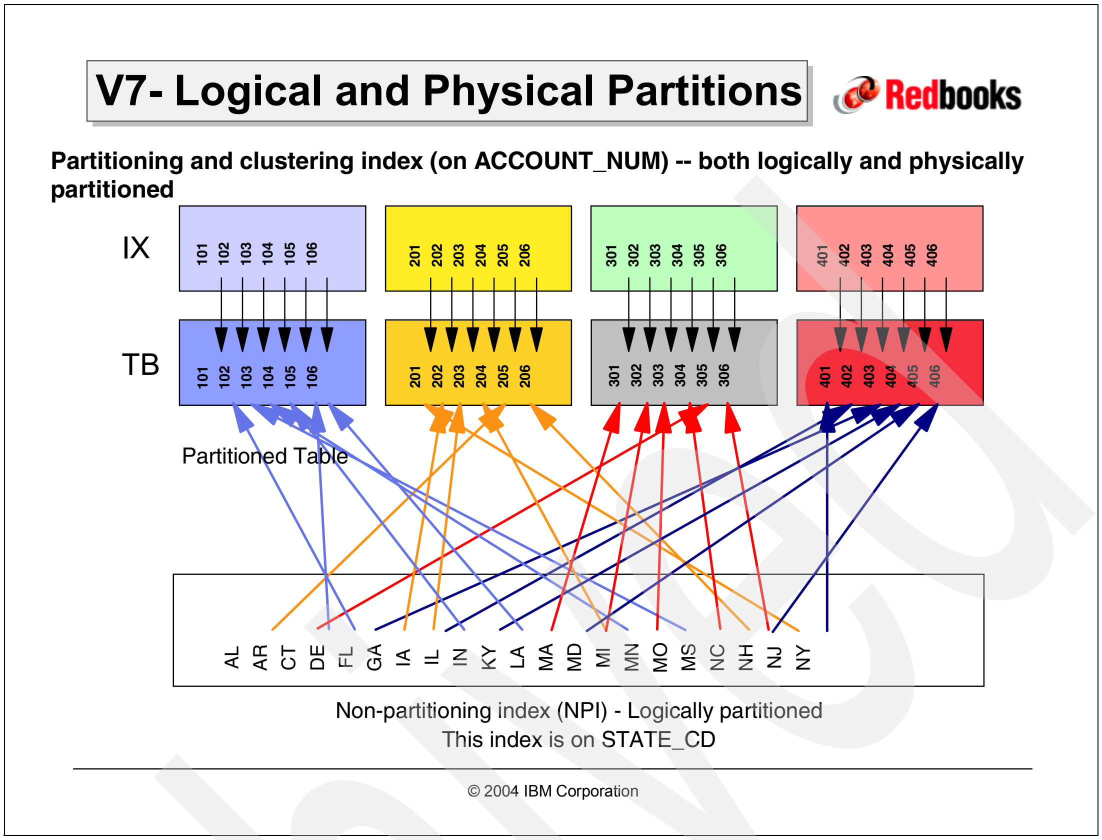
3.5.2 Creating the partitioning index

In V7, it is required that you define a partitioning index for a partitioned table space in order to complete the definition of the table space. The partitioning index specifies a partitioning key that dictates what columns the table is partitioned by, and one `PART` clause per partition to specify which rows (key ranges) go into which partition. The partitioning index also has to be the clustering index, and the `CLUSTER` keyword is required or the statement fails. That is, the partitioning index controls how the table is partitioned and how the table is clustered. This table and index combination is now called *index-controlled partitioning*.



3.5.3 V7 Partitioned table space - index-controlled partitioning

In V7 and prior versions, you only have index-controlled partitioning at your disposal to create a partitioned table. When using index-controlled partitioning, concepts such as “partitioned”, “partitioning” and “clustering” are intertwined because the index that defines the columns and key ranges for the different partitions is the partitioning index, is partitioned (made up of different physical partitions), and is also the clustering index.



3.5.4 Logical and physical partitions

Prior to DB2 V8, any index that does not specify the `PART n VALUES` keywords, and is defined on a partitioned table, is a non-partitioned index (NPI). Prior to DB2 V8, NPIs cannot be physically partitioned, that is, they cannot have multiple physical partitions. They can only be allocated across multiple pieces to reduce I/O contention. However, there does exist a concept of logical partitions. That is, all the keys of an NPI that point to rows of a physical data partition are considered to be a logical partition of the index.

In this visual, we can see the partitioning index that we defined in earlier visuals. The partitioning index partitions the table on ACCOUNT_NUM and there are four partitions (four data partitions and four corresponding index partitions). The other index (at the bottom of the visual) is an NPI on STATE_CD and is composed of one physical data set. Notice that logical partitions exist for the NPI, but they are only used by utilities for claim and drain processing. Note that in V7, this second index is referred to as a non-partitioned index (NPI) or a secondary index.

NPI Challenges - Prior to V8

Redbooks

NPIs increase intersystem read-write interest in data sharing

Sheer size of NPIs over large partitioned tables makes their management as a single object difficult

Recovery of an NPI must be done at the entire index level

Partition-level operations, such as deleting all the data from a partition, are less clean with NPIs

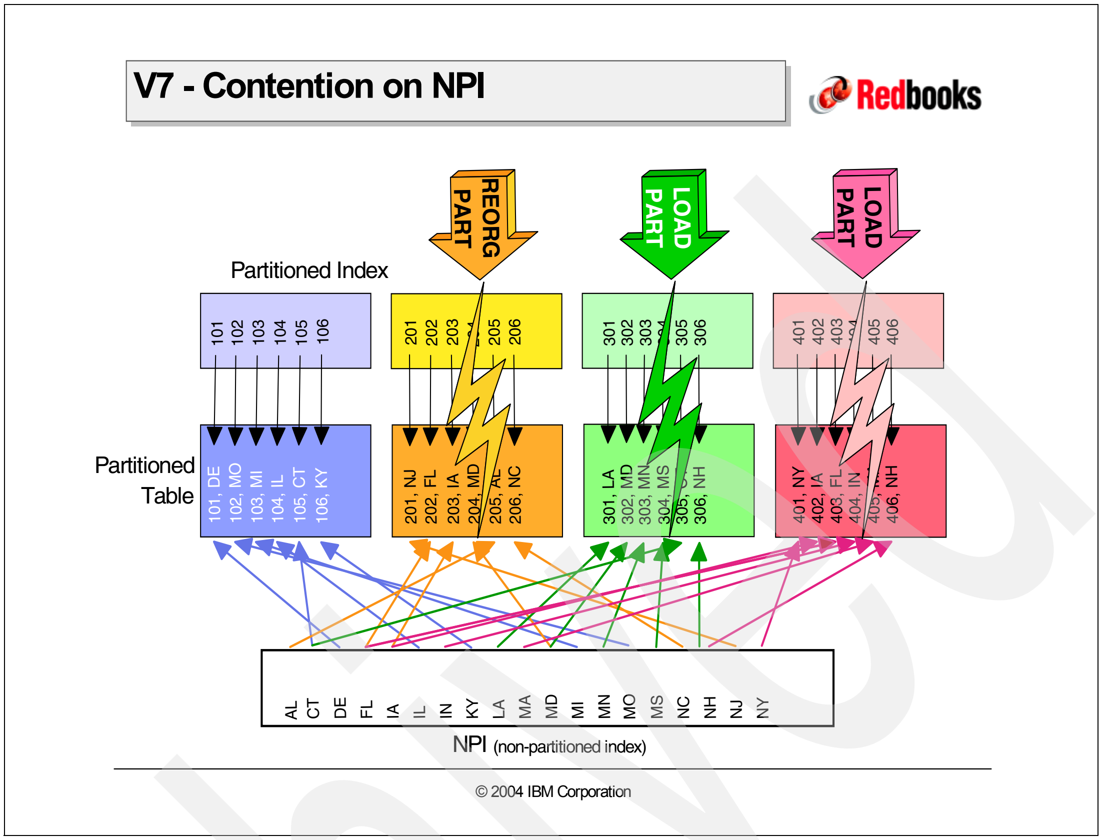
Online REORG of a single partition requires a BUILD2 phase where the entire object is unavailable

Parallel LOAD PART jobs create contention on the NPI

3.5.5 Challenges when using NPIs prior to DB2 V8

The positive aspects of partitioning begin to deteriorate if there are non-partitioned indexes present. More examples follow:

- In a data sharing environment, some customers find benefit from isolating the work on certain members to certain partitions. Such affinity-routing eliminates intersystem read-write interest on physical partitions, thereby reducing data sharing overhead. Affinity routing does not alleviate contention on non-partitioned indexes, since keys that belong to different data partitions are spread throughout the non-partitioned index.
- Recovery from a media failure on a non-partitioned index can only be done at the entire index level. No piece-level rebuild or recovery can be done for a non-partitioning index.
- The sheer size of NPIs over very large partitioned tables makes their management as a single large object difficult. RUNSTATS, REORGs, REBUILDs, etc. take longer clock time than if they could be run on smaller object parts in parallel.
- Partition-level operations become less clean if there are non-partitioned indexes present. For example, to erase the data of a partition, you normally LOAD that partition with an empty input file. This operation quickly "resets" the data partition as well as the partitioned index, but also entails removing key entries from the NPIs that reference the partition being erased. For each row being removed from that partition, DB2 has to look up the key entry for that row in the non-partitioned index and delete it.



3.5.6 V7 and prior - contention on the NPI

As mentioned before, the positive aspects of partitioning begins to deteriorate if there are non-partitioned indexes present. Here we provide some examples related to running utilities:

- When non-partitioned indexes exist on the table space, a BUILD2 phase is performed during online REORG of a partition. This phase uses the shadow index for the index's logical partition to correct RID values in the NPI. During this phase, the utility takes exclusive control of the logical partition. This blocks queries that are not partition-restrictive from operating.
- LOAD PART jobs, run concurrently, contend on non-partitioned indexes because keys of all parts are interleaved. In addition, during a LOAD PART job, key processing against non-partitioned indexes follows insert logic (a row at a time) which is slower than append logic (a page at a time).

V8 Partitioned Tables

Redbooks

Table-controlled partitioning

- Creating a table-controlled partitioned table
- Converting to table-controlled partitioning
- Catalog support

Classification of indexes

- Partitioning/secondary
- Partitioned/non-partitioned

Data-Partitioned Secondary Index

- Creating
- Utility interactions
- Query performance with DPSIs
- Design and planning considerations

Clustering

3.6 V8 Partitioned tables

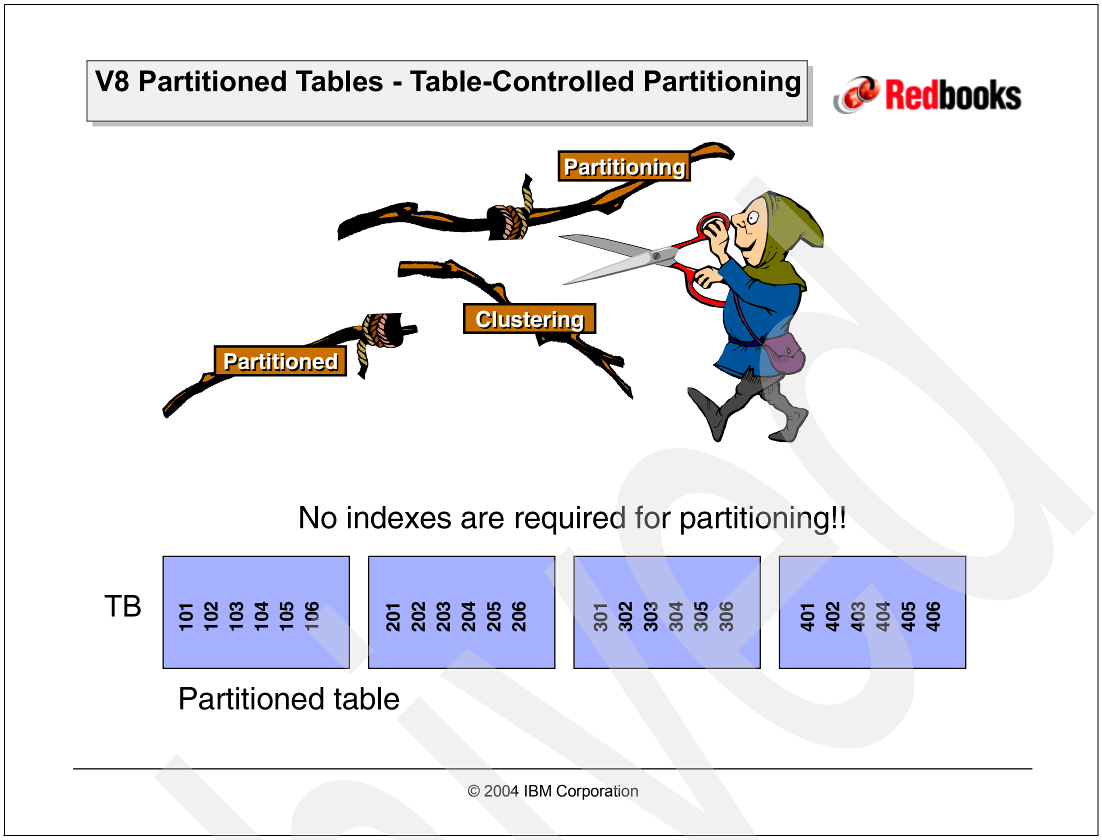
In the following sections we describe the many enhancements to partitioned tables in DB2 Version 8.

First we introduce the “new” way to create partitioned tables, called “table-controlled partitioning”. We also look at how to “convert” from index-controlled to table-controlled partitioning, and where the information about table-controlled partitioning is stored in the DB2 catalog.

With the changes in partitioning, it is important to have correct terminology to describe the different types of indexes on a partitioned table: partitioning versus secondary indexes, and partitioned versus non-partitioned indexes.

DB2 Version 8 also introduces a new type of index, a so-called data-partitioned secondary index. We look at the problems it solves and some design considerations on when and how to use them.

Lastly, we look at the clustering enhancements, where in V8 any index can be the clustering index, and the fact that you can change the clustering index on the fly.



3.6.1 V8 partitioned tables, table-controlled partitioning

By using table-controlled partitioning, clustering, being partitioned, and being the partitioning index are now separate concepts. When using table-controlled partitioning, a table does not require a partitioning index, as the partitioning is done based on the **`PARTITION BY`** clause in the **`CREATE TABLE`** statement. Since the partitioning index is no longer required, another index may also be used as a clustering index. We will discuss clustering in detail in "Clustering indexes" on page 124.

V8 - Creating Partitioned Tables

Redbooks

```
CREATE TABLESPACE tsname NUMPARTS n
        (PARTITION 1  USING . . .
         . . .
         PARTITION n  USING . . . )
      IN dbname;

CREATE TABLE CUSTOMER (
        ACCOUNT_NUM        INTEGER,
        CUST_LAST_NM       CHAR(30),
        . . .
        LAST_ACTIVITY_DT         DATE,
        STATE_CD                 CHAR(2) )
PARTITION BY ( ACCOUNT_NUM      ASC )
        (  PARTITION  1   ENDING AT (199),
           PARTITION  2   ENDING AT (299),
           PARTITION  3   ENDING AT (399),
           PARTITION  4   ENDING AT (499)  )
        IN dbname.tsname;
```

Definition is complete at this point!!

3.6.2 V8 - Creating partitioned tables

Before V8, only *index-controlled partitioning* was supported. With index-controlled partitioning, the partitioning key and partition boundaries are specified on the **CREATE INDEX** statement, when creating a partitioning index on the table. This results in a table that is unusable or "incomplete" until the partitioning index is created.

DB2 V8 introduces *table-controlled partitioning*. That is, when creating a partitioned table, the partitioning key and partition boundaries can be specified on the **CREATE TABLE** statement, as shown in the visual.

When creating a partitioned table (the table space had a **NUMPARTS** keyword specified), table-controlled partitioning is initiated by specifying the new **PARTITION BY** clause on the **CREATE TABLE** statement. The **PARTITION BY** clause identifies the columns and values used to define the partition boundaries. When the new clause is used, the definition of the table is complete and data can be inserted into the table.

Instead of the **PART** and **VALUES** keywords used in DB2 V7 index-controlled partitioning, we now use the **PARTITION** and **ENDING AT** keywords. The old syntax keyword combination (**PART** and **VALUES**) is still supported, but what actually controls whether a table is index-controlled or table-controlled is the **PARTITION BY** clause. However, you cannot code **PART BY**.

Once you have established table-controlled partitioning for a table, index-controlled partitioning is no longer an option for that table. Any attempt to create an index on this table with the **VALUES** or **ENDING AT** keyword is disallowed.

Note: Because the table definition of a table-controlled partitioned table is complete after executing the **CREATE TABLE** statement, no partitioning index is required. So you can have a (table-controlled) partitioned table without a partitioning index.

For some applications, the partitioning column (for example, a date) is not the column that is used to access the data. The partitioning column is often chosen to accommodate for easy data maintenance (like loading additional rows). Now that a partitioning index is no longer required in a table-controlled partitioned table, you can consider getting rid of the partitioning index in those cases. If the partitioning index is only used to define partition boundaries, the partitioning index can be dropped. When you drop the partitioning index of an indexed-controlled partitioned table, the table is automatically converted to table-controlled partitioning.

Both types of partitioning are supported with DB2 V8, but several new enhancements are only supported when using table-controlled partitioning. Note that for the new table-controlled partitioning, the syntax has been changed and wherever you used to code the **PART** keyword, you now code the **PARTITION** keyword. The old syntax is still used to define old style index-controlled partitioning in DB2 V8 (existing DDL will continue to work and create old style partitioning).

The **PART n VALUES** keyword is being replaced by the **PARTITION n ENDING AT** keyword. Although the old syntax is still supported, it is encouraged that you start converting existing DDL to use the new syntax. Also consider changing any processes that generate DDL to use the new syntax. Check with your ISV to see what their plans are to convert to use the new syntax. At some future date, the old syntax may no longer be supported.

Creating a partitioned table - example

Example 3-1 shows how to create a table-controlled partitioned table space.

Example 3-1 Create a table-controlled partitioned table space

```
CREATE  DATABASE  DBBARTP;

CREATE  TABLESPACE  TSBARTP NUMPARTS  4
(PARTITION  1   USING  STOGROUP SYSDEFLT PRIQTY 10 ,
 PARTITION  2   USING  STOGROUP SYSDEFLT PRIQTY 10 ,
 PARTITION  3   USING  STOGROUP SYSDEFLT PRIQTY 10 ,
 PARTITION  4   USING  STOGROUP SYSDEFLT PRIQTY 10 )
 IN DBBARTP;

CREATE  TABLE  CUSTOMER  (
        ACCOUNT_NUM       INTEGER,
        CUST_LAST_NM      CHAR(30),
        LAST_ACTIVITY_DT  DATE,
        STATE_CD          CHAR(2))
  PARTITION BY ( ACCOUNT_NUM    ASC )
  (  PARTITION  1   ENDING AT (199),
     PARTITION  2   ENDING AT (299),
     PARTITION  3   ENDING AT (399),
     PARTITION  4   ENDING AT (499)  )
IN DBBARTP.TSBARTP;
```

Adding the limit keys afterwards

You can also create the table space and table first, and add the limit keys afterwards. For example, a table that uses table-controlled partitioning is identified in the LIKE clause; the new table does not inherit that table's partitioning scheme. If desired, these partition boundaries can be added by specifying ALTER TABLE with the ADD PARTITION BY RANGE clause. The syntax is shown in Figure 3-1.

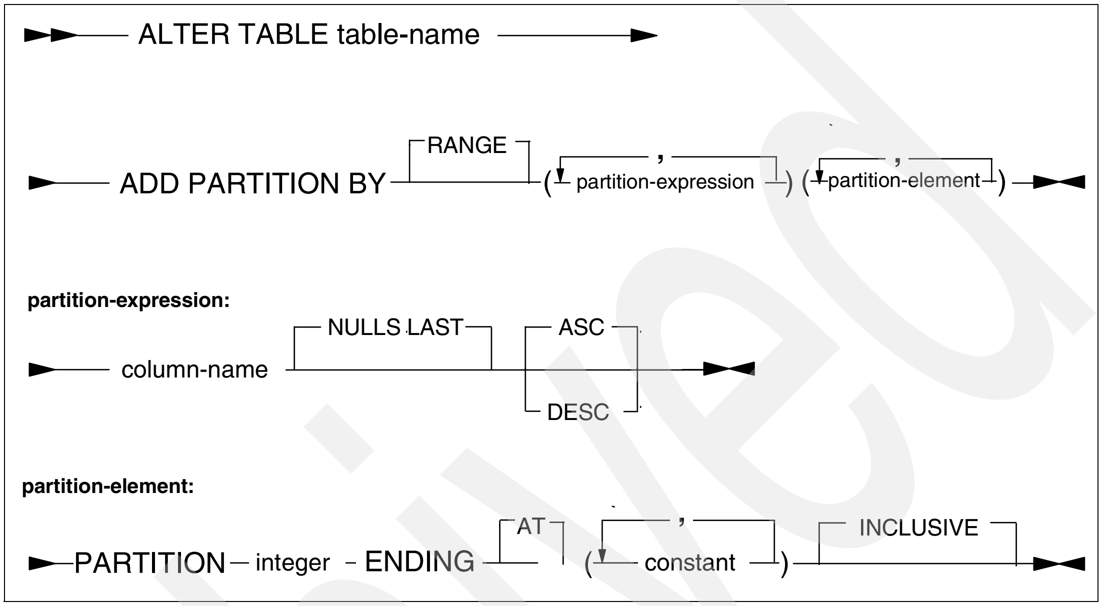


Figure 3-1 Adding a partitioning range

Example 3-2 shows how to use the ADD PARTITION BY RANGE clause on an existing table.

Example 3-2 Using the ADD PARTITION BY RANGE clause

```
CREATE DATABASE BSDBVER2;

CREATE TABLESPACE BSTSVR12 IN BSDBVER2 NUMPARTS 4;

CREATE TABLE BSDBVER2.PARTV12
  ( PARTNUM  CHAR(3)
   ,ALTRNUM  CHAR(6)
   ,COL01    CHAR(13)
   ,COL02    CHAR(13)
   ,COL03    VARCHAR(13) )
IN BSDBVER2.BSTSVR12;

-- You can also use:
-- CREATE TABLE BSDBVER2.PARTV12
--   LIKE other.table_with_same_cols
--   IN BSDBVER2.BSTSVR12;

ALTER TABLE  BSDBVER2.PARTV12 ADD PARTITION BY RANGE
              (PARTNUM ASC,ALTRNUM ASC)
   (PARTITION 1 ENDING AT ('001',X'FFFFFFFFFFFF') INCLUSIVE,
    PARTITION 2 ENDING AT ('002',X'FFFFFFFFFFFF') INCLUSIVE,
    PARTITION 3 ENDING AT ('003',X'FFFFFFFFFFFF') INCLUSIVE,
    PARTITION 4 ENDING AT ('004',X'FFFFFFFFFFFF') INCLUSIVE);
COMMIT;
```

Converting to Table-Controlled Partitioning

No need to DROP/CREATE all existing partitioned tables

DB2 will automatically convert to table-controlled partitioning for you when any of the following SQL statements are executed:

- DROP the partitioning index
- ALTER INDEX NOT CLUSTER on the partitioning index
- ALTER TABLE ... ADD PARTITION
- ALTER TABLE ... ROTATE PARTITION
- ALTER TABLE ... ALTER PARTITION n
- CREATE INDEX ... PARTITIONED
- CREATE INDEX ... ENDING AT ... omitting cluster keyword

Least disruptive approach

- ALTER INDEX xpi NOT CLUSTER of the (current) partitioning index
- ALTER INDEX xpi CLUSTER of the same index

3.6.3 Converting to table-controlled partitioning

For tables that use index-controlled partitioning created in DB2 V8 or in previous releases of DB2, the use of any of the statements listed in this visual will automatically convert the table to use table-controlled partitioning. It is not necessary to drop and recreate the table.

Users are encouraged to convert partitioned tables to use table-controlled partitioning. A non-disruptive method for doing this involves the following two steps:

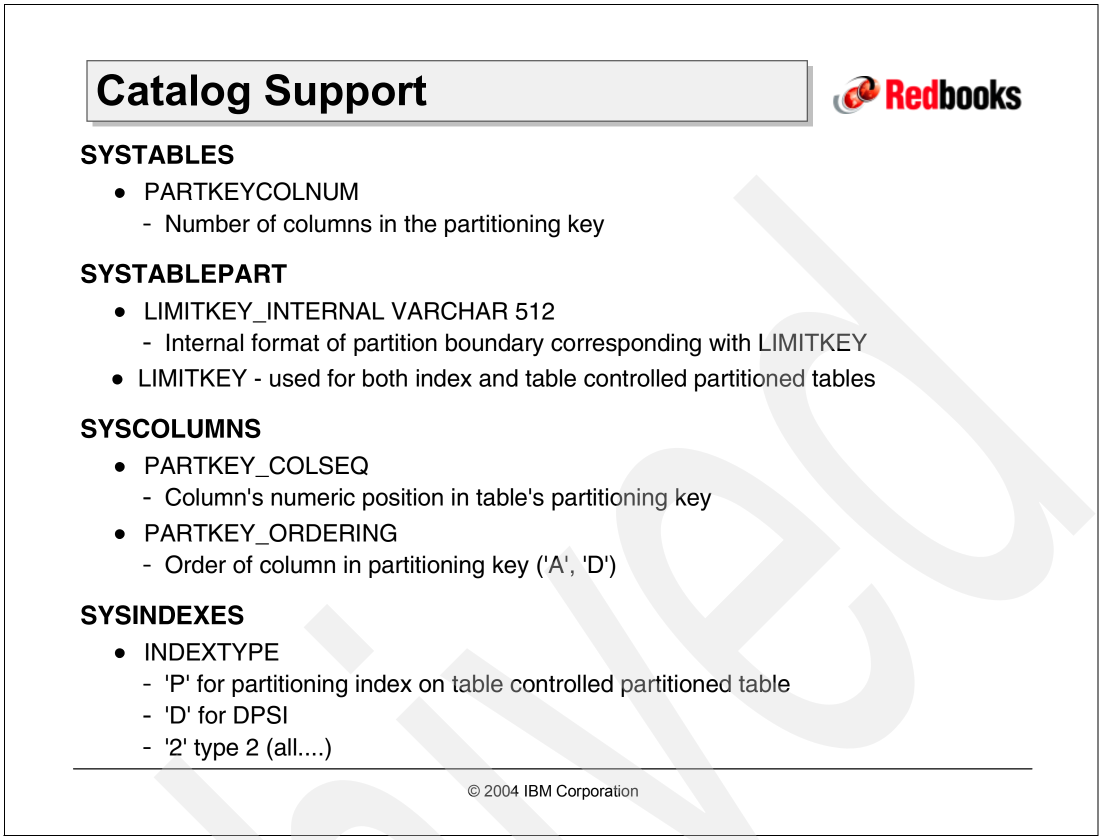
1. `ALTER INDEX` ***ixname*** `NOT CLUSTER` on the partitioning index:
   - The index remains available and functioning as before.
   - The table is converted to table-controlled partitioning and appropriate catalog changes are made.
   - Column CLUSTERING in SYSIBM.SYSINDEXES will be changed from ‘Y’ to ‘N’. DB2 will continue to use this index as the clustering index until another index is explicitly defined with the `CLUSTER` keyword or altered to be the clustering index or you drop the partitioning index.
2. `ALTER INDEX` ***ixname*** `CLUSTER` on the old partitioning index:
   - The index remains available and functioning as before.
   - The table remains table-controlled partitioning.
   - Column CLUSTERING in SYSIBM.SYSINDEXES will change from ‘N’ to ‘Y’. DB2 will continue to use this index as the clustering index. If you attempt to create another index with the `CLUSTER` keyword or to alter another index to be the clustering index, that attempt will fail because only one index can be explicitly defined as clustering.

Note: Table-controlled partitioned tables always enforce the highest (lowest) value on the last partition. This is not the case for index-controlled partitioned tables that are created without the LARGE or DSSIZE keyword. Even though you may have specified a limit key for the last partition, it is not enforced.
Therefore, when such an index-controlled partitioned table is converted to a table-controlled partitioned table by any of the following actions:

- CREATE PARTITIONED INDEX
- DROP INDEX for a partitioning index
- ALTER INDEX with the CLUSTER NO clause

DB2 puts the highest (or lowest) possible value for the partitioning columns into the catalog (SYSTABLEPART.LIMITKEY) for the last partition, potentially overriding the value that was specified at create time. DB2 indicates it has done so by issuing an SQLCODE +20272 “THE LAST PARTITION'S LIMIT KEY VALUE OF old-limit-key-value IS SET TO THE HIGHEST OR LOWEST POSSIBLE VALUE”.

Note: If you later decide to add/rotate a partition to such a partitioned table, you first have to alter the partitioning key to a lower (higher) value, before you can add a new partition or rotate partitions. Adding/rotating a partition requires you to specify a new higher (or lower) partitioning key value for the partition to be added. Since the last partition contains the highest (lowest) possible value after its conversion to a table-controlled partitioned table, you first have to lower (increase) it, using the ALTER TABLE ALTER PARTITION statement, before you can add/rotate a partition.



3.6.4 Catalog support for table-controlled partitioning

For table-controlled partitioning, the limit keys are only stored in columns LIMITKEY and LIMITKEY_INTERNAL of SYSIBM.SYSTABLEPART. No indexes defined on this table will have any values in column LIMITKEY of SYSIBM.SYSINDEXPART. Also, column PARTKEYCOLUMN in SYSIBM.SYSTABLES will have a non-zero value (this is the number of columns in the partitioning key). In addition, column IXNAME of SYSIBM.SYSTABLEPART will be blank.

For index-controlled partitioning, the limit keys are stored in both SYSIBM.SYSINDEXPART and SYSIBM.SYSTABLEPART in column LIMITKEY. Also, column IXNAME of SYSIBM.SYSTABLEPART contains the name of the index-controlled partitioning index.

INDEXTYPE column of SYSIBM.SYSINDEXES

Data Partitioned Secondary Indexes (DPSIs) are identified with a value of 'D' for the INDEXTYPE column of SYSINDEXES. If you want to learn more about DPSI, refer to 3.8, “Data-partitioned secondary indexes” on page 112.

When using table-controlled partitioning, a value of ‘P’ for the INDEXTYPE column of SYSINDEXES indicates that an index is both partitioned and partitioning. In table-controlled partitioning, in order to be a partitioning index, its left-most columns must be the same columns (or be a superset of the columns), in the same order and collating sequence as the columns specified when defining the partitioning of the table.

Indexes on index-controlled partitioned tables, and non-partitioned secondary indexes (non-partitioned and non-partitioning) on table-controlled partitioned tables are identified with a value of '2' for the INDEXTYPE column of SYSINDEXES.

DB2 V8 Classification of Indexes

Redbooks

An index may / may not be correlated with the partitioning columns of the table

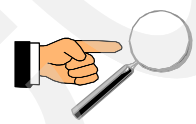
- Partitioning index (PI)
- Secondary index

An index may / may not be physically partitioned

- Partitioned
- Non-partitioned

***Clustering* index:**

- Any index may be the clustering index!!
- The clustering index can be unique / non-unique



3.7 DB2 V8 classification of indexes

Until V7, terms such as secondary index (on a partitioned table), non-partitioned index (NPI), non-partitioning index, or non-clustering index, were often used interchangeably to describe any index other than the partitioning index. With all the enhancements related to partitioning and indexes in DB2 Version 8, it is important that we make sure to use the correct terminology. It is important to distinguish between:

- Partitioning and non-partitioning indexes
- Partitioned and non-partitioned indexes
- Clustering and non-clustering indexes

3.7.1 Index classification

Indexes on partitioned tables can be classified as follows:

- Based on whether or not the columns in the index correlate with the “partitioni**ng**” columns of the table. The partitioning columns are those specified in the `PARTITION BY` clause in the `CREATE TABLE` statement.

  Partitioning index The columns in the index are the same as (are in the same order and have the same collating sequence), or start with the same columns as those specified in the `PARTITION BY` clause of the `CREATE TABLE` statement for table-controlled partitioned tables, or on the `CREATE INDEX` statement for index controlled partitioned tables. A partitioning index can have a super set of the partitioning columns, that is, it can contain all the partitioning columns plus additional columns.

Secondary index Any index where the columns do not coincide with the partitioning columns of the table. We describe these in more detail in 3.7.3, “Secondary indexes” on page 108.

- Based on whether or not an index is physically “partition**ed**”:

  Partitioned index The index is made up of multiple physical partitions (one per data partition), not just index pieces.

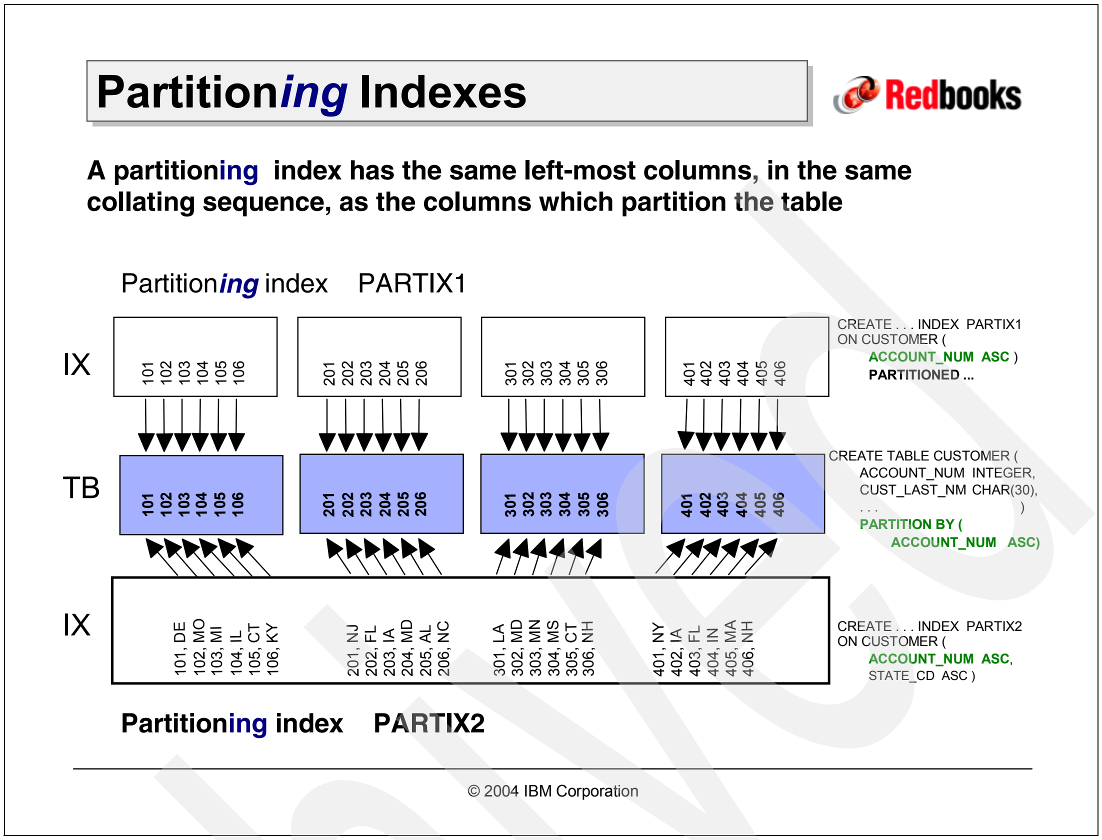
  Non-partitioned index The index is a single physical data set, or multiple pieces.

- Based on whether or not the index determines the **clustering** of the data. Please note that when using table-controlled partitioning, any index may be the clustering index. (With index-controlled partitioning, the partitioning index must be the clustering index). We discuss clustering in more detail in “Clustering indexes” on page 124.

  Clustering index The index determines the order in which the rows are stored in the partitioned table. There can only be one clustering index. The clustering index is either the index that is defined with the `CLUSTER` attribute (explicit clustering index), or the oldest index on the table, if no explicit clustering index is defined (implicit clustering index).

  Non-clustering index The index does not determine the data order in the partitioned table.

  Note that non-partitioned tables can also have any index defined as the clustering index (as in V7).



3.7.2 Partitioning indexes

With DB2 V8, the definition of “partitioning index” has changed. A **partition*ing* index** is no longer necessarily an index that controls how a table is partitioned. Now the term partition*ing* index has two possible meanings:

1. For index-controlled partitioning (old style partitioning):

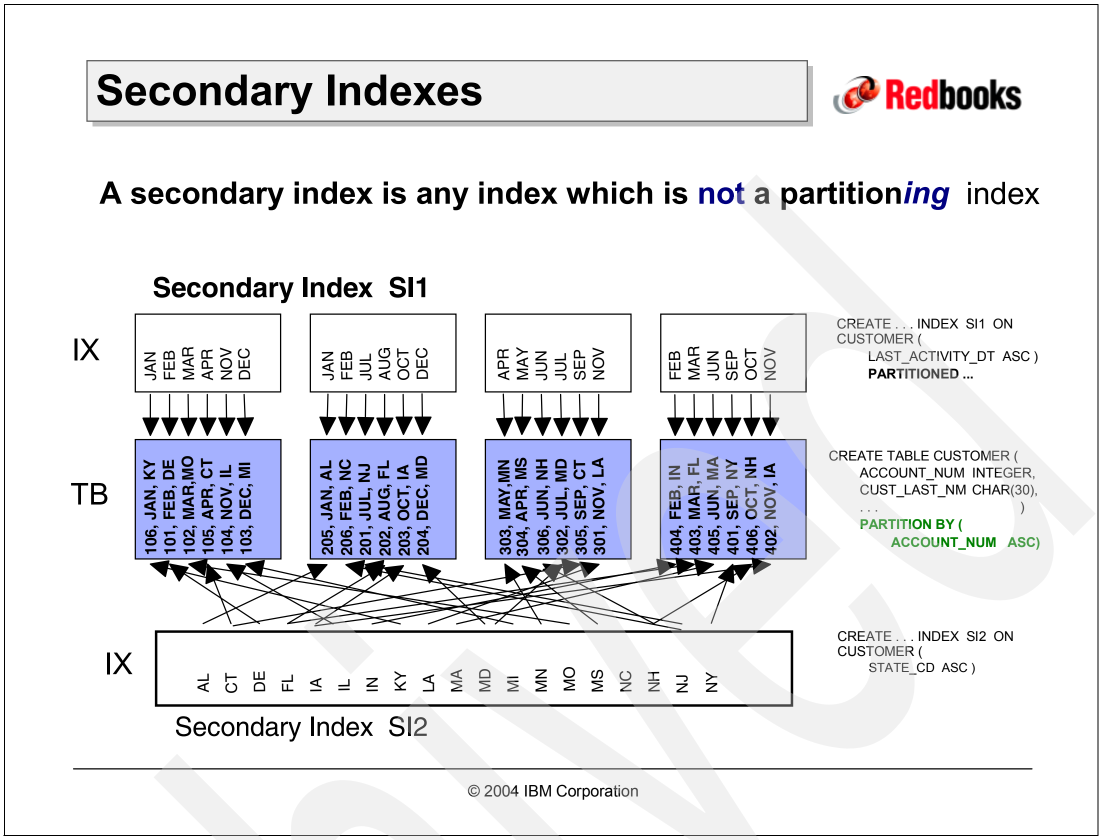
   The term continues to mean that it is the index that controls how the table is partitioned.

2. For table-controlled partitioning (new DB2 V8 style partitioning):

   The term now means that the index has the same left-most key column(s), in the same order, and using the same collating sequence as the columns that control partitioning on the table. As the name implies, table-controlled partitioning means that the table definition itself actually controls how partitioning is done.

In this visual, we have a table-controlled partitioned table. You can tell this by the fact that we specified the `PARTITION BY` clause in the `CREATE TABLE` statement.

Index PARTIX1 is a partition*ing* index because its key has the same left-most column(s), in the same order, and using the same collating sequence as the columns (in this case only one column, ACCOUNT_NUM ASC) which partitions the table. (Index PARTIX1 also happens to be partition*ed*, because we specified the `PARTITIONED` keyword in the `CREATE INDEX` statement.) Even though index PARTIX2 is not partition*ed* (we did not specify the `PARTITIONED` keyword in the `CREATE INDEX` statement), it is also a partition*ing* index because its key also has the same left-most column(s), in the same order, and using the same collating sequence as the columns (in this case, only one column, ACCOUNT_NUM ASC), which partitions the table.



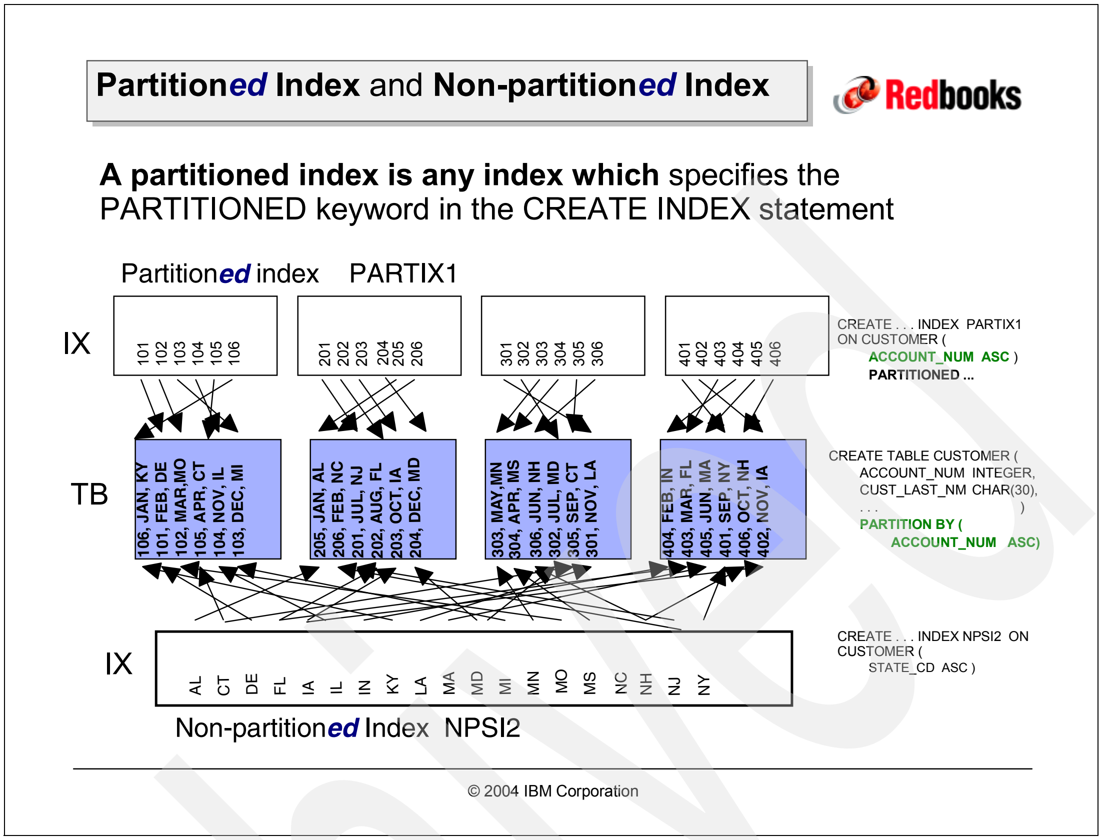
3.7.3 Secondary indexes

Indexes may be created on a table for several reasons: to enforce a uniqueness constraint, to achieve data clustering, but most typically, to provide access paths to data for queries or referential constraint enforcement. While the cost of maintaining any index must always be evaluated against its benefit, several unique factors come into play when deciding whether to add a secondary index to a partitioned table. This is because there are areas where secondary indexes can cause performance and contention problems.

A secondary index is any index that is not a partitioning index. In order for an index to be not partition*ing*, its key must not have the same left-most column(s), or in the same order, or using the same collating sequence as the columns (in this case only one column, ACCOUNT_NUM ASC) which partitions the table.

In this visual, we can see that index SI1 is a secondary index because its key (LAST_ACTIVITY_DT ASC) does not have the same left-most column(s) as those which partition the table (in this case ACCOUNT_NUM ASC). Note that this index also happens to be partition*ed* (because we specified the **PARTITIONED** keyword in the **CREATE INDEX** statement). A partitioned secondary index, or more precisely a Data Partitioned Secondary Index (DPSI), is new in V8. In prior DB2 versions, all secondary indexes were non-partitioned; only the partitioning index could be partitioned.

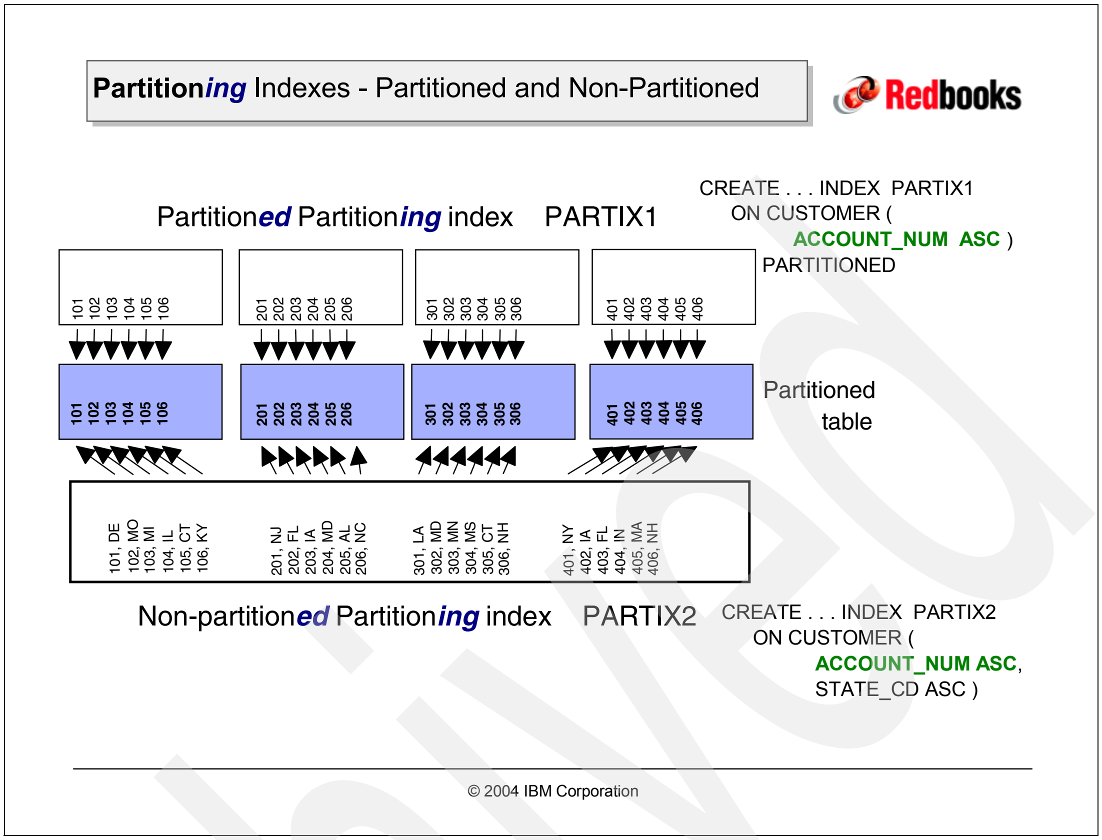
Index SI2 is a secondary index because its key (STATE_CD ASC) does not have the same left-most column(s) as those which partition the table (in this case ACCOUNT_NUM ASC). Index SI2 is also not-partition*ed* (because we did not specify the **PARTITIONED** keyword in the **CREATE INDEX** statement).



3.7.4 Partitioned and non-partitioned indexes

This visual shows the difference between a partition*ed* and a non-partition*ed* index. A partitioned index is made up of multiple physical partitions, one per data partition. The index keys in each index partition correspond to the rows in the same data partition number. That is, index partition 1 only contains keys for those rows found in data partition 1, index partition 2 only contains keys for those rows found in data partition 2, and so on. A partition*ed* index has the keyword **`PARTITIONED`** specified in the **`CREATE INDEX`** statement that defines it. (The partitioning index of an index-controlled partitioned table is also a partitioned index by definition, as it actually defines the partitioning range itself.)

A non-partition*ed* index may be encompassed by one physical data set or multiple data sets if piece size is specified. Non-partition*ed* indexes have a concept of "logical" partitions if the table they are defined on is partition*ed*. A non-partition*ed* index is an index in which the keyword **`PARTITIONED`** is *not* specified in the **`CREATE INDEX`** statement that defined it.

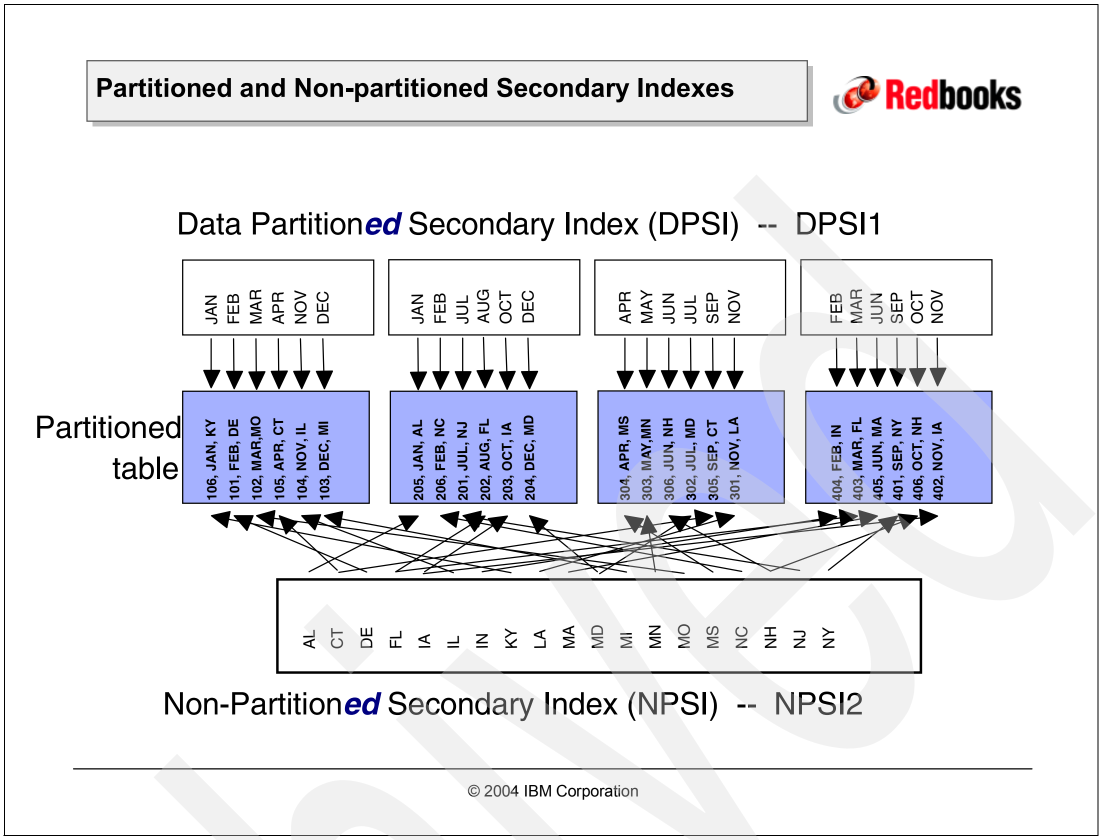


3.7.5 Partitioned and non-partitioned partitioning indexes

The visual above shows two different types of indexes, which combine two of the concepts we talked about in the previous visuals. We start mixing partitioning and partitioned. Starting in Version 8, a partition*ing* index can be partition*ed* (PARTIX1) or non-partition*ed* (PARTIX2).

In the visual, index PARTIX1 is a partition*ed* partition*ing* index. It is partitioned, because it consists of four real partitions just as the partitioned table it is defined on. The fact that an index becomes partitioned does not occur by chance. You must specify the keyword PARTITIONED on the CREATE INDEX statement to do this. This index is also partitioning, because its left-most columns (in this case only one column), is the same as the columns in and in the same collating sequence as the partitioning key (PARTITIONED BY clause) of the underlying table.

Index PARTIX2 is a non-partition*ed* partition*ing* index. It is non-partitioned, because the keyword PARTITIONED was omitted during CREATE INDEX. It is also a partitioning index, because its left-most columns (in this case only one column), is the same as the columns in and in the same collating sequence as the partitioning key (PARTITIONED BY clause) of the underlying table.

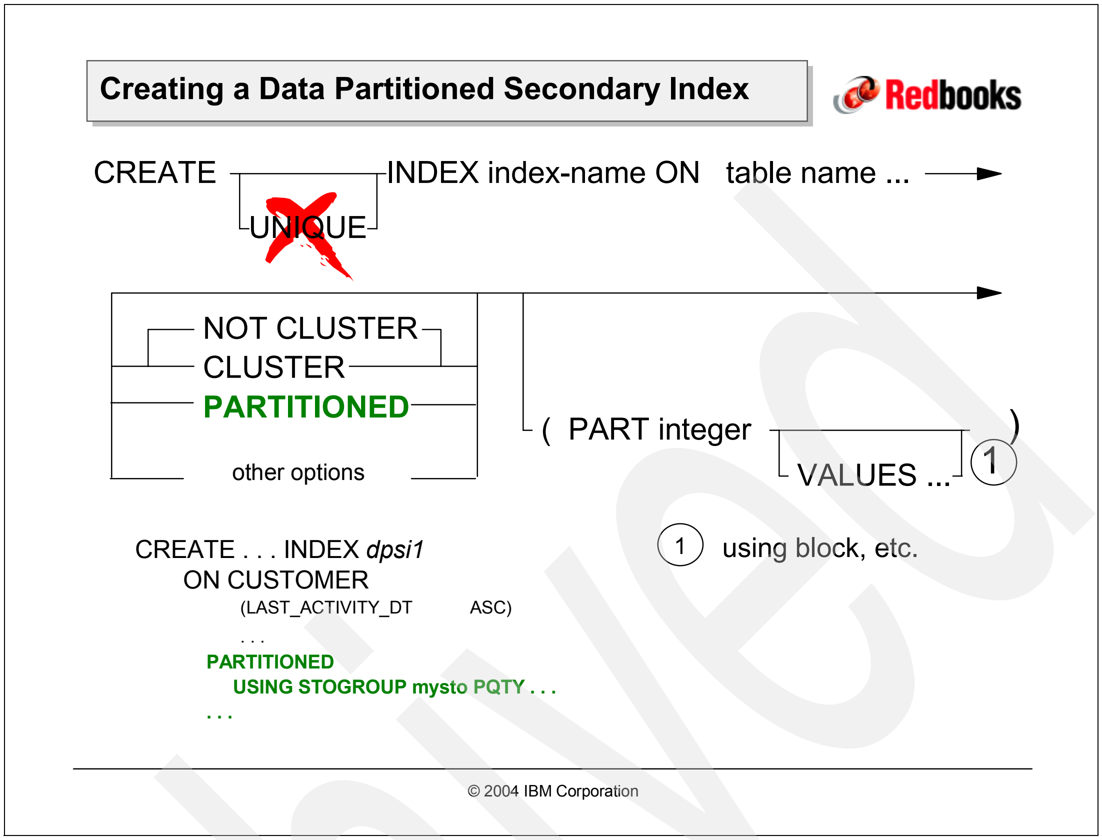


3.7.6 Partitioned and non-partitioned secondary indexes

At the bottom of this visual we also show a secondary index that is made up of a single data set; it is a *Non-Partitioned Secondary Index (NPSI)* “NPSI2”. This is the only type of secondary index that was available before DB2 V8. The NPSI consists of a single data set (or multiple pieces) and is not partitioned according to the table’s partitioning scheme. NPSIs may be defined as unique.

At the top of the visual we see index “DPSI1”, a secondary index that is made up of multiple physical partitions (that match the partitioning scheme of the table); it is a *Data-Partitioned Secondary Index (DPSI)*. It is an index which is partitioned based on data rows. However, the index contains different columns than the partitioning columns or in a different order or collating sequence than those which partition the table. DPSIs must allow duplicates and thus must not be unique. DPSIs are new in V8.

The visual above also illustrates that a single table may support a mix of non-partitioned and data-partitioned secondary indexes. We will discuss DPSIs in more detail next.



3.8 Data-partitioned secondary indexes

As introduced in the previous section, DB2 V8 has the ability to physically partition secondary indexes. The partitioning scheme introduced is the same as that of the table space. That is, there are as many index partitions in the secondary index as table space partitions, and index keys in partition 'n' of the index reference only data in partition 'n' of the table space. Such an index is called a Data-Partitioned Secondary Index (DPSI).

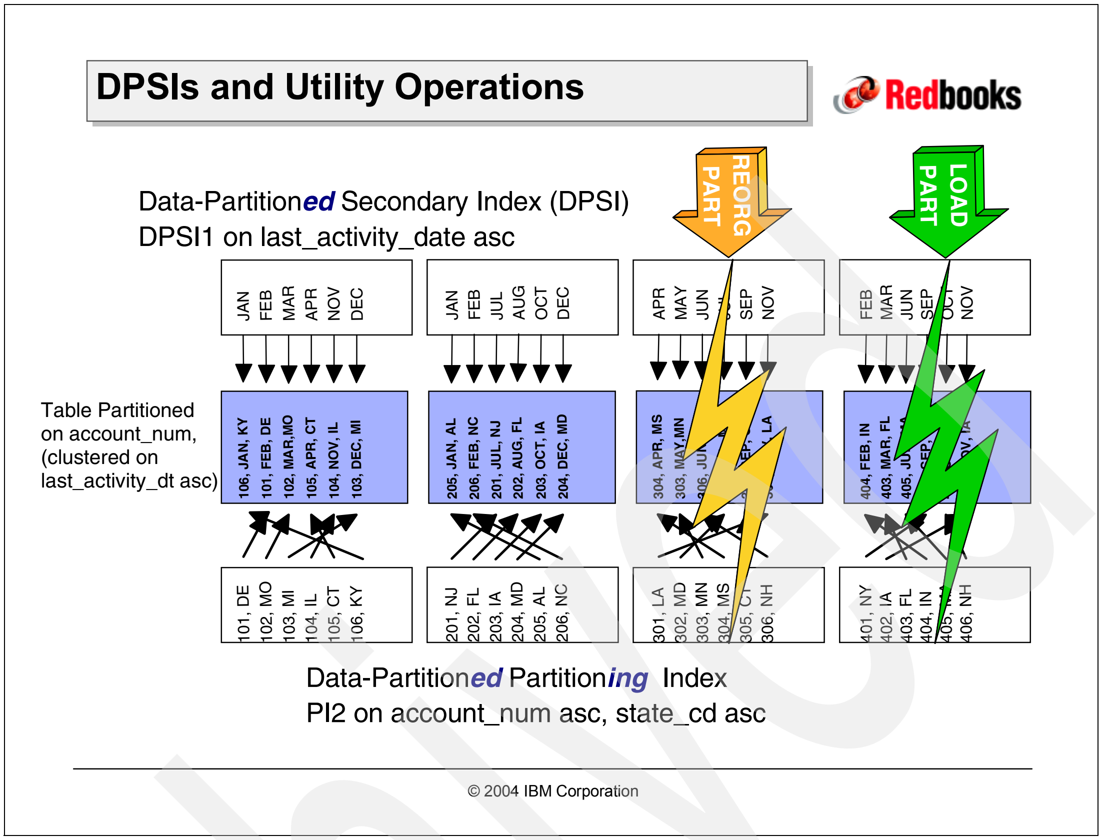
DPSIs introduce a number of new design opportunities, some of which we will discuss next, but first we need to look at how to create a data-partitioned secondary index.

3.8.1 Creating a data-partitioned secondary index

As mentioned before, data-partitioned secondary indexes (DPSIs) are new in DB2 V8. You create a DPSI by specifying the new **`PARTITIONED`** keyword on the **`CREATE INDEX`** statement, as shown in the visual above. When using the **`PARTITIONED`** keyword, a secondary index cannot be created as **`UNIQUE`** or **`UNIQUE WHERE NOT NULL`**. This is to avoid having to search each part of the DPSI to make sure the key is unique.

The index “DPSI1” on the previous visual is a data-partitioned secondary index. (In our case the DPSI is also the clustering index, but that does not have to be the case. You can see that inside each part of the DPSI, the keys are stored by month in ascending order.) Also note that each part of the DPSI potentially has values for all months, as each part of the DPSI stores keys that relate to the rows in the corresponding data partition.

When creating a partitioned index, you cannot specify the size of each *individual* part of the index. You cannot specify a PART clause on the create index statement (for a table-controlled partitioned table). If the sizes of all partitions have to be different, you can use the **`ALTER INDEX ix-name ALTER PARTITION partno PRIQTY value`** statement.



3.8.2 DPSIs and utility operations

By looking at the layout of the keys in the DPSI's parts, it is obvious that this organization promotes high data availability by facilitating efficient utility processing on data partitioned secondary indexes. It also streamlines partition-level operations such as adding and rotating partitions, also introduced in DB2 V8. (See 3.13, "Partition management" on page 166 for more details.

- Elimination of the BUILD2 phase during online REORG of a partition:

  There is no BUILD2 phase processing for DPSIs. Because keys for a given data partition reside in a single DPSI partition, a simple substitution of the index partition newly built by REORG for the old partition is all that is needed. If all indexes on a table are partitioned (partitioned PI or DPSIs), the BUILD2 phase of REORG is eliminated.

- Elimination of LOAD PART job contention and enabling append (load) mode insertion for much more efficient processing:

  There is no contention between LOAD PART jobs during DPSI processing. This is because there are no shared pages between partitions on which to contend. Thus if all indexes on a table are partitioned, index page contention is eliminated.

  Also note that during parallel LOAD PART job execution, each LOAD job inserts DPSI keys into a separate index structure, in key order. This allows the LOAD utility logic to follow an efficient append strategy (instead of doing "row at a time" logic).

- Facilitation of partition-level operations:

  Because keys for a given data partition reside in a single DPSI partition, partition-level operations can take place at a physical versus logical level. Thus, partition-level operations are facilitated to the extent that a table's indexes are partitioned.

3.8.3 Data sharing overhead reduction

Data sharing overhead-reducing strategies, such as affinity routing, can benefit from the use of data partitioned secondary indexes. Because P-locking occurs at the physical partition level, affinity routing is effective for DPSIs. Thus data sharing overhead is reduced to the extent that the table's secondary indexes are partitioned. This should be very useful when taking advantage of data affinities during batch suites.

DPSI Query Performance

Query performance characteristics of DPSIs

- Allows query parallelism
- Queries with predicates only on secondary index columns will need to scan all partitions
- DB2 tries to do partition pruning -- The application needs to code explicit partitioning key predicates to allow for partition pruning when a DPSI exists
- Chosen for queries with predicates on partitioning columns plus secondary index columns

Secondary indexes

- NPSIs
  - pro: favor sequential query performance
  - con: partition-level query or utility operations
- DPSIs
  - pro: favor partition-level query or utility operation
  - con: sequential query performance, although well-suited to partition parallelism

3.8.4 DPSI query performance

DPSIs allow for query parallelism and are likely to be picked by the optimizer for queries with predicates on partitioning columns plus predicates on the secondary index columns.

The physical nature of a DPSI can weaken the profile of some types of queries. Queries with predicates that solely reference columns of the secondary index are likely to experience performance degradation, due to the need to probe each partition of the index for values that satisfy the predicate. Queries with predicates against the secondary index that also restrict the query to a single partition (by also referencing columns of the partitioning index), on the other hand, benefit from the organization. This is called *partition pruning*.

For example, if you are aware of a correlation between DPSI and PI key values, code the PI restriction explicitly when supplying a DPSI predicate to facilitate partition pruning. Let us assume that the partitioning column of a table is DATE, and a data partitioned secondary index exists on ORDERNO. If the company has a policy that the first four digits of ORDERNO are always a four-digit year, you should write queries that include both ORDERNO and DATE in the WHERE clause. For example, instead of coding WHERE ORDERNO BETWEEN 200400001 and 200400999, use WHERE ORDERNO BETWEEN 200400001 AND 200400999 AND DATE BETWEEN '2004-01-01' AND '2004-12-31'. This will allow the pruning of unqualifying partitions from the query.

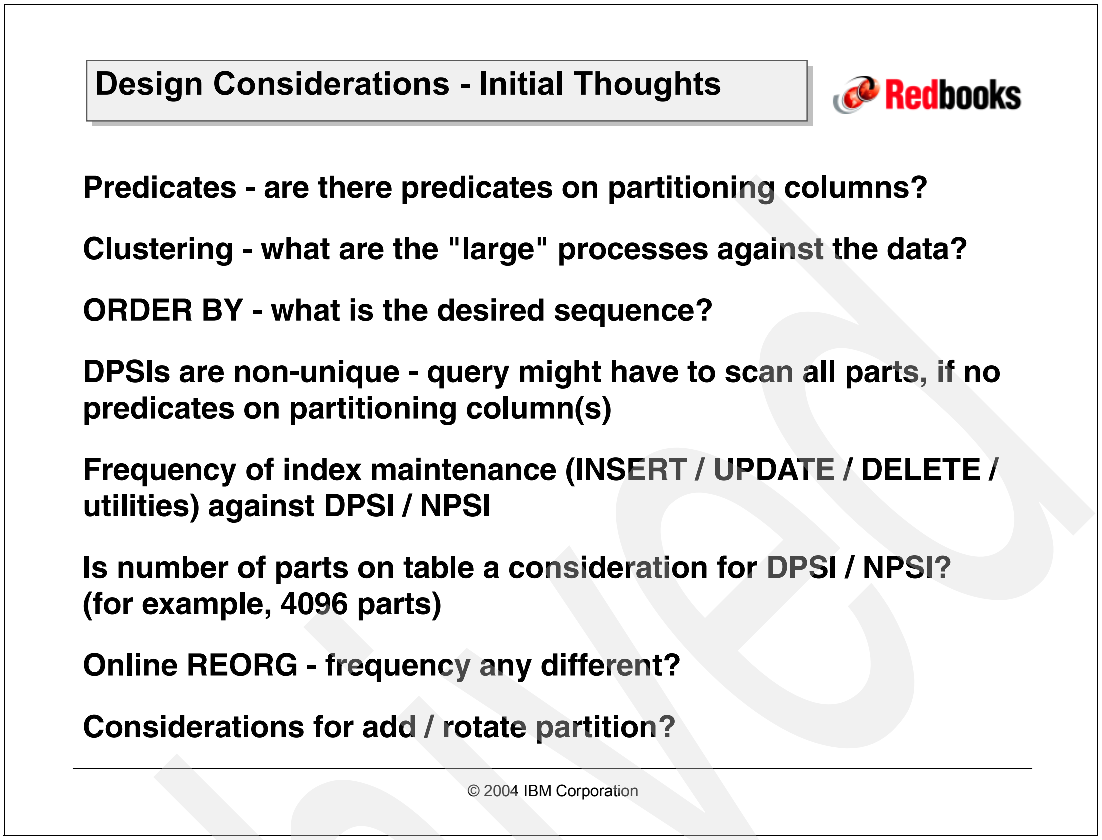
Note: Because DPSIs require a separate probe for each qualified partition, it is very important to reduce the number of qualified partitions as much as possible.

When the qualified partitions cannot be determined at BIND time because the literal value in the predicate is unknown (for example, because a host-variable, parameter marker, or special register is used), then DB2 will determine what the qualified partitions are at execution time. By doing this, only the qualified partitions will be accessed, even if DB2 could not determine which partitions were qualified at BIND time. (This enhancement does NOT require the use of REOPT(VARS)).

In addition, the DB2 access path selection has been enhanced to make use of all leading partitioning key columns when determining the qualified partitions. (Currently, only the first column of the partitioning key is used to determine which partitions qualify. This can result in more partitions being in the range of qualified partitions than actually do qualify.)

This enhancement will not only benefit cases that use a DPSI to access the data, but any case where a partitioned table space was accessed with too many partitions qualified.

The DB2 optimizer is aware of the nature of DPSIs and will take their strong points and weaknesses into account when determining the best access path.



3.8.5 Design considerations - initial thoughts

Data partitioned secondary indexes are not a solution that fits all cases. They must be used wisely. DPSIs only give you an additional design option.

The decision to use a non-partitioned secondary index or a data-partitioned secondary index must take into account both data maintenance practices and the access patterns of the data. We recommend replacing an existing non-partitioned secondary index with a data-partitioned index only if there are perceivable benefits such as easier data or index maintenance, improved data or index availability, or improved performance.

Note also that the capacity to partition secondary indexes provided by this enhancement, coupled with the increase in the number of partitions supported in DB2 V8, increases the design feasibility of using efficient LOAD PART operations to add new data to a table (as opposed to using SQL INSERT operations).

Furthermore, if that design option is taken, the frequency with which REORG INDEX needs to be run on the secondary indexes may decline (since the LOAD operator can reserve free space for future insert, and because REORG of each part no longer needs to be followed by a REORG INDEX of the NPIs to clean them up from all the BUILD2 activity).

Utility Operations - DPSIs

CHECK DATA

- When running on entire table space, sort must be done for DPSI keys. In basically all other cases, sort is avoided.

CHECK INDEX

- Can be run on partition of DPSI, or logical partition of NPSI

RUNSTATS

- May be run against single partitions, including DPSIs. Partition-level statistics are used to update aggregate statistics for the entire table.

Partition parallelism

- DPSIs allow for totally concurrent operations with PART keyword, as do PIs
- LOAD, REORG, REBUILD INDEX, CHECK INDEX

Work data sets may require more space if there is a mixture of DPSIs and NPSIs

3.8.6 Utility operations - DPSIs

Data Partitioned Secondary Indexes also improve the recovery characteristics of your system. DPSIs can be copied and recovered at the partition level. Individual partitions can be rebuilt in parallel to achieve a fast rebuild of the entire index.

A more detailed account of utility and query processing of partitioned secondary indexes is given in 9.12, “Utility changes to support DPSIs” on page 677.

System Planning for DPSIs

DSMAX (and z/OS limit) number of data sets backing a secondary index could increase

Catalog / Directory growth

- DPSIs:
  - SYSINDEXPART
  - SYSCOLDISTSTATS
  - SYSINDEXSTATS
  - DSNDB01.DBD01
- For indexes which are COPY YES:
  - SYSLGRNX
  - SYSCOPY
- For indexes whose HISTORY statistics are collected:
  - SYSINDEXPART_HIST
  - SYSINDEXSTATS_HIST
  - SYSCOLDISTSTATS_HIST

Larger EDM pool requirements

Programs / queries that access SYSINDEXPART may need to change!!

(TYPE2 keyword no longer in CREATE INDEX syntax)

3.8.7 System planning and administration of DPSIs

A secondary index can now be partitioned and you may now have up to 4096 partitions per partitioned table. Each partition requires its own data set. You should take into consideration the DSMAX parameter, as the number of data sets can increase more easily and in larger increments (you can have up to 4096 new data sets per DPSI index).

The ability to have more partitions or indexes that can be partitioned, you will also need additional space in catalog and directory tables. You should consider increasing the size of the catalog tables listed in this visual when you start to use DPSIs extensively, especially when combined with partitioned tables with a large number of partitions.

More storage in the EDM pool will be required since DBDs will be larger due to support of more partitions. The size of the EDM pool should be reviewed if you plan to use many more partitions.

Programs and queries that access index partitioning information in the catalog may need to change. They will need to take into account the different kinds of indexes now available and the increase in the number of partitions.

Displaying a Partitioning Index

Redbooks

```
DSNT360I - ***********************************
DSNT361I - *  DISPLAY DATABASE SUMMARY
           *    RESTRICTED
DSNT360I - ***********************************
DSNT362I -     DATABASE = DBKD0101  STATUS = RW
               DBD LENGTH = 4028
DSNT397I -
NAME     TYPE  PART  STATUS           PHYERRLO . . .
-------- ----  ----- ---------------- -------- . . .
INDEX1   IX     0001 RW
INDEX1   IX     0002 RW
INDEX2   IX     L*   RW
INDEX3   IX    D0001 RW
```

4 digit numeric (nnnn) partition numbers up to 4K)indicate a partition of a partitioned partitioning index

3.9 Displaying indexes in DB V8

In this and the next set of visuals, we look at the output of the DISPLAY database command for the different types of indexes that exist in DB2 Version 8.

3.9.1 Displaying a partitioned partitioning index

When you display a partitioned partitioning index, that is an index that is made up of multiple parts, and has the same left-most columns as the partitioning columns; the display output gives you just the number of the part. In the figure above, INDEX1 is a partitioned partitioning index.

Note that a partitioned partitioning index is the same as the partitioning index of an index-controlled partitioned table space.

Example 3-3 shows another example of a DISPLAY DATABASE command output.

Example 3-3 Display database output

```
DSNT360I  -DB8A ***********************************
DSNT361I  -DB8A *  DISPLAY DATABASE SUMMARY
                *    GLOBAL
DSNT360I  -DB8A ***********************************
DSNT362I  -DB8A     DATABASE = BSDBVER4  STATUS = RW
                   DBD LENGTH = 4028
DSNT397I  -DB8A
NAME     TYPE PART  STATUS            PHYERRLO PHYERRHI CATALOG  PIECE
-------- ---- ----- ----------------- -------- -------- -------- -----
BSTSVR12 TS    0001 RW
BSTSVR12 TS    0002 STOP
BSTSVR12 TS    0003 RW
    -THRU      0004
DIX1     IX   D0001 RW
    -THRU      0004
NIX1     IX    L*   RW
PIX1     IX    0001 RW
PIX1     IX    0002 RO
PIX1     IX    0003 RW
    -THRU      0004
******* DISPLAY OF DATABASE BSDBVER4 ENDED      **********************
DSN9022I  -DB8A DSNTDDIS 'DISPLAY DATABASE' NORMAL COMPLETION
***
```

In the example above, DIX1 is the DPSI, NIX1 is a non-partitioned index, and PIX1 is a partitioned partitioning index.

Displaying Non-Partitioned Indexes

```
DSNT360I - ***************************************
DSNT361I - *  DISPLAY DATABASE SUMMARY RESTRICTED
DSNT360I - ***************************************
DSNT362I -      DATABASE = DBKD0101  STATUS = RW
                DBD LENGTH = 4028
DSNT397I -
NAME     TYPE PART   STATUS            PHYERRLO . . .
-------- ---- -----  ----------------- -------- . . .
INDEX1   IX    0001  RW
INDEX1   IX    0002  RW
INDEX2   IX    L*    RW
INDEX3   IX   D0001  RW
INDEX3   IX   D0002  RW
INDEX4   IX   L0001  RECP
INDEX4   IX   L0002  RW
```

Non-partitioned indexes (NPSIs or NPPIs) are shown by L* or Lnnnn.
L* is used when there is no information to be displayed at a logical partition level.
Lnnn indicates information about a logical partition.

3.9.2 Displaying non-partitioned indexes

Non-partitioned indexes are displayed with an L; either L*, when all logical partitions have the same status, or Lxxxx for individual logical parts. In the example above, INDEX2 is a non-partitioned index with no logical parts in any special status, therefore L*. (For more information on logical partitions, see 3.5.4, “Logical and physical partitions” on page 94.

INDEX4 is also a non-partitioned index and here logical part #1 (L0001) is in recovery pending status.

Note that the usage of “L” applies to both non-partitioned secondary indexes (NPSIs) as well as non-partitioned partitioning indexes (NPPIs).

Displaying DPSIs

```
DSNT360I - ****************************************
DSNT361I - *  DISPLAY DATABASE SUMMARY RESTRICTED
DSNT360I - ****************************************
DSNT362I -      DATABASE = DBKD0101   STATUS = RW
                DBD LENGTH = 4028
NAME     TYPE PART   STATUS            PHYERRLO . . .
-------- ---- -----  ----------------- -------- . . .
INDEX1   IX    0001  RW
INDEX1   IX    0002  RW
INDEX2   IX    L*    RW
INDEX3   IX   D0001  RW
INDEX3   IX   D0002  RW
INDEX4   IX   L0001  RECP
```

Dnnnn indicates information about partitions of a DPSI

3.9.3 Displaying data-partitioned secondary indexes

When you display a data-partitioned secondary index (DPSI), you will notice that each part of the DPSI is preceded by a D, for example D0001.

In the figure above, INDEX3 is a DPSI consisting of two parts, D0001 and D0002.

Clustering Indexes

Any index can be the clustering index

The CLUSTER keyword is now optional when creating a partitioning index

Ordering of rows for INSERT and REORG:

- Partitioning columns determine the proper partition for the row
- The clustering index determines the location within the partition (assuming availability of space at that location)

The clustering index can be changed using ALTER INDEX

Steps to change the clustering index

- ALTER INDEX index1 NOT CLUSTER
- ALTER INDEX index2 CLUSTER
- (Followed by REORG to change the sequence of existing rows)

3.10 Clustering indexes

Historically, the partitioning index for partitioned tables also had to be the clustering index. These two attributes (partitioning and clustering) are unbundled in Version 8.

In DB2 V8, any index on a table-controlled partitioned table can be the clustering index, including a secondary index.

The `CLUSTER` keyword is now optional when creating a partitioning index. Please note that if you are attempting to create an index-controlled partitioned table space, if you do not specify the `CLUSTER` keyword in the `CREATE INDEX` statement for the partitioning index, then that table is converted to table-controlled partitioning and the table definition is complete.

The partitioning columns (those specified in the `PARTITION BY` clause) determine the proper partition for the placement of rows. The clustering index (the index defined with the `CLUSTER` keyword or the first index defined on the table) controls the clustering or location within the partition. If there are no indexes defined on the table space, then clustering is done using the partitioning columns specified in the `PARTITION BY` clause.

You now also have the capability to change the clustering index with an `ALTER INDEX` statement. Since only one index can be explicitly defined as a clustering index, you must follow these steps to change the clustering:

1. `ALTER INDEX` *ixname* `NOT CLUSTER` on the current clustering index.

   Clustering continues to be done according to this index until a new clustering index is explicitly defined

2. **ALTER INDEX** *ixname* **CLUSTER** on the index you wish to be the new clustering index.

   New rows will be clustered according to the new clustering index, old rows will remain in their current location.

3. REORG the table space to re-arrange the rows in the new clustering index order.

   All existing rows will be re-arranged in the new clustering sequence. Any new rows will be inserted with the new clustering sequence.

   It is worth noting that for *table-controlled partitioning*, the SYSIBM.SYSTABLEPART columns IXNAME and IXCREATOR contain blanks, and column LIMTKEY_INTERNAL has the highest value of the limit key of the partition in internal format.
   For *index-controlled partitioning*, columns IXNAME and IXCREATOR contain the index name and creator, and column LIMITKEY_INTERNAL has blanks.

If no explicit clustering index is specified for a table, the V8 REORG utility recognizes the first index created on each table as the implicit clustering index when ordering data rows.

If explicit clustering for a table is removed (changed to NOT CLUSTER), that index is still used as the implicit clustering index until a new explicit clustering index is chosen.

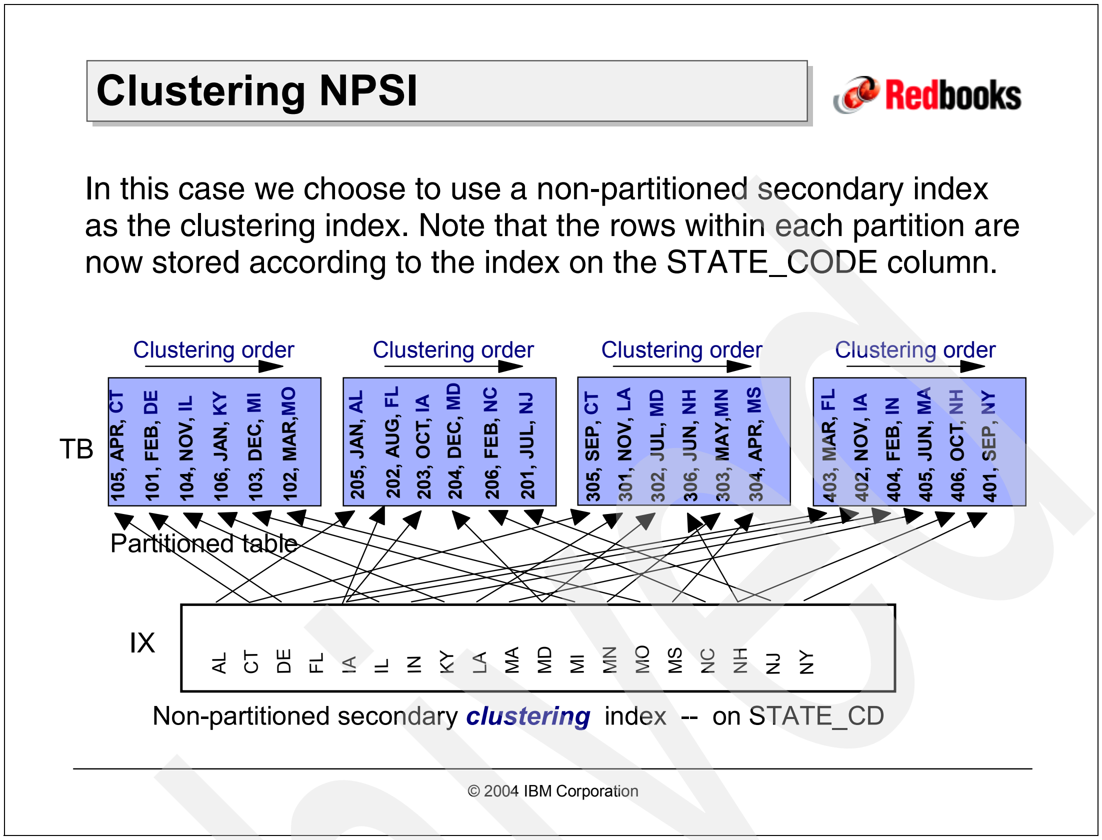
A Clustering Index Can be a Secondary Index

```
CREATE INDEX DPSICLST ON CUSTOMER
    (LAST_ACTIVITY_DT     ASC)
    . . .
    PARTITIONED
    CLUSTER

    OR

CREATE INDEX NPSICLST ON CUSTOMER
    (STATE_CD         ASC)
    CLUSTER
```

NOTE:
DPSI cannot be unique

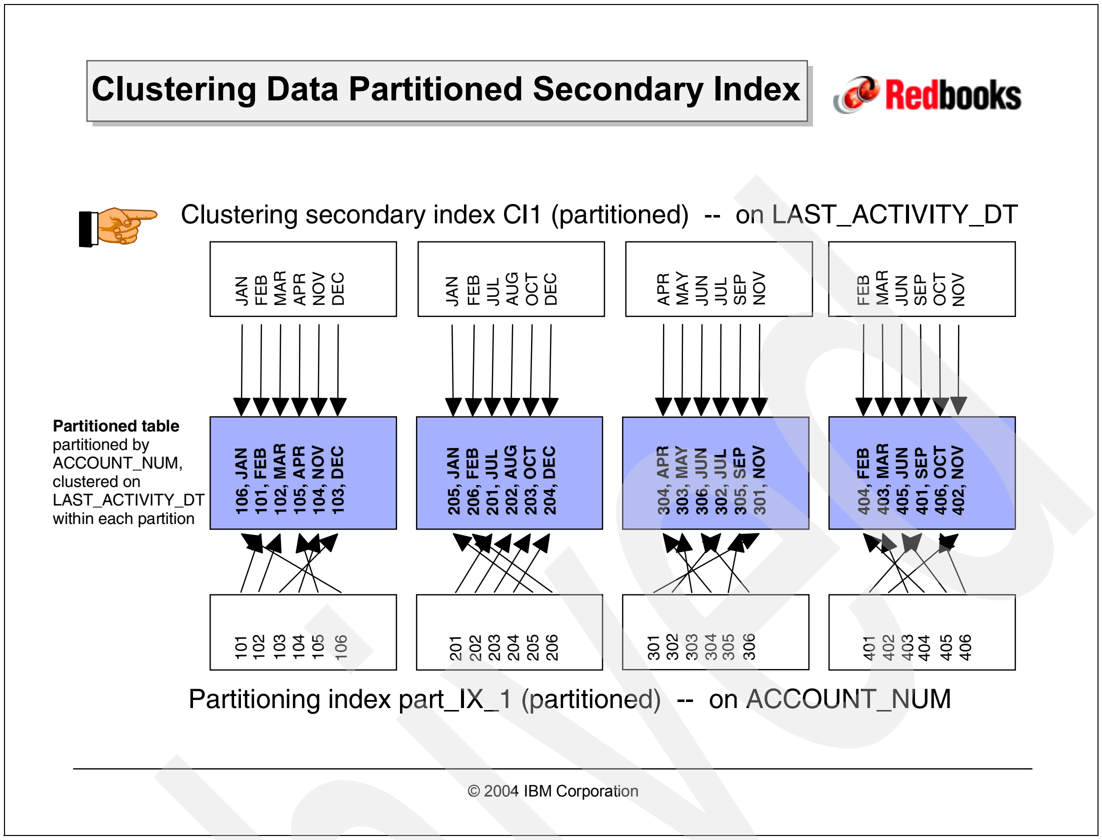
3.10.1 A clustering index can be a secondary index

The clustering index can be any index, you are no longer restricted to the partitioning index having to be the clustering index. This visual shows two possibilities. Index DPSICLST is a data partitioned secondary index and it can be the clustering index. Index NPSICLST is a non-partitioned secondary index and it can be the clustering index. However, only one index may be explicitly defined as the clustering index at any one time. The clustering index may also be unique (when it is not a DPSI, as DSPIs cannot be defined as unique).



3.10.2 Clustering NPSI

In this case we chose to use a non-partitioned secondary index as the clustering index. Note that the rows within each partition are now stored according to the index on the STATE_CODE column.



3.10.3 Clustering DPSI

In this case shown in the figure above, we chose to use a data-partitioned secondary index as the clustering index. Note that the rows within each partition are now stored according to the index on the LAST_ACTIVITY_DT column. The clustering index is based on a secondary index, and therefore by definition different from the partitioning columns. The partitioning columns are only used to determine in which partition to insert the rows; inside each partition the order of the rows is based on the clustering index, a DPSI in this case.

Online Schema Changes

Overview of online schema evolution

Altering Tables

Altering Indexes

Versioning

Partition management enhancements

- Adding, rotating, changing, and rebalancing partitions

Other Schema Changes

- Padded/Not Padded
- Cluster/Not Cluster

Rebuild pending index avoidance

3.11 Online schema changes

In DB2 Version 8 we make great strides to improve data availability. V8 allows you to make a number of schema changes without having to drop and recreate the objects.

We start out with an overview of the things that you normally have to do to implement a change to your schema. We also list the things that were done in previous releases and versions to allow you to make more online schema changes.

DB2 Version 8 allows you to make a number of changes (ALTERs) to tables and indexes without having to drop and recreate the objects (and its dependent objects).

To implement online schema changes, DB2 V8 has implemented a new versioning infrastructure.

V8 also allows you to add partitions to a partitioned table space, as well as rotate (roll-off), and rebalance partitions, without having to drop and recreate the object.

Lastly, we discuss a number of the schema changes, like changing the clustering index, and switching between padded and not padded indexes.

The Availability Story ...

Application maintenance

- Packages, versioning (V2.3)

Code maintenance (installation & PTFs)

- Data Sharing (V4)

Data maintenance

- Online RUNSTATS, COPY, REORG, and LOAD (V3-V7)

Schema maintenance (ALTER, CREATE, DROP)

- Simple ALTER (primary quantity etc.), add column, rename table already supported
- Online schema evolution for tables and indexes (V8)

TO SOLVE THE MAIN AVAILABILITY ISSUE CUSTOMERS FACE TODAY

3.11.1 The availability story

Over the last versions of DB2, significant enhancements have already been implemented to reduce the unavailability window.

Application maintenance

DB2 has implemented packages and package versioning since Version 2 Release 3. By using this versioning technique, you can prepare the DB2 packages for the new version of the application in advance, and once the new application code gets activated, DB2 automatically picks up the new version of the package, without any service interruption.

Code maintenance

Installing a new version of DB2 or applying maintenance (PTFs) normally requires that you stop and start your DB2 subsystem. During that time, your DB2 applications are not available for your customers. With the introduction of DB2 data sharing in Version 4, you can stop and start individual members (DB2 subsystems) to activate maintenance or a new DB2 release, and applications can use other, active, members to run while certain members of the data sharing group are down for maintenance.

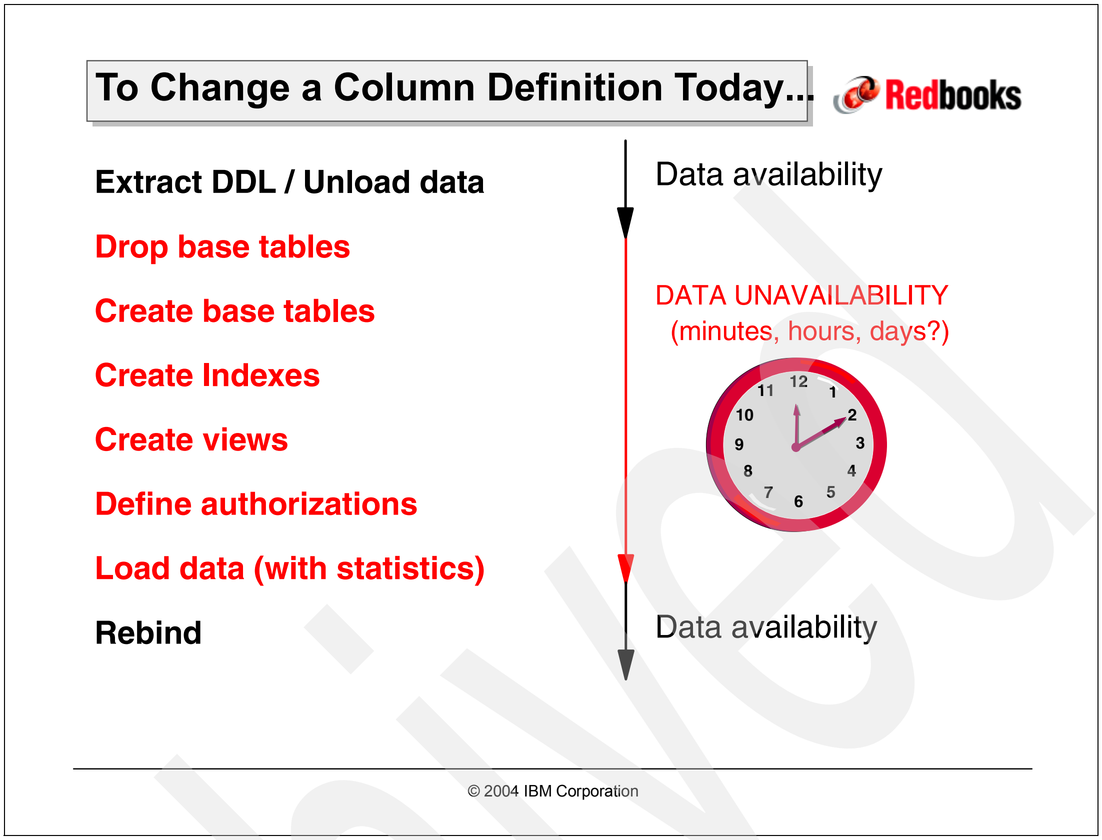
Data maintenance

Another area in which a lot of work has been accomplished is to keep the data available as much as possible. Data requires maintenance every so often. Sometimes data gets disorganized and needs a REORG, sometimes additional data needs to be loaded, and during that maintenance time, the data should be available to the applications as much as possible. DB2 utilities have come a long way over the last releases, for example, by introducing online REORG, inline copy and statistics, and online LOAD RESUME.

Schema maintenance

Starting in Version 8, DB2 takes on a new challenge, that is, to reduce the unavailability window when making changes to the data definition of DB2 objects.

However, it should be noted that V8 does not support schema versioning. DB2 will always convert the rows to the latest table format.



3.11.2 Performing schema changes today (V7)

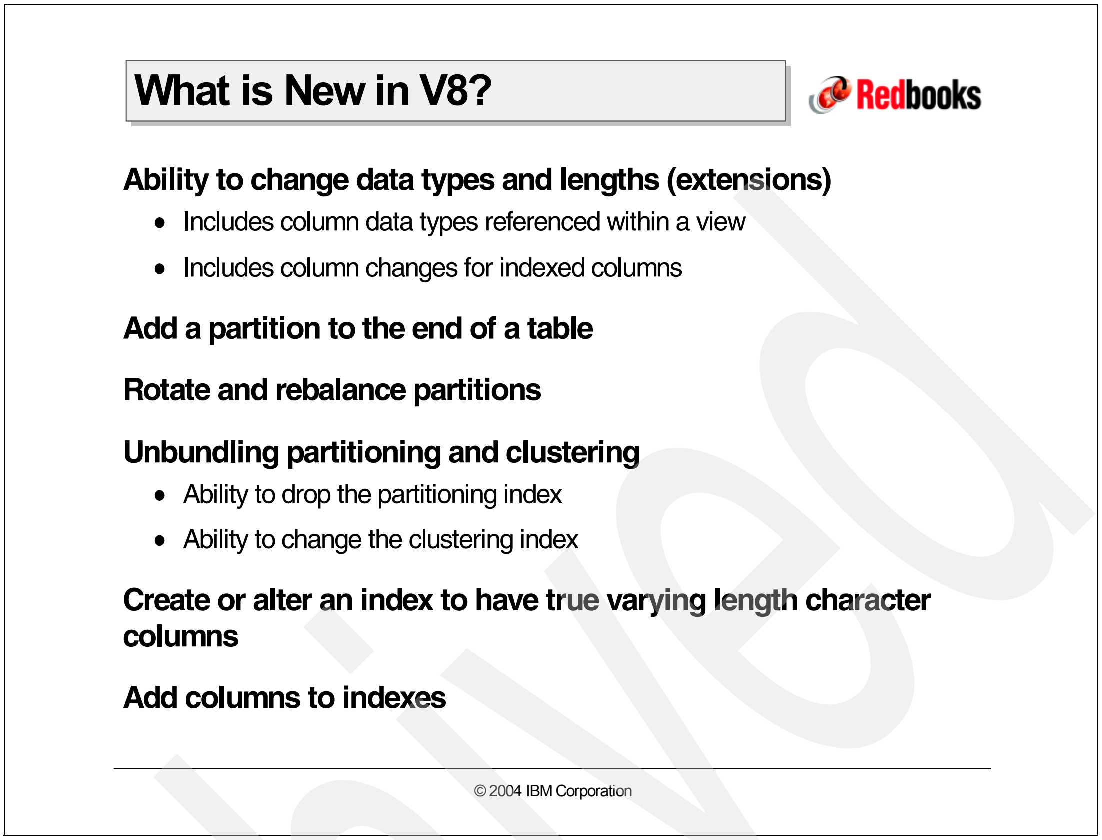
In the past, DB2 releases have implemented most DDL ALTER enhancements without actively addressing the problem of data unavailability while modifying object attributes. Some of these obstacles have also been removed via APARs where the required change was fairly simple. See Table 3-1 for a list of some of the changes:

Table 3-1 Improvements in availability using DDL ALTER

DB2 Version	Improvement
DB2 V1	Add a column to an existing table without any outage.
DB2 V4	Allow changing the PRIQTY, SECQTY, PCTFREE, FREEPAGE, CLOSE and ERASE attributes without taking locks on the SKCT/SKPT.
DB2 V5	Allow new PRIQTY(n), SECQTY(m) specifications without requiring the object to be stopped. The new SECQTY is picked up immediately on the next extension, while the PRIQTY gets picked up when the data sets are deleted and redefined.
	Allow expanding the length of varying length character columns (with or without indexes).
	Allow renaming of tables.
DB2 V6	Allow changing partition boundaries without dropping the table.
DB2 V7	Allow adding and removal of work file table spaces without having to stop the work file database.

Many changes to table, table space, and index schemas (DDL) prior to DB2 Version 8 require that you adhere to the following procedure to implement them:

1. Unload the data, extract the DDL and authorizations of the object you are about to change, and all dependent objects, like tables, indexes, views, synonyms, triggers, etc.
2. Drop the object.
3. Create the object with the new definition and reestablish authorizations for the object.
4. Recreate all dependent objects, such as views and indexes, etc., and their authorizations.
5. Reload the data.
6. Rebind plans and packages.
7. Test that all is OK.



3.11.3 What is new in V8

As 24x7 availability becomes more critical for applications, the need grows for allowing changes to database objects reflected in the catalog and the DBD while minimizing the impact upon availability. We call this *Online Schema Evolution* (or Online Schema Changes or Online Alter).

In an ideal world, this enhancement would provide support for changes to all object attributes without losing availability. DB2 V8 lays the groundwork for allowing many changes, while implementing a reasonable subset of these changes.

The following schema changes are allowed in DB2 V8:

- Extend CHAR(n) column lengths.
- Change type within character data types (CHAR, VARCHAR).
- Change type within numeric data types (SMALLINT, INTEGER, FLOAT, REAL, FLOAT, DOUBLE, DECIMAL), as long as the existing row values can all fit within the range allowed by the new data type.
- Change type for graphic data types (GRAPHIC, VARGRAPHIC).
- Allow column data type changes for columns that are referenced within a view.
- Allow these column changes for columns that are part of an index.
- Add a column to an index.
- Drop the partitioning index (or create a table without one).
- Change the clustering index.

- Create or alter an index to have not padded varying length character columns within a key.
- Allow an alter of identity columns.
- Add a partition to the end of a table which extends the limit value.
- Rotate partitions.
- Support automatic rebalancing of partitions during REORG.
- Loosen the restrictiveness of indexes in Recover or Rebuild Pending.

Key Benefits

Redbooks

Availability!

Avoids wasted space as column lengths can be defined for today's maximums and extended in the future if necessary

Reduces the number of data sets to be managed as partitions can be added when required - particularly important for time series data

Enables rolling partition designs

May reduce the number of indexes required as you may be able to drop partitioning index and define a secondary index as the clustering index

3.11.4 Key benefits

The key benefits to online schema evolution are:

- Availability:

  Changes to the schema can be done without having to make the objects unavailable for extended periods of time in order to drop and redefine them.

- You can avoid wasting disk space:

  Columns can be defined with today's needs in mind, they can be extended in the future as requirements change.

- Reduce the number of data sets to be managed:

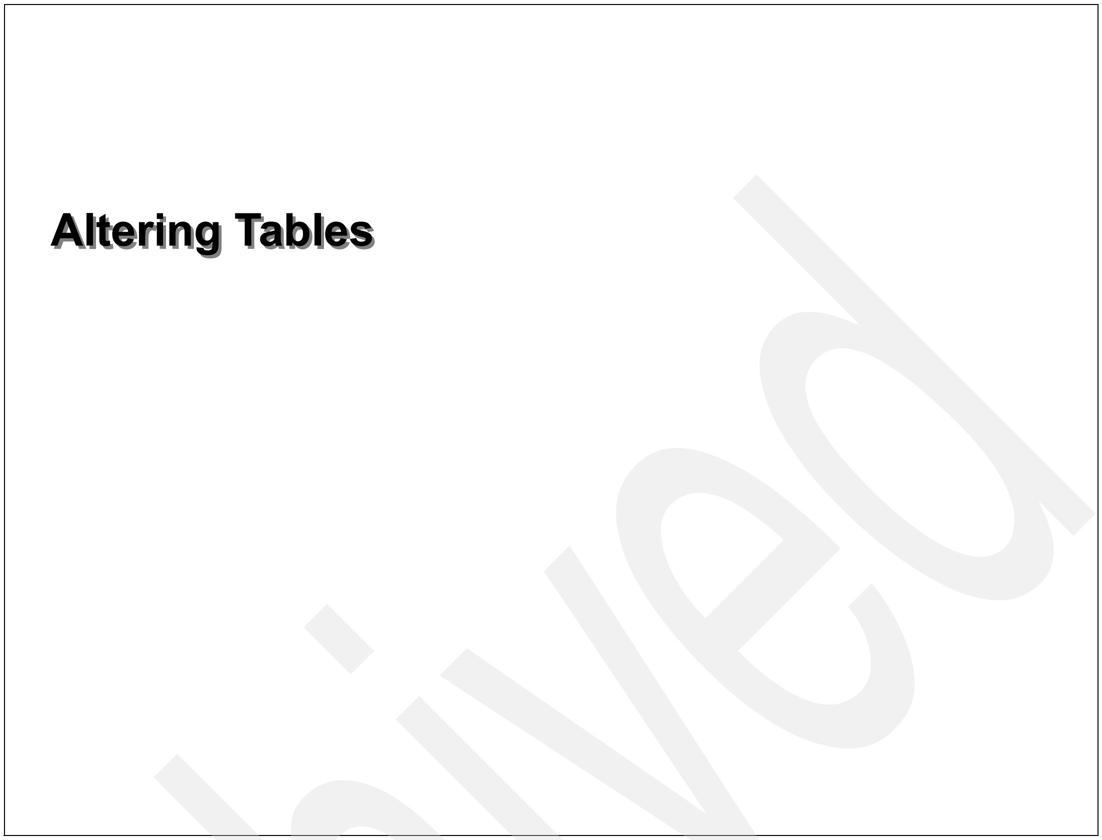
  Partitions can be added as needed.

- Enables rolling partition design:

  You can roll partitions and reuse them for new data ranges.

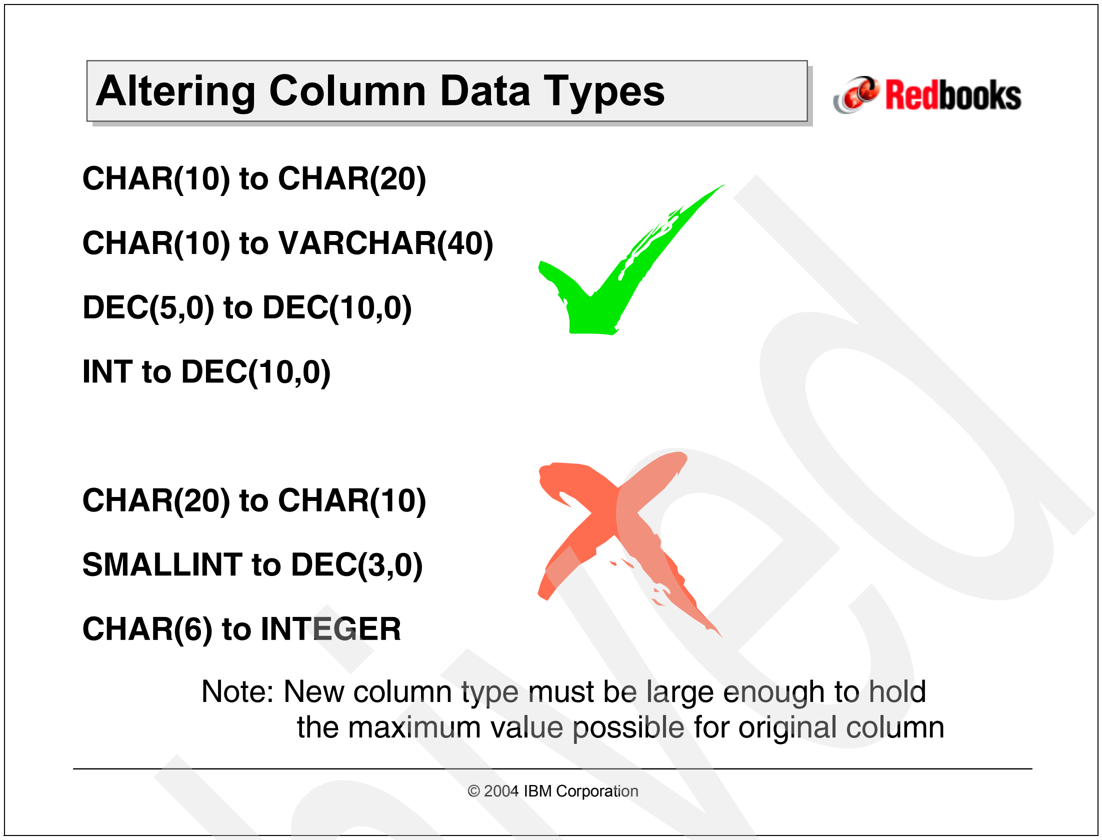
- May reduce the number of indexes required:

  An index is no longer required to be able to partition a table, and a secondary index can now be the clustering index.



3.11.5 Altering tables

First we discuss schema changes related to tables, more specifically data type changes for character and numeric data types.



3.11.6 Altering column data types

Designing databases and objects for applications is more forgiving than in the past. The problem with underestimating the size of objects is lessened with the ability to change column data types without losing availability to the data. Designing applications can give more consideration to saving space and reducing the number of data sets up front without the fear of being locked in at a future point by initial schema decisions.

Supported Alter Data Types

FROM DATA TYPE	TO DATA TYPE
smallint	integer
smallint	float(1-21) or real
smallint	foat(22-53) or double
smallint	>= decimal(5,0)
integer	float(22-53) or double
integer	>=decimal(10,0)
float(1-21) or real	float(22-53) or double
<=decimal(7,s)	float(1-21) or real
<=decimal(15,s)	float(22-53) or double
decimal(p,s)	decimal(p+a,s+b)
char(n)	char(n+x)
char(n)	varchar(n+x)
varchar(n)	char(n+x)
varchar(n)	varchar(n+x)
graphic(n)	graphic(n+x)
graphic(n)	vargraphic(n+x)
vargraphic(n)	vargraphic(n+x)
vargraphic(n)	graphic(n+x)

If a decimal is converted to floating point, column cannot have unique index or a unique constraint

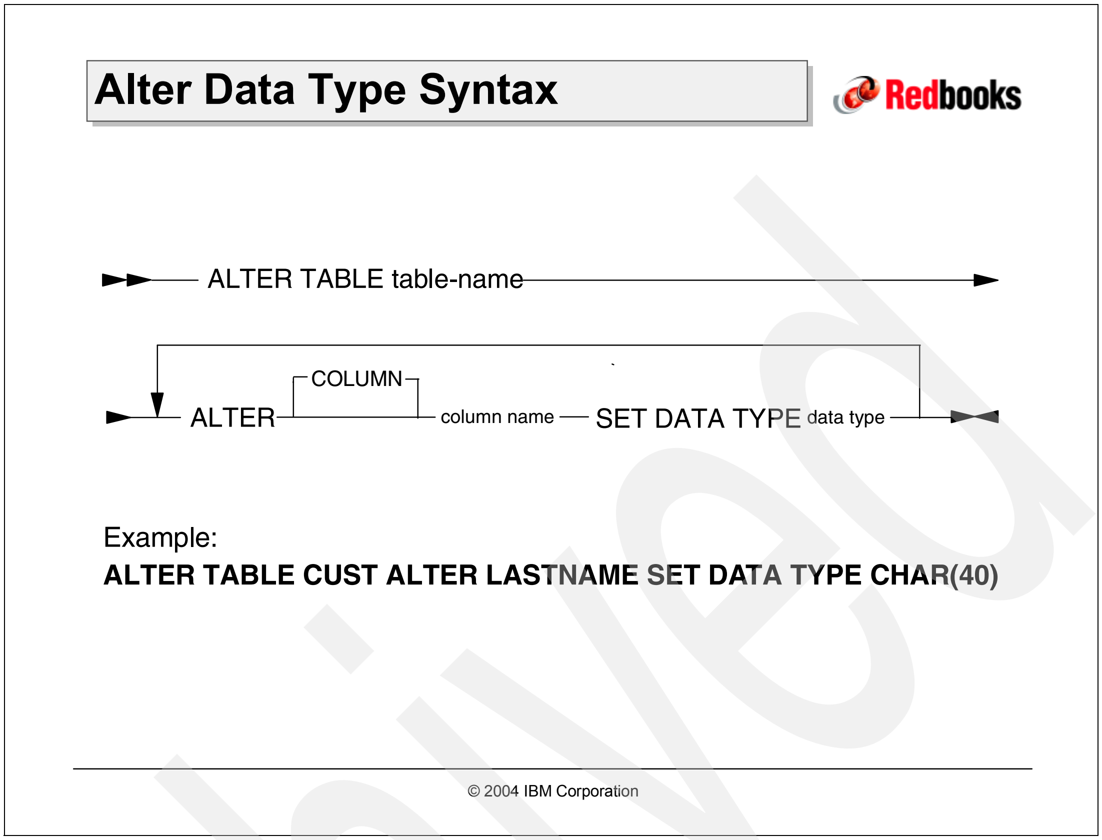
For decimal data types "a" plus "b" must be greater than zero or there is no change

For character data types "x" can be greater than or equal to zero

3.11.7 Supported alter data types

A column data type may be altered if the data can be converted from the old type to the new without losing significance. This basically means that the new column definition has to allow for “larger” values than the current column definition. In V8 you can change numeric and character data types. You can change smallint to int, to decimal, to float, real, or double, as long as the maximum value that you can store in the new data type is greater than the maximum value of the old data type. The same is true for character data. In addition, you can change between CHAR and VARCHAR, and vice versa, as well as between GRAPHIC and VARGRAPHIC, and vice versa.

Note that when a DECIMAL data type is changed to FLOAT, the column cannot be part of a unique constraint. This is because the FLOAT data type is stored as a floating point number, and floating point numbers are approximate numbers. Therefore, two existing decimal values that are different can end up being converted to the same floating point number, which would mean a unique constraint violation.



3.11.8 Alter data type syntax

To support changing the column data type of a column in an existing table, the SET DATA TYPE clause of the ALTER TABLE ALTER COLUMN has been enhanced to support these additional changes. In this visual, we show the syntax and an example of the use of the SET DATA TYPE clause.

What Happens to the Table?

Information about new table layout is stored in:

- Catalog and directory
- New system pages inside the page set

If this is the first ALTER TABLE ALTER COLUMN, the original definition (V0) is captured in the SYSOBDS catalog table

Maximum 255 alters per table space before a REORG is required

Table space is placed in AREO* (advisory REORG-pending)

- Accessible but some performance degradation until REORG

Plans, packages and cached dynamic statements referring to the changed column are invalidated

Runstats values for columns are invalidated

3.11.9 What happens to the table?

Here we discuss some considerations regarding how the tables are handled.

Current definition is captured in catalog and directory

All definitions of the “active” version of the object are stored in the catalog/directory (version 0 (zero)), and in the table space or index space (system pages) itself. Objects become “self-defining”.

Maximum 255 alters per table space before a REORG is required

You can make 255 changes to a table space before you have to REORG it. A change can be multiple alters within one commit scope.

Table space is placed in Advisory Reorg Pending (AREO) state*

When the ALTER completes, the table space is placed in an Advisory Reorg Pending (AREO*) state (see 3.15, “Reviewing DBET states used by online schema evolution” on page 189 for more details on this new database exception state). Access to a table (SELECT, INSERT, UPDATE and DELETE) can continue with columns containing data in multiple version formats, but there is a performance degradation until the table space is reorganized. To reduce the performance impact, it is recommended to schedule a REORG after executing the ALTER statement that changes the data type of a column in a table.

The principal reasons for the performance degradation are:

- Disablement of the “fast” column processing occurs.
- The altered columns will have to be converted to the new format when they are accessed.

Plans, packages, and cached dynamic statements

Plans, packages, and cached dynamic statements referencing the changed table are invalidated. If auto-rebind is enabled, the plans and packages referencing the changed table space are automatically rebound during the next access if not manually rebound before that time.

RUNSTATS

Note that after altering the column's data type or column length, the RUNSTATS values in some catalog tables for that column become unusable. Therefore it is a good idea to these statistics again after changing the data type of a column. More precisely:

- The statistics found in SYSCOLUMNS are converted to the new data type when the column is altered. So HIGH2KEY, LOW2KEY and COLCARDF is still usable.
- Cardinality statistics (TYPE 'C' statistics) in SYSCOLDIST remain usable.
- Single and multi-column frequency statistics (TYPE 'F' statistics) are no longer usable.

To indicate that some statistics may have become unusable, the STATSTIME column in the following catalog tables is updated to '0001-01-**02**-00.00.00.000000':

- SYSIBM.SYSCOLSTATS
- SYSIBM.SYSCOLUMNS
- SYSIBM.SYSCOLUMNS_HIST

What Happens to the Data?

Existing data remains unchanged

On SELECT, data will be materialized to the latest format

On INSERT / UPDATE, the entire row will be changed to latest format

Reorg changes all rows to the latest format

3.11.10 What happens to the data?

After a table column data type is changed (via an ALTER statement), the new definition *immediately* applies for all data in the associated table. No existing data is converted to the new version format.

When rows are retrieved, they are materialized in the new format indicated by the catalog and system pages contained within the object. Likewise, when a data row is modified or inserted, the entire row is saved using the new definition.

When the object is reorganized, all rows are converted into the format of the latest definition (see 3.12, “Versioning” on page 156 for details).

What Happens to Dependent Indexes?

Redbooks

New index version created for each index that references altered column

- Up to 16 versions per index

Version information stored in system (version) pages in the index

- Like in V7 after ALTER TABLE ALTER COLUMN to extend a varchar column

Immediate access for character data type extensions

- Index placed in AREO*

Delayed access for numeric data types

- Index placed in rebuild pending (RBDP) to avoid severely degraded performance
- Static plans/packages accessing the index are invalidated
  - Auto rebind may choose different access path
- Dynamic queries will avoid RBDP indexes
- Deletes are allowed
- Updates and inserts are allowed for non-unique indexes

3.11.11 What happens to dependent indexes?

When the data type or length of a column is altered on a table and that column is defined in an index, the index is altered accordingly.

If a table has multiple indexes, a change for a table column results in a new table version and a new index version for each index that contains the column. Indexes created on different tables in the same table space or unchanged columns in the same table are not affected. If a change is made to a non-indexed column, it results in a new table (and table space) version but not a new index version.

All new keys inserted are in the new index format.

If an entire index is rebuilt from the data, all the keys are converted to the latest format. Utilities which may rebuild an entire index include:

- REBUILD INDEX
- REORG TABLESPACE
- LOAD REPLACE

If data type changes, reorganization of the entire index (REORG INDEX) materializes all keys to the format of the latest version.

Whether or not the index is immediately available after a column in the index incurred a data type change, depends on the data type of the column being changed.

Immediate index availability

In DB2 V5 and V6, the ALTER TABLE statement was enhanced to provide the ability to increase the length of VARCHAR columns. If an index on the altered table had a key containing altered columns, index versioning support allowed immediate access to the index. This index versioning design is extended to support immediate access of indexes containing keys from all forms of fixed length or varying length character and graphic columns.

So in V8, all changes for character data type columns that are also part of an index, result in immediate index availability. This includes columns defined as CHAR, VARCHAR, GRAPHIC, or VARGRAPHIC. The index is placed in AREO* state, to indicate that a REORG is recommended.

Delayed index availability

In some cases, supporting immediate changes with index versioning would result in severely degraded performance. To avoid this, the index is placed into Rebuild Pending (RBDP) instead (for both COPY NO and COPY YES indexes). Availability to the index is delayed until the index is rebuilt.

Therefore in V8, changes for numeric data type columns that are used in an index, are immediate with delayed index availability. This includes columns defined as SMALLINT, INTEGER, DECIMAL or NUMERIC, FLOAT, REAL, or DOUBLE.

Limiting the scope of the unavailability for dynamic SQL for RBDP index

To limit the scope of the unavailability of the data for dynamic SQL when the index is in RBDP state, in V8:

- Deletes are allowed for table rows, even if there are indexes in RBDP
- Updates and inserts are allowed for table rows, even if their corresponding non-unique indexes are in RBDP state
- Inserting or updating data rows which result in inserting keys into an index that is in RBDP state is disallowed for unique or unique where not null indexes
- For dynamic SQL queries, DB2 does not choose an index in RBDP for an access path

And Referencing Views and Check Constraints ?

Redbooks

Views referencing affected columns are regenerated

- Ordered regeneration, underlying view done first
- Errors during regeneration causes the ALTER to rollback
  - ALTER VIEW *viewname* REGENERATE - limited usage only
- All plans, packages, and dynamic cached statements referencing the view are invalidated
- If there are a large number of views, you may want to consider recreating views manually for performance reasons

Check constraints dependent on the column being altered are regenerated

Triggers dependent on the column being altered are also handled

3.11.12 How about referencing views and check constraints?

When a column is altered in a base table, the views that reference the column are immediately regenerated. If one of the views cannot be regenerated, then the ALTER TABLE statement fails on the first error encountered.

A change to any column within a view invalidates all plans, packages, and dynamic cached statements that are dependent on that view.

When a column data type is altered, the precision and scale of the decimal arithmetic result needs to be recalculated. The value of the CURRENT PRECISION special register that is in effect for the ALTER TABLE is used to regenerate all the views affected by the altered column. Since a single CURRENT PRECISION setting is used for all the views, it is possible the ALTER TABLE can fail with an SQLCODE -419 or complete with a precision calculated for view columns that does not work well for an application. In this case the user has to DROP and CREATE the view in order to correct the problem.

If an ALTER TABLE fails because of a problem regenerating a view, the failing SQLCODE and tokens identifying which ALTER failed is returned and the entire ALTER TABLE statement fails.

If a check constraint is dependent on the column being altered, it is also “regenerated”. The regeneration may also fail in the case where different options are in use during the regeneration than the options in use at the time the check constraint was created. The options are the decimal point indicator and quote delimiter. The failing SQLCODE and tokens identifying which ALTER failed are returned.

V8 also adds a new DDL statement ALTER VIEW *viewname* REGENERATE. This statement has two purposes:

- Regeneration of views which have been marked invalid during catalog migration. A view is considered as invalid when column STATUS in catalog table SYSIBM.SYSTABLES contains an 'R', and the TABLESTATUS column a ‘V’. During catalog migration, DB2 continues processing even when a view regeneration fails. DB2 does not want to stop the migration process because a single view on the catalog is no longer usable. You can use the ALTER VIEW name REGENERATE statement to do those views manually after the migration is complete.
- To have DB2 update the stored form of the view to the new V8 structure — rather than doing this on the fly each time it is used.

When there is a trigger defined on a column that is changed, that is also automatically handled by the system.

Considerations When Changing Data Types

Before doing the ALTER, assess what programs need to change

- Host variables may need to be extended to cater for extra length
- Host variable numeric data types may need to change to match new column data type

Schedule REORG soon after the ALTER to minimize performance degradation

Rebuild any affected indexes and rebind plans/packages

- After index rebuild, statements in the dynamic statement cache referencing the table are invalidated, so they can pick up the IX
- Bind or ensure that auto rebind occurs for static plans/packages

Schedule RUNSTATS to repopulate column and index statistics

Alter table add column still handled the "old" way when at V0

3.11.13 Considerations when changing data types

When making schema changes, applications are usually affected. Changes in the schema must be closely coordinated between database objects and applications to avoid “breaking” existing applications. For example, if a column is extended from CHAR(n) to CHAR(n+m), the processing application truncates the last m bytes if the application is not changed to handle the longer column. With this feature, be sure to assess which programs need to change.

The creation of new versions for objects can degrade performance of existing access paths. Schema changes should be planned to balance the trade-off between performance and availability expectations within a customer environment. Typically, the best time to make schema changes to minimize the poor performance impact is before a scheduled reorganization of an object.

Rebuild any affected indexes and rebind plans and packages for applications using static SQL. This will prevent the system from picking an inefficient access path during automatic rebind because the best suited index is currently not available. When the index is rebuilt and the RBDP status is reset, statements in the dynamic statement cache related to the table for which the index was rebuilt are automatically invalidated, and the next time the statement is executed, a new access path is determined, and can now pick the index that was rebuilt.

Schedule RUNSTATS to repopulate the catalog with accurate column and index statistics.

Ever since V1 of DB2, you have been able to add a (nullable) column to an existing table, without having to drop and recreate the object; online schema evolution before its time. When you add a column to an existing table in Version 8, this change will not create a new version if this is the first alter on the table, in other words, if the table is still at version 0.

If you have executed a version-generating ALTER on that table in the past (in other words, the table is not at V0), then adding a column to an existing table will create a version.

Restrictions

Data types must be compatible and lengths must be the same or longer

Disallowed for ROWID, LOB, DATE, TIME, TIMESTAMP columns or a distinct type

Data types and lengths cannot be altered when

- Column is part of a referential constraint or has a FIELDPROC
- An EDITPROC or VALIDPROC exists on the table
- Part of a materialized query table
- The column is defined as an identity column or seclabel

Default values and check constraints are handled

V8 does NOT support schema versioning (or recovery other than a point-in-time recovery of the catalog)

3.11.14 Restrictions

DB2 V8 takes the first steps to avoid outages due to schema changes. However, certain restrictions are still in place, such as these:

- Data types must be compatible and lengths must be the same or longer.
- Online schema changes are not allowed on columns that are defined as ROWID, LOB, DATE, TIME, TIMESTAMP, as well as columns that use a distinct type.
- Data type and lengths cannot be altered when:
  - The column is part of a referential constraint.
  - The column has a FIELDPROC defined.
  - An EDITPROC or VALIDPROC exists on the table.
  - The table is used in a materialized query table (if the table you are ALTERing is a materialized query table, or a materialized query table is defined on the table you are altering), irrespective of whether or not the column you are altering is part of the SELECT clause that makes up the MQT.
  - The column is defined as an identity column.
  - The column is defined as a security label column.

Operational Impact

Table space and indexes will contain data in multiple version formats until REORG or LOAD REPLACE or REBUILD (for IX)

Recovering data

- Whether to current or PIT, there is no problem, as image copy and/or SYSOBDS contain version information
- Log processing will insert and update data to new format
- Just restoring data from a previous version

Note: It is recovery of data rather than schema

Moving data using off-line utilities (for example, DSN1COPY) may require extra steps

- REPAIR VERSIONS
- Recommendation: Keep track of DDL history of your tables when moving data using off-line utilities

3.11.15 Operational impact

SYSODBS only contains the original version (V0), the one prior to the first new version. Table space and indexes contain system pages with all changed definitions. REORGs and MODIFYs must be used to convert all the data to the latest definition.

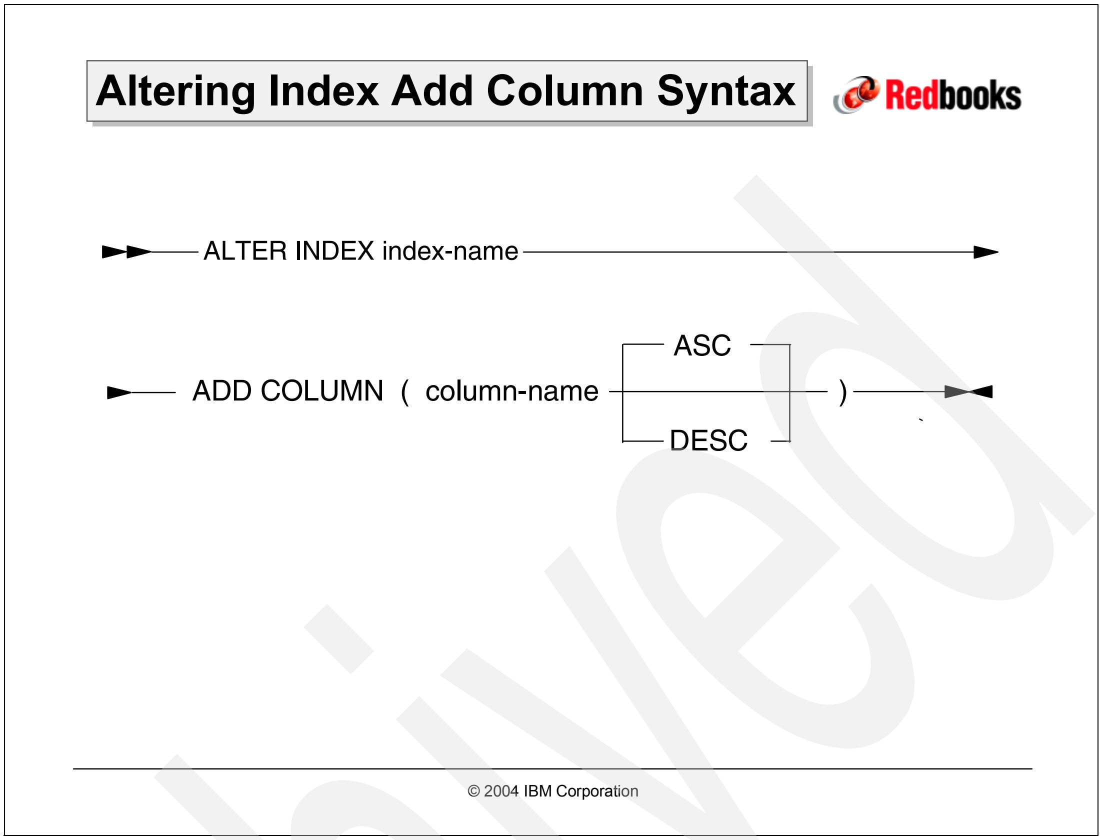
Recovery of data rather than schema means that the data is recovered to a point-in-time (PIT); you do not recover the schema to a point-in-time.

Moving data between objects or subsystems using DSN1COPY is more complicated. You need to execute the REPAIR utility using the VERSIONS keyword to resync the catalog information with the data. As before, as a "golden rule", you need to track the DDL history of your tables, because you may have to perform different actions depending on different sequences of DDL statements, when using offline utilities to move the data to a different table. For more information about the use of REPAIR VERSIONS, see 9.10.2, "REPAIR - use of versions" on page 671.

Altering Index Attributes

3.11.16 Altering index attributes

In Version 8, some index attributes can also be changed ‘in-flight’ using an ALTER INDEX statement, without causing the index to become unavailable.



3.11.17 Altering index add column syntax

In DB2 V8, the syntax of the ALTER INDEX statement is changed to allow for the addition of columns to the end of an index, for example:

```
ALTER INDEX CUST_IDX ADD COLUMN NEW_COL ASC;
```

Alter Index Add Column

Ability to add a column to the end of an index

When the column preexists in the table, the index is placed in RBDP

If it is a new column, add it to the table and index in the same UOW, then the index will be placed in AREO* instead

Examples:

```
ALTER TABLE CUST ADD COLUMN NEW_COL;
ALTER INDEX CUST_IDX ADD COLUMN NEW_COL ASC;
COMMIT;
```

Immediate availability! (Index placed in AREO*)

```
ALTER TABLE CUST ADD COLUMN NEW_COL;
COMMIT;
ALTER INDEX CUST_IDX ADD COLUMN NEW_COL ASC;
COMMIT;
```

Delayed availability! (Index placed in RBDP)

3.11.18 Alter index add column

Columns can now be appended to the end of an existing index key with the ALTER INDEX statement.

- If the index is not defined (created with the DEFINE NO keyword), no restricted state is set and a new index version is not created.
- If the index is defined and the column is added to the table in the same unit of work that the column is also added to the index, the index is immediately available for access, and the index is placed in Advisory Reorg Pending (AREO*) state.
- However, if the column was not added to the table in the same unit of work, the index is placed in a Rebuild Pending state (RBDP) and a new index version is generated.

This support allows maximum availability for the situations where new columns are added to a table, and these new columns are also desired as part of an existing index. By making changes in one unit of work, there is no loss of availability. The alternative is to drop the index, and then create a new index with the column. When creating a new index, there is always a period of unavailability while the index is being created.

Restrictions

Cannot exceed 64 columns in an index

Length maximum

- 2000 - n for padded, where n is the number of nullable columns
- 2000 - n - 2m, where m is the number of varying length columns

Disallowed for

- System defined indexes (indexes on Catalog and Directory)
- Partitioning indexes of index-based partitioned table spaces
- Indexes enforcing a primary or unique constraint or referential constraint
- A unique index required for a ROWID column defined as GENERATED BY DEFAULT
- An auxiliary index

3.11.19 Restrictions

This visual shows the restrictions in for adding columns to the end of an index. You cannot exceed a total of 64 columns per index. The total length of the index key columns cannot exceed 2000 bytes minus the number of nullable columns and minus 2 times the number of varying length columns in the index.

You may not alter add columns to the end of an index for the following indexes:

- An index that is a system-defined catalog index
- An index that enforces a primary key, unique key, or referential constraint
- A partitioning index when index-controlled partitioning is being used
- A unique index required for a ROWID column defined as GENERATED BY DEFAULT
- An auxiliary index

RBDP Considerations

Redbooks

Index must be rebuilt - reorg will not suffice

Dynamic statement cache

- Changes which result in indexes being placed in RBDP will flush dynamic statement cache

Once the index is available again, revert to optimal access paths by

- Rebinding affected plans/packages for static SQL
- Automatic invalidation of related statements in the dynamic statement cache when RBDP is reset.
  At next execution, statement is prepared again, and can go back to original access path
- Manual invalidation of DSC can be done using
  RUNSTATS UPDATE NONE REPORT NO

3.11.20 RBDP considerations

There are a few things to consider when an index is placed in a Rebuild Pending (RBDP) state:

- Indexes in RBDP must be rebuilt using the REBUILD INDEX utility; a REORG will not remove the RBDP state.
- Changes which result in indexes being placed in RBDP will flush any statements in the dynamic statement cache that use those indexes. Static plans/packages are not invalidated. For non-unique indexes, static plans/packages that do not use the index to retrieve data will continue to function as before. For unique indexes placed in RBDP, those plans/packages that are dependent on them will get a -904 (unavailable resource) on an update of an existing row or insert of a new row. Deletes are no problem.
- Once the index has been rebuilt, you should rebind the affected plans/packages and you should invalidate your dynamic statement cache. In V8, you can invalidate the dynamic statement cache by running RUNSTATS with **`UPDATE NONE REPORT NO`** keywords.

Versioning

Up to 256 active versions for table space, up to 16 for indexes

- Active versions include those within page set and all available image copies
- Unaltered objects remain at version 0
- If maximum reached, alter fails with SQLCODE -4702

If you have a large number of ALTERs:

- Make several ALTERs to an object within a single unit of work (counts as one)
  - You cannot have an ALTER and DML in the same unit of work SQLCODE -910
- REORG and MODIFY before you reach the maximum of 256 versions per table space
  - MODIFY to delete image copies
- "Rebuild" index before you reach the maximum of 16 versions per index space

3.12 Versioning

To support online schema evolution, DB2 has implemented a new architecture to track object definitions at different times during its life by using versions.

Altering existing objects may result in a new format for tables, table spaces, or indexes that indicates how the data should be stored and used. Since all the data for an object and its image copies cannot be changed immediately to match the format of the latest version, support for migrating the data over time is implemented by using versions of tables and indexes. This allows data access, index access, recovery to current, and recovery to a point-in-time while maximizing data availability.

Versioning existed before DB2 V8 for indexes (after an indexed VARCHAR column in a table had been enlarged). It was tracked using the IOFACTOR column of SYSINDEXES. In DB2 V8, the first ALTER that creates a new index version switches to DB2 V8 versioning by setting the OLDEST_VERSION and CURRENT_VERSION columns to the existing versions in the index. And to support the table data type changes mentioned before, versioning in Version 8 is also implemented for tables and table spaces.

Version generating ALTER statements

The following statements result in a new version for the affected tables and/or indexes:

```
ALTER TABLE table-name ALTER COLUMN column-name SET DATA TYPE altered-data-type
ALTER INDEX index-name ADD COLUMN column-name
```

Multiple ALTER COLUMN SET DATA TYPE statements in the same unit of work on the same object are included in one new schema version.

The following ALTER statements do not result in a new version:

```
ALTER TABLE table-name ADD PARTITION ENDING AT constant
ALTER TABLE table-name ALTER PARTITION n ENDING AT constant
ALTER TABLE table-name ROTATE PARTITION FIRST TO LAST
ALTER TABLE table-name ADD PARTITION BY RANGE (column-name)
ALTER INDEX index-name NOT CLUSTER
ALTER INDEX index-name CLUSTER
ALTER INDEX index-name NOT PADDED
ALTER INDEX index-name PADDED
```

The following cases also do not generate a new version:

- When the table space or index was created as DEFINE NO and contains no data.
- When a varying character or varying graphic column length is extended.
- When an ALTER TABLE specifies the same data type and length so the definition is not changed.
- ALTER TABLE ADD COLUMN. This feature is available in V7 and will continue to work the same way as it does in V7, as long as you have not executed any other version generating SQL statement. If your table is no longer at V0 when you do the alter, ALTER TABLE ADD COLUMN will also generate a version.

Version limits

A table space can have up to 256 different active versions (versions 0-255) while an index can have up to 16 different active versions (versions 0-15). Active versions include those within the pageset and all available image copies (in SYSCOPY).

The range of *active versions* is all versions that exist for rows in the page set itself as well as the versions that exist in image copies registered in SYSCOPY. If the maximum number of active versions is reached, the SQL statement fails with:

```
DSNT408I SQLCODE = -4702, ERROR: THE MAXIMUM NUMBER OF ALTERS ALLOWED HAS BEEN
EXCEEDED FOR TABLE
```

Unaltered objects remain at version 0 (zero).

Storing version information

The version information is stored in the DB2 catalog as well as inside the page set in what we call system pages.

Catalog Support for Versioning

Oldest for pageset and all available copies - updated by utilities such as MODIFY and REORG

Used to allocate next version number

Current version number of the table or index

	OLDEST_VERSION	CURRENT_VERSION	VERSION	VERSION "0" DATA
SYSTABLESPACE	X	X		
SYSTABLEPART	X			
SYSTABLES			X	
SYSINDEXES	X	X	X (data version)	
SYSINDEXPART	X			
SYSCOPY	X			
SYSOBDS				X

Note that versioning is tracked at table space and index level

- Each change to a table in a segmented table space will be assigned the next available version number based on CURRENT_VERSION number in SYSTABLESPACE

3.12.1 Catalog support for versioning

As can be seen in the visual above, versioning information for an object is kept in the catalog tables SYSIBM.SYSTABLESPACE, SYSIBM.SYSTABLEPART, SYSIBM.SYSINDEXES, SYSIBM.SYSINDEXPART, SYSIBM.SYSTABLES, and SYSIBM.SYSCOPY.

In addition, the new catalog table SYSIBM.SYSOBDS contains one row for each table space OBD or index that can be recovered to an image copy and that has more than one active version. Only the first active version (the definition of the object when it was created — Version 0 (zero)) of the OBD is placed in SYSODBS, the records are cleaned up when version numbers are consolidated, and active versions are reduced to only one active version.

A table space starts out with all data in tables at version zero. When an ALTER creates a new version, it gets the next available number after the active table space CURRENT_VERSION. Once version 255 is reached, numbering starts again with version 1 if it can be reclaimed. A version of 0 indicates that a version creating ALTER statement has never been issued for the corresponding table or table space.

Versioning information inside the page set

The version information relevant to the data is stored inside the page set in system pages. Storing the version information inside the page set makes the objects self-defining. The system pages contain one record for each active version of the page set. (To make sure that system pages are included in *incremental* image copies, you can specify the new SYSTEMPAGES YES keyword.)

Minimize Number of Active Versions

REORG TABLESPACE and LOAD REPLACE materialize data and keys to latest version

- When inline COPY is used, a row is inserted with the most recent (CURRENT_VERSION) version number in the OLDEST_VERSION column in SYSCOPY
- REORG sets all tables with non-zero versions in a segmented table space to the current table space version
- Deletes obsolete version information in system pages

For COPY YES indexes, REBUILD INDEX materializes the index keys

For COPY NO indexes, REORG, REBUILD or LOAD materialize the index keys, AND update the OLDEST_VERSION in the catalog

New versions are not created in the following circumstances

- If object is specified as DEFINE NO
- For NOT PADDED indexes with VARCHAR/VARGRAPHIC extensions
- When ALTER TABLE specifies existing data type and length so there is no change
- ALTER INDEX ... CLUSTER / NOT CLUSTER
- ALTER INDEX ... PADDED / NOT PADDED
- ALTER TABLE ADD COLUMN of a version 0 table

3.12.2 Minimize the number of active versions

To reduce the number of versions, you must first materialize all the rows to the latest version format. Depending on the object type (table space and COPY YES index) you can use REORG, LOAD REPLACE, or REBUILD INDEX to achieve this.

Then you can run the MODIFY utility to update the version information in the catalog. For table spaces, and indexes defined as COPY YES, the MODIFY utility must be run to update OLDEST_VERSION for either SYSTABLEPART and SYSTABLESPACE, or SYSINDEXPART and SYSINDEXES. If there are COPY, REORG, or REPAIR VERSIONS SYSCOPY entries (ICTYPE of “blank” and STYPE of “V”) for the table space, MODIFY updates OLDEST_VERSION to be the lowest value of OLDEST_VERSION found from matching SYSCOPY rows. If no SYSCOPY rows remain for the object, MODIFY sets OLDEST_VERSION to the lowest version data row or key that exists in the active pageset.

For indexes defined as COPY NO (those indexes do not store information in SYSIBM.SYSCOPY), a REORG, REBUILD, or LOAD utility that resets the entire index before adding keys, updates OLDEST_VERSION in SYSIBM.SYSINDEXES to be the same as CURRENT_VERSION.

When rebuilding an entire index (partitioned or non-partitioned) defined as COPY NO, REBUILD updates the new catalog columns SYSINDEXES.OLDEST_VERSION and SYSINDEXPART.OLDEST_VERSION to the value of SYSINDEXES.CURRENT_VERSION.

If the index is partitioned, REBUILD INDEX also updates SYSINDEXES.OLDEST_VERSION if the lowest OLDEST_VERSION from all SYSINDEXPART entries has changed.

The SYSCOPY records inserted by COPY utility have the OLDEST_VERSION column filled in with the lowest version of data within the copied object.

Reclaiming Versions

Redbooks

Run MODIFY RECOVERY for table spaces and COPY YES indexes

Deletes any unwanted SYSCOPY entries

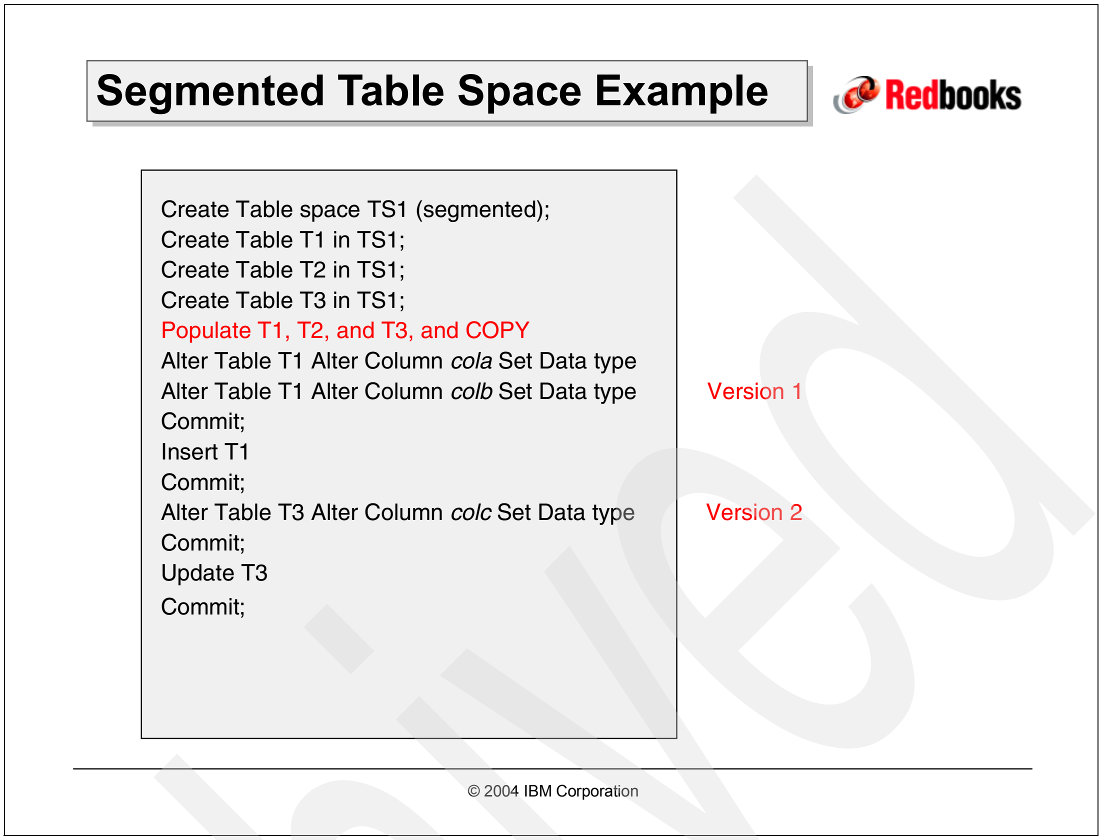
Determines oldest version in page set and related SYSCOPY entries and updates the catalog

- Updates SYSTABLEPART / SYSINDEXPART
- For nonpartitioned, also updates SYSTABLESPACE / SYSINDEXES
- For partitioned, updates SYSTABLESPACE / SYSINDEXES if oldest version across all the partitions has changed

3.12.3 Reclaiming versions

When reorganizing a partitioned table space that has indexes defined as COPY NO, REORG updates the new catalog column SYSINDEXPART.OLDEST_VERSION to the value of SYSINDEXES.CURRENT_VERSION for those indexes (this is not done for non-partitioned indexes unless reorganizing the whole table space). If the index is partitioned, REORG also updates SYSINDEXES.OLDEST_VERSION if the oldest OLDEST_VERSION from all SYSINDEXPART entries has changed. For non-partitioned table spaces, REORG also updates SYSINDEXES.OLDEST_VERSION to the same value.

The REORG TABLESPACE utility resets table spaces and all indexes which are in AREO* state. SYSCOPY records with an ICTYPE value of 'X' (for REORG LOG YES) or 'W' (for REORG LOG NO) and an STYPE value of 'A' are inserted for each data partition where REORP is reset. SYSCOPY records include the current version number (in column OLDEST_VERSION), at the time of reorganization. The REORG TABLESPACE utility inserts SYSCOPY records for each index that is built as part of the reorganization.



3.12.4 Segmented table space example

Let us take a look at a segmented table space and how versioning is tracked. Assume that we have a table space TS1 and three tables (T1, T2 and T3) and that we have populated these tables and taken an image copy of the table space. We then do two alters to change two of the columns of table T1, and we commit the schema change in one commit scope.

At this point we have generated version 1 of TS1. Applications continue to update, insert, and delete rows from T1. We then decide that we also need to alter a column in table T3, and we commit that schema change. At this point we have generated version 2 of TS1. Applications continue to update, insert, and delete rows from T3.

Version Status

SYSTABLES NAME	VERSION
T1	1
T2	0
T3	2

SYSTABLESPACE NAME	OLDEST_ VERSION	CURRENT_ VERSION
TS1	0	2

SYSTABLEPART TSNAME	OLDEST_ VERSION
TS1	0

SYSCOPY TSNAME	TIMESTAMP	ICTYPE	OLDEST_ VERSION
TS1	1	F (FULL)	0

Version status

Here we can see the details of the operations performed in the previous visual. SYSCOPY will contain an entry for the image copy that we created prior to doing the alters. Column OLDEST_VERSION of the SYSCOPY row will contain a '0' since the oldest version of the data is the original version. Once the alters are done, in SYSTABLES, table T1 will have a value of 1, T3 will have a value of 2, and T2 will continue to have a value of 0 for column VERSION since it was not altered is still at its original definition.

Additionally, SYSTABLESPACE will also reflect the changes we have made. Column OLDEST_VERSION will contain 0 since we still have data at the original version and an image copy with data in the original version. Since this is not a partitioned table space, there is only one row in SYSTABLEPART and it will still have a value of 0 for column OLDEST_VERSION.

So, at this point we have three active versions, that is, the data is in any of three forms: Version 0 format, which is the original definition before any of the alters are done; version 1 format, which is the definition after the first set of alters (done on the same commit scope); and version 2 format, which is the definition after the second alter (second commit scope).

All three tables contain rows in the original format (version 0), since we loaded data before any alters were done. In addition, since we inserted a new row into table T1 right after the first version was generated (first ALTER), table T1 will contain data in version 1.

Version Status Following a REORG

Redbooks

SYSTABLES NAME	VERSION
T1	2
T2	0
T3	2

SYSTABLESPACE NAME	OLDEST_ VERSION	CURRENT_ VERSION
TS1	0	2

SYSTABLEPART TSNAME	OLDEST_ VERSION
TS1	0

Table versions have been collapsed as all data has been materialized

SYSCOPY TSNAME	TIMESTAMP	ICTYPE	OLDEST_ VERSION
TS1	1	F (FULL)	0
TS1	2	W (REORG)	2
TS1	2	F (FULL INLINE)	2

Version status following a REORG

Now let us see what happens after we run a REORG with an inline image copy on table space TS1. The REORG reformats all the data to the latest definition (or version). Table T1 in SYSTABLES is updated to reflect the fact that it is now at the latest version (version 2).

Note that table T2 remains at version 0 since no alters were performed against it. In SYSTABLESPACE, the OLDEST_VERSION for TS1 remains at 0 since we still have an entry in SYSCOPY that will allow us to recover our data to that version; the CURRENT_VERSION will be 2. OLDEST_VERSION in SYSTABLEPART for TS1 is unaffected since this is not a partitioned table space.

Version Status Following a MODIFY

SYSTABLES NAME	VERSION
T1	2
T2	0
T3	2

SYSTABLESPACE NAME	OLDEST_ VERSION	CURRENT_ VERSION
TS1	**2**	2

SYSTABLEPART TSNAME	OLDEST_ VERSION
TS1	**2**

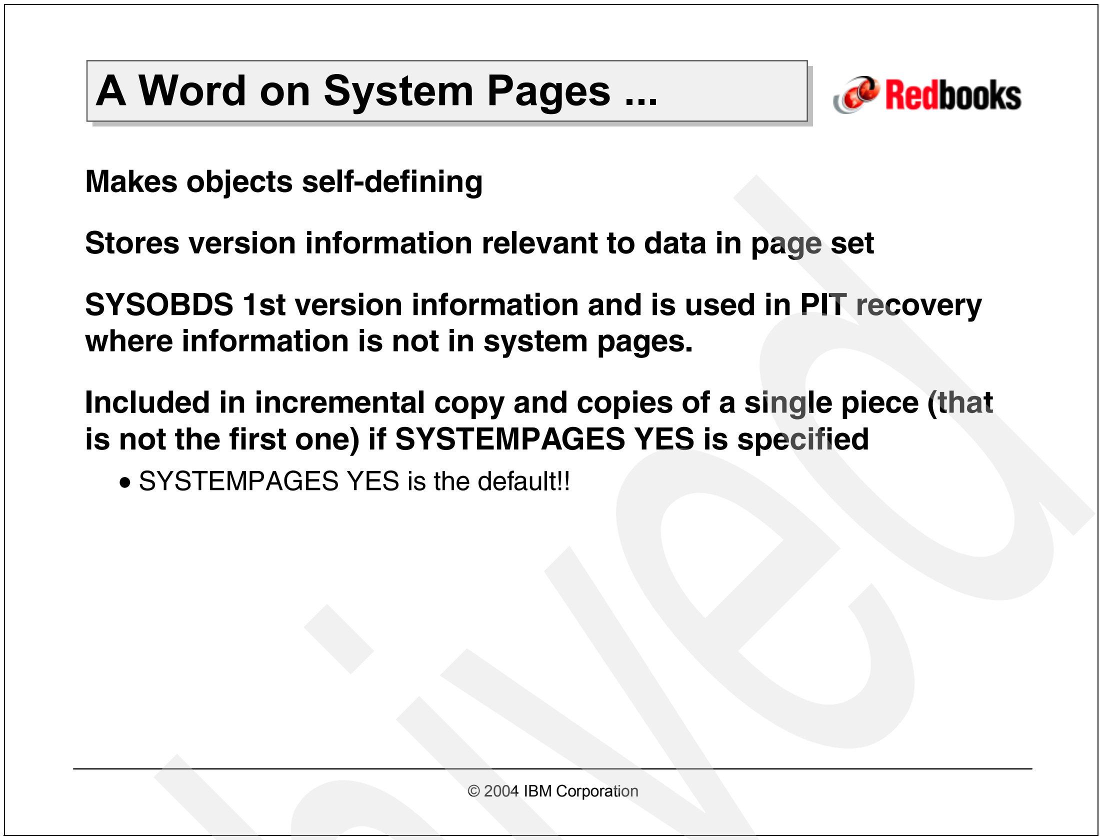
MODIFY has updated with lowest version in TS and ICs

MODIFY RECOVERY Delete Age(1) has deleted first copy

SYSCOPY TSNAME	TIMESTAMP	ICTYPE	OLDEST_ VERSION
~~TS1~~	~~1~~	~~F (FULL)~~	~~0~~
TS1	2	W (REORG)	**2**
TS1	2	F (FULL INLINE)	**2**

Version status following a MODIFY

Now, let us see what happens when we run a MODIFY DELETE AGE(1) and delete the image copy row created prior to the REORG from SYSCOPY. Since we ran a REORG and all the data for the altered objects is at version 2 and there are no more image copy records in SYSCOPY with an older OLDEST_VERSION number, MODIFY updates the OLDEST_VERSION column of SYSTABLESPACE and SYSTABLEPART for TS1 (and sets it to 2 (two) in this case. It is set to 2 because that is the oldest version that is used in the table space or any entry in SYSCOPY after the MODIFY.



3.12.5 A word on system pages

System pages is a new term in V8 to indicate pages in DB2's physical data set that do not contain real data (except for space map pages). A system page can be:

- A header page
- A dictionary page of a compression dictionary
- A version page (also known as system page for OBDRECs), containing information about different row layouts of the data that occurs in the table.

There can be zero, one, or multiple system pages for OBDRECs in a DB2 page set or partition. They can appear anywhere in the data set and are anchored off the header page.

The V8 COPY utility has a new SYSTEMPAGES (YES is the default and recommended) keyword. This allows some utilities (for example, UNLOAD and offline utilities) to access and interpret the data because they have the data formats in the system pages. (See 9.9.1, "COPY utility SYSTEMPAGES option" on page 669 for details.)

Partition Management

Add partition

Rotate partition

Alter partition boundaries

Rebalance partitions

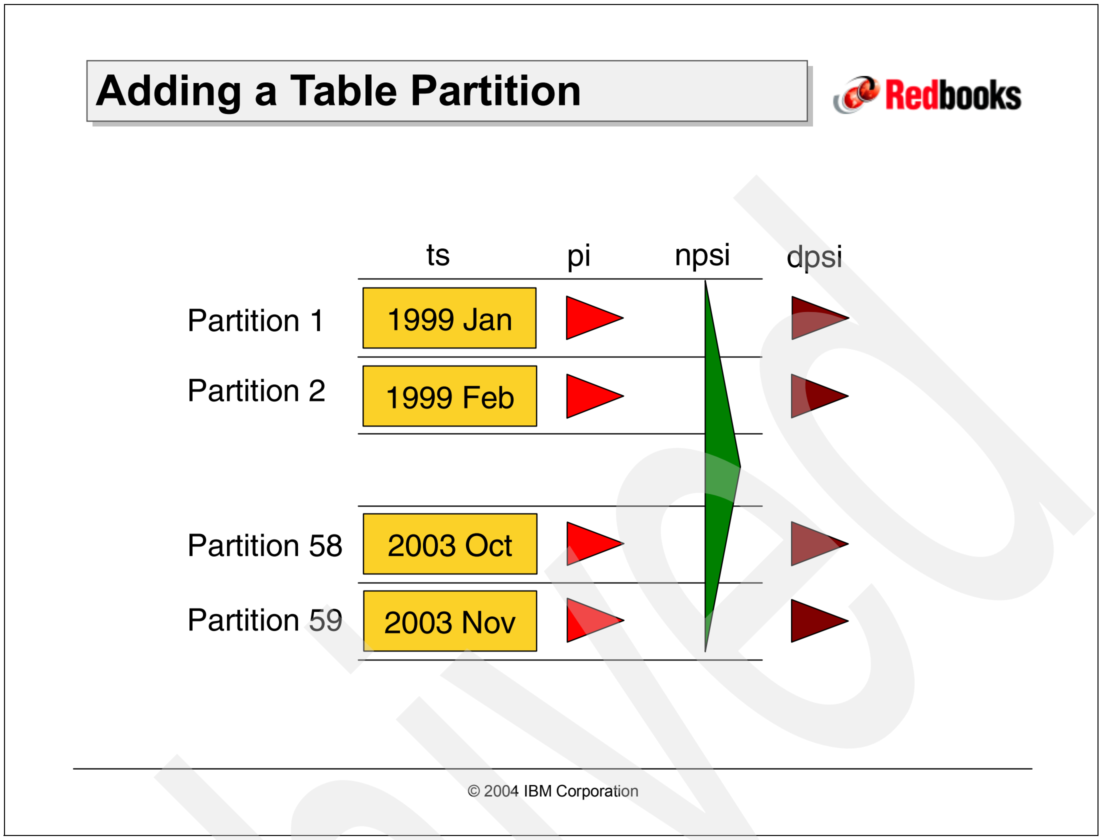
3.13 Partition management

A common partitioning scheme is to partition by date. Typically, the oldest data resides in partition 1 and the youngest data resides in part *n*. As time progresses and more data is collected, the usual desire is to add partitions to the table space to house the new periods of data. At some point, enough history has been collected, and the desire becomes one of two things:

1. To discard the oldest partition's worth of data and re-use that partition to hold the newest period's data. This reflects a cyclic use of some set number of partition numbers.
2. To roll-off the oldest partition of data and roll-on a new partition in which to collect the next period's data. This delays the need to re-use a partition number until the name space is exhausted (that is, until the number wraps). A variation on this is to make the data in "rolled off" partitions available to queries only on demand. The partition is hibernating, as it were, but can be awakened to participate in special queries.

Another common partitioning scheme is to partition so that you achieve partitions of similar size. This method is mainly used when you want to run utilities and application processes in parallel to reduce total elapsed time and with reduced contention.

DB2 V8 has the ability to immediately add partitions, rotate partitions, change the partitioning key values for *table-controlled* (see "V8 partitioned tables, table-controlled partitioning" on page 98 for more information) partitioned tables via the ALTER TABLE statement, and rebalance partitions. Here we give a brief description of these enhancements. More detailed information can be found in the utilities chapter.

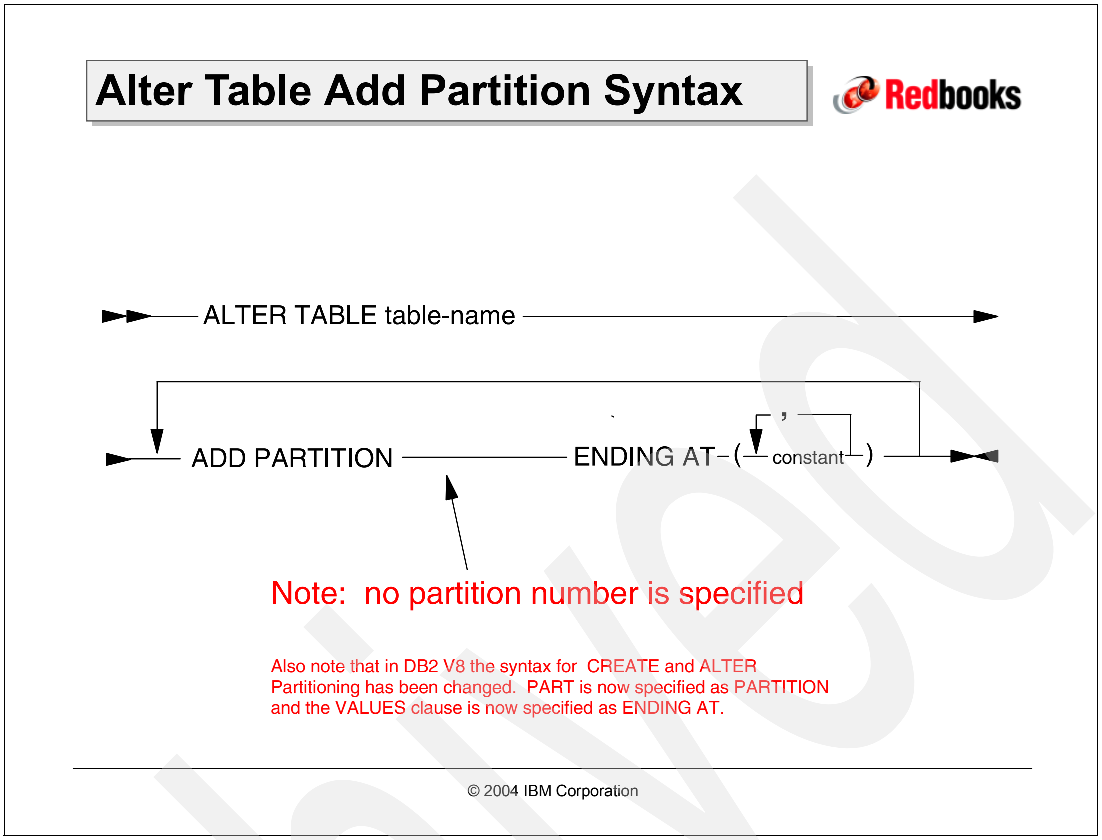


3.13.1 Adding a partition to a partitioned table space - current situation

Up until DB2 V7, an application that requires inserting time based data, the data is often stored by month, with a separate partition assigned for each month. The design usually leans toward allocating the maximum number of partitions allowed up front. This probably means allocating 254 partitions (with most of them initially with minimum space allocation) so that the application had a life span of about 20 years before running out of partitions.

In the example in the figure above, we have not been that wise. We currently have 59 partitions allocated and in use, each partition containing a month's data starting with January 1999. Currently, we want to get ready to store the data for December 2003, but we are out of partitions. We only have 59 partitions defined in our table space. In previous versions of DB2, we would have had to drop the entire table space, redefine it with additional partitions, and reload the data again.

But with Version 8, this is no longer required. We can just add a new partition at the end to store December's data, as we show in the next couple of pages.

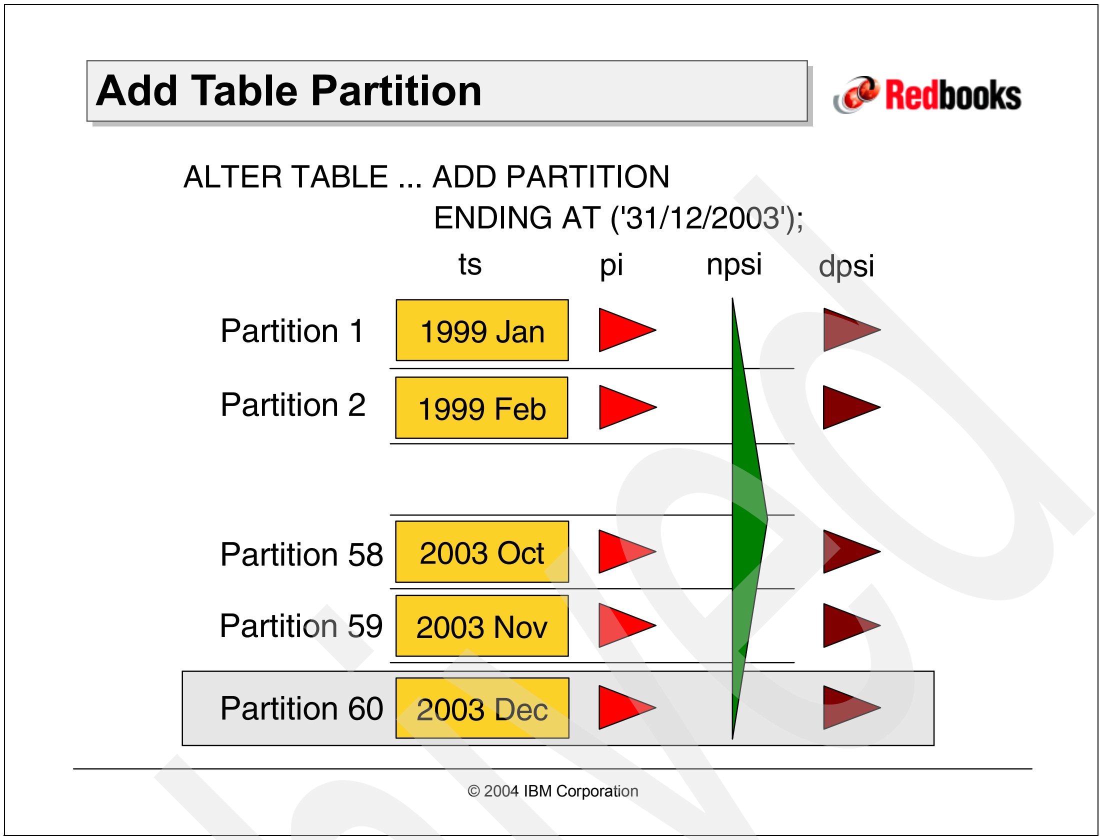


3.13.2 ALTER TABLE ADD PARTITION syntax

The visual above shows the syntax of the new ALTER TABLE ADD PARTITION statement.

Note that the statement is referencing the table (ALTER TABLE), not the table space (the statement is not an ALTER TABLESPACE). As a consequence, there is no syntax at the table level to specify the size of the new partition. (How the size of the new partitioned is determined will be explained shortly).

Also note that you do not specify a partition number on the ALTER TABLE statement, only a new limit key for the new partition. DB2 knows how many partitions are currently being used and will allocate a data set with the next higher partition number (when the table space is storage group defined; otherwise you must allocate a VSAM file prior to running the ALTER TABLE statement).



3.13.3 Add a partition to a partitioned table space

The figure above shows how easy it is to add a partition to a partitioned table space in DB2 Version 8. Just execute the ALTER TABLE ADD PARTITION statement and you are ready to go. You can start inserting / loading rows into the new partition immediately.

With DB2 Version 8, it is much easier to add partitions at a later date. Therefore, you can now start out with a limited number of partitions in your applications, as many as you currently need, and then reevaluate your partition needs within the next 12 to 18 months. This results in “managing” many fewer objects that today may be pre-allocated without being immediately used.

Adding a Partition - Considerations

Redbooks

Immediate availability

- Table is quiesced and all related plans, packages and cached statements are invalidated
  - Necessary as access path may be optimized to read only certain partitions
  - Auto rebinds will occur but you may wish to issue rebinds manually
- Table space does not need to be stopped

Next *physical* partition number is assigned

Attributes of previous *logical* partition are used - for example, PRIQTY

- Run ALTER TABLESPACE statements afterwards

If the previous partition is in REORP, new one will be too

Last partition limit key is enforced - a change in behavior!

3.13.4 Adding a partition - considerations

With Version 8, users are able to dynamically add partitions to a partitioned table. You can add partitions up to the maximum limit that is determined by the parameters specified when the partitioned table was initially created. When you add a partition, the next available *physical partition number* is used:

- When objects are DB2-managed (STOGROUP defined), the next data set is allocated for the table space and each partitioned index.
- When objects are user-managed (USING VCAT), these data sets must be pre-defined. The data sets for the data partition and all partitioned indexes must be predefined using the VSAM access method services (IDCAMS) DEFINE command for the partition to be added (that is the value of the PARTITIONS column in SYSTABLESPACE plus one), before issuing the ALTER TABLE ADD PARTITION statement.

Note that no partition number is supplied on the ALTER TABLE ADD PARTITION statement. The partition number is selected by DB2 based on the current number of partitions of the table.

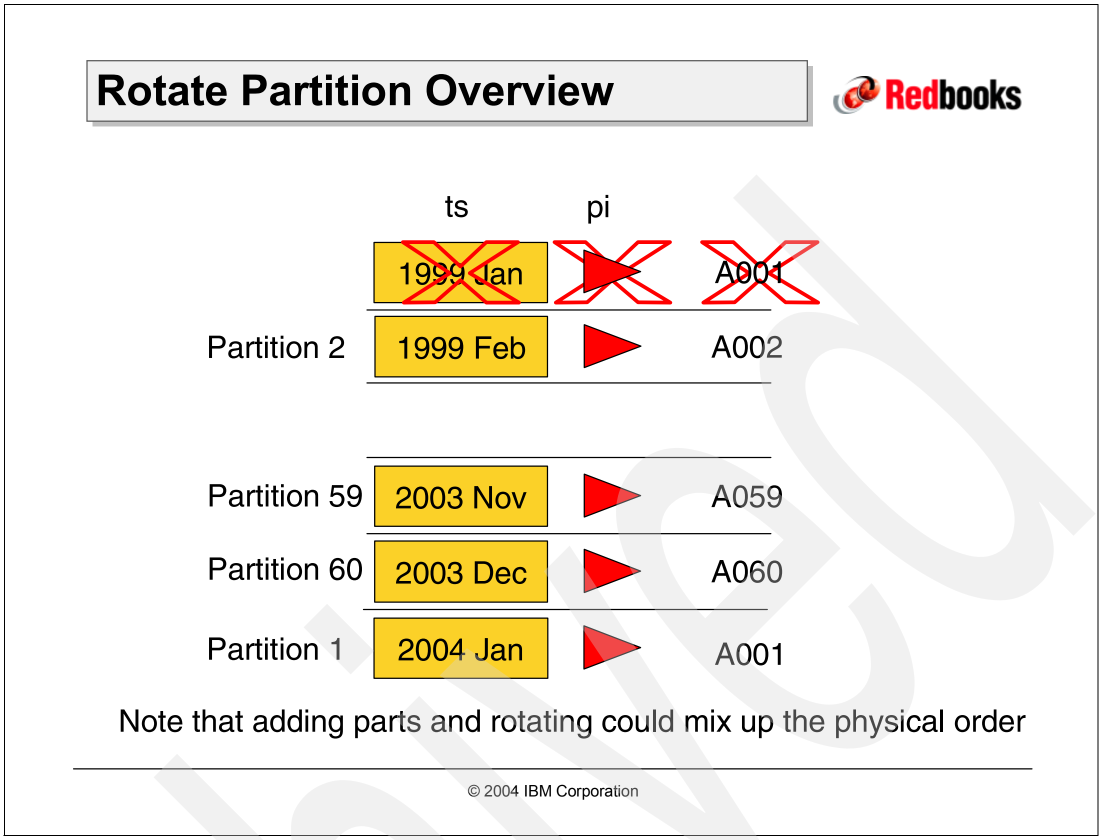
The newly added partition is immediately available. None of the objects need to be stopped before issuing the ALTER statement.

The table is quiesced and all related plans, packages, and cached statements are invalidated. This is necessary as the access path may be optimized to read only certain partitions. Automatic rebinds will occur (if allowed), but you may wish to issue rebinds manually.

Since you cannot specify attributes like PRIQTY, the values of the previous *logical* partition are used. Therefore, you will probably want to run an ALTER TABLESPACE statement afterwards to provide accurate space parameters, before starting to use the newly added partition. However, you have to remember that you specify the *physical* partition number on the ALTER TABLESPACE command.

Adding partitions to a partitioned table does not affect the ability to do point-in-time recovery of the object in any way. When you recover the table space to a point-in-time before the partition was added, the new partition is not deleted; it is only emptied.

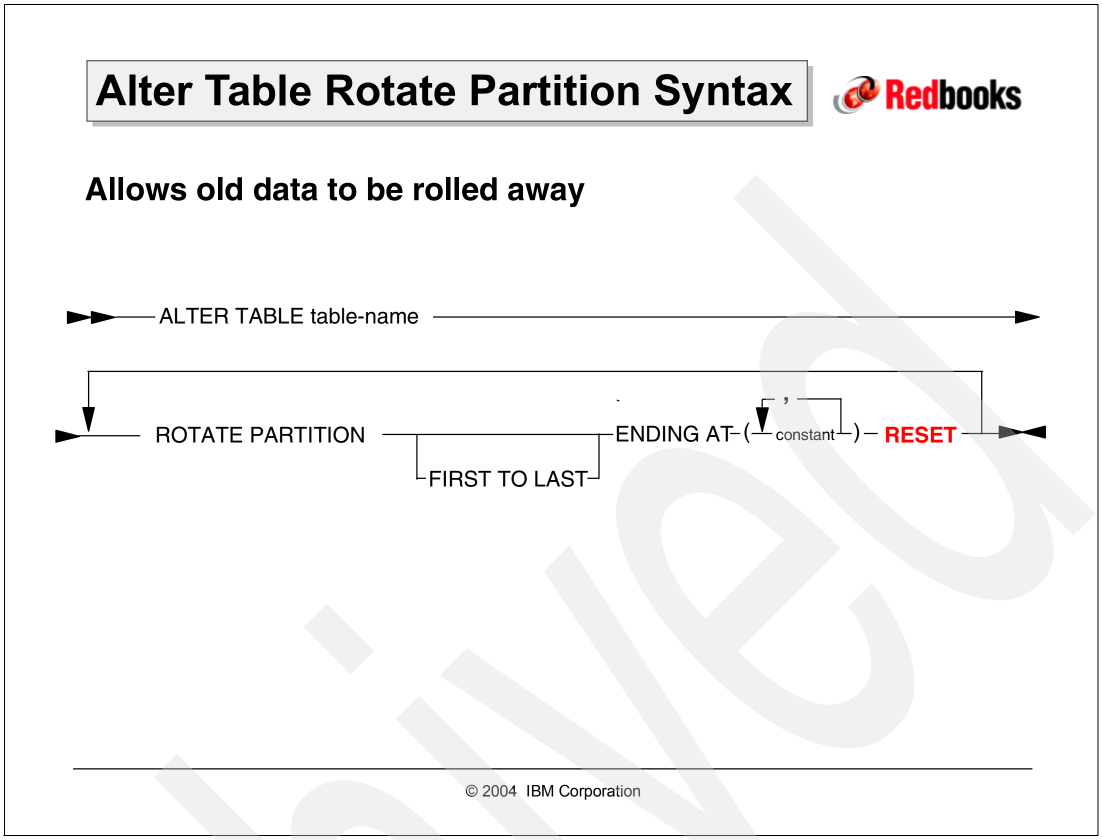
If you are altering a partitioned table space where currently the highest partitioning key is not enforced, and add a partition at the end, the last partitioning key limit is enforced.



3.13.5 Rotate partition - overview

When designing an application that requires storing the data for only a certain amount of time, (such as for legal reasons), consider a rolling partitions design. Now that there is the ability to easily rotate and reuse partitions over time in Version 8, it is easier to manage a limited number of partitions that are set up based upon dates.

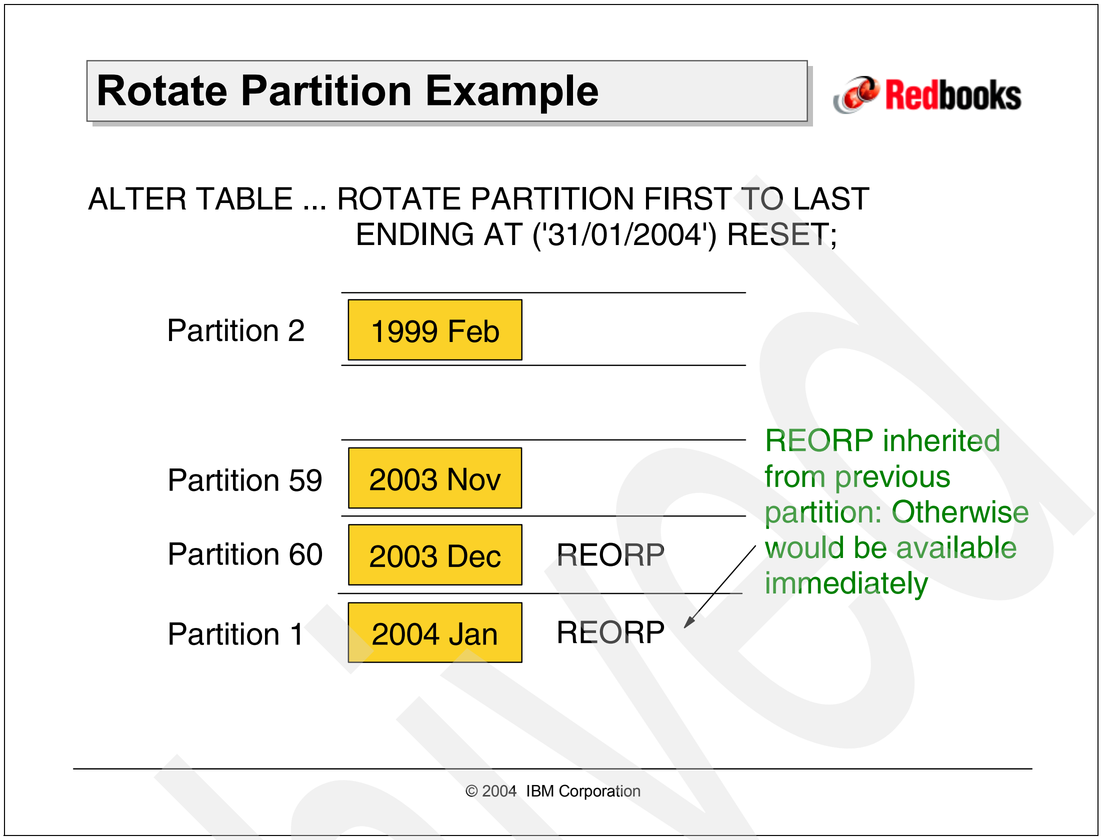
Rotating partitions allows old data to “roll-off” while reusing the partition for new data with the ALTER TABLE ROTATE PARTITION FIRST TO LAST statement. A typical case is where 13 partitions are used to continuously keep the last 12 months of data. When rotating, you can specify that all the data rows in the oldest (or logically first) partition are to be deleted, and then specify a new table space high boundary (limit key) so that the partition essentially becomes the last logical partition in sequence, ready to hold the data that is added.



3.13.6 Alter table rotate partition syntax

The figure above shows the new ALTER TABLE ROTATE PARTITION syntax. Note the RESET keyword at the end. This is to draw your attention to the fact that the information in the existing table space partition is going to be deleted (reset).

RESET in this context should not be confused with the RESET keyword that is used in the "VSAM-world" to indicate the reset of the HURBA to zero.



3.13.7 Rotate partition example

The figure above shows an example of an ALTER TABLE ROTATE PARTITION statement. In this case partition 1 is emptied (all rows are deleted) and reused to store the data of January 2004 (new limit key '31/01/2004').

Normally the emptied partition is immediately available. However, because partition 60 (the previous last logical partition) is in Reorg Pending (REORP) status, partition 1 will inherit that status and will be in REORP as well. We will have to run a REORG before both partitions can be used by our applications.

Rotate Partition Effects

Redbooks

Immediate availability (no REORG necessary)

Lowest logical partition is assigned to become the next logical partition

Specified boundary must be new "high value"

Last partition limit key is enforced

Recover to previous PIT blocked as SYSCOPY and SYSLGRNX entries are deleted

3.13.8 Rotate partition effect

The partition that was rolled off is immediately available after the SQL statements are successfully executed. No REORG is necessary (unless the previous logical partition is REORP, then the rolled off partition will also be placed in REORP).

The current lowest logical partition is looked up and its corresponding physical partition will become the new highest logical partition.

When you execute the ALTER TABLE ROTATE PARTITION statement, the new limit key must be higher than the limit key of the last logical partition at that time, and the last partition limit key is enforced.

> **Important:** It is important to know that recovery to a previous point-in-time is blocked after running the rotate statement. You cannot recover to a point-in-time before the ALTER TABLE ROTATE PARTITION was executed. This is because all SYSCOPY and SYSLGRNX entries are deleted when the rotate statement is executed. In addition, DB2 does not support an “undo” of the rotate statement itself. After the rotate, DB2 has no knowledge of what the original limit key for the rolled-off partition was.

Considerations When Rotating Partitions

Data is deleted, so you may want to unload it first

Rotate will issue individual deletes

- Consider the impact on logging and performance
- DBD lock held for the duration of the ROTATE

Suggestion:

- Run LOAD ... PART n REPLACE
  with a dummy SYSREC data set to clear out the partition first
- Run cross-loader to combine the LOAD REPLACE with the ALTER TABLE ROTATE PARTITION statement

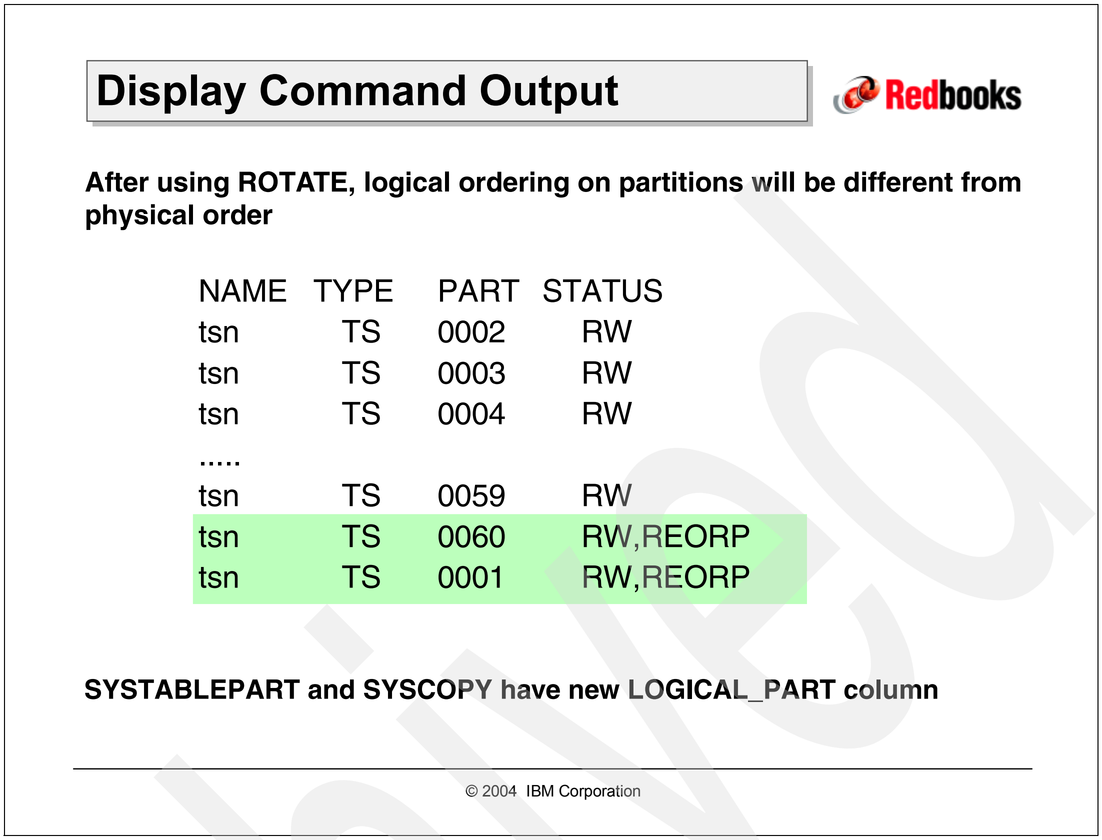
3.13.9 Considerations when rotating partitions

Because the data of the partition being rolled off is deleted, you may want to consider running an unload job before rotating the partition.

When rotating partitions of a partitioned table, consideration should be made for the time needed to complete the DDL statement.

The RESET keyword indicates that you reset the data of an existing partition. DB2 will *delete* all the rows of the existing partition that you are about to reuse. To speed up the delete process, you may want to consider doing a LOAD REPLACE with an empty data set before running the ALTER TABLE .. ROTATE PARTITION statement. The reset operation requires that the keys for the rows of that partition must also be deleted from all non-partitioned indexes. Each NPI must be scanned to delete these keys; therefore, the process can take an extended amount of time to complete as each NPI is processed serially.

Additional consideration must be given for the time needed to delete data rows if processing must be done a row at a time. Additional delete row processing may be required for referential integrity relationships (SET NULL and DELETE CASCADE), or when there are delete triggers.



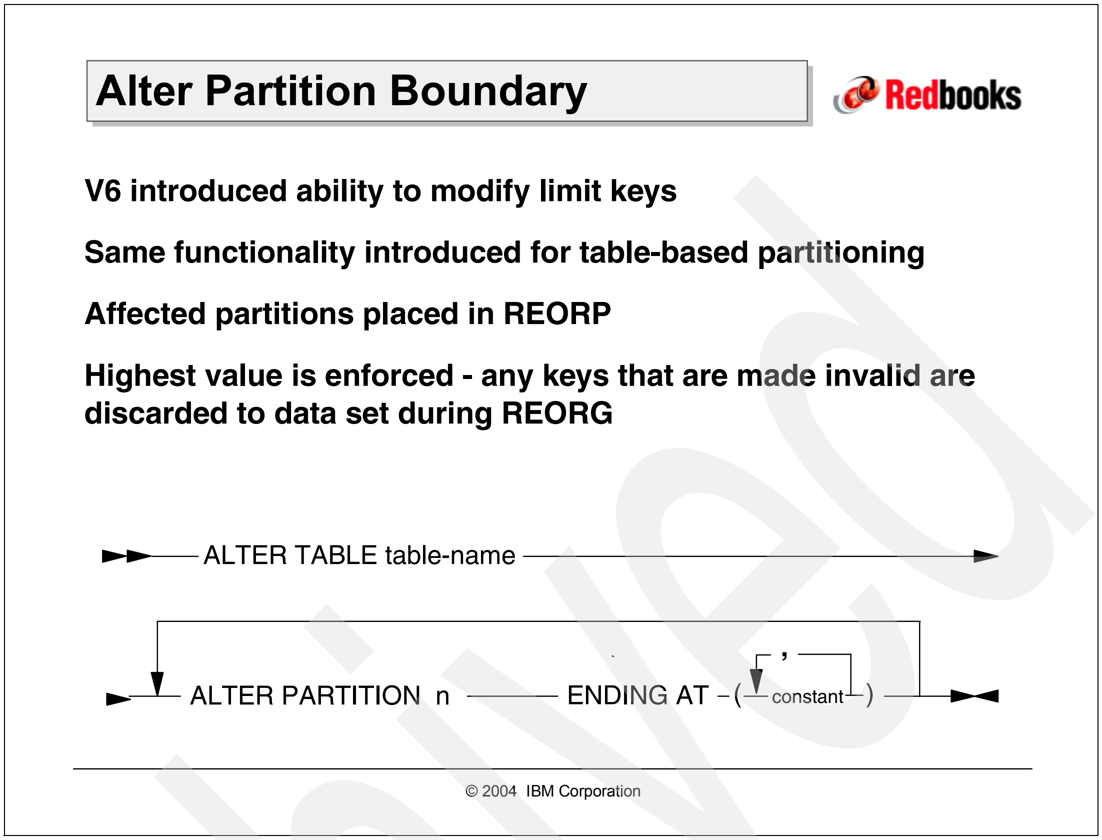
3.13.10 Display command output

After using the new ALTER TABLE ROTATE PARTITION statement, the logical and physical order of the partitions is no longer the same. The display command lists the status of table space partitions in logical partition. Logical order is helpful when investigating ranges of partitions that are in REORP. It enables one to more easily see groupings of adjacent partitions that may be good candidates for reorganization. When used in conjunction with the new SCOPE PENDING keyword of REORG, a reasonable subset of partitions can be identified if one wants to reorganize REORP ranges in separate jobs.

The logical partition number can also be found in the catalog in several places. Both SYSTABLEPART and SYSCOPY have a LOGICAL_PART column (Table 3-2).

Table 3-2 Logical versus physical partition

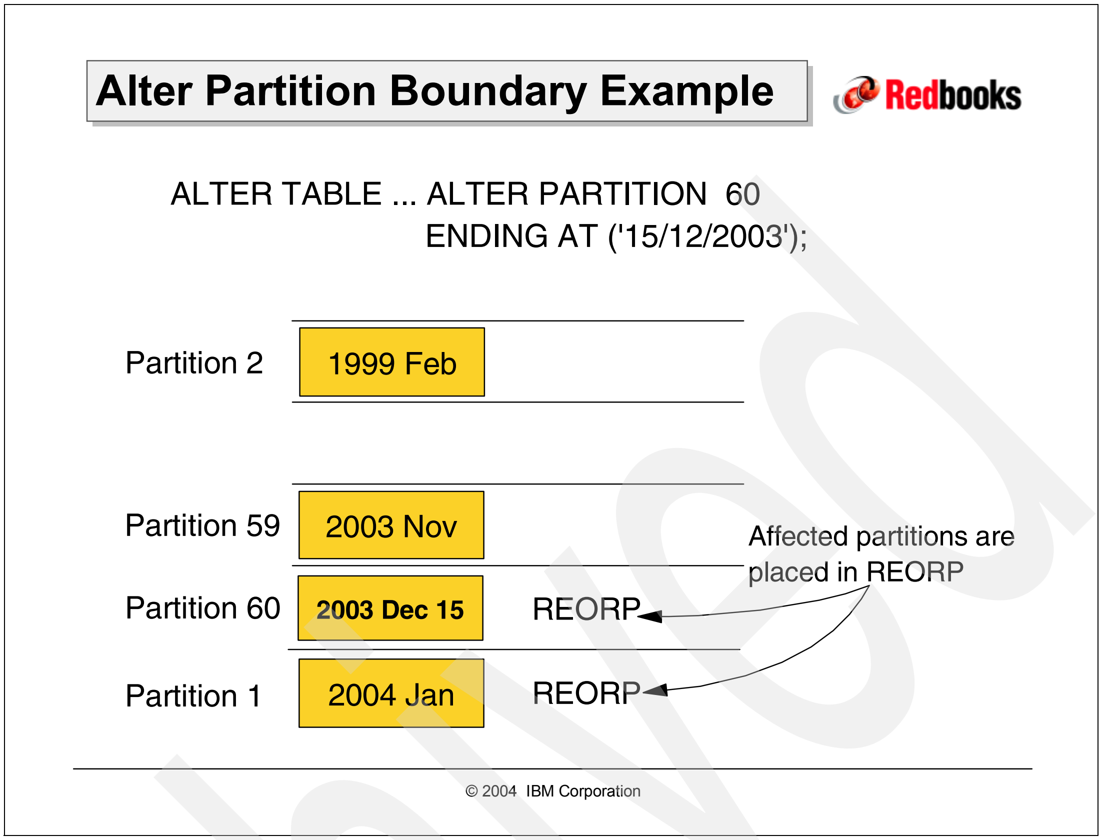
	Physical partition	Logical partition
SYSCOPY	DSNUM	LOGICAL_PART
SYSTABLEPART	PARTITION	LOGICAL_PART



3.13.11 Alter partition boundary

In DB2 V6, the ability to modify limit keys for table partitions was introduced. The enhancement in DB2 V8 introduces the same capability for table-based partitioning with the ALTER TABLE ALTER PART VALUES statement. The affected data partitions are placed into a Reorg Pending state (REORP) until they have been reorganized.

> **Attention:** Note that any ALTER, related to a partitioned table that specifies a partition number, always has to specify the *physical partition number*. It is the user's responsibility to derive the physical partition number from the logical partition number whenever required. SYSIBM.SYSTABLEPART and SYSIBM.SYSCOPY have a new LOGICAL_PART column to assist with the conversion between logical and physical partition number.



3.13.12 Alter partition boundary example

The figure above shows an example of how to change the limit key of a partition for a table-controlled partitioned table space. As with index-controlled table spaces, the affected partition (and potentially the next logical partition) are put in REORP status.

Note that you specify the physical partition number on the ALTER TABLE ALTER PARTITION statement.

Rebalance Partitions

Redbooks

REORG ... REBALANCE allows you to automatically rebalance rows across selected partitions

DB2 determines the required key ranges during REORG

- You may not get perfect rebalancing when many duplicate keys exist

Can be done using REORG SHRLEVEL REFERENCE

3.13.13 Rebalance partitions

Rebalancing partitions is done by means of the REORG TABLESPACE utility. Specifying the REBALANCE option when specifying a range of partitions to be reorganized allows DB2 to set new partition boundaries for those partitions, so that all the rows that participate in the reorganization are evenly distributed across the reorganized partitions. (However, if the columns used in defining the partition boundaries have many duplicate values within the data rows, even balancing is not always possible.)

Rebalancing is ideal when no skewing of data between partitions is required, or needs to be catered for. It has an advantage over changing the partition boundaries using the ALTER TABLE ALTER PARTITION. ENDING AT statement, in that the partitions involved in the rebalancing are not put into REORP status.

You can specify REBALANCE with online REORG, REORG TABLESPACE *SHRLEVEL REFERENCE*.

Upon completion, DB2 invalidates plans, packages, and the dynamic statement cache that reference the reorganized object.

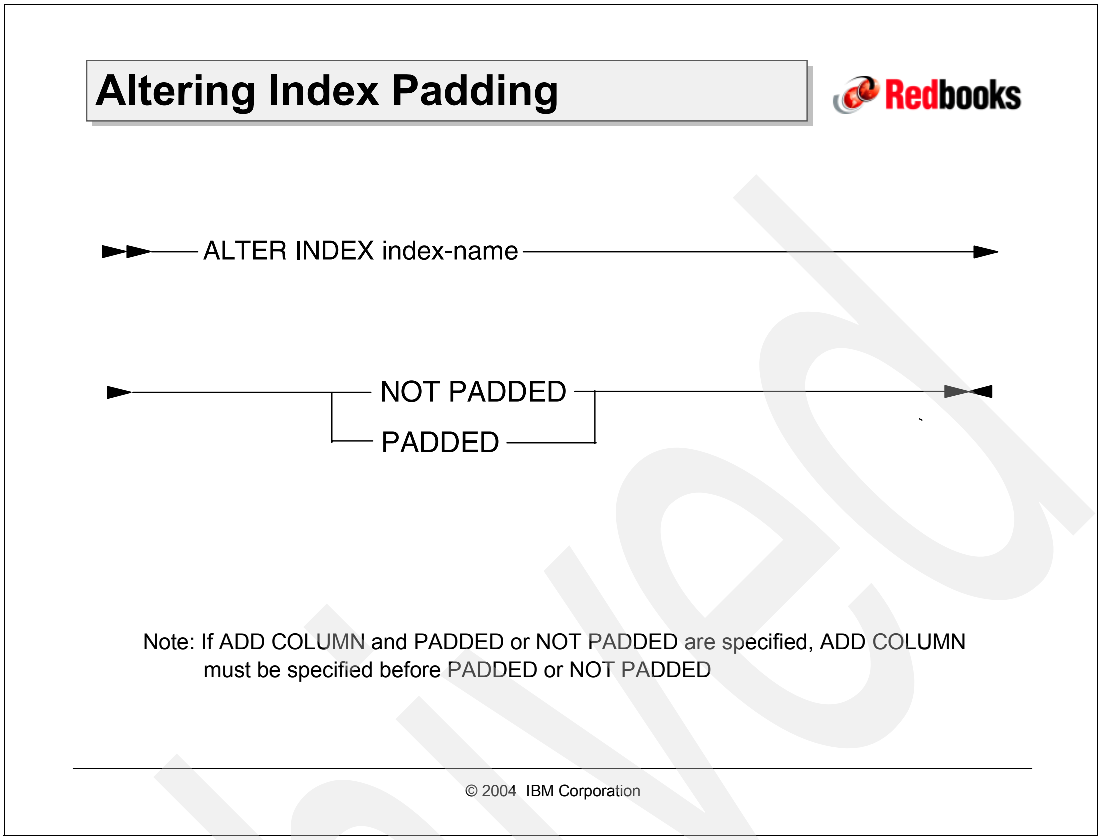
More details can be found in 9.5, “REORG REBALANCE” on page 657.

Other Index Enhancements

3.14 Other index enhancements

In this section we cover a number of miscellaneous index enhancements, such as:

- Altering an index between PADDED and NOT PADDED and vice versa
- Altering the clustering index
- Avoiding indexes in Rebuild Pending
- Index creation enhancements



3.14.1 Altering index padding attribute

In Versions prior to Version 8, varying length columns in the index are always padded with blanks up to their maximum length. When you specify a large number as the maximum length for a VARCHAR column, and only use a few bytes, your indexes that contain that column will be very big, because of the index key padding.

In Version 8, varying length columns are no longer always padded to their full length when they are part of an index key. When specifying NOT PADDED during the creation of an index, padding does not occur and the keys are stored as true varying length keys. Varying length indexes are marked in the new SYSINDEXES column, PADDED, with a value of 'N'. NOT PADDED is the default at create time.

You can also alter an index between PADDED and NOT PADDED. The syntax is shown in the figure above.

Alter Index Not Padded/Padded

Prior to V8, data is varying length in the table but always padded to the maximum length in indexes

V8 provides a new index option NOT PADDED.
Then varying-length columns are no longer be padded to the maximum length in index keys.

True column is stored in the index (length and data)

Enables index-only access for varchar data

Likely to reduce index storage requirement

ALTER INDEX PADDED/NOT PADDED sets index to RBDP

- Index must be rebuilt from data
- Optimizer can then choose index for index only access

3.14.2 Alter index not padded/padded

An index can be changed from `PADDED` *to* `NOT PADDED` or from `NOT PADDED` *to* `PADDED` using ALTER INDEX. In both cases the index is placed in Rebuild Pending (RBDP) state. Even though the data is temporarily unavailable (while the index is in RBDP), there is no need to drop and recreate the index. In addition, DB2 V8 can avoid using an index that is in RBDP in many cases (as discussed in 3.14.4, “RBDP Index avoidance” on page 185).

One of the major benefits of using NOT PADDED indexes, is that DB2 can do true index-only access with NOT PADDED indexes. This is discussed in 10.3.1, “Varying-length index keys performance benefits” on page 736.

When the database design has used VARCHAR columns wisely (that is, not using the VARCHAR data type for very small columns, or character columns that are always filled to the maximum size), NOT PADDED indexes can reduce the size of the index (including the number of levels).

Alter Clustering Attribute of Indexes

Clustering has been unbundled from partitioning

- A move to table-controlled rather than index-controlled partitioning
- A partitioning index does not have to be the explicit clustering index

Change clustering index with two steps

- ALTER INDEX index1 NOT CLUSTER
  - Will continue to be used until new clustering index is defined
- ALTER INDEX index2 CLUSTER
  - Immediate effect - inserts follow new clustering but needs REORG!

3.14.3 Alter the clustering attribute of indexes

In V8, you can change the clustering order in a partitioned or non-partitioned table space without dropping the index.

In case of a partitioned table, when using index-controlled partitioning, the partitioning index is also the clustering index. On the other hand, when using table controlled partitioning, any index can be the clustering index.

In addition, you can change the clustering attribute of an index by using the `CLUSTER` and `NOT CLUSTER` options of `ALTER INDEX`. As before, only one clustering index is allowed for any table.

When you change an index to NOT CLUSTER, it continues to be the clustering index until you define another index with the CLUSTER attribute. When you change an index to become the clustering index, all new rows will be inserted using the new clustering index. To rearrange the existing row according to the new clustering index, you must perform a REORG.

For more details, see also 3.10, “Clustering indexes” on page 124.

RBDP Index Avoidance

Data Manager bypasses indexes in rebuild pending (index is ignored)

- For all DELETE processing
- For all non-unique indexes for UPDATEs and INSERTs

Optimizer avoids RBDP indexes as follows:

- Dynamic PREPARE
  - Indexes in RBDP avoided
- Cached
  - If cached, PREPARE is bypassed
  - Invalidation occurs during ALTER
- Static BIND (RBDP index NOT avoided)
  - Indexes in RBDP can still be chosen, and will get resource unavailable at execution time
- Reoptimization
  - Acts the same as initial BIND or PREPARE

Utilities invalidate the cache for tables where index RBDP state is reset

3.14.4 RBDP Index avoidance

Today, when an index is in Rebuild Pending (RBDP), the optimizer is unaware of that and can still pick that index during access path selection when you are executing a query in your QMF™ application, for example.

In Version 8, DB2 will try to avoid using indexes in RBDP.

Data Manager will bypass indexes in Rebuild Pending (index is ignored):

- For all DELETE processing
- For all non-unique indexes for UPDATEs and INSERTs

A unique index in RBDP cannot be avoided for UPDATE or INSERT because DB2 needs the index to be available to be able to enforce uniqueness.

The DB2 optimizer avoids RBDP indexes as follows:

- Dynamic PREPARE:
  - Indexes in RBDP are avoided.
- Cached statements:
  - If the statement is cached, the PREPARE is bypassed.
  - Invalidation occurs during ALTER.
- Static BIND does not avoid an index that is in RBDP:
  - Indexes in RBDP can still be chosen, and will get Resource Unavailable at execution time.

- Reoptimization:
  - Reoptimization acts the same as an initial BIND or PREPARE.
- DB2 utilities invalidate the dynamic statement cache for tables where index RBDP state is set or reset.

> **Note:** An index in RECOVER-pending (RECP) is also avoided during access path selection of a dynamically prepared SQL statement.

Index Creation Enhancements

Redbooks

Deferred indexes avoided by dynamic access path selection

- Index avoidance also applies to DEFER YES indexes in RBDP

CREATE INDEX invalidates dynamic statement cache

- Allows optimizer to pick the newly created index without any additional user action (such as manual invalidation using RUNSTATS)

3.14.5 Index creation enhancements

In this section we describe two enhancements related to DB2 index creation, namely:

- Deferred indexes do not prevent table access
- Invalidation of dynamically cached statements after CREATE INDEX

Deferred indexes do not prevent table access

Prior to V8, DB2 already allows you to create deferred indexes. Deferred indexes are indexes that are created using the DEFER YES clause. When creating a deferred index, the description of the index and its index space is added to the catalog, but it is not built. It is placed in Rebuild Pending (RBDP) status. (The only exception to this is if the table on which the index is defined is empty). The advantage of deferring the creation of the index tree is that multiple indexes, which are defined on the same table, can be built in a single run rather than scanning the table multiple times. In addition, these indexes can be built using the REBUILD INDEX utility, which is faster than building the index at CREATE INDEX time.

The problem with deferred indexes is that, prior to DB2 V8, they block table access if the optimizer selects an access path that uses the deferred index for a given statement. The execution of the statement results in SQL code -904 and reason code 00C900AE, which indicates that an attempt was made to access an index that is in Rebuild Pending state.

As mentioned in the previous topic, with DB2 V8, the optimizer does not consider indexes in RBDP state during the dynamic prepare of a statement. As deferred indexes are initially in RBDP state, this enhancement also applies to indexes created with DEFER YES.

To take advantage of deferred indexes as soon as possible after they are (re)built, cached statements, which refer to the base table of a deferred index, are invalidated when the index is rebuilt and reset from RBDP state.

Invalidation of dynamically cached statements after CREATE INDEX

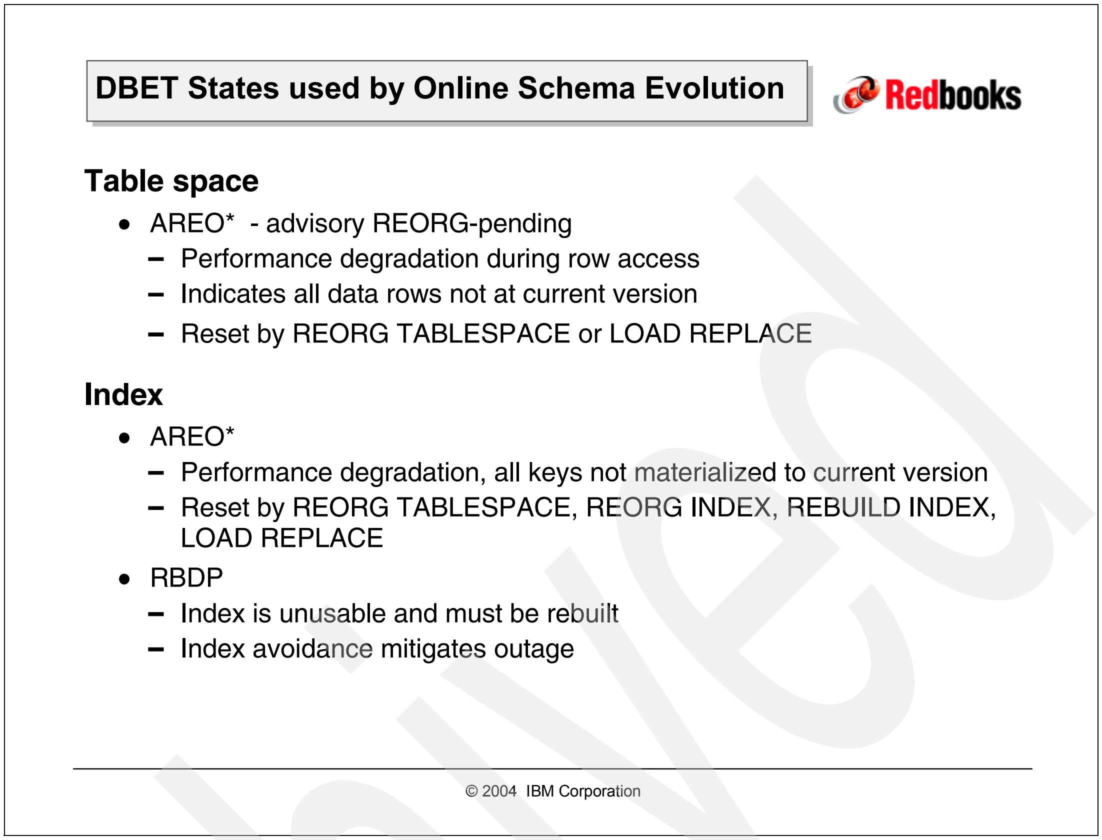
If an SQL statement encounters performance problems, a typical remedy is to create an appropriate index that the DB2 optimizer can choose to speed up the processing of this statement. However, if the statement is cached in the dynamic statement cache (DSC), the newly created index is ignored for this statement, prior to DB2 V8, in single subsystem environments. DB2 keeps the access path that the optimizer has previously determined. The optimizer can consider the index for the statement only after that statement has been removed from DSC, either because it was displaced from DSC or because the database administrator explicitly invalidated it.

While adding to operational complexity, there is a circumvention for this problem that addresses known indexes. The statements can be manually invalidated after a new index is created, for example, by running the RUNSTATS utility with the options UPDATE NONE REPORT NO in DB2 V8. However, when using ERP solutions, indexes are often created when maintenance is applied to the ERP package. Therefore, their existence is not obvious to the database administrator. The existing cached statements often cannot take advantage of the new indexes for a considerable amount of time.

In data sharing environments, the creation of an index invalidates cached statements that refer to the base table of the new index already before DB2 V8. In this case, the cached statements are even quiesced, which means that they cannot continued to be used within active transactions.

DB2 V8 introduces analogous behavior for both data sharing and non-data sharing. If a new index is created, the affected cached statements are invalidated, but not quiesced. That means that statements that are used in active transactions can still be employed until these transactions complete. Indexes that are created with the option DEFER YES, which means that the index tree is not immediately built, also invalidate cached statements.

Both enhancements improve the usability of newly created indexes (whether created deferred or not). This eliminates the exposure of having the optimizer select an index that is not yet ready to be used, as well as the DBA having to run RUNSTATS to invalidate the statements in the DSC that can potentially benefit from the newly created index. They make DB2 require less manual interaction and hence contribute to a less error-prone approach.



3.15 Reviewing DBET states used by online schema evolution

In support of online schema changes, DB2 V8 introduces a new Database Exception Table state (DBET), Advisory Reorg (AREO*). Advisory Reorg indicates that the table space, index, or partition identified should be reorganized for optimal performance.

The AREO* state is reset for a *table space* by running the REORG TABLESPACE or LOAD REPLACE utilities.

The AREO* state is reset for an *index* by running by running the REORG TABLESPACE, REORG INDEX, REBUILD INDEX, or LOAD REPLACE utilities.

Utilities can be run to reset AREO* for individual partitions without being restricted by AREO* in adjacent partitions. The AREO* state is not reset by a START FORCE command, but can be reset using the REPAIR utility.

The DISPLAY DATABASE command shows the new DBET state AREO* for all objects.

Review of New DBET States

Object type	Action taken	Resulting condition
Table Space	ALTER TABLE ALTER COLUMN (any data type)	AREO*
Index Space	ALTER TABLE ALTER COLUMN of VARCHAR to CHAR	AREO*
	ALTER TABLE ALTER COLUMN of CHAR to VARCHAR	AREO*
	ALTER TABLE ALTER COLUMN of GRAPHIC to VARGRAPHIC	AREO*
	ALTER TABLE ALTER COLUMN of VARGRAPHIC to GRAPHIC	AREO*
	ALTER TABLE ALTER COLUMN of CHAR to CHAR	AREO*
	ALTER TABLE ALTER COLUMN of GRAPHIC to GRAPHIC	AREO*
	ALTER TABLE ALTER COLUMN of VARGRAPHIC to VARGRAPHIC	AREO*
Index Space	ALTER INDEX ADD COLUMN	RBDP (1)
Index Space	ALTER INDEX from NOT PADDED to PADDED	RBDP
	ALTER INDEX from PADDED to NOT PADDED	RBDP

The AREO* and RBDP states are not reset by a START FORCE command, but can be reset using the REPAIR utility.

(1) Can be AREO* if column added to the table in the same UOW as the ALTER INDEX ADD COLUMN

3.15.1 Review of new DBET states

The figure above gives an overview of the different types of ALTERs you can do and their effect on the DBET state of your table space and/or index.

System Level Point-In-Time Recovery

Redbooks

New utilities for

Backing up the system

- BACKUP SYSTEM
  - Less disruptive than -SET LOG SUSPEND
  - Not a consistent backup (like -SET LOG SUSPEND)

Restoring the system

- RESTORE SYSTEM
  - Restores consistency (no outstanding URs when done)

Where system is an entire

- DB2 subsystem
- DB2 data sharing group

Support for DB2 Tracker site

3.16 System level point-in-time recovery

Version 8 provides enhanced backup and recover capabilities at the DB2 subsystem or data sharing group level. The purpose is to provide an easier and less disruptive way to make fast volume-level backups of an entire DB2 subsystem or data-sharing group with minimal disruption, and recover a subsystem or data-sharing group to any point-in-time, regardless of whether you have uncommitted units of work.

The primary focus of this solution is to provide a fast and easy way to bring back an entire DB2 system or data sharing group to a point-in-time that you know is OK. It is can be used for off-site disaster recovery situations, but its primary goal is to bring back the data on the original volumes to an arbitrary point-in-time as quickly as possible.

Two new utilities provide the vehicle for system level point-in-time recovery:

- The *BACKUP SYSTEM* utility provides fast volume-level copies of DB2 databases and logs.
- The *RESTORE SYSTEM* utility recovers a DB2 system to an arbitrary point-in-time. RESTORE SYSTEM automatically handles any creates, drops, and LOG NO events that might have occurred between the backup and the recovery point-in-time.

Using the new BACKUP SYSTEM utility, you can copy both the data and logs, or only the data. Previously, to make a system level backup, you had to issue the SET LOG SUSPEND command, which stops the logging and thus prevents any new database updates.

A BACKUP SYSTEM job does not stop the logging. However, it does quiesce some system activities, but the process is less disruptive than SET LOG SUSPEND processing. (The list of quiesced system activities during SYSTEM BACKUP processing are described in 3.16.3, “BACKUP SYSTEM operation” on page 197.) The BACKUP SYSTEM utility can operate on an entire data-sharing group, whereas the SET LOG SUSPEND command has to be issued for each data-sharing member.

As a further enhancement to taking system level backups, the SET LOG SUSPEND command now quiesces 32K page writes and data set extensions. (Note that quiescing 32K page writes is only required for page sets where the CI size of the underlying VSAM data set is not 32K. This was not possible in the past as all VSAM data sets had to use a 4K CI size. However, V8 gives you the possibility to have the VSAM CI size match the DB2 page size. See 3.18.1, “Control interval larger than 4KB” on page 206 for more details.)

The BACKUP SYSTEM and RESTORE SYSTEM utilities rely on new DFSMShsm services, and SMS constructs, like copy pools and copy pool backups, in z/OS V1R5 that automatically keep track of which volumes need to be copied.

A *copy pool* is a set of SMS managed storage groups that can be backed up and restored with a single command. Each DB2 system has up to two copy pools, one for databases and one for logs. The name of each copy pool that is to be used with DB2 must use the following naming conventions:

DSN$locn-name$cp-type

The variables that are used in this naming convention have the following meanings:

DSN	Unique DB2 product identifier
$	Delimiter
locn-name	DB2 location name.
cp-type	Copy pool type. Use DB for the database copy pool and LG for the log copy pool.

You have to set up both COPYPOOLs properly in order to successfully implement system level point-in-time recovery:

- The “log” COPYPOOL should be set up to contain the volumes that contain the BSDS data sets, the active logs, and their associated ICF catalogs.
- The “database” COPYPOOL should be set up to contain all the volumes that contain the databases and the associated ICF catalogs.

Archive logs should, if possible reside on different DASD and COPYPOOLs. If that is not possible, they should be with part of the “log” COPYPOOL.

Attention: It should be noted that in this new environment, IBM recommends that ICF catalogs are created and reside with the logs, and separate ICF catalogs are created that reside with the databases.

You can specify the number of copy versions to be maintained on disk (maximum 85) by using the VERSIONS attribute.

A *copy pool backup* is a new storage group type that is used as a container for copy pool volume copies. Each volume in the copy pool storage group needs to have a corresponding volume in the copy target storage group.

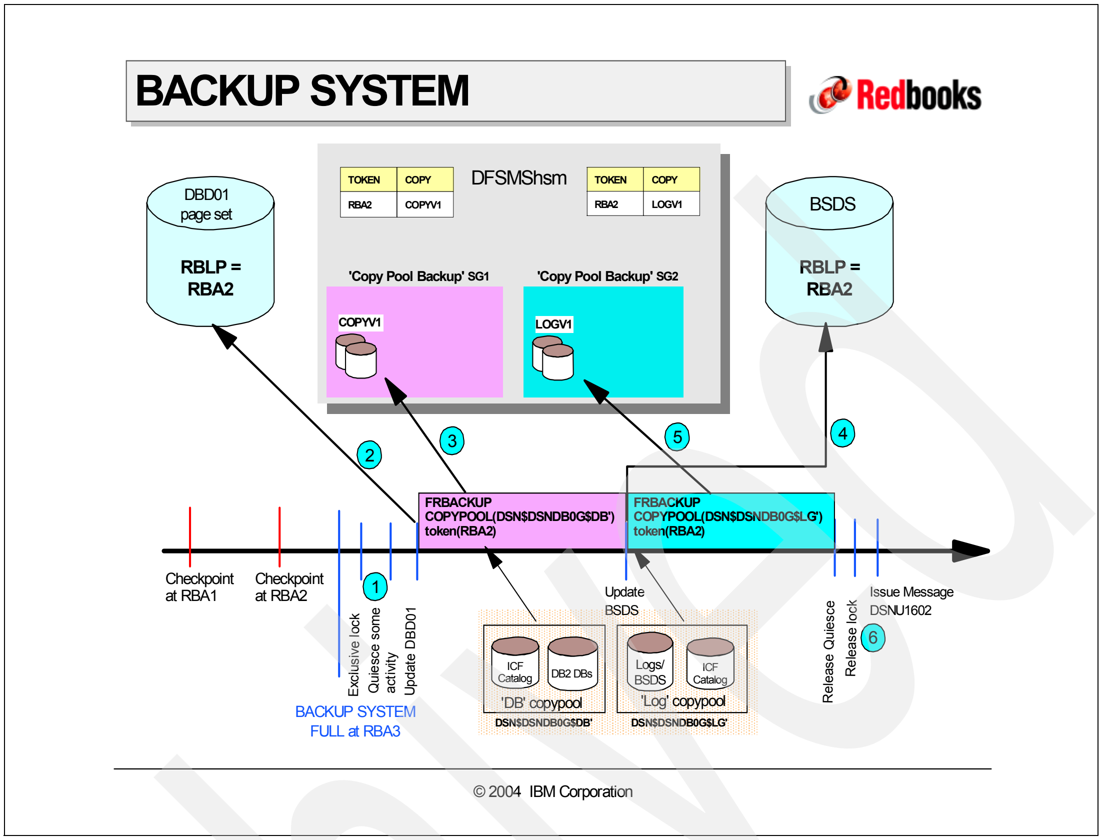
3.16.1 Prerequisites for this feature

To use the BACKUP SYSTEM and RESTORE SYSTEM utilities, all data sets that you want to back up and recover must be SMS-managed data sets. Additionally, you must have the following requirements:

- z/OS V1R5 or above
- Disk control units that support ESS FlashCopy ® API (FlashCopy V2 is recommended)
- SMS copy pools that are defined by using the DB2 naming convention
- Defined SMS copy pool backup storage groups

DB2 has to be running in new-function mode.

It is worth noting that you can use RESTORE SYSTEM *LOGONLY* without z/OS V1.5 or Disk control units that support ESS FlashCopy ® API.



3.16.2 BACKUP SYSTEM

The figure above provides an overview of how the BACKUP SYSTEM utility operates.

You can use the new **BACKUP SYSTEM** utility to back up an entire DB2 subsystem or an entire DB2 data sharing group with a single command. The BACKUP SYSTEM utility has two options that you can specify:

- **FULL:**

  In this case, the backup contains both the logs and databases. This is referred to as a *full system backup*. Full system backups can be used to recover the system to the point-in-time (PIT) at which the copy was taken. After restoring the backups, you can use a normal DB2 restart to bring the system back to consistent state. DB2 restart processing will take care of any outstanding units of work, by either completing their commit processing, or rolling back the changes they made so far (in case they did not reach a commit point). In this case you do not need to use the RESTORE SYTEM utility, although it can be used if you want. When taking a FULL backup, the copies of the log copypool are always taken after the database copypool copies.

- **DATA ONLY:**

  This type of copy only contains the “database” COPYPOOL. A BACKUP SYSTEM DATA ONLY does NOT copy the “log” COPYPOOL. Such a copy is referred to as a *data-only system backup*. Data-only system backups can be used in conjunction with the new RESTORE SYSTEM utility to recover the system to an arbitrary PIT.

These system backups (full and data-only) are recorded in the BSDS (up to 50 entries) and the header page of DBD01. The information in DBD01 is called the *Recovery Base Log Point (RBLP)* and is the RBA (non-data sharing) or LRSN(data sharing - the RBLP LRSN is determined by taking minimum of all member level RBLP values) of the time the most recent system backup ran. The information in DBD01 is recorded before the actual copy is initiated. This way it can be used as a starting point for log processing after a backup has been restored. The information in the BSDS is recorded after receiving word back from DFSMShsm that the copies are “logically” complete.

BACKUP SYSTEM invokes DFSMShsm services (so-called fast replication services, in this case the FRBACKUP command) to either take a data-only or a full system backup via DASD volume copy functions. The backups may (and probably do) contain uncommitted data. This is not a problem. the data is brought back to consistency (no outstanding in-flight, in commit, or in abort units of work by a DB2 restart operation or by using the RESTORE SYSTEM utility. While the BACKUP SYSTEM utility is active in the system, certain DB2 system activities are quiesced:

- System checkpoints
- 32K page writes (for page sets with a VSAM CISZE that is different from the DB2 page size)
- Writing page set close control log records (PSCRs)
- Data set creation, extensions, renaming, and deletion

However, the log write latch is not obtained by the BACKUP SYSTEM utility, as is done by the SET LOG SUSPEND command. Therefore, using the BACKUP SYSTEM utility should be less disruptive in most cases. There is no need to suspend logging when using the BACKUP SYSTEM utility, as DB2 is now in control of taking the volume copies, whereas you are in charge to initiate copying the data, in case of using -SET LOG SUSPEND.

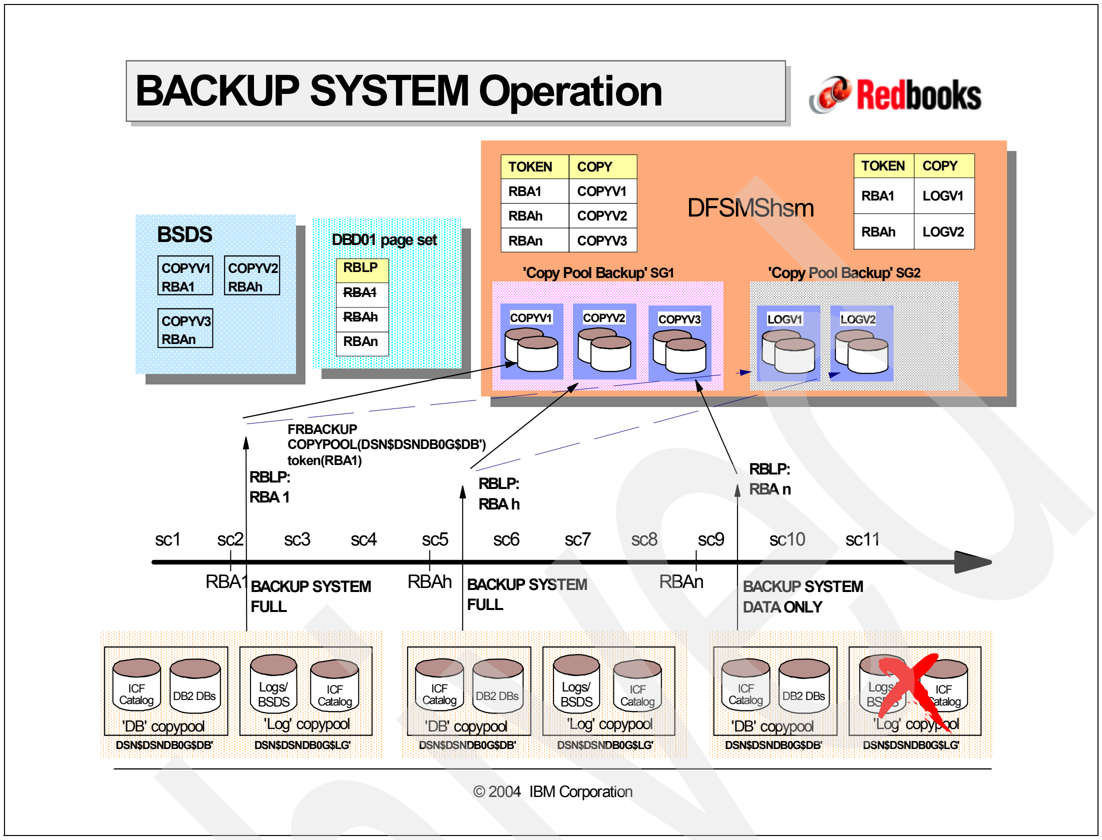
In a data sharing environment, the BACKUP SYSTEM utility will fail, when it detects any member that is in a “failed” or “not normally quiesced” state.

The BACKUP SYSTEM utility completes after the “logical” copies have completed. This should typically be within a few seconds.Taking copies of the volumes in the COPYPOOL is done in parallel.

Here is the sequence of events that occur when running the BACKUP SYSTEM utility, as shown in the figure above. The example uses BACKUP SYSTEM FULL, so we take a backup of both the database and the log copy pool.

1. Preparation: To make sure there is only one BACKUP SYTEM job running in the system at any one time, DB2 uses a new lock type to serialize the execution of BACKUP and RESTORE SYTEM utilities. So the first thing to do is to obtain that lock in exclusive mode. DB2 also has to drain some minimal system activity, to maintain system integrity. (They are listed earlier in this section). Therefore, the next thing to do is to quiesce those system activities.
2. Update the header page of DBD01 to update the Recovery Base Log Point (RBLP). The RBLP value is based on system checkpoints in non-data sharing and system checkpoints and group buffer pool checkpoints in a data sharing environment.
3. Invoke the DFSMShsm fast replication function to make a volume-level backup of all the volumes in the database copy pool.
4. Update the BSDS with the with the system copy information, recording the RBLP that is associated with the copy.
5. Invoke the DFSMShsm fast replication function to make a volume-level backup of all the volumes in the log copy pool, because we specified the FULL option when invoking the BACKUP SYSTEM utility.

6. Free up the system activities there were quiesced at the beginning of the utility invocation, release the exclusive lock, and put out the message DSNU1602I “BACKUP SYSTEM UTILITY COMPLETED”.

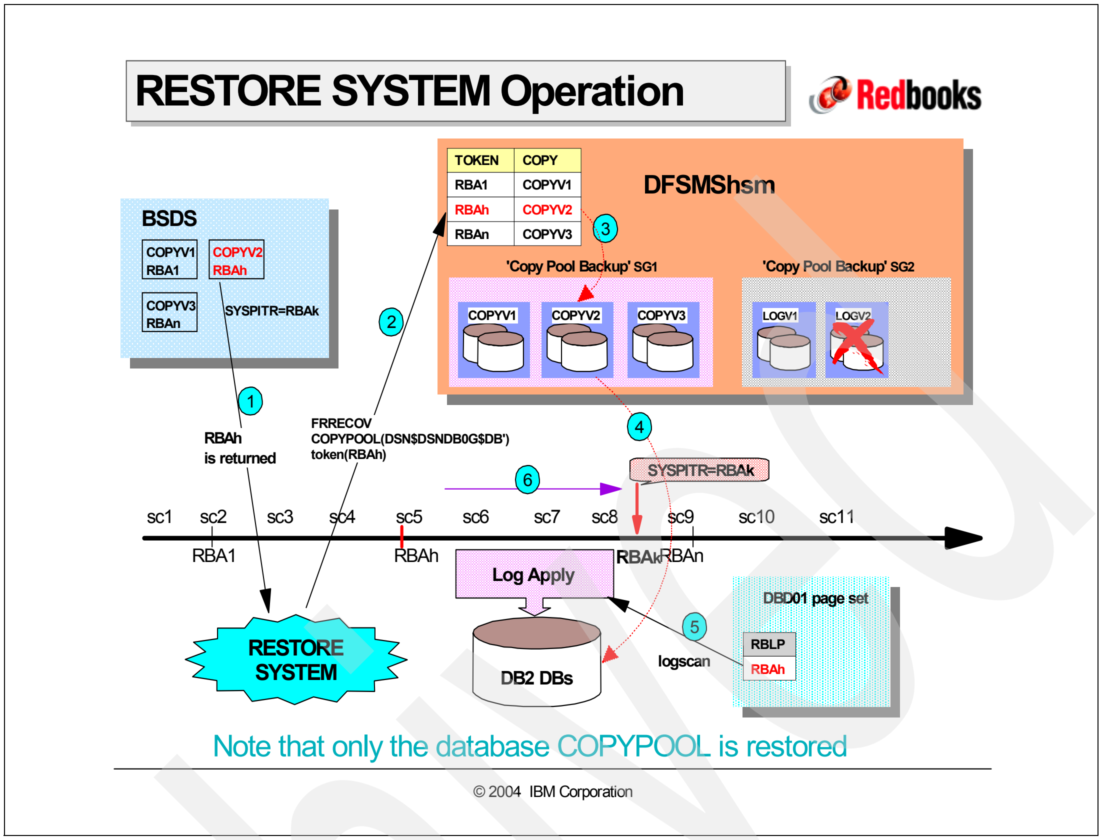


3.16.3 BACKUP SYSTEM operation

The figure above shows what happens when several BACKUP SYSTEM utilities are executed. The first and the second execution are BACKUP SYSTEM FULL jobs. This means that both the data base copy pool and log copy pool are backed up each time (COPYV1 and LOGV1) and (COPYV2 and LOGV2). The third execution is a BACKUP SYSTEM DATA ONLY execution. This means that only the data base copy pool is backed up (COPYV3).

DFSMShsm tracks all copies that are made of any copy pool (up to 85), and identifies them by a token (that is passed to it by DB2, via the BACKUP SYSTEM utility). It is worth noting that today all backup versions of a copy pool have to reside on disk.

Note that all the BACKUP SYSTEM executions (up to a maximum of 50) are tracked in the BSDS, where as the RBLP in DBD01 is updated (overwritten) each time the BACKUP SYSTEM is run.



3.16.4 RESTORE SYSTEM operation

The figure above shows how a RESTORE SYSTEM utility operates. In this example, the LOGONLY option is not used. Before we describe this example, we give a general description of how RESTORE SYSTEM works.

Depending on the type of system backup that is available, and the PIT that you want to go back to, you have different options to restore the system.

Restoring a DB2 system to the PIT of a prior system level backup

To restore a DB2 system to the PIT of a prior system level backup, you need to have a copy of both the databases and the logs.

1. Obtain a full system backup. This can either be done through:
   - The execution of a BACKUP SYSTEM FULL utility, or
   - Creating the copies manually, via:
     i. Issuing a SET LOG SUSPEND to quiesce system activity.
     ii. Taking backups of all DB2 data, as well as the logs. This can be done by using COPYPOOLs and issuing the DFSMShsm commands manually
     iii. Issuing the SET LOG RESUME command.
2. In case you need to restore the system, stop DB2. In case of a data sharing group, stop all members.
3. Use HSM commands to restore the database and log copypools:

```
FRRECOV * COPYPOOL(cpname) VERIFY(Y) Token(X'C4C2D7F1B7EF7EA8AD2DBE0E000007A5E090')
```

4. If data sharing, delete the CF structures used by DB2
5. Restart DB2. For a data sharing system, restart all non-dormant members. During restart all in-flight, in-abort and in-commit URs get resolved.
6. If data sharing, execute GRECP/LPL recovery. This recovers changed data that was stored in the CF at the time of the backup.

Note that in this scenario, we do not use the new RESTORE SYSTEM utility.

Restoring a DB2 system to arbitrary PIT

To restore the system to an arbitrary point-in-time, it is sufficient to have DATA ONLY backups available. Of course, you can also use FULL system backups. Both are created using the BACKUP SYSTEM utility.

This procedure uses the RESTORE SYSTEM utility. To be able to run the RESTORE SYSTEM utility, the system has to be in *System Recover Pending* mode. A DB2 system goes into System Recover Pending mode after a conditional restart is performed, using a special type of conditional restart record called a “SYSPITR” conditional restart control record (CRCR).

It is your responsibility to find the point-in-time that you want to go back to, and that LRSN-RBA is specified in the SYSPITR conditional restart record.

DB2 V8 has been enhanced and allows you to specify an exact RBA as an RBA to recover to, in a non-data sharing environment. In previous DB2 versions, you had to round up the “log truncation RBA” to the next higher CI boundary during a conditional restart. (In a data sharing environment you could already specify an exact LRSN as a “log truncation” point.)

In DB2 V8 you can specify an exact RBA that you want to use during a conditional restart as a “log truncation point”, or as an RBA for a SYSPITR conditional restart record. If the RBA value specified equals the RBA of a logical log record, the log is truncated at the end of this log record. If the RBA value specified does not equal the beginning of a logical log record, issue message DSNJ096E and fail the restart, as this is an indication that the specified RBA value is incorrect.

The RBA value specified with the ENDLRSN parameter is an “inclusive” value as opposed to the RBA (on a 4K CI boundary) specified with the ENDRBA parameter. Notice that when using an exact RBA for a normal conditional restart, you use the ENDLRSN parameter (whereas you use the ENDRBA keyword when specifying a RBA that is at a 4K CI boundary). When doing a conditional restart for system level point-in-time recovery, you always use the SYSPITR keyword.

You also have to make sure that all logs (active and archive), between the time of the system level backup and the SYSPITR that you want to recover to, are available to the RESTORE SYSTEM utility.

Creating a SYSPITR CRCR and performing a conditional restart

A CRCR record is, as always, created using the Change Log Inventory (DSNJU003) utility. A new option called “SYSPITR” is added to the CRESTART keyword. The syntax is:

CRESTART CREATE SYSPITR=*log-point*

Here, *log-point* is an RBA (non-data sharing) or an LRSN (data sharing) that represents the PIT to which the system is to be recovered.

After the conditional restart completes, the system enters into System Recover Pending mode. This means that:

- Only the RESTORE SYSTEM utility is allowed to execute.
- The DB2 data remains unavailable until the RESTORE SYSTEM utility has completed.
- DB2 has to be recycled to reset the System Recover Pending mode.

In a data sharing environment, each non-dormant member needs to be restarted with the PITR CRCR, with all members specifying the same log truncation LRSN. After the conditional restart of all members is complete, you can run the RESTORE SYSTEM utility on (only) one of the members. This utility has to start and complete on the same member. It cannot be restarted on a different member. After RESTORE SYSTEM successfully completes, you must bring down all members to reset the System Recover Pending mode.

Executing the RESTORE SYSTEM utility

As mentioned before, the RESTORE SYSTEM utility can only be executed when the system is in System Restore Pending mode, or on a DB2 Tracker site.

You can specify a single option on the **RESTORE SYSTEM** utility:

- **No option specified:**

  In this case, RESTORE SYSTEM first restores the version of the database copypool (created by the BACKUP SYSTEM utility) that was taken immediately prior to the specified logpoint (of the PITR CRCR). Then it recovers from that point onwards using the log.

- **LOGONLY:**

  By using this option, you indicate that the database volumes have already been restored and the restore phase will be skipped. In this case, the RESTORE SYSTEM utility will only apply outstanding log changes to the databases. This option is useful if the user (or some automation tool) has already restored the database volumes prior to invoking the RESTORE SYSTEM utility. RESTORE SYSTEM LOGONLY uses the recovery base log point (RBLP) that is recorded in DBD01 to determine the starting point of the log apply phase (just like RESTORE SYSTEM does). Note that the RBLP is only updated by the BACKUP SYSTEM utility and the -SET LOG SUSPEND command. Therefore, if you take flashcopies without a prior -SET LOG SUSPEND command, RESTORE SYSTEM LOGONLY may go back much farther in time for the start of the log apply phase than you expected.

Note: The RESTORE SYSTEM utility does not restore the log COPYPOOL. The ability to back up logs with a FULL backup is designed to allow for the recovery of an entire subsystem, via means other than the RESTORE SYSTEM utility. Such a recovery can be realized by direct invocation of DFSMShsm services, as shown in “Restoring a DB2 system to the PIT of a prior system level backup” on page 198.

As indicated above, each COPYPOOL can have multiple VERSIONS. You cannot specify a specific desired version, other than implicitly via the log truncation point (specified as the PITR CRCR). When you specify the log truncation point, you have determined the version of the COPYPOOL that will be used. RESTORE SYSTEM automatically recovers from the latest version prior to the log truncation point. The restore of the database volume from the COPYPOOL is done in parallel.

After the data is restored, the RESTORE SYSTEM utility uses the recovery base log point (RBLP) that is recorded in DBD01 to determine the starting point of the log apply phase. The log apply phase uses the fast log apply (FLA) function to recover objects in parallel.

The consistency for LOG NO utilities is established when, during the logapply phase of RESTORE SYSTEM, a log record is encountered that represents the open of a table space or index space with recovery(no). In this case, table spaces will have to be put in RECP state, and index spaces will have to be put in either RECP or RBDP state, depending on their COPY attribute. These objects should be recovered to a different point-in-time, prior to the log truncation point using image copies, or rebuilt from the data in the case of a COPY NO index.

Note: With DB2 Version 8, system level point-in-time recovery is an all-or-nothing approach, as it is operating at the subsystem level or at the data sharing group level. This architecture might be extended in the future to handle backup and recovery at a more granular level, such as an application or a single table space.

RESTORE SYSTEM example

In the figure above, we show the sequence of events for a RESTORE SYSTEM utility in a non-data sharing environment.

Before you can execute the RESTORE system utility, you must first determine the point-in-time you want to go back to. This is no different from what you need to do today for any type of conditional restart or point-in-time recovery. You have to find a point-in-time where the system was OK. Assume we determine that at RBAk all is still OK, and we decide to return to that point-in-time. Then you have to create a “SYSPITR” conditional restart record specifying RBAk, and restart DB2. DB2 will start in System Recover Pending mode, and the only thing you can do is execute the RESTORE SYSTEM utility.

1. The first thing RESTORE SYSTEM does is to look at the point-in-time you want to recover to, as indicated by the SYSPITR CRCR RBA/LSRN (RBAk), and look for the backup system entry in the BSDS that immediately precedes that point-in-time (RBA/LRSN). We find that RBAh is the last BACKUP SYSTEM run. (We have a more recent BACKUP SYSTEM utility execution available at RBAn, but that is beyond RBAk that we want to return to, and therefore cannot be used.
2. Invoke DFSMShsm functions to restore the version of the data base copy pool that is associated with the entry that is retrieved from the BSDS.
3. DFSMShsm will analyze the version token passed to it from DB2 and locate the appropriate backup in its copy pool backup.
4. DFSMShsm will then invoke the fast replication functionality to restore all the volumes belonging to the database copy pool in parallel.
5. Now that the database copy pool has been restored, DB2 can look at the header page in DBD01 that has just been restored, to retrieve the RBLP (RBAh). This is the starting point of the forward log recovery.
6. The RESTORE SYSTEM then starts the log apply phase and uses the fast log apply (FLA) function to recover all objects in parallel from the log, up to the point-in-time (RBA/LRSN) specified by the conditional restart (RBAk).

DB2 Tracker support

DB2 Version 5 introduced so-called “Tracker site” support, as a way to DB2 disaster recovery.

RESTORE SYSTEM LOGONLY can also be used on the tracker site to establish a recovery cycle. In case you run the RESTORE SYSTEM LOGONLY utility at the tracker site, the tracker site does not have to be in “system recover pending mode”. Note also that RESTORE SYSTEM LOGONLY, running at the tracker site, updates the RBLP (so it knows where to start the next recovery cycle).

A DB2 tracker site is a separate DB2 subsystem, or data sharing group, that exists solely for the purpose of keeping shadow copies of your primary site's data. No independent work can be run on the tracker site. From the primary site, you transfer the BSDS and the archive logs, and the tracker site runs periodic LOGONLY recoveries to keep the shadow data up-to-date.

If a disaster occurs at the primary site, the tracker site becomes the takeover site. Because the tracker site has been shadowing the activity on the primary site, you do not have to constantly ship image copies. The takeover time for the tracker site might be faster because DB2 recovery does not have to use image copies.

For more information, see "Using a tracker site for disaster recovery" in *DB2 Administration Guide,* SC18-7413.

More online ZPARMs

Additional online changeable ZPARMs in DB2 V8:

DSN6SPRM		DSN6SYSP	DSN6FAC
CHGDC	PARTKEYU	EXTRAREQ	RESYNC
EDPROP	SRTPOOL	EXTRASRV	MAXTYPE1
SYSADM	XLKUPDLT		IDTHTOIN
SYSADM2	MAXKEEPD		POOLINAC
SYSOPR1			TCPKPALV
SYSOPR2		**DSN6GRP**	TCPALVER
CACHEDYN		IMMEDWRI	

3.17 More online ZPARMs

In order to minimize the events which make it necessary to recycle your DB2 subsystem to make ZPARM parameter changes effective, the online change for subsystem parameters was introduced in DB2 V7. This function allows a user to load a new subsystem parameter load module into storage without recycling DB2. To do this, you can use the normal installation parameter update process to produce a new load module, which includes any desired changes to parameter values. You can then issue the -SET SYSPARM command to load the new module in order to affect the change.

Not all subsystem parameters are dynamically changeable in DB2 V7. Refer to *DB2 Universal Database for OS/390 and z/OS Command Reference version 7*, Appendix A.2, SC26-9934 for a list of online changeable subsystem parameters. An overview is shown in Table 3-3.

DB2 V8 adds some more online changeable parameters. *Refer to DB2 UDB for z/OS Version 8 DB2 Command Reference,* SC18-7416. Actually all new DSNZPARMs introduced in Version 8 are online changeable.

Table 3-3 Subsystem parameters online changeable for V8

Parameter	Macro	Panel	Panel Field
CHGDC	DSN6SPRM	DSNTIPO	DPROP Support
EDPROP			
SYSADM		DSNTIPP	System Admin 1
SYSADM2			System Admin 2
SYSOPR1			System Operator 1
SYSOPR2			System Operator 2
CACHEDYN		DSNTIP4	Cache dynamic SQL
SRTPOOL		DSNTIPC	Sort pool size
XLKUPDLT		DSNTIPI	X lock for searched U/D
MAXKEEPD		DSNTIPE	Max kept dynamic stmt
PARTKEYU		DSNTIP4	Update part key cols
RESYNC	DSN6FAC	DSNTIPR	Resync interval
MAXTYPE1			max type 1 inactive
IDTHTOIN			Idle thread time-out
POOLINAC		DSNTIP5	Pool thread time-out
TCPKPALV			TCP/IP keepalive
TCPALVER			TCP/IP already verified
IMMEDWRI	DSN6GRP	DSNTIP4	Immediate write
EXTRAREQ	DSN6SYSP	DSNTIP5	Extra query blocks - DB2 Requester
EXTRASRV			Extra query blocks - DB2 Server

For most parameters, online change is transparent, with the change taking effect immediately. There are a few parameters for which this is not the case. The behavior exhibited by the system upon change for these parameters is discussed below.

PARTKEYU

After you change the partitioning key columns update parameter from YES to NO, from YES to SAME, or from SAME to NO, your attempt to inappropriately update the value in a partitioning key column results in SQLCODE -904, resource unavailable.

Important: In addition, when PARTKEYU is set to YES, in DB2 Version 8, drain locks are no longer acquired when moving between partitions after updating the partitioning key.

SYSADM/SYSADM2

You can only change the install SYSADM (SYSADM and SYSADM2) subsystem parameters using the SET SYSPARM command using the current install SYSADM authority. If you do not have the proper authority, DB2 issues message DSNZ015 for each ID you tried to change. In this case, the current value of the load module parameter remains in effect. The rest of the load module is not affected, that is, all other changes will take effect.

As with changing this parameter offline, revoking SYSADM authorization from the install SYSADM user ID, does not cascade. However, if you REVOKE SYSADM from the user that was just replaced as an install SYSADM, then that REVOKE will cascade.

To perform an online update of SYSADM/SYSADM2 system parameters, use either a primary or secondary authorization ID having install SYSADM authority.

SYSOPR1/SYSOPR2

You can only change the Install SYSOPR (SYSOPR1 and SYSOPR2) subsystem parameters using the SET SYSPARM command if either your primary or any secondary authorization ID has install SYSADM authority. If you do not have the proper authority, DB2 issues message DSNZ015 for each ID you tried to change. In this case, the current value of the load module parameter remains in effect. The rest of the load module is not affected, that is, all other changes will take effect.

CACHEDYN

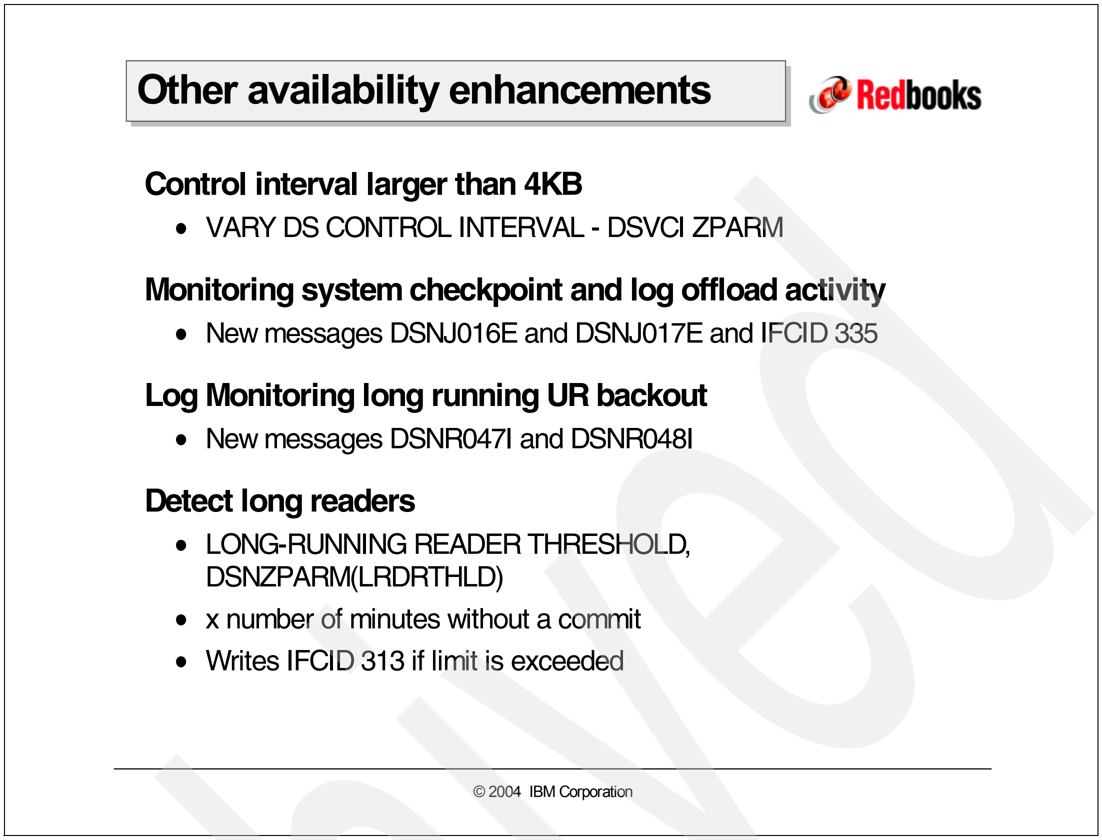
When you change this value from YES to NO, existing statements in the cache will not be used by new threads, but the statements will remain in the pool until they are no longer referenced. Therefore, changing this value to NO, you cannot immediately reduce the EDM pool size allocated for dynamic statements.

MAXKEEPD

If you change the value for MAXKEEPD, those changes take effect after the next COMMIT.

XLKUPDLT

Your changes to XLKUPDLT will not affect the previous settings of currently running statements. All future statements coming in are affected immediately.



3.18 Other availability enhancements

DB2 V8 also takes actions to increase availability, allowing you to specify a CIZIZE that matches your DB2 page size, as well as trying to anticipate and warn you about potential availability problems, and take more automatic actions to reduce the outage to a minimum. The following enhancements were made to further increase availability:

- Control interval larger than 4 KB
- Monitoring system checkpoint activity and log offload activity
- Monitoring long running unit of recovery (UR) backout
- Detecting long readers
- Lock escalation IFCID
- Partitioning key update enhancements
- Automatic and less disruptive LPL recovery
- SMART DB2 extent sizes for DB2 managed objects
- Logging manageability enhancements

3.18.1 Control interval larger than 4KB

DB2 V8 introduces support for CI sizes of 8, 16, and 32 KB, activated by the default of a new DSNZPARM, DSVCI, in panel DSNTIP7. This is valid for user-defined and DB2 stogroup-defined table spaces. Index spaces only use 4 KB pages. If you decide to activate the new CI sizes, once you are in new-function mode (NFM), all new table space page sets will be allocated by DB2 with a CI corresponding to the page size. The page sets already existing at the time of migration to DB2 V8 will be later converted by the first execution of LOADs or REORGs. When using DB2 user-defined objects (USING VCAT), it is your responsibility to define the underlying VSAM cluster with the correct CISIZE.

The DB2 install procedure will also prepare the correct JCL for the user-defined DB2 catalog table spaces, and will convert them to the new page size during the enabling-new-function mode (ENFM) phase.

The new CI sizes reduce integrity exposures, and relieves some restrictions on concurrent copy (of 32 KB objects) and the use of striping (of objects with a page size > 4KB). This enhancement can potentially reduce elapsed time for table space scans.

3.18.2 Monitoring system checkpoints and log offload activity

With DB2 for OS/390 and z/OS V7, you have the chance to monitor the actuality of system checkpoints and the status of the log offload activity using DB2 command -DISPLAY LOG, which was introduced in DB2 for OS/390 V6. Sometimes, however, there are situations where DB2 stops taking system checkpoints or gets stuck during its log offload activity. Since both situations might cause problems during operations, DB2 for z/OS V8 provides you with new messages, which help you to identify problems faster. In addition, an IFCID 335 record will be produced if the statistics class 3 trace is active:

- `DSNJ016E csect-name WARNING -SYSTEM CHECKPOINT PROCESSOR MAY`
  `HAVE STALLED. LAST CHECKPOINT WAS TAKEN date-time`
- `DSNJ017E csect-name WARNING -OFFLOAD TASK HAS BEEN ACTIVE SINCE`
  `date-time AND MAY HAVE STALLED`

3.18.3 Log Monitoring long running UR backout

During V7 *restart processing*, DB2 issues progress messages during the forward and backward phase of the restart. Message DSNR031I is issued in 2-minute intervals, showing the current log RBA being processed and the target RBA to complete the restart phase.

With DB2 UDB for z/OS V8, a new progress message is issued during the *backout of postponed abort URs* as well. This message is first issued after the process has at least exceeded one two-minute interval and is repeated every two minutes until the backout process is complete:

- `DSNR047I csect-name POSTPONED ABORT BACKOUT PROCESSING LOGRECORD`
  `AT RBA rba1 TO RBA rba2`

For *in-abort URs*, there is also a new progress message. As for DSNR047I, the new message DSNR048I is also issued every two minutes:

- DSNR048I message:

```
DSNR048I csect-name UR BACKOUT PROCESSING LOG RECORD
                    AT RBA rba1 TO RBA rba2 FOR
                    CORRELATION NAME =xxxxxxxxxxxx
                    CONNECTION ID =yyyyyyyy
                    LUWID =logical-unit-of-work-ID=token
                    PLAN NAME =xxxxxxxx
                    AUTHID =xxxxxxxx
                    END USER ID =xxxxxxxx
                    TRANSACTION NAME =xxxxxxxx
                    WORKSTATION NAME =xxxxxxxx
```

3.18.4 Detecting long readers

When looking for "bad" transactions in your system, there are three types of transactions that you want to watch out for:

- Long running updaters that do not commit
- Transactions that generate a lot of logging, that potentially needs to be undone in case of a rollback
- Long running readers that do not commit

In V5, DB2 introduced the UR CHECK FREQ (URCHKTH) DSNZPARM. When a unit of recovery (UR) has been active for more than the number of system checkpoints specified in this DSNZPARM, DB2 produces a DSNR035I message.

In V7, DB2 added another DSNZPARM (URLGWTH) UR LOG WRITE CHECK. DB2 will produce a DSNJ031I message, when a UR produces more than URLGWTH log records without committing. This check has more granularity than UR CHECK FREQ.

V8 takes this one step further by also alerting you when you have a long-running reader in the system. It is probably less-well known that not only updates have to commit frequently. Also read-only transactions should commit regularly. This is to allow utilities (that need to drain the object) to take control of the object, for example, during the switch phase of online REORG, the utility needs full control of the object to switch between the shadow data set and the actual data set used by DB2.

DB2 for z/OS Version 8 introduces another DSNZPARM (LRDRTHLD), LONG-RUNNING READER THRESHOLD. When you specify a non-zero value, DB2 records the time that a task holds a read claim. For parallel operations, DB2 records this information only for the originating task. When a task holds a claim longer than the number of minutes specified in LRDRTHLD, and for each multiple of that number of minutes, DB2 writes an IFCID 313 record. IFCID 313 contains the total number of minutes that the claim has been held. No console message is displayed.

Other availability enhancements - 2

Redbooks

Lock holder can inherit WLM priority from lock waiter

Lock escalation IFCID 337

Partitioning key update enhancements

Improved LPL recovery

SMART DB2 extent sizes for DB2 Managed object

- MGEXTSZ - Enable Sliding Secondary Quantity
- TSQTY / IXQTY - specify default primary allocation for TS / IX

Logging manageability enhancements

- Allow cancellation of a rollback operation after a log data set access error
- -DISPLAY ARCHIVE command enhancements
- Automatic scan for prior checkpoint during point in time recovery with conditional restart

3.18.5 Lock holder can inherit WLM priority from lock waiter

Assume that a transaction executes with a low WLM priority and makes updates to the database. Because it is updating, it must acquire X-locks on the pages that it modifies and then these X-locks are held until the transaction reaches a commit point. This is business as usual. If such a transaction holds an X-lock on a row that another transaction is interested in, the other transaction is forced to wait until the first transaction (with a low WLM priority) commits. If the lock waiter performs very important work and is thus assigned a high WLM priority, it would be desirable that it is not slowed down by other transactions that execute in a low priority service class.

To reduce the time the high priority transaction has to wait for that lock to be given up by the low priority transaction, it would be better if the waiting transaction could temporarily assign its own priority to the transaction that holds the lock until this transaction frees the locked resource.

The WLM component of z/OS 1.4 provides a set of APIs that can be used to accomplish this. This service is called *WLM enqueue management*. In z/OS 1.4, this support is limited to transactions running on a single system.

DB2 V8 exploits WLM enqueue management. When a transaction has spent roughly half of the lock time-out value waiting for a lock, then the WLM priority of the transaction, which holds the lock, is increased to the priority of the lock waiter if the latter has a higher priority. If the lock holding transaction completes, it resumes its original service class. In case multiple transactions hold a common lock, this procedure is applied to all of these transactions.

3.18.6 Lock escalation IFCID

Many installations put a lot of work into avoiding or minimizing lock escalation, as it can be very disruptive to the concurrency in your system. However, DB2 V7 does not produce an IFCID record when lock escalation occurs, thus making it difficult for performance monitors to report on lock escalation.

Therefore, today most people monitor console message DSNR031I. It is produced whenever lock escalation occurs. However, performance monitors prefer a single interface for all performance related information through the DB2 Instrumentation Facility Component.

For this reason, DB2 Version 8 introduces a new IFCID 337 to report on lock escalations. IFCID 337 is added to the statistics trace class 3 and performance trace class 6. If activated, IFCID 337 is written whenever lock escalation occurs. The record contains a superset of the information that is reported in message DSNI031I.

Message DSNI031I is enhanced to include the collection ID in order to be consistent with IFCID 337.

```
DSNI031I csect-name - LOCK ESCALATION HAS OCCURRED FOR RESOURCE NAME = name, LOCK STATE
= state, PLAN NAME:PACKAGE NAME = id1:id2, COLLECTION-ID = id11, STATEMENT NUMBER
= id3, CORRELATION-ID = id4, CONNECTION-ID = id5, LUW-ID = id6, THREAD-INFO =
id7:id8:id9:id10
```

For more information about the content of IFCID 337, see C.1.4, “Lock escalation trace record” on page 1047.

3.18.7 Partitioning key update enhancements

Since DB2 V5, an enhancement introduced via the maintenance stream allowed you to update the partitioning key. However, updating the partitioning key can have a severe impact on availability (because of reduced concurrency during the update operation) when updates occur frequently, and the keys need to be moved from one partition to another. If the update requires moving the data row from one partition to another, DB2 V7 tries to take exclusive control of certain parts of the objects to perform the update by acquiring DRAIN locks. Because of this, no other application can access the range of partition affected by the update of values in the partitioning key columns. Following is the list of objects on which DB2 V7 takes exclusive control to perform the update of a partitioning key that causes a move of the data from one partition to another:

- The partition of the table space from which the row is moving, the partition of the table space to which the row is moving, and all partitions in between
- The partition of the partitioning index from which the partitioning key is moving, the partition of the partitioning index to which the partitioning key is moving, and all partitions in between
- Non-partitioning indexes defined on the table space

DB2 V8 no longer needs to take exclusive control of the objects to perform the update operation. Concurrency is allowed if an update of a partitioning key changes the partition to which the row belongs.

3.18.8 Improved LPL recovery

As stated before, the need for higher and higher availability of data residing in DB2 for OS/390 and z/OS subsystems is growing all the time. One source of unplanned outages that some of you encounter from time to time is that some data is not accessible because the object has been placed into logical page list (LPL) status.

The logical page list is a list of pages that are logically in error. They cannot be referenced until the pages are recovered. For example, DB2 typically puts pages into the LPL when I/O or coupling facility read or write errors are encountered.

DB2 V8 introduces several enhancements to LPL recovery.

Reason ID for adding a page to the LPL

When pages are added into the LPL, DB2 issues message DSNB250E to tell the page range, database and pageset/partition names of the LPL pages. DB2 V8 adds the reason why the pages are added in the LPL to this message.

Automatic recovery of LPL pages

After the pages are added in the LPL, DB2 V8 attempts to initiate the automatic LPL recovery processor. If the automatic LPL recovery completes successfully, the pages are deleted from the LPL and DB2 issues an existing message, DSNI021I, to indicate the completion.

If the automatic LPL recovery fails, DB2 V8 issues message DSNI005I to indicate the failure of the automatic LPL recovery.

A new message, DSNB357I is issued to inform you about the fact that pages have been added to LPL, which might not be automatically recovered.

Less disruptive LPL recovery

Even more important than automatic LPL recovery is that with DB2 V8, during LPL recovery all pages that are not in the LPL are accessible by SQL. Up to DB2 V7, the entire table space or partition was unavailable during LPL recovery.

The LPL recovery processor (by way of the START DATABASE command or the new automatic LPL recovery feature), makes a write claim instead of a drain on the object that is being recovered. As a result, “good pages” in the object are available to SQL users, and performance is improved because the claim is less disruptive than a drain.

More details can be found in 11.3, “Improved LPL recovery” on page 863.

3.18.9 SMART DB2 extent sizes for DB2 managed objects

DB2 V8 adds a new DSNZPARM called MGEXTSZ (with a global scope) which enables the use of a “sliding secondary quantity” for DB2 managed page sets (stogroup defined). The range of values are: NO, YES. The default value for MGEXTSZ is NO.

The idea is that DB2 will adjust the size of the secondary extent, when extending the data set, in such a way that the maximum allowable data set size is reached, before the maximum number of extents for that data set (255) is reached.

The MGEXTSZ parameter only impacts the allocation of secondary extents. When MGEXTSZ=YES, when calculating secondary quantities, DB2 uses a sliding scale. With the sliding scale, the initial secondary allocations are small, and later secondary allocations are larger.

If you want large secondary quantities, starting with the first secondary allocation, specify a large SECQTY value in CREATE TABLESPACE or CREATE INDEX. When the value you specify as SECQTY is more than what DB2 calculates as the “sliding quantity”, DB2 honors the specified SECQTY from the CREATE or ALTER TABLESPACE or INDEX.

Using the ALTER TABLESPACE and ALTER INDEX statements, you can change the PRIQTY and SECQTY values to -1, to let DB2 use the default primary space allocation and calculate secondary space allocations using a sliding scale.

If the value of subsystem parameter MGEXTSZ is NO, DB2 always uses that SECQTY value (or its default value).

Two new DSNZPARMS (TSQTY and IXQTY), that were introduced in DB2 V6 and V7 by the PTF for APAR PQ53067, can be specified on the DB2 installation panels in V8:

- TSQTY specifies the number of space in KB for the *primary space allocation quantit*y for DB2-managed table spaces that are created without the USING clause. A value of 0 (zero) indicates that you want to use standard defaults (now 1 cylinder).
- IXQTY fulfils a similar role but for index spaces.

TSQTY and IXQTY have a global scope. TSQTY applies to non-LOB table spaces. For LOB tables paces a 10x multiplier will be applied to TSQTY to provide the default value for PRIQTY. IXQTY applies to all indexes.

The default value for TSQTY and IXQTY remains 0 (zero) as in previous versions. However, zero now (V8) means a primary allocation of 1 cylinder for both table spaces and indexes. (When using the installation clist in migration mode, DB2 will draw your attention to this change in allocation when specifying zero, by displaying message “DSNT520I DB2’s default sizes for index and table spaces are increased in V8”.)

DB2 always honors the PRIQTY value specified by the user and recorded in the associated PQTY column in SYSTABLEPART or SYSINDEXPART catalog table.

This enhancement should benefit all users of DB2. It is particularly beneficial for ERP/CRM, ported and new applications. For example, when implementing an ERP/CRM solution, it is difficult to know which objects will be used by the functions that you will exploit in the CRM/ERP package. In addition, it is often difficult for any type of application to have a good understanding of the object sizes at design time.

With this enhancement, in a lot of cases, there is no longer the need for a user to specify the secondary quantity for a DB2-managed pageset, or even both the primary and secondary quantities. The methodology used will not lead to heavy over-allocation and waste excessive space. For example, it will not exceed DSSIZE and PIECESIZE.

This feature delivers autonomic selection of data set extent sizes with a goal of preventing extent errors before reaching maximum data set size.

The new default allocations in cylinders, as well as the sliding (bigger) secondary allocations, can also result in better performance of mass inserts, prefetch operations, and the LOAD, REORG and RECOVER utilities.

3.18.10 Logging manageability enhancements

In this section we discuss another set of enhancements that make it easier for you to work with DB2 log data.

Allow cancellation of a rollback after a log data set access error

DB2 Version 7 introduced the ability to cancel backout processing for in-abort and postponed-abort units-of-recovery (URs) using the -CANCEL THREAD NOBACKOUT and -RECOVER POSTPONED CANCEL commands. V7 also added the ability to “retry” for must-complete log data set access errors. The retry option allows you to correct the log access error, and then reply to a WTOR.

Because of HSM recalls and automatic tape subsystems, long-running rollbacks may not be detected until an unrecoverable log data set access error occurs, for example an archive data set has been deleted. APAR/PTF PQ88983/UQ89479 adds the ability to cancel backout processing when a log data set access error can not be corrected.

In case a log data set access error occurs during backout processing (regular backout or during postponed-abort processing) that can not be corrected, you can now use the -CANCEL THREAD NOBACKOUT or -RECOVER POSTPONED CANCEL commands, before replying to the outstanding WTOR.
After replying to the WTOR, log manager detects that a log-read request has been canceled, and passes back a new reason code to the log-read requestor indicating the request has been canceled. The log-read requestor (ABORT or the -RECOVER POSTPONED command processor) can then complete the canceled backout processing.

An example using -CANCEL THREAD NOBACKOUT is shown in Example 3-4.

Example 3-4 Cancelling a thread after a log data set access error

```
DSNJ104I = DSNJR206 RECEIVED ERROR STATUS 00000004
           FROM DSNPCLOC FOR DSNAME=DSNC810.ARCHLOG1.A0000045
DSNJ104I = DSNJR206 RECEIVED ERROR STATUS 00000004
           FROM DSNPCLOC FOR DSNAME=DSNC810.ARCHLOG2.A0000045
*DSNJ153E = DSNJR006 CRITICAL LOG READ ERROR
                     CONNECTION-ID = TEST0001
                     CORRELATION-ID = CTHDCORID001
                     LUWID = V81A.SYEC1DB2.BB246F9D6799=10
                     REASON-CODE = 00D10345
*26 DSNJ154I = DSNJR126 REPLY Y TO RETRY LOG READ REQUEST, N TO ABEND

=CAN THD(10) NOBACKOUT

DSNV426I = DSNVCT THREAD Œ10Œ HAS BEEN CANCELED
R 26,Y
IEE600I REPLY TO 26 IS;Y
DSNJ151I = DSNJR003 IN READING RBA Œ00000DA18000Œ A
           NON ZERO REASON CODE WAS RETURNED. CONNECTION-ID=TEST0001,
           CORRELATION-ID=CTHDCORID001, REASON-CODE=00D10319, MEMBER-ID=0
DSNB250E = DSNILABR A PAGE RANGE WAS ADDED TO THE
           LOGICAL PAGE LIST
           DATABASE NAME=TESTDB
           SPACE NAME=TESTS
           DATA SET NUMBER=1
           PAGE RANGE XŒ00000000Œ TO XŒ FFFFFFFFŒ
           START LRSN=XŒ00000CFEDBCAŒ
           END LRSN=XŒ0000101961E4Œ
           START RBA=XŒ00000CFEDBCAŒ
           LPL TRACE ID=00000001
           LPL REASON TYPE=LOGAPPLY
DSNI033I = DSNILABR PAGESET TESTDB .TESTTS PART (n/a)
           IS MARKED REFP AND LPL ON BEHALF OF UR 00000CFEDB3A.
           RECOVERY TO RBA 00000CFEDBC9 IS REQUIRED.
DSNR042I = DSNRUA02 WARNING - UR ROLLBACK HAS BEEN
```

```
CANCELLED AND IS INCOMPLETE FOR
CORRELATION NAME = CTHDCORID001
CONNECTION ID = TEST0001
AUTHID = SYSADM
PLAN NAME = DONSQL1
URID = 00000CFEDB3A
```

-DISPLAY ARCHIVE command enhancements

Prior to this enhancement, the -DISPLAY ARCHIVE command only displays information about tape archive log read activity. No output is displayed for disk archive log read activity or if an archive log data set is being recalled by HSM.

APAR/PTF PQ90936/UQ90532 enhances the -DISPLAY ARCHIVE command output. It now includes:

- DASD archive log read activity.
- A status of 'RECALL' if an archive log data set is being recalled by HSM.
- The 4-digit device number.
- 8 bytes of the correlation-ID for system agents (2 byte RMID and 6 byte function ID), instead of using "RESTART" for all system agents.

A sample output is shown in Example 3-5.

Example 3-5 Enhanced -DIS ARCHIVE output

```
-DT25 DIS ARCHIVE
DSNJ322I -DT25 DISPLAY ARCHIVE REPORT FOLLOWS
              COUNT                  TIME
            (TAPE UNITS)           (MIN,SEC)
DSNZPARM         2                    0,00
CURRENT          2                    0,00
===============================
ADDR   STATUS   CORR-ID   VOLSER   DATASET_NAME
03B0   BUSY     USER01    A00001   DSNT2AR1.DT25.D04169.T1334426.A0012701
      RECALL    03RCRSC   MIGRAT   DSNT2AR1.DT25.D04169.T1334426.A0012704
A99B   BUSY     14DRSTRT  ARN690   DSNT2AR1.DT25.D04169.T1346176.A0012705
BDDD   BUSY     10LPLALR  ARN738   DSNT2AR1.DT25.D04170.T1506437.A0012743
END OF DISPLAY ARCHIVE REPORT.
```

Automatic scan for prior checkpoint during point in time recovery

DB2 checkpoint information, required for DB2 restart, is recorded in the BSDS data set in a wrap-around queue. This checkpoint queue contains information about the most recent 100 checkpoints. During a point-in-time (PIT) or disaster recovery, the recovery point (or log truncation point) may be prior to the RBA of the oldest checkpoint recorded in the BSDS. Currently, this causes the Change Log Inventory (DSNJU003) job, that defines the log truncation point for a conditional restart, to fail with message:

```
DSNJ407E NO VALID CHECKPOINT RBA FOUND.
```

The user must then manually run the DSN1LOGP utility to find the RBA of the begin-checkpoint log record for most recent complete checkpoint prior to the log truncation point, and manually specify this RBA on the CRESTART statement of the DSNJU003 job (CHKPTRBA parameter). This can be a time consuming task, especially for a multi-member data sharing group, and can greatly prolong the PIT or disaster recovery time.

With this enhancement, introduced by APAR/PFT PQ93548/UQ92875, if a prior checkpoint can not be found in the BSDS by the DSNJU003 utility when processing the CRESTART statement, the job will not fail. An indicator is inserted into the CRCR record that a backward scan of the log to locate the checkpoint will be required during the subsequent restart, as shown in Example 3-6.

Example 3-6 Message DSNJ407I indicating scanning at restart

```
CRESTART CREATE,ENDRBA=0000127E1000
DSNJ407I  DSNRJFCK WARNING - NO VALID CHECKPOINT RBA FOUND.
LOG WILL BE SCANNED AT RESTART
```

The output from the Print Log Map (DSNJU004) utility is shown in Example 3-7.

Example 3-7 DSNJU004 output indicating scan required

```
CRCR IDENTIFIER 0007
 USE COUNT   0
 RECORD STATUS
     CRCR ACTIVE
     CRCR NOT USED
 PROCESSING STATUS
     FORWARD = YES
     BACKOUT = YES
 STARTRBA                       NOT SPECIFIED
 ENDRBA                         0000127E1000
 ENDLRSN                        NOT SPECIFIED
 EARLIEST REQUESTED RBA         000000000000
 FIRST LOG RECORD RBA           000000000000
 ORIGINAL CHECKPOINT RBA        000000000000
 NEW CHECKPOINT RBA (CHKPTRBA)  SCAN REQUIRED
 CRCR CREATED            21:16:36 JULY 07, 2004
 RESTART PROGRESS                STARTED     ENDED
                                 =======     =====
     CURRENT STATUS REBUILD         NO         NO
     FORWARD RECOVERY PHASE         NO         NO
     BACKOUT RECOVERY PHASE         NO         NO
```

During the actual conditional restart, at the beginning of the recovery phase of the subsequent DB2 restart, DB2 issues a new message (DSNR054I) indicating that a scan of the log to locate the restart checkpoint is (automatically) started. DB2 begin reading the log backwards from the log truncation point, until a complete checkpoint is found. Once the prior checkpoint is found, update the BSDS CRCR record with the checkpoint information, and proceed with the recovery phase of restart. A sample DB2 conditional restart job log is shown in Example 3-8.

Example 3-8 Conditional restart job log

```
DSNY001I = SUBSYSTEM STARTING
DSNJ127I = SYSTEM TIMESTAMP FOR BSDS= 04.189 14:14:15.10
*37 DSNJ245I = CONDITIONAL RESTART RECORD INDICATES TRUNCATION AT RBA
0000127E1000. REPLY Y TO CONTINUE, N TO CANCEL
R 37,Y
DSNJ002I = FULL ACTIVE LOG DATA SET
DSNAME=DSNC910.LOGCOPY1.DS02, STARTRBA=0000126EE000,
ENDRBA=0000127E0FFF
```

```
DSNJ001I = DSNJW307 CURRENT COPY 1 ACTIVE LOG DATA
SET IS DSNAME=DSNC910.LOGCOPY1.DS03,
STARTRBA=0000127E1000,ENDRBA=0000131B8FFF
DSNJ002I = FULL ACTIVE LOG DATA SET
DSNAME=DSNC910.LOGCOPY2.DS02, STARTRBA=0000126EE000,
ENDRBA=0000127E0FFF
DSNJ001I = DSNJW307 CURRENT COPY 2 ACTIVE LOG DATA
SET IS DSNAME=DSNC910.LOGCOPY2.DS03,
STARTRBA=0000127E1000,ENDRBA=0000131B8FFF
DSNJ099I = LOG RECORDING TO COMMENCE WITH
STARTRBA=0000127E1000
S V91ADBM1
DSNR001I = RESTART INITIATED
DSNR054I = RESTART...BEGIN SCAN FOR CHECKPOINT
DSNR003I = RESTART...PRIOR CHECKPOINT RBA=0000127DCC1A
DSNR004I = RESTART...UR STATUS COUNTS
IN COMMIT=0, INDOUBT=1, INFLIGHT=0, IN ABORT=0, POSTPONED ABORT=0
DSNR007I = RESTART...STATUS TABLE
T CON-ID CORR-ID AUTHID PLAN S URID DAY TIME
- -------- ------------ -------- -------- - ------------ --- --------
S TEST0001 CTHDCORID001 SYSADM DONSQL1 D 00001260B3D9 188 11:45:59
DSNR005I = RESTART...COUNTS AFTER FORWARD RECOVERY
IN COMMIT=0, INDOUBT=1
DSNR007I = RESTART...STATUS TABLE 675
T CON-ID CORR-ID AUTHID PLAN S URID DAY TIME
- -------- ------------ -------- -------- - ------------ --- --------
S TEST0001 CTHDCORID001 SYSADM DONSQL1 D 00001260B3D9 188 11:45:59
DSNR006I = RESTART...COUNTS AFTER BACKWARD RECOVERY
INFLIGHT=0, IN ABORT=0, POSTPONED ABORT=0
DSNR002I = RESTART COMPLETED
DSN9022I = DSNYASCP ' STA DB2' NORMAL COMPLETION
```

4

SQL enhancements

List of Topics

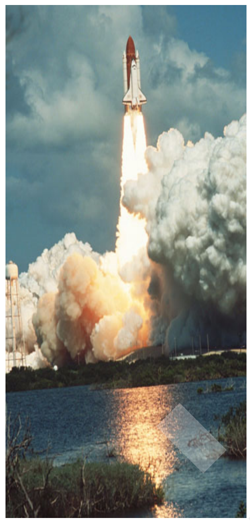
- Breaking limitations
- Dynamic scrollable cursors
- Multi-row FETCH and INSERT
- GET DIAGNOSTICS statement
- Common table expressions and recursive SQL
- Identity column enhancements
- Sequence objects
- Scalar fullselect
- Multiple DISTINCT clauses
- INSERT within SELECT statement
- Miscellaneous enhancements

This chapter describes the enhancements to SQL. It consists of the following topics:

- Breaking limitations
- Dynamic scrollable cursors
- Multi-row FETCH and INSERT
- GET DIAGNOSTICS statement
- Common table expressions and recursive SQL
- Identity column enhancements
- Sequence objects
- Scalar fullselect
- Multiple DISTINCT clauses
- INSERT within SELECT statement
- Expressions in GROUP BY
- Miscellaneous enhancements:
  - Qualified column names in INSERT and UPDATE
  - IS NOT DISTINCT FROM
  - Explain STMTCACHE
  - Read-only using update locks
  - New built-in functions
  - New special registers
  - Session variables
  - Transparent ROWID
  - SELECT INTO with ORDER BY

Breaking SQL Limitations

Redbooks



• Table name length (*)	18 to 128
• Column name length	18 to 30
• Index key size	255 to 2000
• Character Literals	255 to 32704
• Hex literal digits	255 to 32704
• Predicates	255 to 32704
• SQL statement length	32 KB to 2 MB
• Tables in a join	15 to 225

(*) Also applies to (most) other DB2 objects like views, aliases, index, triggers, synonyms,...

4.1 Breaking SQL limitations

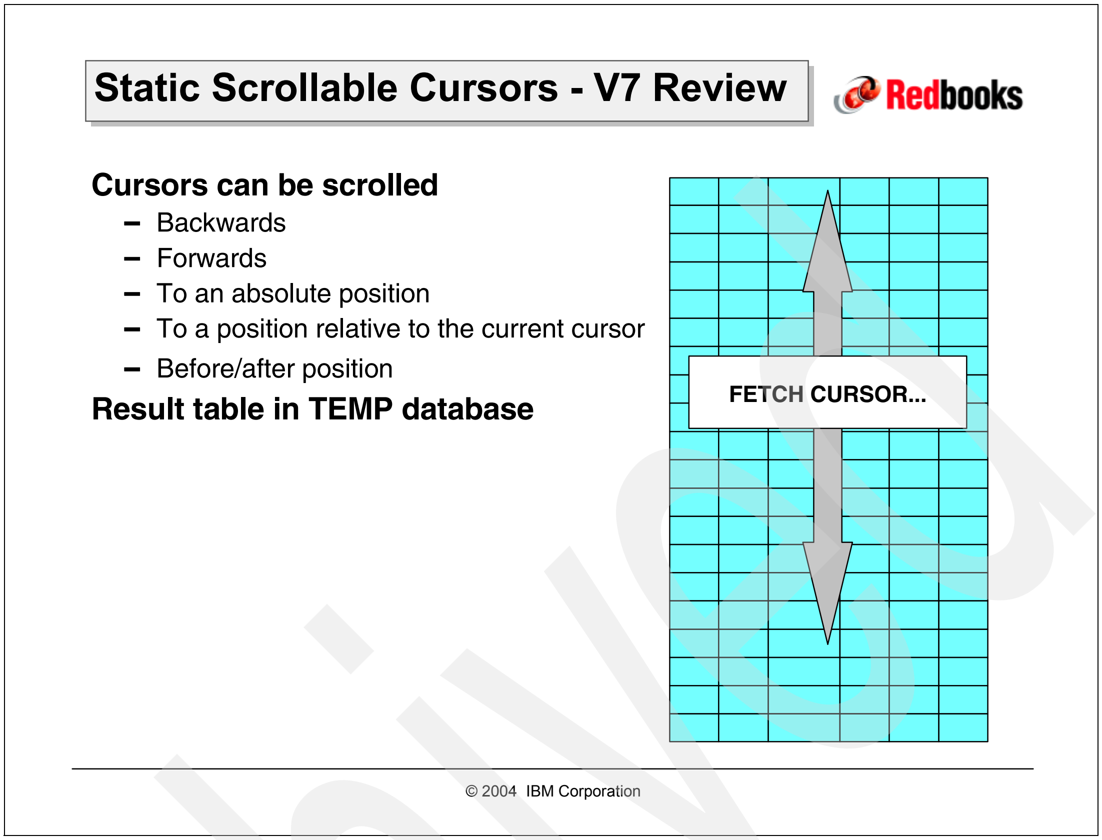
Applications for DB2 UDB for z/OS are very often developed on other platforms and then ported. In doing this, care has to be exercised because of the restrictions on the number of characters that can be used for the object names. For example, table names and column names are restricted with DB2 V7 to 18 characters. DB2 V8 support for long names goes a long way to help easily port applications from other platforms. Long names require the DB2 V8 new-function mode to be active; during the catalog migration process DB2 will ALTER the existing definitions to the new ones.

Refer to Table 4-1 for a complete list of changed limits.

Table 4-1 Changed limits in V8

Item	Changed from	Changed to
Table check constraint name, correlation name, name of an alias, cursor (except for DECLARE CURSOR WITH RETURN), index, procedure, synonym, function, view, table	18 bytes	128 bytes
name of a column	18	30 bytes. Stored in DB2 catalog as VARCHAR(128), but limited to 30 Unicode bytes.
Name of cursor created with DECLARE CURSOR WITH RETURN	18 bytes	30 bytes
Name of a location	16 bytes	16 bytes. Stored in DB2 catalog as VARCHAR(128), but only 16 characters are used.
Name of package	8 bytes	8 bytes. Stored in DB2 catalog as VARCHAR(128), only 8 characters are used for packages created with BIND PACKAGE command, but can be 128 bytes long if created as a result of CREATE TRIGGER command.
Auxiliary table, collection, distinct type (both parts of two-part name), host identifier, JARs, parameter, sequence, specific, statement, savepoint, SQL parameter	18 bytes	128 bytes
SQL variable, SQL label, SQL condition	64 bytes	128 bytes
Maximum length of character constant	255 bytes	32704 UTF-8 bytes
Maximum length of hexadecimal constant	254 digits	32704 hexadecimal digits
Maximum length of graphic string constant	124 DBCS characters	32698 UTF-8 bytes

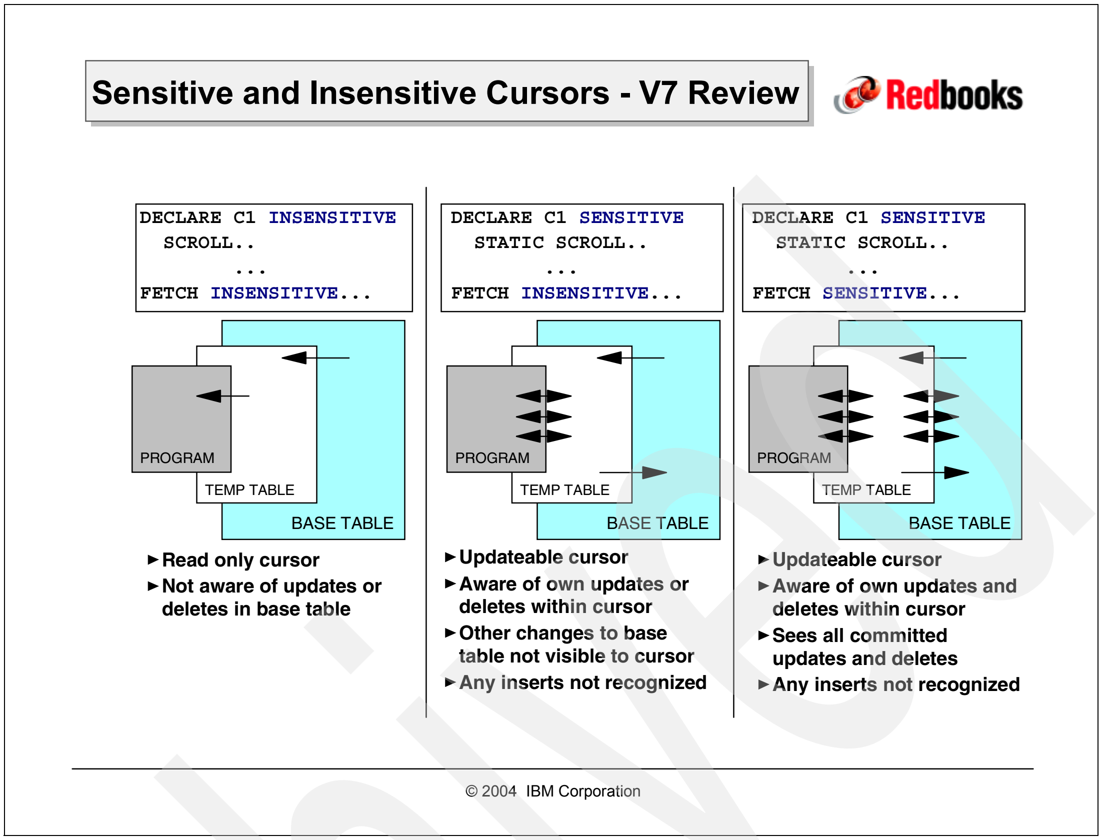
Item	Changed from	Changed to
Maximum length of hexadecimal graphic string constant	N/A	32704 hexadecimal digits
Maximum number of partitions in a partitioned table space or partitioned index	64 for table spaces that are not defined with LARGE or a DSSIZE greater than 2G 254 for table spaces that are defined with LARGE or a DSSIZE greater than 2G	64 for table spaces that are not defined with LARGE or a DSSIZE greater than 2G 4096 depending on what is specified for DSSIZE or LARGE and the page size.
Maximum size of a partition (table space or index)	For table spaces that are not defined with LARGE or a DSSIZE greater than 2G: 4 gigabytes, for 1 to 16 partitions 2 gigabytes, for 17 to 32 partitions 1 gigabyte, for 33 to 64 partitions For table spaces that are defined with LARGE: 4 gigabytes, for 1 to 254 partitions For table spaces that are defined with a DSSIZE greater than 2G: 64 gigabytes, for 1 to 254 partitions	For table spaces that are not defined with LARGE or a DSSIZE greater than 2G: 4 gigabytes, for 1 to 16 partitions 2 gigabytes, for 17 to 32 partitions 1 gigabyte, for 33 to 64 partitions For table spaces that are defined with LARGE: 4 gigabytes, for 1 to 4096 partitions For table spaces that are defined with a DSSIZE greater than 2G: 64 gigabytes, depending on the page size (1 to 256 partitions for 4KB, 1 to 512 partitions for 16KB, 1 to 1024 partitions for 32KB, and 1 to 2048 for 32KB)
Maximum length of an index key	255 bytes less the number of key columns that allow nulls	Partitioning index: 255 - n Non-partitioning index that is padded: 2000 - n Non-partitioning index that is not padded: 2000 - n -2m Where n is the number of columns in the key that allow nulls and m is the number of varying-length columns in the key
Longest SQL statement	32765 bytes	2097152 bytes
Maximum length of a sort key	4000 bytes	16000 bytes
Maximum length of the SQL path	254 bytes	2048 bytes
Largest table or table space	16 terabytes	128 terabytes



4.2 Static scrollable cursors - V7 review

DB2 V7 introduced static scrollable cursors. When a static scrollable cursor is opened, the qualifying rows are copied to a declared temporary table automatically created by DB2 in a TEMP database defined by you. User-declared temporary tables and system-declared temporary tables share the set of table spaces that are defined in the TEMP database.

When a static scrollable cursor is opened, DB2 stores the RID of each qualifying base row in the result table. The result table has a fixed number of rows. Scrolling is performed on the temporary table in both the forward direction and backward direction. DB2 deletes the result table when the cursor is closed.



4.2.1 Sensitive and insensitive cursors - V7 review

DB2 V7 introduced keywords to control the positioning within a cursor, and whether the data in the result set is maintained with the actual rows in the base table.

In V7, you can declare a scrollable cursor in one of the following two ways:

- INSENSITIVE:

  This causes the scrollable cursor to be read-only. You can only specify INSENSITIVE on the FETCH statement, which is the default if the cursor is declared INSENSITIVE. A FETCH INSENSITIVE request retrieves the row data from the result table. The cursor is not sensitive to the updates, deletes, and inserts made to the base table.

- SENSITIVE STATIC:

  This causes the scrollable cursor to be updateable. You can specify either of the following on the FETCH statement:

  – INSENSITIVE:

    The application can see the changes it has made itself, using positioned updates and deletes through the cursor. These changes are visible to the application because DB2 updates both the base table and the result table when a positioned update or delete is issued by the application. However, updates and deletes to the base table made outside the cursor are not visible. The cursor is also not sensitive to inserts.

– SENSITIVE:

This is the default if the cursor is declared SENSITIVE STATIC. A FETCH SENSITIVE request retrieves the row data from the result table (stored in a declared temporary table). However, as part of a SENSITIVE FETCH, the row is verified against the underlying table to make sure it still exists and qualifies. Using FETCH SENSITIVE, the application can see the changes it has made using positioned and searched updates and deletes. The application can also see the committed updates and deletes made by other applications or outside the cursor. However, the cursor is not sensitive to inserts.

Note that all V7 scrollable cursors use a declared temporary table (DTT) to store the result table of the cursor. The DTT is populated at OPEN CURSOR time.

New in V8 - Dynamic Scrollable Cursors

Scrollable cursor that provides access to the base table rather than a DTT -- allows visibility of updates and ***inserts*** done by you or other users

Defaults to single row FETCH, so DDF applications should use:

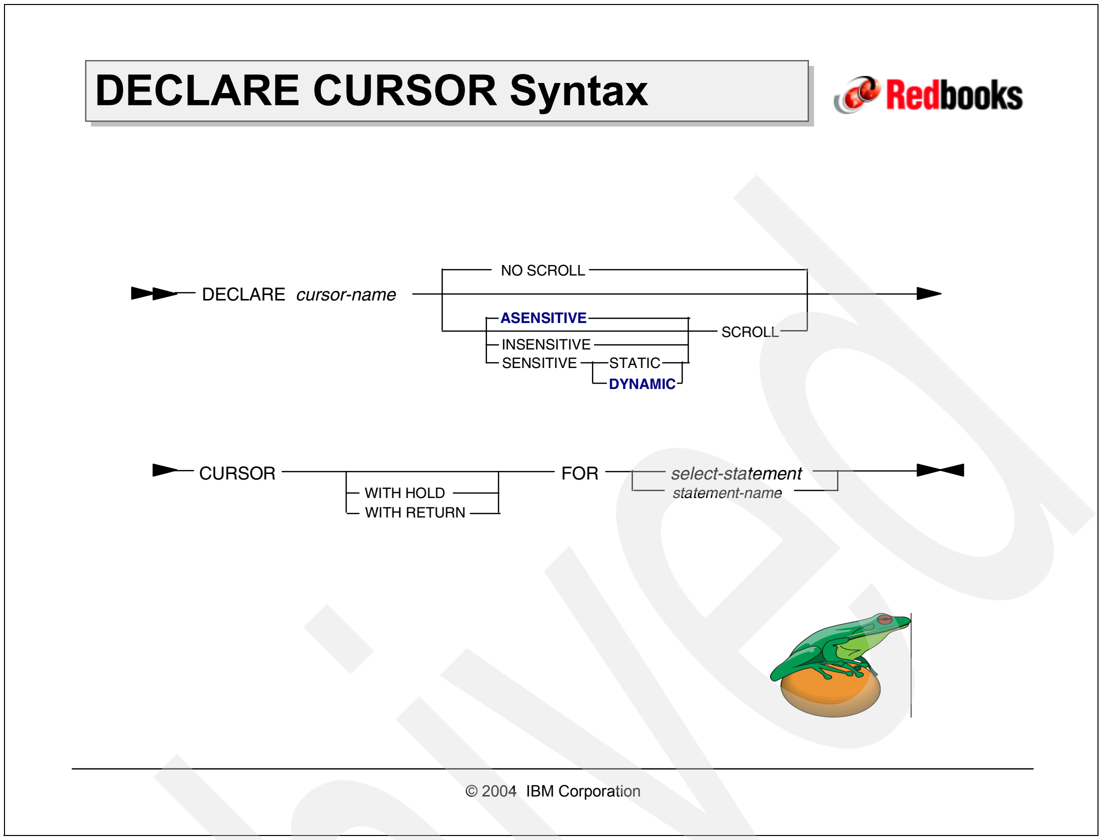
- Multi-row FETCH
- Positioned UPDATE/DELETE for multi-row FETCH

```
DECLARE C1 SENSITIVE DYNAMIC SCROLL
    CURSOR FOR
        SELECT C1, C2
          FROM T1;
```


4.3 Dynamic scrollable cursors - V8 new function

DB2 V8 introduces dynamic scrollable cursors. A dynamic scrollable cursor does not materialize the result table at any time. Instead, it scrolls directly on the base table and is therefore sensitive to all committed inserts, updates, and deletes. Dynamic scrollable cursors are supported for the index scan and table space scan access paths. DPSIs also support dynamic scrollable cursors.

Dynamic scrollable cursors can be used with row-level operations or rowset-level operations. Rowset-level operations are discussed in 4.4, “Multi-row FETCH and INSERT” on page 239.



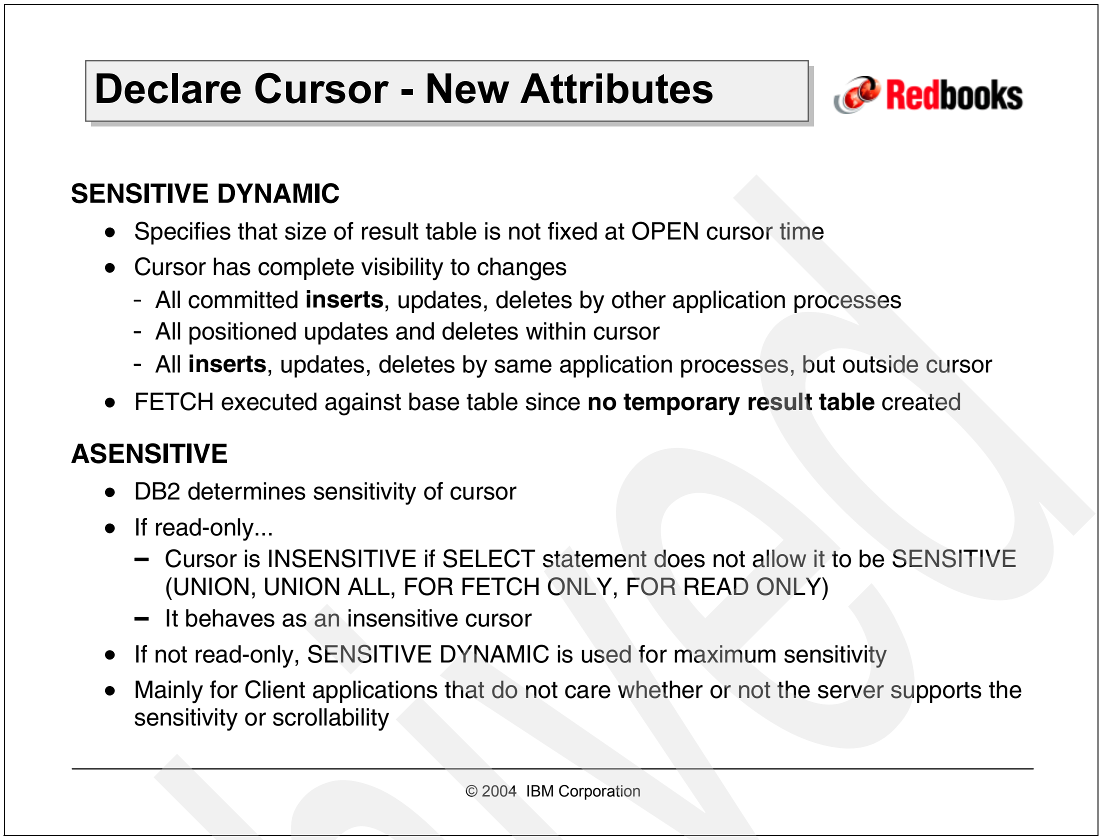
4.3.1 DECLARE CURSOR syntax

This visual shows the new syntax for declaring a cursor in DB2 V8.

NO SCROLL, which is the default, indicates that the cursor is non-scrollable.

SCROLL specifies that the cursor is scrollable. For a scrollable cursor, whether the cursor has sensitivity to inserts, updates, or deletes depends on the cursor sensitivity option in effect for the cursor. The sensitivity options include the following possibilities:

- ASENSITIVE, introduced in DB2 V8 (this is the default for a scrollable cursor)
- INSENSITIVE, as in DB2 V7
- SENSITIVE STATIC, as in DB2 V7
- SENSITIVE DYNAMIC for dynamic scrollable cursors, introduced in DB2 V8



4.3.2 Declare cursor - new attributes

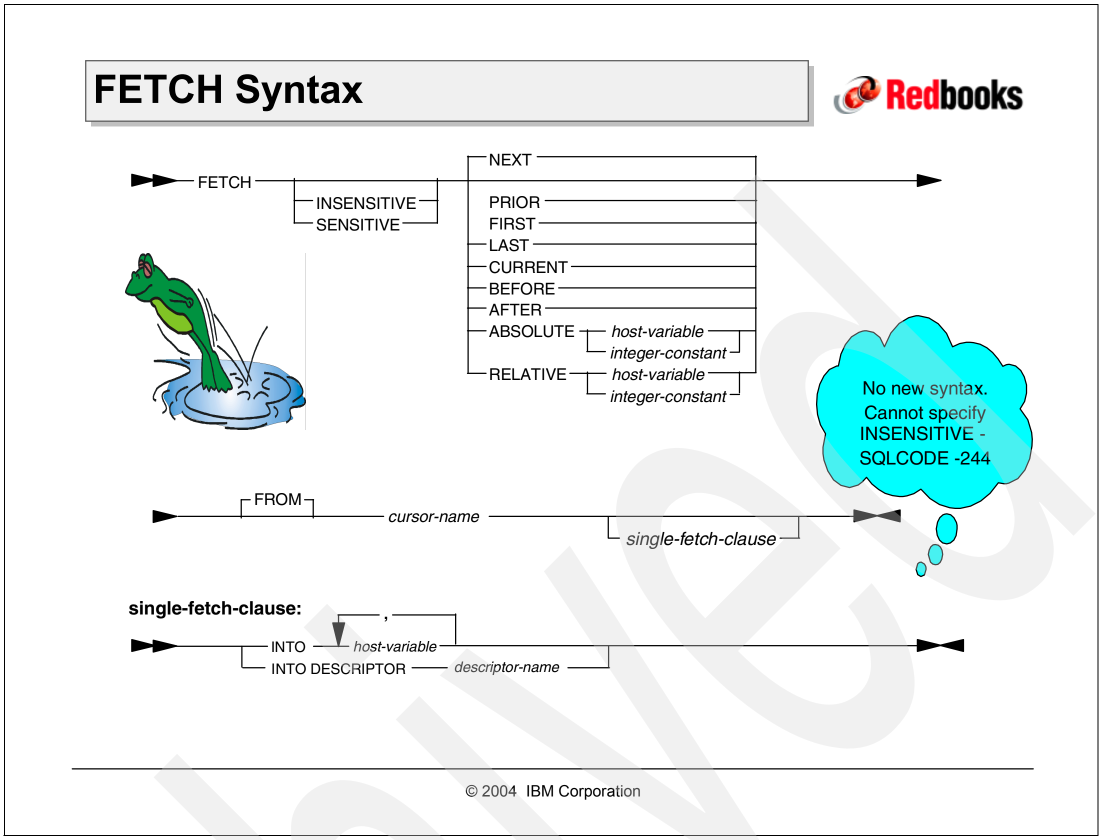
SENSITIVE DYNAMIC specifies that the result table of the cursor is dynamic, in that the size of the result table may change after the cursor is opened as rows are inserted into or deleted from the underlying table, and the order of the rows may change. Rows inserted, updated, or deleted with INSERT, UPDATE, and DELETE statements executed by the same application process are immediately visible. Rows inserted, updated, or deleted with INSERT, UPDATE and DELETE statements executed by other application processes are visible once committed.

Because the FETCH statements are executed against the base table, no temporary result table is created. The SELECT statement of a cursor that is defined as SENSITIVE DYNAMIC cannot contain an INSERT statement.
For client applications that do not care whether or not the server supports the sensitivity or scrollability, you can use the ASENSITIVE option, to let DB2 determine whether the cursor behaves as SENSITIVE DYNAMIC or INSENSITIVE depending on the complexity (updateability) of the associated SELECT statement.

- If the cursor is specified as ASENSITIVE and is read-only, it behaves as an INSENSITIVE scrollable cursor.

> **Note:** An INSENSITIVE scrollable cursor behaves the same as in V7, and used the TEMP database to store its qualifying rows, as shown in 4.2.1, “Sensitive and insensitive cursors - V7 review” on page 223.

- If the cursor is specified as ASENSITIVE and is not read-only, DB2 provides the maximum allowable sensitivity, which is SENSITIVE DYNAMIC.



4.3.3 FETCH syntax using dynamic scrollable cursors

There is no new syntax for the FETCH statement in DB2 V8 to support dynamic scrollable cursors.

However, there is a restriction in the FETCH orientation syntax. Any violation of the restriction results in SQLCODE -244, as explained in the next visual.

Implications on FETCH

INSENSITIVE not allowed with FETCH statement (SQLCODE -244) if

- The associated cursor is declared as SENSITIVE DYNAMIC SCROLL
- The cursor is declared ASENSITIVE and DB2 chooses the maximum allowable sensitivity of SENSITIVE DYNAMIC SCROLL

There are no "holes" as there is no temporary result table

- Special case: If FETCH CURRENT or FETCH RELATIVE +0 requested but row on which cursor is positioned was deleted or updated so that it no longer meets selection criteria (SQLCODE +231)
  For example, can occur with ISOLATION(CS) and CURRENTDATA(NO)

Inserts by the application itself are immediately visible -- inserts by others are visible after commit

Order is always maintained

- If current row is updated, the cursor is positioned before the next row of the original location and there is no current row

4.3.4 Implications on FETCH

SQLCODE -244 indicates that the sensitivity option specified on FETCH conflicts with the sensitivity option in effect for cursor cursor-name. If a cursor is declared INSENSITIVE, the FETCH statement can only specify INSENSITIVE or nothing. If a cursor is declared SENSITIVE, the FETCH statement can specify INSENSITIVE, SENSITIVE, or nothing. This is as in DB2 V7.

The restriction in DB2 V8 is that the keyword INSENSITIVE is not allowed with the FETCH statements if the associated cursor is either:

- Declared as SENSITIVE DYNAMIC SCROLL, or
- Declared ASENSITIVE, and DB2 selects the maximum allowable sensitivity of SENSITIVE DYNAMIC SCROLL for the associated SELECT statement.

Since there is no temporary result table, there are no holes except in the case of the current row. An SQLCODE +231 is returned if FETCH CURRENT or FETCH RELATIVE +0 is requested, but the row on which the cursor is positioned has been deleted or the row has been updated so that it no longer meets the selection criteria. This situation can only occur when using ISOLATION (CS) and CURRENTDATA(NO), which allows the row to be retrieved without taking a lock.

The order is always maintained. If a column for an ORDER BY clause is updated, then the next FETCH statement behaves as if the updated row were deleted and re-inserted into the result table at its correct location. At the time of a positioned update, the cursor is positioned before the next row of the original location and there is no current row, making the row appear to have moved. If the row is deleted and the next cursor operation is FETCH CURRENT, a warning SQLCODE +231 is raised.

Locking with Dynamic Scrollable Cursors

ISOLATION CS recommended for maximum concurrency

- RR and RS severely restrict update of table by other users, therefore defeating purpose of SENSITIVE DYNAMIC

Lock is held on current row as base table is always accessed directly

As before, if ISOLATION UR is the BIND option, and the SELECT statement contains FOR UPDATE OF, ISOLATION promoted to CS

As normal, if cursor is defined as WITH HOLD, locks are released at commit unless DSNZPARM RELCURHL=NO, in which case locks held for held cursors are not released at commit

4.3.5 Locking with dynamic scrollable cursors

Dynamic scrollable cursors are useful when it is important for the application to see updated rows as well as newly inserted rows. The purpose of using SENSITIVE DYNAMIC is defeated if the isolation is RR or RS, as the update of table by other users is severely restricted. Therefore isolation CS is recommended for maximum concurrency.

Locks may be held on the current row. Remember that there is no temporary result table and the base table is accessed directly by the FETCH statement.

If isolation UR is specified as a bind option and the associated SELECT statement contains a FOR UPDATE OF clause, DB2 promotes UR to CS. If WITH UR is specified in the SELECT statement along with the FOR UPDATE OF clause, DB2 returns SQLSTATE 42801, SQLCODE -173.

As in previous versions, DSNZPARM RELCURHL specifies whether DB2 should, at commit time, release a data page or row lock on which a cursor that is defined WITH HOLD is positioned. This lock is not necessary for maintaining cursor position. If you choose YES, the default, DB2 releases this data page or row lock after a COMMIT is issued for cursors that are defined WITH HOLD. Specifying YES can improve concurrency. If you choose NO, DB2 holds the data page or row lock for WITH HOLD cursors after the COMMIT. This option is provided so that existing applications that rely on this lock can continue to work correctly.

UPDATE Using Dynamic Scrollable Cursors

Redbooks

Optimistic locking is used by static scrollable cursors

- Static scrollable cursors hold no locks on base tables after OPEN CURSOR
- Check at update time whether the row has not changed

Optimistic locking is not used by dynamic scrollable cursors

- Locks against the underlying table while fetching, similar to non-scrollable cursors

Positioned UPDATE and DELETE always allowed if cursor not read-only and the page or row lock was acquired successfully

4.3.6 UPDATE using dynamic scrollable cursors

DB2 cannot use optimistic concurrency control for dynamic scrollable cursors, as it is working against the real table. For updateable dynamic scrollable cursors that use ISOLATION(CS), DB2 holds row or page lock on the current row in the base table (DB2 does not use a temporary global table when using dynamic scrollable cursors). The most recently fetched row or page from the base table is locked to maintain data integrity for a positioned update or delete.

Optimistic concurrency control - review

For packages and plans that contain updateable *static* scrollable cursors, ISOLATION(CS) lets DB2 use optimistic locking. DB2 can use optimistic concurrency control to shorten the amount of time that locks are held in the following situations:

- Between consecutive FETCH operations
- Between FETCH operations and subsequent positioned UPDATE or DELETE operations

Optimistic locking control consists of the following steps:

1. When the application opens the static scrollable cursor, DB2 fetches the qualifying rows into the DTT. When doing so, DB2 will try to use lock avoidance to minimize the amount of locking required.
2. When the application requests a positioned update or delete operation on the row, DB2 performs the following steps:
   a. Finds the corresponding row in the base table.
   b. Locks the row.
   c. Re-evaluates the predicate to ensure that the row still qualifies for the result table.

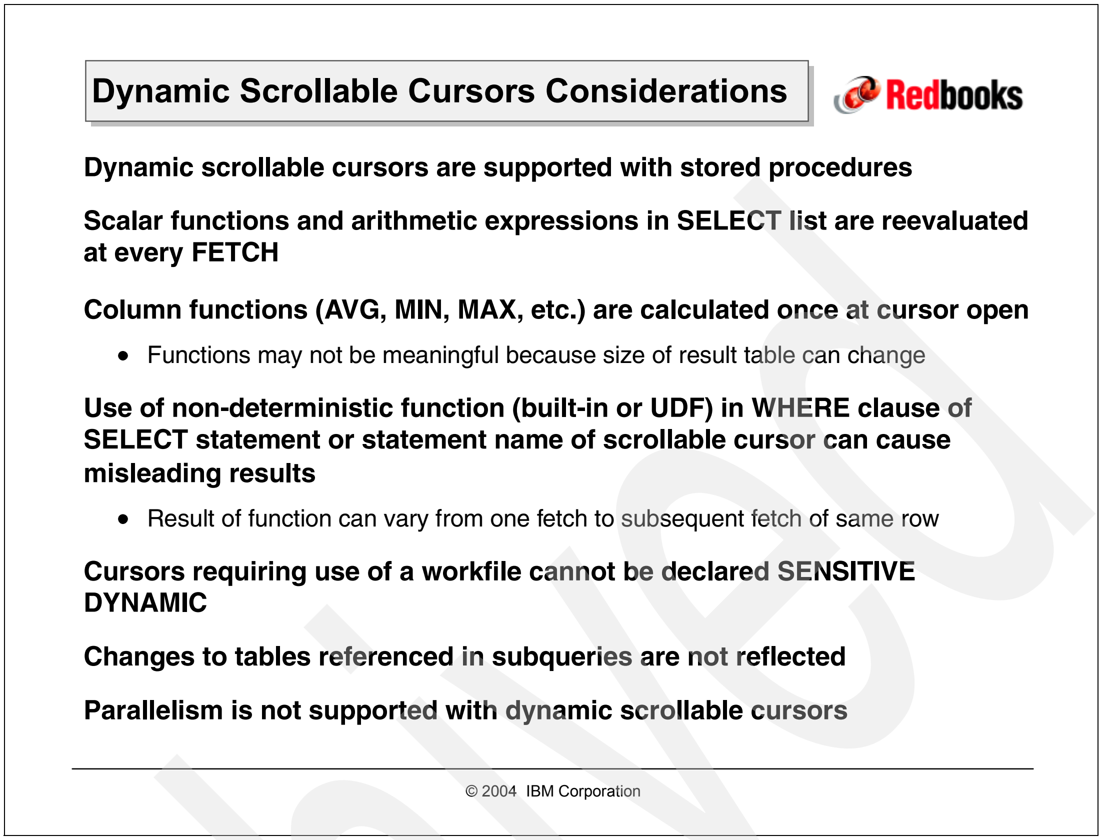
d. For columns that are in the result table, compares current values in the row to the values of the row when step 1 was executed. DB2 performs the positioned update or delete operation only if the values match.

However, as mentioned at the beginning of this section, *optimistic locking cannot be used for dynamic scrollable cursors*.

Read-only cursors - review

If the result table is read-only, the cursor is read-only. The result table is read-only if one or more of the following statements is true about the SELECT statement of the cursor:

- The first FROM clause identifies or contains any of the following items:
  - More than one table or view
  - A catalog table with no updateable columns
  - A read-only view
  - A nested table expression
  - A table function
  - A system-maintained materialized query table
- The first SELECT clause specifies the keyword DISTINCT, contains an aggregate function, or uses both.
- The SELECT statement of the cursor contains an INSERT statement.
- The outer subselect contains a GROUP BY clause, a HAVING clause, or both clauses.
- It contains a subquery such that the base object of the outer subselect, and of the subquery, is the same table.
- Any of the following operators or clauses are specified:
  - A UNION or UNION ALL operator
  - An ORDER BY clause (except when the cursor is declared as SENSITIVE STATIC scrollable)
  - A FOR FETCH ONLY or a FOR READ ONLY clause
- It is executed with isolation level UR and a FOR UPDATE clause is not specified.



4.3.7 Dynamic scrollable cursors considerations

Dynamic scrollable cursors are supported with stored procedures. The stored procedure itself can update through a dynamic scrollable cursor. However, the program calling the stored procedure is restricted from updating using the allocated cursor. (This is the case for any type of cursor.)

Scalar functions and arithmetic expressions in SELECT list are re-evaluated at every FETCH.

Column functions (AVG, MIN, MAX, etc.) are calculated once at open cursor time. Functions may not be meaningful because the size of result table can change.

The use of non-deterministic functions (built-in or UDF) in the WHERE clause can cause misleading results, because the result of the function can vary from one FETCH to a subsequent FETCH of the same row.

Cursors requiring the use of a workfile cannot be declared SENSITIVE DYNAMIC. For example, you cannot associate the following SQL statement with a dynamic scrollable cursor, as the entire result table must be materialized to a workfile:

```
SELECT COL1, MAX(COL2), COUNT(*) FROM T1 GROUP BY COL1 ORDER BY 3
```

Changes to tables referenced in subqueries are not reflected. For example, the subquery in the following SQL statement is executed once when the cursor is opened, and the subquery value does not change if there are any changes to the table referenced in the subquery:

```
SELECT * FROM EMPLOYEES WHERE SALARY > ( SELECT AVG(SALARY) FROM EMPLOYEES )
```

We cannot over emphasize the fact that because of this, some cursors will return not exactly the expected data (cursors with column functions (AVG, MIN, MAX functions, etc.). So you have a potential “logical” data integrity exposure that must be understood before you decide to use dynamic scrollable cursors.

Dynamic scrollable cursors are not the answer to every problem, and should not be used by default instead of static scrollable cursors. Static scrollable cursors may be a better solution for some online applications where you have to scroll backwards to previously displayed screens and show the data as it was initially displayed.

Parallelism is not supported with dynamic scrollable cursors.

DRDA Considerations

DB2 Linux, UNIX, and Windows V8.1 clients support dynamic scrollable cursors with FP4

- Only via the ODBC interface
- Only when calling DB2 for z/OS V8

Dynamic scrollable cursors do not support the DRDA limited block fetch protocol

- To achieve blocking, use multi-row fetch operations. This is currently supported between:
  - DB2 for z/OS V8 systems
  - DB2 ODBC distributed clients using dynamic scrollable cursors and DB2 for z/OS V8

4.3.8 DRDA considerations

Dynamic scrollable cursors are also allowed in a DRDA environment. They are fully supported in a distributed environment between DB2 for z/OS V8 systems.

DB2 V8.1 clients on Linux, UNIX, and Windows platforms have also implemented dynamic scrollable cursors via ODBC (or DB2 CLI) with FixPak 4 when communicating with a DB2 for z/OS V8 system. You can make a statement a dynamic scrollable cursor by calling the SQLSetStmtAttr() function with the SQL_ATTR_CURSOR_TYPE statement attribute set to SQL_CURSOR_DYNAMIC.

When using dynamic scrollable cursors in a distributed environment using DRDA, limited block fetch is not used, and one row is fetched at a time from the remote server. DB2 cannot use block fetch as it has to evaluate the predicates during each FETCH operation because the base table may have changed, as dictated by the semantics of the “dynamic” keyword.

You may want to consider declaring your dynamic scrollable cursor WITH ROWSET POSITIONING to “simulate” a limited block fetch effect. When using rowset cursors, DRDA always sends a rowset across the wire, also for dynamic scrollable cursors.

At the time of writing of this publication, rowset cursors are supported:

- Between DB2 for z/OS V8 systems.
- Between DB2 ODBC/CLI clients on distributed platforms that use dynamic scrollable cursors to retrieve data on a DB2 for z/OS V8. Note that dynamic scrollable cursors on DB2 distributed clients always use multi-row fetch (when obtaining data from a DB2 for z/OS V8 system).

Dynamic Scrollable Cursor Example

Redbooks

Example of how an application program might use dynamic scrollable cursor

```
DECLARE ORDERSCROLL SENSITIVE DYNAMIC SCROLL CURSOR FOR
  SELECT ORDERNUM, CUSTNAME, ORDERAMT, ORDERDATE
  FROM ORDERS
  WHERE ORDERAMT > 1   FOR UPDATE OF COMMENTS;

** Open scrollable cursor
OPEN ORDERSCROLL;
** Fetch forward from scrollable cursor
** LOOP-TO-FILL SCREEN
  DO 3 TIMES
    FETCH FROM ORDERSCROLL INTO :hv1, :hv2, :hv3, :hv4;
  END
** Fetch RELATIVE from scrollable cursor
** Skip forward 1 row
  FETCH RELATIVE +1 FROM ORDERSCROLL INTO :hv1, :hv2, :hv3, :hv4;
** Skip backward 5 rows
  FETCH RELATIVE -5 FROM ORDERSCROLL INTO :hv1, :hv2, :hv3, :hv4;
```

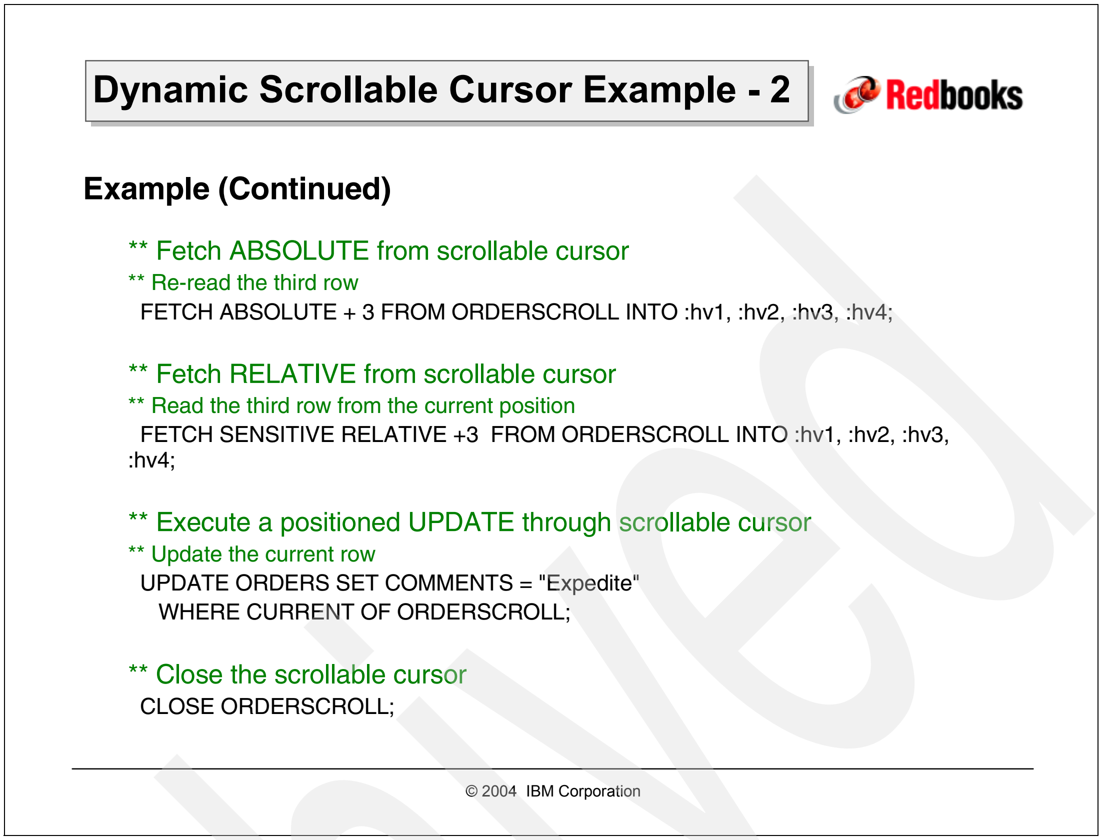

4.3.9 Dynamic scrollable cursor example - 1

This visual shows some SQL statements using dynamic scrollable cursors that you can embed in your programs.

Cursor positioning and serialization

At OPEN CURSOR time, the cursor is positioned before the first row. After FETCH, the fetched row becomes the current row and the cursor is positioned on this row. At this point, the UPDATE or DELETE WHERE CURRENT OF statement operates on the current row under the existing rules of positioned updates and deletes.

When FETCH returns an EOF condition (SQLCODE +100), the cursor is positioned after the last row if the scroll number is positive (or FETCH NEXT is executed), and the cursor is positioned before the first row if the scroll number is negative (or FETCH PRIOR is executed). With an updateable dynamic scrollable cursor, the most recently fetched row from the base table remains locked in order to maintain its position for a positioned UPDATE or positioned DELETE (in case FOR UPDATE OF is specified on the cursor definition).



4.3.10 Dynamic scrollable cursor example - 2

The visual shows some SQL statements using dynamic scrollable cursors that you can embed in your programs.

Note that when you declare the cursor as dynamic scrollable, and SQLCODE +100 is returned, you can continue to FETCH rows. However, once end of file is reached, FETCH NEXT will always give +100; that is, once you get SQLCODE +100, do not expect to see rows inserted after the last successful fetch (prior) to +100, and similarly for FETCH PREVIOUS. Once you get a +100, if you want to continue fetching, you need to reposition via FETCH FIRST/LAST, etc.

Cursor Type Comparison

Cursor Type	Result Table	Visibility of Own Changes	Visibility of Others' Changes	Updatability (*)
Non-Scrollable (SQL contains a Join or Sort, etc)	Fixed, workfile	No	No	No
Non-Scrollable	No workfile, base table access	Yes	Yes	Yes
INSENSITIVE SCROLL	Fixed, declared temp table	No	No	No
SENSITIVE STATIC SCROLL	Fixed, declared temp table	Yes (INSERTs not allowed)	Yes (Not INSERTs)	Yes
SENSITIVE DYNAMIC SCROLL	No declared temp table, base table access	Yes	Yes	Yes

4.3.11 Cursor type comparison

The visual summarizes the characteristics of different types of cursors. Only the last row is new in DB2 V8.

Multi-Row FETCH and INSERT

What is it? .....

- Multi-row FETCH:
  - A single FETCH statement can retrieve multiple rows of data from the result table of a query as a rowset
    - A rowset is a group of rows of data that are grouped together and operated on as a set
    - Supports dynamic and static SQL (Fetch always static)
- Multi-row INSERT:
  - A single SQL statement can insert one or more rows into a table or view
  - Multi-row INSERT can be implemented as either static or dynamic SQL

Benefits .....

- Enhances usability and power of SQL
- Performance is improved by eliminating multiple trips between application and database engine; for distributed access, reduced network traffic

4.4 Multi-row FETCH and INSERT

DB2 V8 introduces support for multiple row processing for both the FETCH and INSERT statements. In prior versions of DB2, an application has to execute multiple SQL FETCH statements, one for each row to be retrieved from a table, and multiple SQL INSERT statements, one for each row to be inserted into a table.

In V8, a single FETCH statement can be used to retrieve multiple rows of data from the result table of a query as a *rowset*. A rowset is a group of rows that are grouped together and operated on as a set. For example, you may fetch the next rowset. Fetching multiple rows of data can be done with both scrollable and non-scrollable cursors. New syntax on the FETCH statement allows the specification of the number of rows to be returned in the rowset for each fetch. The maximum rowset size is 32767.

The multiple-row FETCH statement can only be embedded in an application program. It is an executable statement that cannot be dynamically prepared. This only applies to the FETCH statement. You can of course dynamically prepare a statement with rowset positioning.

Prior to DB2 V8, an SQL INSERT statement using the VALUES clause can insert one row of data into a table or view. The INSERT INTO ... SELECT FROM ... allows zero or more rows to be inserted into a table or view named in the INTO clause, by retrieving qualified rows from the table or view named with the FROM clause. With the multi-row INSERT enhancement in DB2 V8, an INSERT statement using the VALUES clause can insert one or more rows into a table or view with a single SQL statement.

There are two forms of multiple-row INSERT: one static, and one dynamic form. These DB2 V8 enhancements help to lower the statement execution cost, and in a distributed environment, also the network cost. Multiple trips between the application and the database are no longer required, and there are fewer send and receive messages over the network.

Multi-Row FETCH

Redbooks

DECLARE CURSOR statement

Host variable arrays

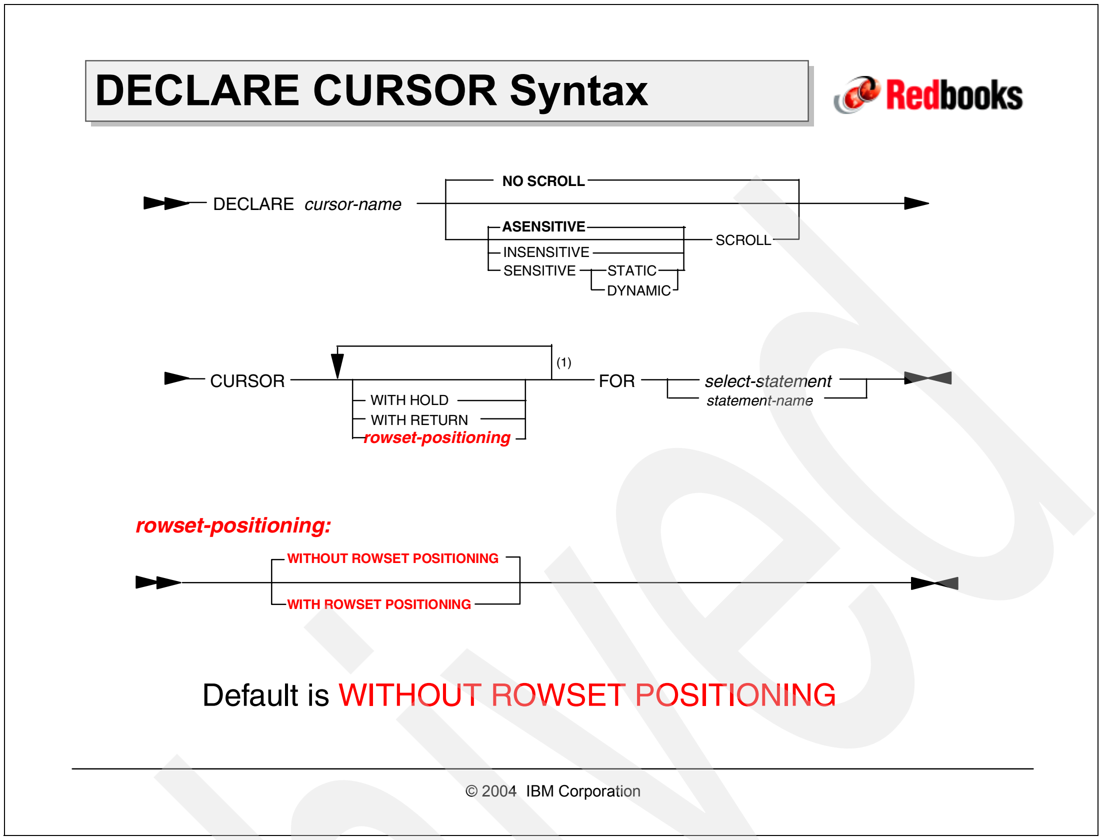
FETCH statement

Positioned UPDATE statement

Positioned DELETE statement

4.5 Multi-row FETCH

Multi-row FETCH requires a proper understanding of how to set up the environment in terms of declaring the cursor, setting up the host variable arrays, and performing the FETCH operations. Using multi-row FETCH not only has an impact on how you handle FETCH operations in your application, but also has repercussions on how the positioned UPDATE and positioned DELETE statements are handled. We discuss the details in the following visuals.



4.5.1 DECLARE CURSOR syntax

The visual shows the changes to the DECLARE CURSOR syntax. The *row-set positioning* block specifies whether multiple rows of data can be accessed as a rowset on a single FETCH statement for this cursor. The default is WITHOUT ROWSET POSITIONING.

- WITHOUT ROWSET POSITIONING:

  Specifies that the cursor can only be used to return a single row for each FETCH statement, and that the FOR n ROWS clause cannot be specified on a FETCH statement for this cursor. Doing so results in an SQLSTATE 24523 or SQLCODE -249.

- WITH ROWSET POSITIONING:

  Specifies that this cursor can be used to return either a single row or multiple rows, as a rowset, with a single FETCH statement. The FOR n ROWS clause of the FETCH statement controls how many rows are returned on each FETCH statement. Cursors declared WITH ROWSET POSITIONING may also be used with row positioned FETCH statements.

Important: Using WITHOUT ROWSET POSITIONING does not mean that there is no DRDA blocking when using distributed access. It only indicates that one row at a time is returned to the application host variables.

DECLARE CURSOR Example

Redbooks

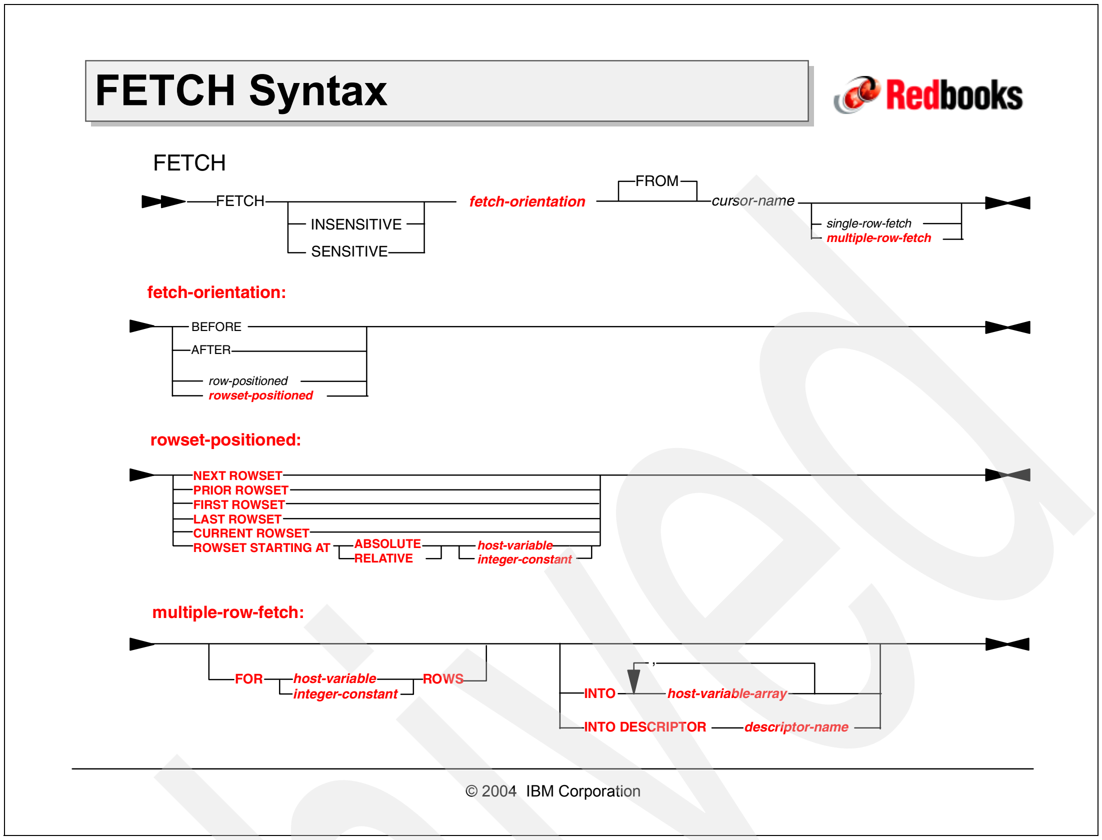
Declare C1 as the cursor of a query to retrieve a rowset from table EMP

```
EXEC SQL
  DECLARE C1 CURSOR
  WITH ROWSET POSITIONING
  FOR SELECT * FROM EMP;
```

WITH ROWSET POSITIONING specifies whether multiple rows of data can be accessed as a rowset on a single FETCH statement

4.5.2 DECLARE CURSOR example

This visual shows an example of how to declare the cursor for a query to retrieve a rowset from the table EMP. Note that you do not specify the size of the rowset on the DECLARE CURSOR statement. That is done at FETCH time.



4.5.3 FETCH syntax

Two new syntax blocks, *multiple-row-fetch* and *rowset-positioned*, have been introduced for multi-row FETCH.

The *rowset-positioned* block is similar to the existing *row-positioned* block in DB2 V7. The rowset-positioned clause specifies positioning of the cursor with rowset-positioned fetch orientations NEXT ROWSET, PRIOR ROWSET, FIRST ROWSET, LAST ROWSET, CURRENT ROWSET, ROWSET STARTING AT ABSOLUTE, ROWSET STARTING AT RELATIVE, just as the existing row-positioned clause specifies positioning of the cursor with row-positioned fetch orientations NEXT, PRIOR, FIRST, LAST, CURRENT, ABSOLUTE, and RELATIVE.

The multiple-row-fetch syntax block is similar to the existing single-row-fetch block in DB2 V7, except that there is an additional clause FOR n ROWS, where n can either be an integer constant or a host variable. When using a host variable, you can vary the number of rows fetched in a rowset for each FETCH, if needed.

FETCH Examples

EXAMPLE 1:

Fetch the previous rowset and have the cursor positioned on that rowset

```
EXEC SQL
   FETCH PRIOR ROWSET FROM C1 FOR 3 ROWS INTO...
                       --    OR  --
EXEC SQL
    FETCH ROWSET
       STARTING AT RELATIVE -3 FROM C1 FOR 3 ROWS INTO...
```

EXAMPLE 2:

Fetch 3 rows starting with row 20 regardless of the current position of the cursor

```
EXEC SQL
    FETCH ROWSET STARTING AT ABSOLUTE 20
        FROM C1 FOR 3 ROWS INTO...
```


4.5.4 FETCH examples

This visual shows some simple examples of the FETCH statement using rowsets.

Given that the cursor C1 is defined as:

```
DECLARE C1 CURSOR WITH ROWSET POSITIONING FOR SELECT * FROM EMP
```

- Fetch 3 rows starting with row 20 regardless of the current position of the cursor, and cause the cursor to be positioned on that rowset at the completion of the fetch:

```
FETCH ROWSET STARTING AT ABSOLUTE 20 FROM C1 FOR 3 ROWS INTO :HVA1,:HVA2, ...
```

- Fetch the previous rowset, and have the cursor positioned on that rowset:

```
FETCH PRIOR ROWSET FROM C1 FOR 3 ROWS INTO :HVA1,:HVA2, ... or
FETCH ROWSET STARTING AT RELATIVE -3 FROM C1 FOR 3 ROWS INTO :HVA1,:HVA2, ...
```

- Fetch the first 3 rows and leave the cursor positioned on that rowset at the completion of the fetch:

```
FETCH FIRST ROWSET FROM C1 FOR 3 ROWS INTO :HVA1,:HVA2, ...
```

In the foregoing example:

- The FOR n ROWS clause specifies that with a single FETCH statement in the application program, DB2 fetches n rows. The value of ‘n’ determines the ROWSET size.
- The rowset is the group of rows that are returned by the single FETCH statement from the result table of the query.
- :HVA1 and :HVA2 following the INTO clause are the names of the host variable arrays. For more information on host variable arrays, see 4.5.5, “Host variable arrays” on page 246.

Single row and multiple row fetches can be mixed for a rowset cursor. If FOR n ROWS is NOT specified and the cursor is declared for rowset positioning, then the size of rowset is the same as the previous rowset fetch (as long as it was the previous fetch for this cursor), or the previous fetch was a FETCH BEFORE or FETCH AFTER and the fetch before that was a rowset fetch. Otherwise the rowset is 1.

Tip: To avoid any unexpected behavior, we recommend that you always code the FOR n ROWS clause.

Note that you can also specify the FETCH FIRST n ROWS only clause in the SELECT statement of the cursor. However, this clause has a very different meaning:

- In the SELECT statement, the FETCH FIRST n ROWS ONLY clause controls the maximum number of rows that can be accessed with the cursor. When a FETCH statement attempts to retrieve a row beyond the number specified in the FETCH FIRST n ROWS ONLY clause of the SELECT statement, an end of data condition (SQLCODE +100) occurs. In other words, it controls the total number of the rows of the result set.
- In a FETCH statement, the FOR n ROWS clause controls the number of rows that are returned for a single FETCH statement.

Host Variable Arrays

Redbooks

A host variable array is an array in which each element of the array contains a value for the same column; column-wise binding

Can only be referenced in multi-row FETCH and INSERT

Host variable arrays are supported in

- COBOL
- PL/I
- C and C++

Assembler support is limited to cases where USING DESCRIPTOR is allowed.

In general, arrays may not be arrays of structures

4.5.5 Host variable arrays

When using ROWSET operations, each FETCH retrieves column values for multiple rows. It is therefore necessary to set up host variable arrays to receive multiple values for each column. A host variable array is an array in which each element of the array contains a value for the same column. The number of host variable arrays that you must specify in a multi-row fetch or multi-row insert statement is the same as for single-row select and insert statements; that is, one host variable for each column in the table that you want to retrieve or insert. Host variable arrays can only be referenced in multi-row FETCH and INSERT operations.

To handle nulls, you also have to have an indicator array and you specify the name of this array following the host variable array. In the following example, COL1 is the host variable array and COL1IND is its indicator array. Assuming that COL1 has 10 elements (for fetching a single column of data for multiple rows of data), then COL1ID must also have 10 entries:

```
EXEC SQL
   FETCH C1 FOR 5 ROWS
   INTO :COL1:COL1IND
END_EXEC.
```

Host variable arrays are supported in COBOL, PL/I, C, and C++. Assembler support is limited to cases where USING DESCRIPTOR is allowed. (When using the USING DESCRIPTOR clause, you use the SQLDA instead.) The DB2 precompiler does not recognize the declaration of host variable arrays in Assembler programs. The programmer is responsible for allocating storage areas correctly. Multi-row FETCH is not supported in REXX, FORTRAN, and SQL procedure applications.

Our JDBC drivers currently do not support multi-row fetch and insert to pass data back and forth between the Java application and DB2. (They do use it under the covers in some cases, but that is transparent to the application.). However, we can expect to see support for this in the not-too-distant future.

At the time of writing of this publication (DB2 for Linux, UNIX, and Windows Version 8.1 FixPak 4), there is no support for either multi-row fetch or insert with embedded SQL.

The ODBC/CLI driver on the distributed platform has limited support for multi-row fetch and insert. The DB2 LUW ODBC driver supports "array fetch" and most of the DB2 for z/OS multi-row INSERT is supported.

ODBC on z/OS currently does not, but that may change in the foreseeable future.

COBOL Example

Declare cursor C1 and fetch 10 rows using a multi-row FETCH statement

```
01 OUTPUT-VARS.
   05 NAME OCCURS 10 TIMES.
      49 NAME-LEN       PIC S9(4) USAGE COMP.
      49 NAME-TEXT      PIC X(40).
   05 SERIAL-NUMBER     PIC S9(9) USAGE COMP OCCURS 10 TIMES.
01 IND-VARS.
   10 INDSTRUC1         PIC S9(4) USAGE COMP OCCURS 10 TIMES.
   10 INDSTRUC2         PIC S9(4) USAGE COMP OCCURS 10 TIMES.

PROCEDURE DIVISION.
EXEC SQL
  DECLARE C1 SCROLL CURSOR WITH ROWSET POSITIONING FOR
  SELECT NAME, SERIAL#   FROM EMPLOYEE
END-EXEC.
EXEC SQL   OPEN C1   END-EXEC.
EXEC SQL
  FETCH FIRST ROWSET   FROM C1 FOR 10 ROWS
    INTO :NAME:INDSTRUC1,:SERIAL-NUMBER:INDSTRUC2
END-EXEC.
```


4.5.6 COBOL example

The COBOL example on the visual demonstrates how to retrieve the first rowset consisting of 10 rows using a single FETCH statement.

The program retrieves two columns, NAME defined as VARCHAR(40) and SERIAL# defined as INTEGER from EMPLOYEE table. Both columns are nullable and so there is need for indicator variable arrays, as well as normal arrays for both column values.

Note that the cursor must be declared as a scrollable cursor. This is because we use the FETCH FIRST ROWSET clause. FETCH FIRST ROWSET can only be used if the cursor is declared scrollable.

Note also how the host variable arrays and indicator variable arrays are set up. You can use DCLGEN to set up the host variables and indicator variables. However, you must edit the DCLGEN output to include the OCCURS clause appropriately to set up the arrays.

PL/I Example

You can retrieve 10 rows from table DEPARTMENT with:

```
DCL DEPTNO(10)       CHAR(3),
       DEPTNAME(10)   CHAR(29) VAR,
       MGRNO(10)        CHAR(6),
       ADMRDEPT(10)   CHAR(3);
DCL IND_ARRAY1(10) BIN FIXED(15);
DCL IND_ARRAY2(10) BIN FIXED(15);
DCL IND_ARRAY3(10) BIN FIXED(15);
DCL IND_ARRAY4(10) BIN FIXED(15);
EXEC SQL
  DECLARE C1 SCROLL CURSOR WITH ROWSET POSITIONING FOR
  SELECT * FROM DEPARTMENT;
EXEC SQL  OPEN C1;
EXEC SQL
  FETCH FIRST ROWSET   FROM C1 FOR 10 ROWS
  INTO :DEPTNO:IND_ARRAY1, :DEPTNAME :IND_ARRAY2,
         :MGRNO:IND_ARRAY3, :ADMRDEPT:IND_ARRAY4;
```


4.5.7 PL/I example

The PL/I example on the visual demonstrates the use of a (scrollable) cursor to retrieve the first rowset consisting of 10 rows using a single FETCH statement.

The program retrieves four columns, DEPTNO defined as CHAR(3), DEPTNAME defined as VARCHAR(29), MGRNO defined as CHAR(6), and ADMRDEPT defined as CHAR(3) from DEPARTMENT table.

Note how the host variable arrays and indicator variable arrays are set up. You can use DCLGEN to have the host variables and indicator variables set up. However, you should edit DCLGEN output appropriately to set up the arrays.

C/C++ Example

Declare an integer and varying character array to hold columns retrieved from a multi-row fetch statement

```
long serial_num [10];
struct {
        short len;
        char data [18];
       } name [10];

...

EXEC SQL
  DECLARE C1 CURSOR FOR SELECT NAME, SERIAL_NO FROM EMPLOYEE
                 WITH ROWSET POSITIONING;
...

EXEC SQL
  OPEN C1;
...

EXEC SQL
  FETCH FIRST ROWSET FROM C1 FOR 10 ROWS INTO :NAME, :SERIAL_NUM;
```


4.5.8 C/C++ example

The C/C++ example in the figure above, demonstrates the use of a cursor to retrieve the first rowset consisting of 10 rows using a single FETCH statement.

The program retrieves two columns, NAME defined as VARCHAR(18), and SERIAL_NO defined as INTEGER from the EMPLOYEE table.

Note that we added a dimension parameter [10] for both host variable arrays. Also note that a VARCHAR column uses a C structure format.

Using Multi-Row FETCH with Scrollable Cursors

As holes may occur, ensure at least one indicator variable array is defined for a column

- Even if no nullable columns exist add an indicator variable array for at least one column
- If nullable columns exist all indicator variable arrays provided are updated if a hole is found

Value of -3 indicates hole

SQLCODE +222 is also returned

CUSTNO	ADDRESS_NULLABLE	IND-VAR	CUST_TYPE
1000	M		P
2000		-1	B
		-3	
4000	F		P

4.5.9 Using multi-row FETCH with scrollable cursors

A new value of -3 for an indicator variable indicates that values were not returned for the row because a hole was detected. The value of -3 is only used for multiple-row FETCH statements. You need to provide an indicator variable array for at least one column, even if there are no nullable columns in the result table. If multiple indicator variable arrays are provided, then the indication of the hole is reflected in each indicator array.

The purpose of an indicator variable is to indicate when the associated value is the null value, or that values were not returned because a hole was detected. The value is:

- -1 if the value selected was the null value, as in prior versions.
- -2 if the null value was returned due to a numeric conversion or arithmetic expression error that occurred in the SELECT list of an outer SELECT statement, as in prior versions.
- -3 if the null value was returned because a hole was detected for the row on a multiple row FETCH, and values were not returned for the row. In cases where -3 is set to indicate a hole, SQLSTATE 02502, SQLCODE +222, is also returned for that row.

If no indicator variable arrays are provided for a multiple-row FETCH statement, and a hole is detected, an error is returned (SQLSTATE 24519, SQLCODE -247).

In the example on the visual, CUSTNO and CUST_TYPE are NOT NULL columns and ADDRESS_NULABLE column is nullable. IND-VAR is the indicator array for ADDRESS_NULLABLE. For the second row in the rowset, (CUSTNO = 2000, the address is unknown, and the ADDRESS_NULLABLE column is not filled in, and the value for the

IND-VAR column is -1. For the third row in the rowset, the value for the IND-VAR column is -3, indicating a hole. No values are returned for any host variables. If the update or delete hole is part of the last rowset that is returned, a +222 SQLCODE is returned (not SQLCODE +100).

Rowsets

A ROWSET is a group of rows from the result table of a query, which are returned by a single FETCH statement (or inserted by a single (multi-row) INSERT statement)

The program controls how many rows are returned in a rowset (it controls the size of the rowset)

- Can be specified on the FETCH ... FOR n ROWS statement (n is the rowset size and can be up to 32767)

Each group of rows is operated on as a rowset

Ability to intertwine single row and multiple row fetches for a multi-fetch cursor

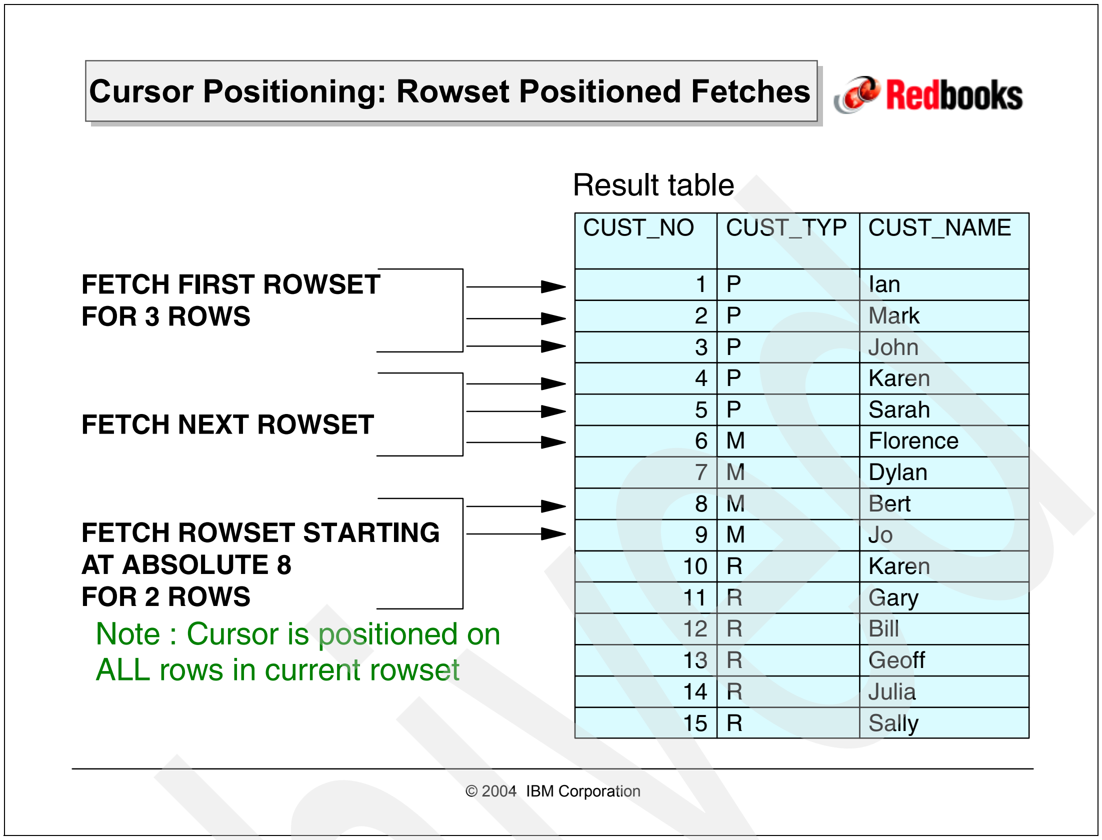
```
FETCH FIRST ROWSET STARTING AT ABSOLUTE 10
  FROM CURS1
  FOR 6 ROWS INTO :hva1, :hva2;
```


4.5.10 Rowsets

We have seen some examples of multi-row FETCH operations. A formal definition of a rowset is a group of rows for the result table of a query that are returned by a single FETCH statement. The maximum size of the rowset is 32767 and is controlled by the FETCH statement (FOR n ROWS clause). Each group of rows retrieved by a FETCH statement is operated on as a rowset. It is possible to have FETCH statements retrieving single rows and multi-row FETCH statements intertwined in a program.

When the cursor is opened, the associated SELECT statement is evaluated. How the FETCH statements operate on the result table, and cursor positions on the result table, depends on how the FETCH statement is issued.

The example on the visual shows a FETCH statement. The STARTING at ABSOLUTE 10 clause causes retrieval of rows starting at row 10. The FOR 6 ROWS clause causes rows 10 to 15 to be retrieved, and the cursor is positioned on all these 6 rows.



4.5.11 Rowset positioned fetches

FETCH FIRST ROWSET causes retrieval of rows starting from the first row in the result table. The rowset size is 3 because FOR 3 ROWS is specified on the FETCH.

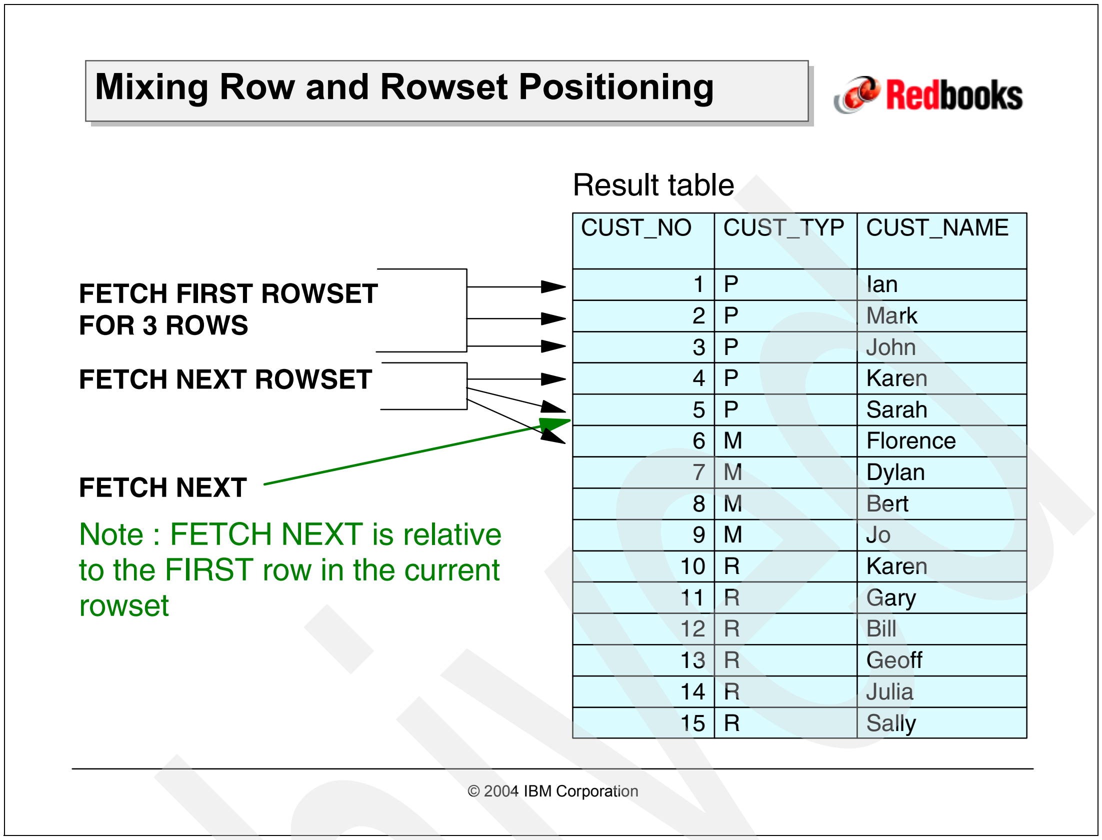
FETCH NEXT ROWSET causes retrieval starting from the first row after the previous rowset, in this case row 4. The rowset size continues to be 3 since FOR n ROWS is not specified.

FETCH ROWSET STARTING AT ABSOLUTE 8 FOR 2 ROWS cause retrieval of rows starting from row 8 in the result table. The rowset size is now 2 rows because the FETCH statement specifies a value (2) other than 3 for n.

Consider another scenario assuming that the cursor has just been opened:

- FIRST FETCH ROWSET FOR 3 ROWS ... cursor is positioned on rowset on rows 1,2,3.
- FETCH NEXT ROWSET ... cursor is positioned on rowset on rows 4,5,6.
- FETCH CURRENT ROWSET ... cursor is still positioned on the same rowset (rows 4,5,6).
- FETCH PRIOR ROWSET FOR 4 ROWS ... cursor is positioned on a “partial” rowset on rows 1,2,3 and returns a warning message. The reason for the warning is that we want to go back more rows in the result table than there are rows. It is OK to specify a different number of rows (4) on the FETCH PRIOR ROWSET, than was specified (explicitly or implicitly) on the previous FETCH (3).

  Although the rowset is logically obtained by fetching backwards from before the current rowset, the data is returned to the application starting with the first row of the rowset to the end of the rowset.



4.5.12 Mixing row and rowset positioning

You are allowed to mix rowset fetches and row fetches from the same cursor. However, the result may be different from what you expect.

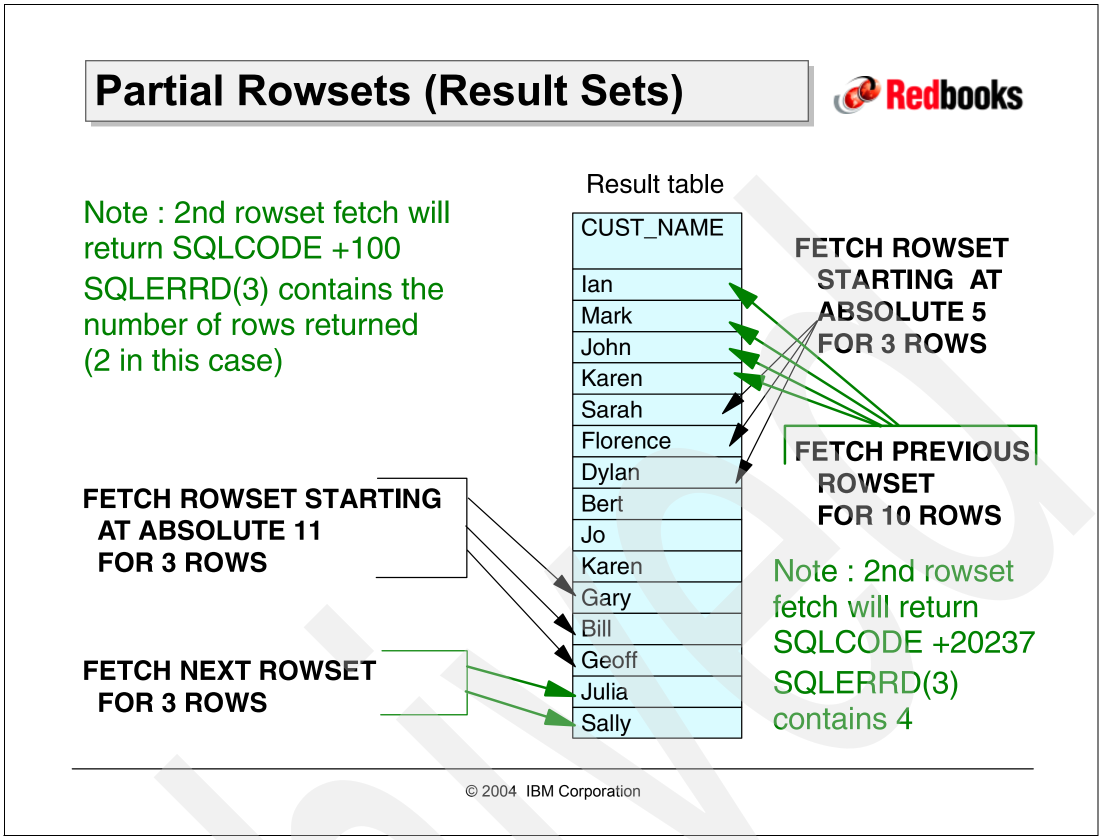
The example in the figure above starts with a FETCH FIRST ROWSET FOR 3 ROWS. This fetches the rowset with customer numbers 1, 2, and 3.

The next statement is a FETCH NEXT ROWSET. Because we did not specify the size of the rowset, the rowset size of the previous call is used. This means we fetch another rowset with three rows with customer numbers 4, 5, and 6.

Now we issue a row fetch by doing a FETCH NEXT or FETCH. In this case a single row is returned. The row that is returned is the next row after the first row that makes up the current rowset. That means that in this case, customer number 5 is returned (again). (The positioning for a row fetch is relative to the first row in the current rowset.)

Because it may cause confusion, we do not recommend mixing rowset and row fetches on the same cursor. It does work, but you have to know very well what you are doing in order not to retrieve the wrong data.

In case you want to retrieve customer number 7 (instead of 5), you can use a FETCH NEXT ROWSET FOR 1ROW. In that case a rowset of one row is returned but we continue where we left off on the previous rowset fetch.



4.5.13 Partial rowsets (result sets)

Another important topic to address when using multi-row fetch is what happens in case you reach the end of a result set. When you do an "old-fashioned" row fetch, when you fetch beyond the end of the result set you receive an SQLCODE +100 and you know that you are done (you do not have to look at the values of the fetch that returned the SQLCODE +100, as there is no more data).

Life becomes more complicated when using rowset fetches. When you fetch past the end of the result set, you also receive an SQLCODE +100. However, in this case, it may not be sufficient to stop processing the data. The last rowset that is returned may still contain a number of valid rows that you need to process; there may just not have been enough rows left in the result set to fill up a complete rowset.

This case is illustrated on the left side of the figure above. The second rowset fetch asks for 3 rows. However, the result set only contains 2 more rows. In this case those 2 remaining rows are returned in the provided host variable arrays and an SQLCODE +100 is returned. To determine how many rows were returned in the rowset, you have to examine the SQLERRD(3) flag in the SQLCA (or use GET DIAGNOSTICS). In this case SQLERRD(3) contains the value 2.

The same situation can occur when you fetch beyond the beginning of the result set. This is shown on the right-hand side in the figure above. If you are positioned on the rowset for rows 5,6,7 after the first rowset fetch and issue a FETCH PREVIOUS ROWSET FOR 10 ROWS, you go beyond the start of the result set. In this case you receive a different SQLCODE +20237, and SQLERRD(3) will contain 4, and the provided host variable array will contain the four previous values in the result set.

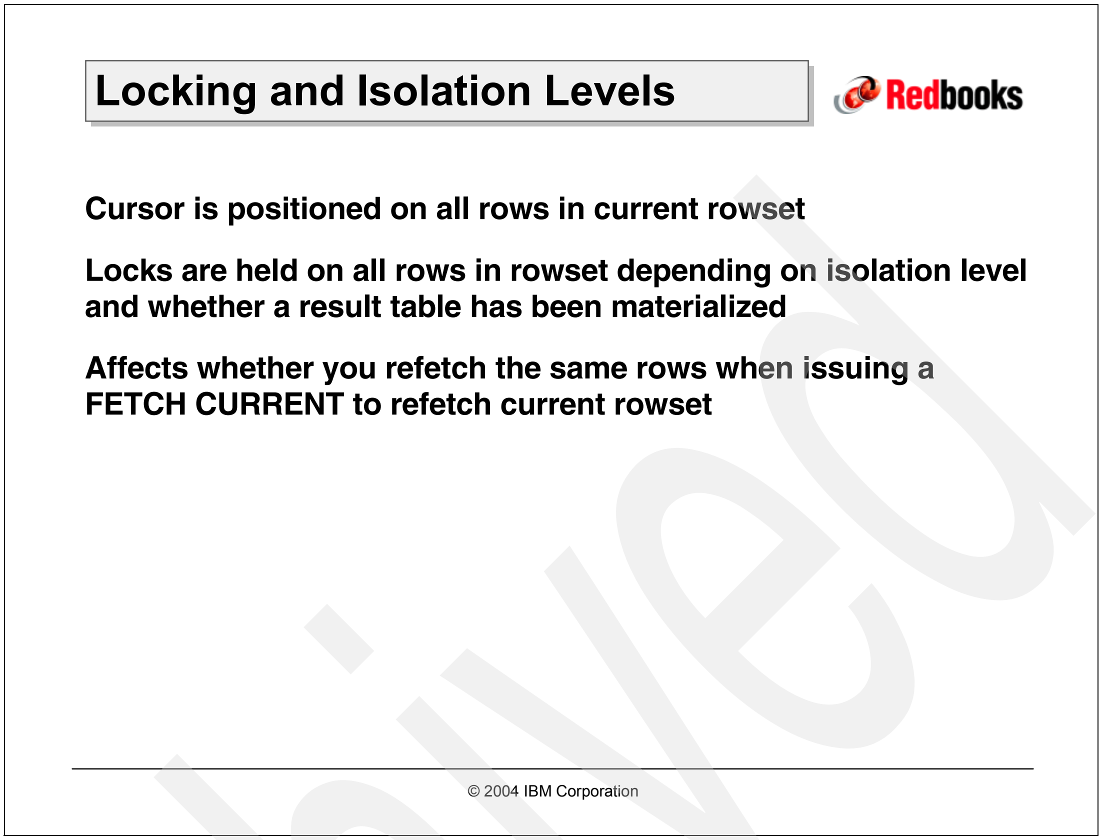
Fetching beyond the result set ABSOLUTE or RELATIVE

If you fetch beyond the end of the result set, or beyond the beginning of the result set, using a FETCH ABSOLUTE or FETCH RELATIVE, you will receive an end of data condition.

So in case you are positioned on row 10 in the result set of the figure on the previous page (containing 15 rows), and you issue a:

- FETCH ROWSET STARTING AT ABSOLUTE 20
- FETCH ROWSET STARTING AT RELATIVE -20

Here, no rows will be returned. SQLERRD(3) will be 0 (zero) and an SQLCODE +100 is returned. The cursor itself will be positioned after (first fetch statement) or before (second fetch statement) depending on the direction of the fetch.



4.5.14 Locking and isolation levels

The isolation level of the statement (specified implicitly or explicitly) can affect the result of a rowset-positioned FETCH statement. This is possible, for example, when changes are made to the tables underlying the cursor when isolation level UR is used with a dynamic scrollable cursor, or with other isolation levels when rows have been added by the application fetching from the cursor.

This is not any different in versions prior to DB2 V8. The only change in DB2 V8 is that locks may be held on multiple rows.

Considerations with Static Scrollable Cursors

Redbooks

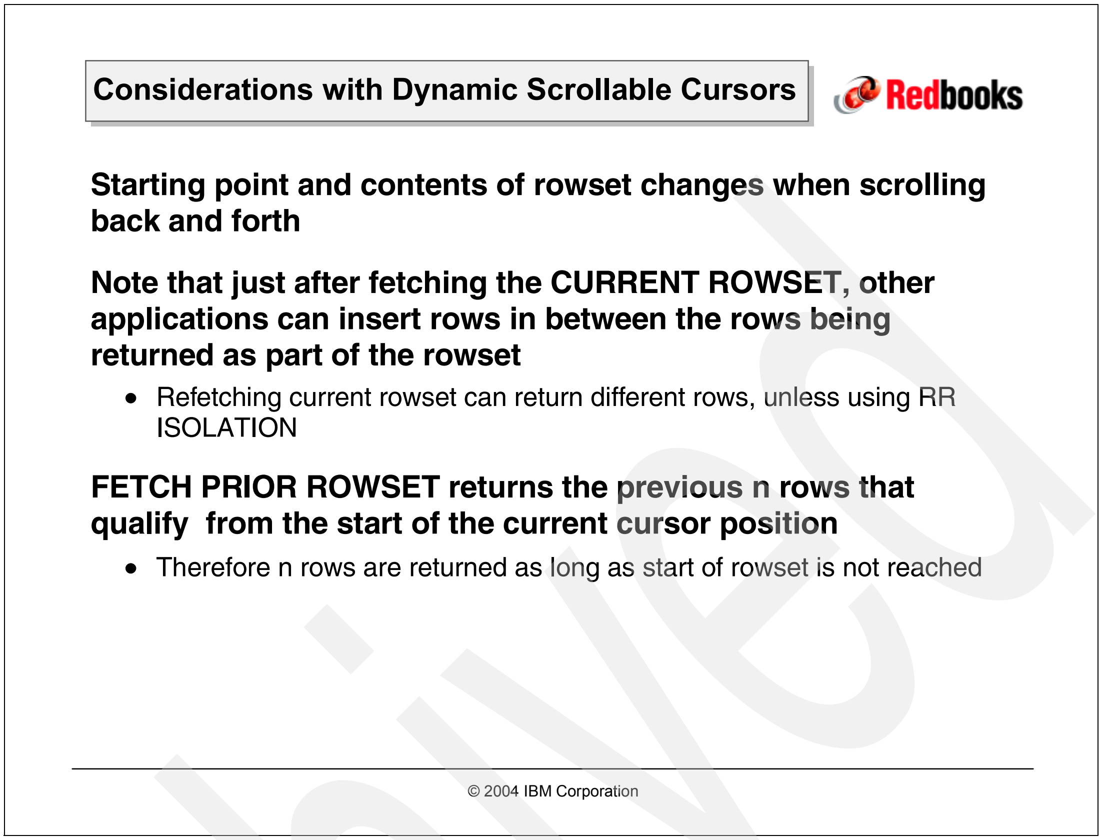
When scrolling with rowsets:

- With insensitive fetches, updates by the application itself can cause changes and holes
- With sensitive fetches, updates by other applications can also cause changes and holes
- For example, FETCH PRIOR ROWSET may return update or delete holes in place of rows that were fetched before

Row contents can change between fetches

4.5.15 Considerations with static scrollable cursors

Be aware of the considerations shown on the visual when using multi-row FETCH with static scrollable cursors. Depending on the type of fetching you do, you can see updates or deletes, made through your cursor (insensitive fetch), or also updates and deletes made by other applications (sensitive fetch). In both cases, you can have update and/or delete holes in your rowset. These are the same considerations with static scrollable cursors in V7.



4.5.16 Considerations with dynamic scrollable cursors

The current row of a cursor cannot be updated or deleted by another application process if it is locked. Unless it is already locked because it was inserted or updated by the application process during the current unit of work, the current row of the cursor is not locked if:

- The isolation level is UR, or
- The isolation level is CS, and
  - The result table is of the cursor is read-only.
  - The bind option is CURRENTDATA(NO) (which allows for lock avoidance), and DB2 actually managed to avoid having to take a lock.

The following situations can occur depending on the isolation level, other activity in the system and the fetch orientation you use:

- PRIOR ROWSET:

  With a dynamic scrollable cursor and isolation level UR, the content of a prior rowset can be affected by other activity within the table. It is possible that a row that previously qualified for the cursor, and was included as a member of the "prior" rowset, has since been deleted or modified before it is actually returned as part of the rowset for the current statement. The same is true for cursor stability with CURRENTDATA(NO), when lock avoidance is used. To avoid this behavior, use an isolation level other than UR or CS with CURRENTDATA(NO).

- CURRENT ROWSET:

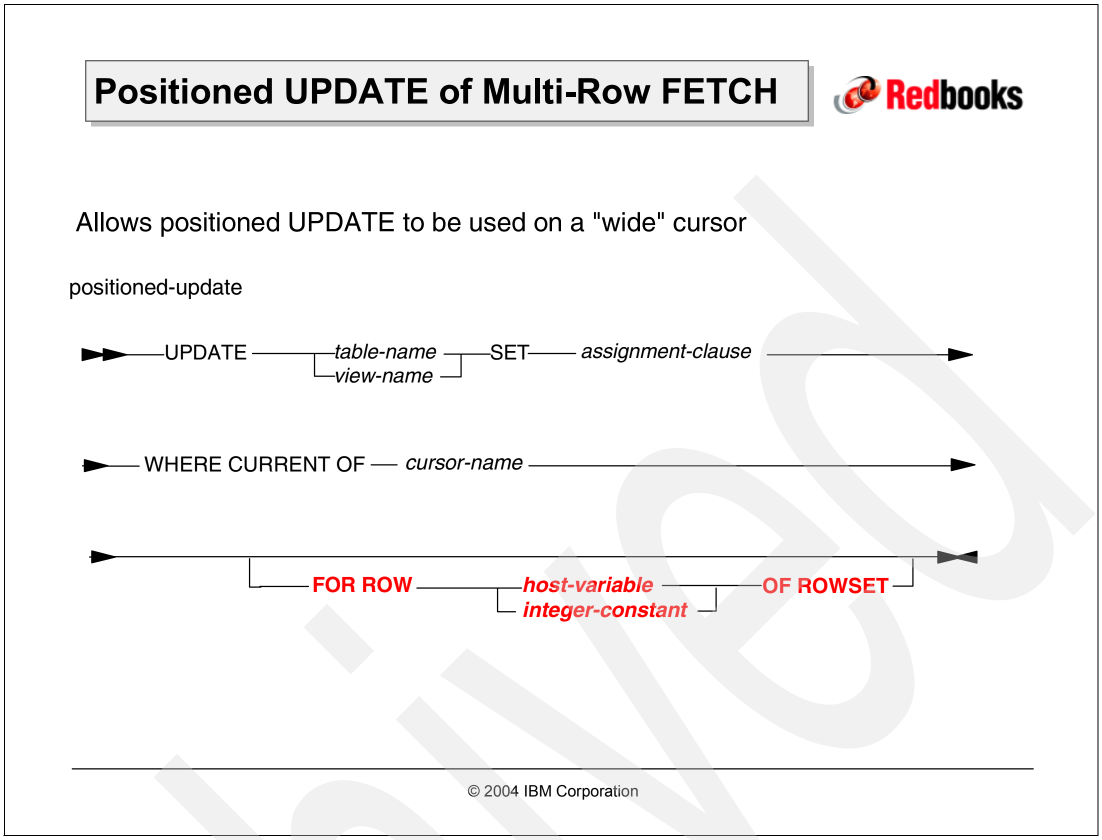
  With a dynamic scrollable cursor, additional rows can be added between rows that form the rowset that was returned to the user. With isolation level RR, these rows can only be added by the application fetching from the cursor. For isolation levels other than RR, other applications can insert rows that can affect the results of a subsequent FETCH CURRENT ROWSET. To avoid this behavior, use a static scrollable cursor instead of a dynamic scrollable cursor.

- LAST ROWSET:

  With a dynamic scrollable cursor and isolation level UR, the content of the last rowset can be affected by other activity within the table. It is possible that a row that previously qualified for the cursor, and was included as a member of the "last" rowset, has since been deleted or modified before it is actually returned as part of the rowset for the current statement. To avoid this behavior, use an isolation level other than UR.

- ROWSET STARTING AT RELATIVE n (where n is a negative number):

  With a dynamic scrollable cursor and isolation level UR, the content of a prior rowset can be affected by other activity within the table. It is possible that a row that previously qualified for the cursor, and was included as a member of the "prior" rowset, has since been deleted or modified before it is actually returned as part of the rowset for the current statement. To avoid this behavior, use an isolation level other than UR.



4.5.17 Positioned UPDATE of multi-row FETCH

You can issue the positioned UPDATE statement in one of the following two ways:

- WHERE CURRENT OF ... as in prior versions and this is the default.

  When the UPDATE statement is executed, the cursor must be positioned on the row or rowset of the result table:

  - If positioned on a single row, that row is updated.
  - If positioned on a rowset, all rows corresponding to rows of the current rowset are updated.

- WHERE CURRENT OF ... followed by FOR ROW ... OF ROWSET, which is new in V8.

  This enables you to update a specific row in a rowset.

Positioned UPDATE Examples

Redbooks

Cursor CS1 is positioned on a rowset consisting of 10 rows of table T1:

Example 1**:**

The following UPDATE statement is used to update all 10 rows of the rowset:

EXEC SQL UPDATE T1 SET C1 = 5 WHERE CURRENT OF CS1 END-EXEC

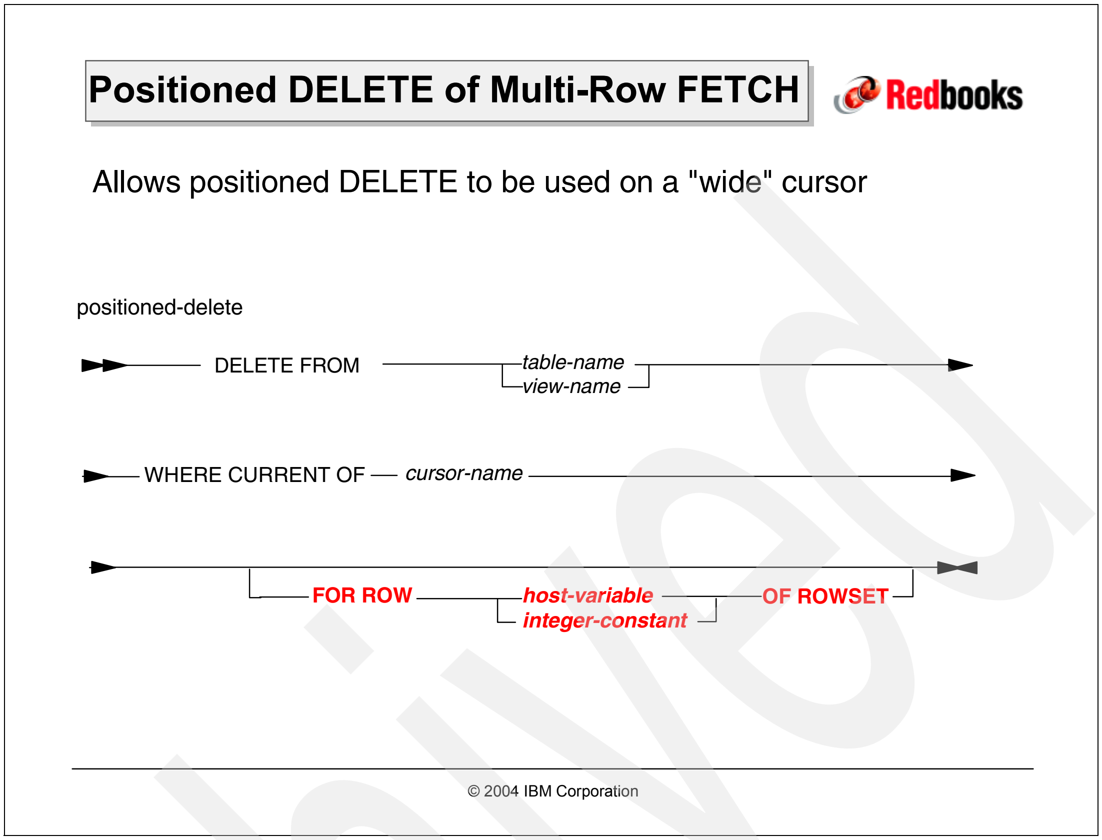
Example 2:

The following UPDATE statement is used to update row 4 of the rowset:

```
EXEC SQL
UPDATE T1  SET COL1='ABC'
           WHERE CURRENT OF CS1 FOR ROW 4 OF ROWSET
END-EXEC
```


Positioned UPDATE examples

If you use the UPDATE statement in conjunction with multi-row FETCH processing, notice that without the "FOR ROW n OF ROWSET" clause, all rows of the current rowset are updated, which would have a dramatically different effect on data being changed.



4.5.18 Positioned DELETE of multi-row FETCH

You can issue the positioned DELETE statement in one of the following two ways:

- WHERE CURRENT OF ... as in prior versions, and this is the default.

  When the DELETE statement is executed, the cursor must be positioned on a row or rowset of the result table:

  – If positioned on a single row, that row is deleted and, after deletion, the cursor is positioned before the next row of the result table. If there is no next row, cursor is positioned after the last row.

  – If positioned on a rowset, rows corresponding to rows of the current rowset are deleted and, after deletion, the cursor is positioned before the next rowset of the result table. If there is no next rowset, the cursor is positioned after the last rowset.

- WHERE CURRENT OF ... FOR ROW ... OF ROWSET, which is new in DB2 V8.

  This enables you to delete a specific row in a rowset.

Positioned DELETE Examples

Cursor CS1 is positioned on a rowset consisting of 10 rows of table T1:

Example 1:

The following DELETE statement is used to delete all 10 rows of the rowset:

```
EXEC SQL DELETE FROM T1 WHERE CURRENT OF CS1 END-EXEC
```

Example 2:

The following DELETE statement is used to delete row 4 of the rowset:

```
EXEC SQL
DELETE FROM T1
WHERE CURRENT OF CS1 FOR ROW 4 OF ROWSET
END-EXEC
```


Positioned DELETE examples

If you use the DELETE statement in conjunction with multi-row FETCH processing, notice that without "FOR ROW n OF ROWSET" clause, all 10 rows are deleted, which would have a dramatically different effect on data being deleted.

Multi-Row INSERT

Redbooks

New third form of insert

- INSERT with FOR "n" ROWS is used to insert multiple rows into the table or view using values provided in a host variable array

FOR "n" ROWS

- For static, it is valid to specify FOR "n" ROWS on the INSERT statement (for dynamic INSERT, specify FOR "n" ROWS on the EXECUTE statement)
- Input provided with host variable array -- each array represents cells for multiple rows of a single column

VALUES clause allows specification of multiple rows of data

- Host variable arrays used to provide values for a column on INSERT
- Example: VALUES (:hva1, :hva2)

4.6 Multi-row INSERT

Prior to DB2 V8, inserting rows into a table can be done in one of the following ways:

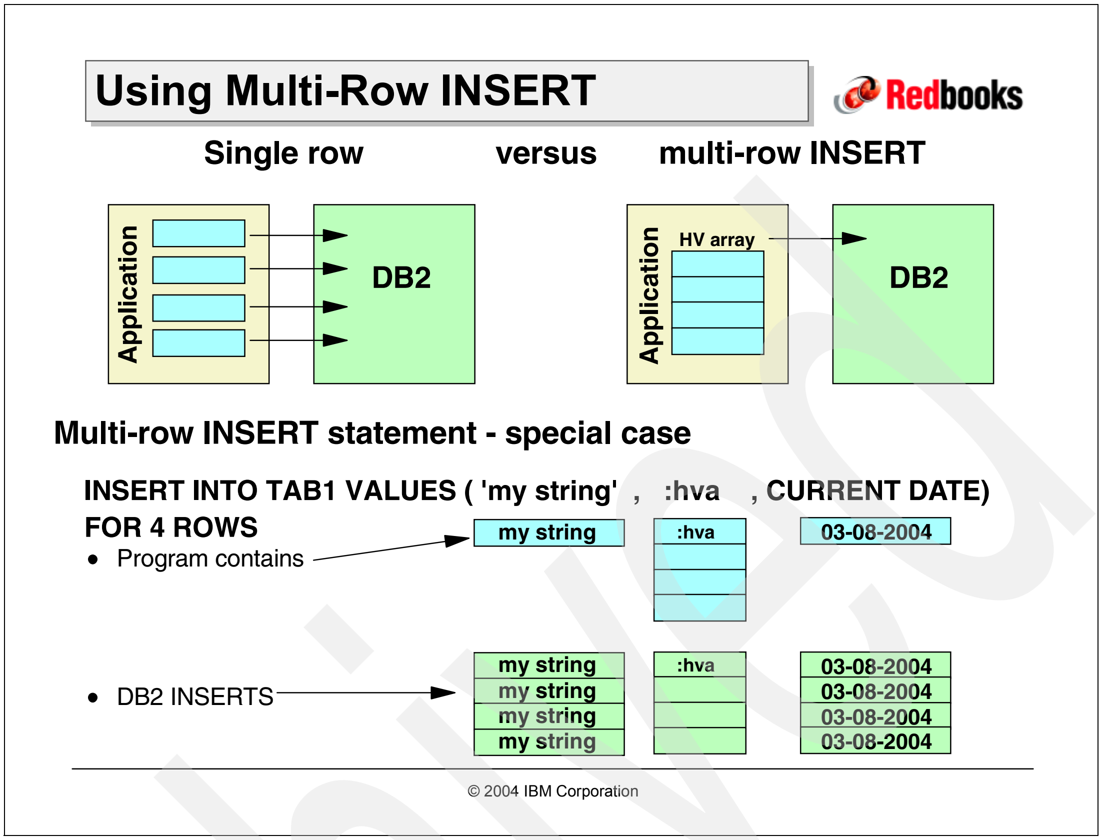
- INSERT with VALUES is used to insert a single row into the table using values provided or referenced.
- INSERT with SELECT is used to insert one or more rows into the table using values from other tables or views.

DB2 V8 has introduced another way to insert multiple rows using values provided in host variable arrays.

To use a multiple-row FETCH or INSERT statement with a host variable array per column, the application must define one or more host variable arrays that can be used by DB2. Each language has its own conventions and rules for defining a host variable array (see “Host variable arrays” on page 246).

A host variable array corresponds to the values for one column of the result table for FETCH, or column of data to be inserted for INSERT. The first value in the array corresponds to the value for that column for the first row, the second value in the array corresponds to the value for the column in the second row, and so on. DB2 determines the attributes of the values in the array based on the declaration of the array. Host variable arrays are used to return the values for a column of the result table on FETCH, or to provide values for a column on INSERT.

Multi-row INSERT can be used to reduce network traffic when input is presented through a remote application.

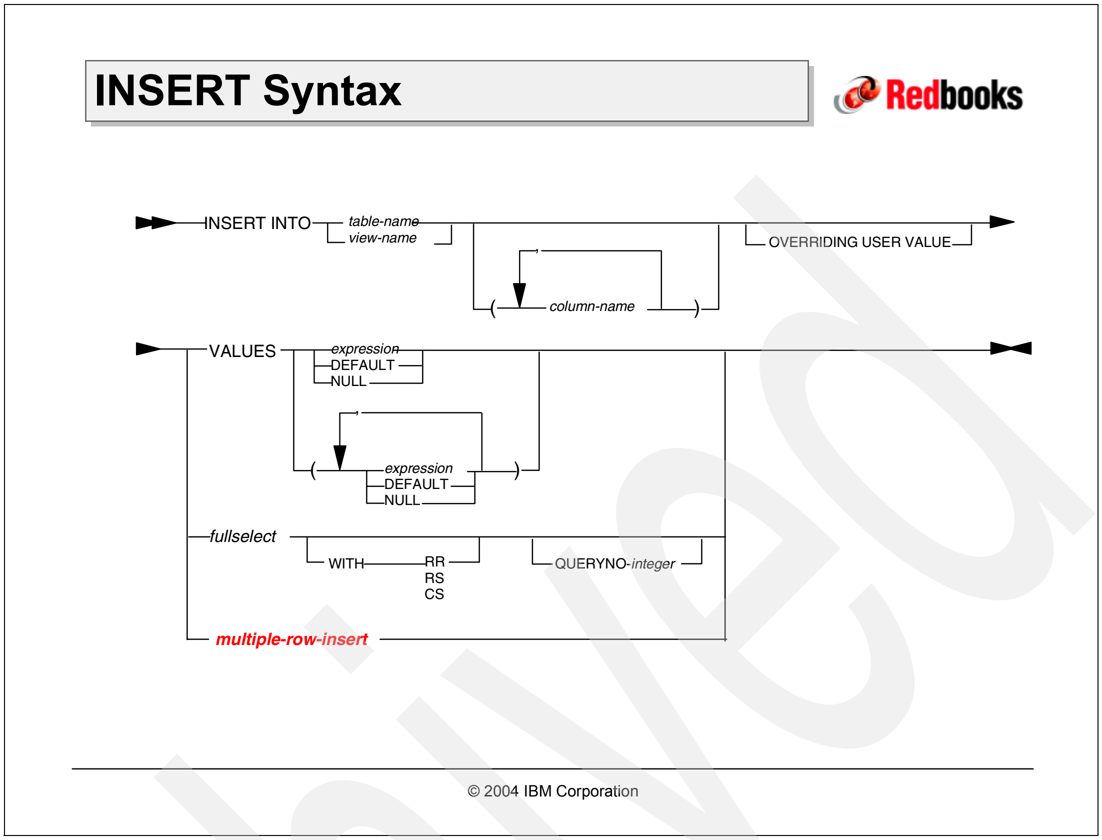


4.6.1 Using multi-row INSERT

The figure above is divided horizontally into two parts.

The top part of the picture shows the difference between individual row inserts and multi-row insert. In the first case (single-row insert) we have as many trips from the application to DB2 as we insert rows. In the second case (multi-row insert), in a single trip across the API, we insert 4 rows (the maximum is 32K rows in a single insert).

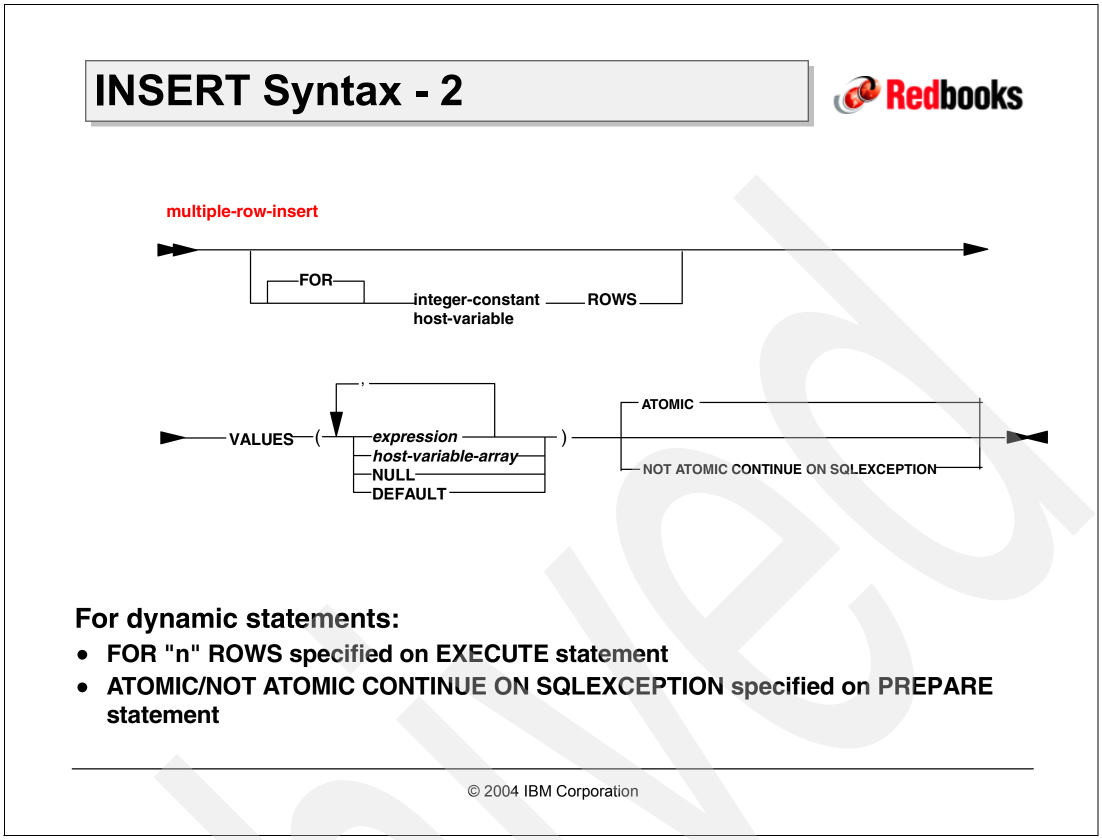
The bottom part of the picture shows that when using multi-row insert, not all variables have to be a host variable array. In the example we use a fixed character string and a special register. During the insert, the values of those are “duplicated” as many times as required to match the FOR n ROWS clause. (Note that the literal “my string”' can also be a normal host variable.



4.6.2 INSERT syntax changes

This visual shows a new block *multiple-row-insert has been* introduced in the INSERT syntax to facilitate multi-row INSERT operation.

This is applicable to *static* SQL INSERT. There are two flavors, static and dynamic.



4.6.3 INSERT syntax - multiple-row-insert block

The visual shows the details of the new *multiple-row-insert* block.

The FOR n ROWS clause is used to insert multiple rows into a table or view. To facilitate this, the VALUES clause has the *host-variable-array* specification.

ATOMIC/NOT ATOMIC CONTINUE ON SQLEXCEPTION

ATOMIC (default)

- If the insert for any row fails, all changes made to database by that INSERT statement are undone

NOT ATOMIC CONTINUE ON SQLEXCEPTION

- Inserts are processed independently
- If errors occur during execution of INSERT, processing continues
- Diagnostics are available for each failed row through GET DIAGNOSTICS
- SQLCODE indicates if:
  - All failed
    - SQLSTATE 22530, SQLCODE -254
  - All were successful, but warnings
    - SQLSTATE 01659, SQLCODE +252
  - At least one failed
    - SQLSTATE 22529, SQLCODE -253

4.6.4 ATOMIC and NOT ATOMIC CONTINUE ON SQLEXCEPTION

ATOMIC or NOT ATOMIC CONTINUE ON SQLEXCEPTION clause is provided so that the application can specify if it wants the multiple-row INSERT to succeed or fail as a unit, or if it wants DB2 to proceed despite a partial failure (one or more rows).

ATOMIC specifies that if the insert for any row fails, then all changes made to the database by any of the inserts, including changes made by successful inserts, are undone. This is the default.

When NOT ATOMIC CONTINUE ON SQLEXCEPTION is specified, the inserts are processed independently. This means that if one or more errors occur during the execution of an INSERT statement, then processing continues and any successful inserts made during the execution of the statement are not undone. You can use the GET DIAGNOSTICS statement to keep track of this. For more information on GET DIAGNOSTICS, see 4.8, “GET DIAGNOSTICS statement” on page 279.

A consideration regarding ATOMIC or NOT ATOMIC CONTINUE ON SQLEXCEPTION is the amount of data you are inserting. Inserting 32K rows into a table whose rows are 32K bytes long (the row is in a 32K page and 1 row/page) consumes 1G of space in the application and would log > 1G of data, so rollback could be painful.

Another consideration is the effect on triggers.With a multiple row INSERT statement, when triggers are processed is dependent on the atomicity option in effect for the statement:

- ATOMIC:

  The inserts are processed as a single statement, any statement level triggers are invoked once for the statement, and transition tables include all of the rows inserted.

- NOT ATOMIC CONTINUE ON SQLEXCEPTION:

  The inserts are processed separately, any statement level triggers are processed for each inserted row, and transition tables include the individual row inserted. With this option in effect when errors are encountered, processing continues, and some of the specified rows do not end up being inserted. In this case, if an insert trigger is defined on the underlying base table, the trigger transition table includes only rows that are successfully inserted.

ATOMIC is easier to restart/reposition from since it is an “ALL or NONE” type of process.

INSERT - Example 1

Redbooks

Insert a variable number of rows using host variable arrays for column values. Assume table T1 has one column and that a variable (:hv) number of rows of data are to be inserted into T1.

```
EXEC SQL INSERT INTO T1 FOR :hv ROWS
         VALUES (:hva:hvaind) ATOMIC;
```

In this example, :hva represents the host variable array and :hvaind represents the array of indicator variables

4.6.5 INSERT - example 1

Notice the use of the keyword ATOMIC.

Therefore either all rows are inserted, or in case of a failure no rows are inserted. It is either all or nothing.

Note that if the host variable array has fewer elements than n specified in the FOR n ROWS clause, SQL error is returned (SQLSTATE 42873, SQLCODE -246). If the host variable array has more elements than specified in the FOR n ROWS clause, the excess elements are just ignored.

INSERT - Example 2

Redbooks

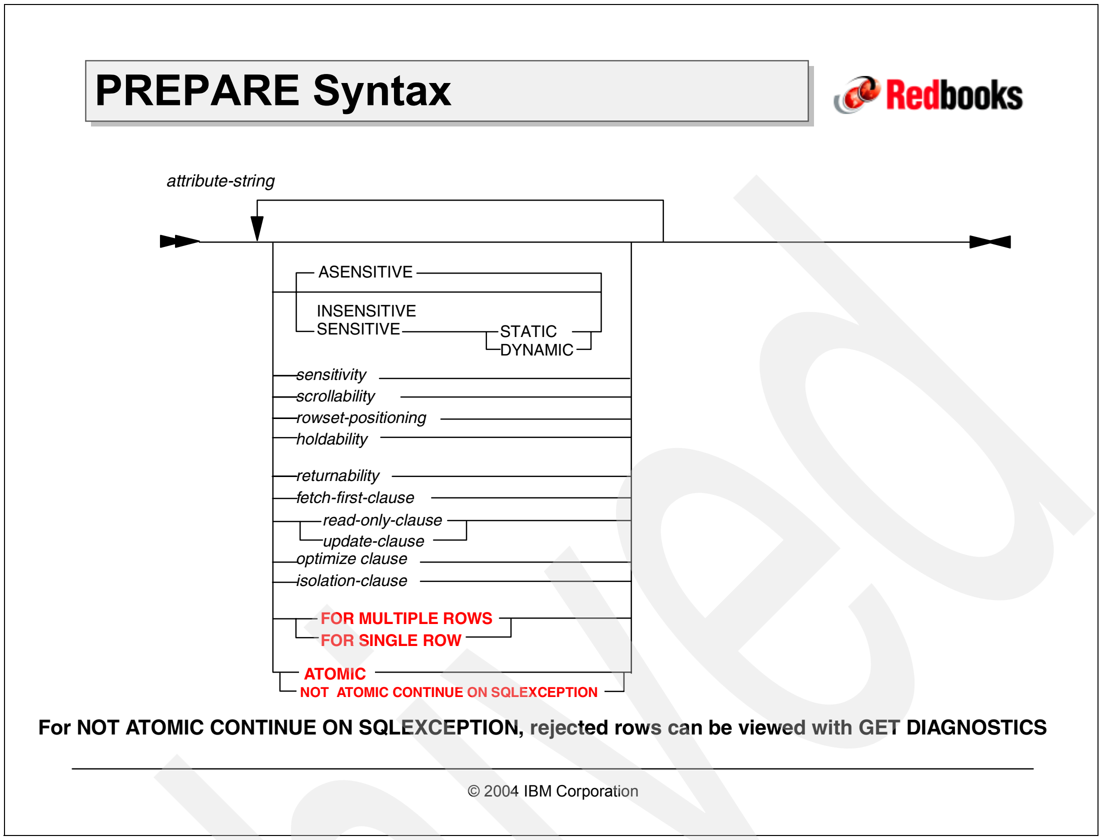
Insert multiple rows using host variable arrays for column values.
Table T2 has 2 columns C1 and C2. INSERT 10 rows of data into T2.
Values to be inserted are in host variable arrays :hva1 and :hva2.
The corresponding indicator variable arrays are :hvaind1 and :hvaind2.

```
EXEC SQL INSERT INTO T2 (C1, C2) FOR 10 ROWS
         VALUES (:hva1:hvaind1, :hva2:hvaind2)
         NOT ATOMIC CONTINUE ON SQLEXCEPTION;
```


4.6.6 INSERT - example 2

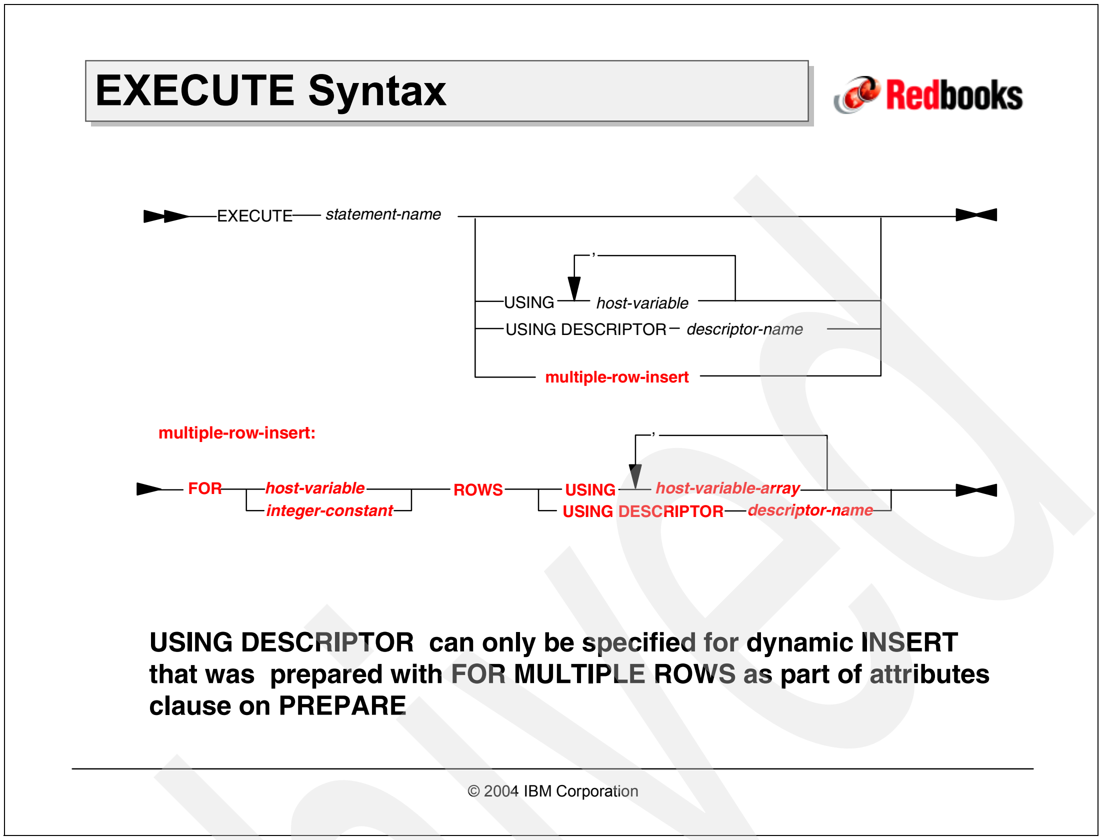
Notice the use of the keyword NOT ATOMIC CONTINUE ON SQLEXCEPTION.

If the insert of one or more of the rows fails, the application is notified of the failure through the SQLCA and must then issue a GET DIAGNOSTICS (see “GET DIAGNOSTICS statement” on page 279) in order to determine the failing record(s). All the other rows are successfully inserted.



4.6.7 PREPARE syntax

The visual shows the changed PREPARE syntax to facilitate inserting multiple rows of data with a single dynamic SQL INSERT statement. The keywords ATOMIC and NOT ATOMIC CONTINUE ON SQLEXCEPTION have the usual meaning.



4.6.8 EXECUTE syntax

The visual shows the changed PREPARE syntax to facilitate inserting multiple rows of data with a single dynamic SQL INSERT statement.

PREPARE/EXECUTE Example

Assume that table PROG has 9 columns. Prepare and execute a dynamic INSERT statement to insert 5 rows of data into PROG.

```
stmt = 'INSERT INTO PROG (C1, C2, C3, C4, C5, C6, C7, C8, C9)
          VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)';
attrvar = 'FOR MULTIPLE ROWS ATOMIC'
nrows = 5
EXEC SQL PREPARE ins_stmt ATTRIBUTES :attrvar FROM :stmt;
EXEC SQL EXECUTE ins_stmt FOR :nrows ROWS
            USING :V1, :V2, :V3, :V4, :V5, :V6, :V7, :V8, :V9
```

In this example, each host variable in the USING clause represents an array of values for the corresponding column of the target of the INSERT statement

4.6.9 PREPARE/EXECUTE example

Notice the difference in syntax between static SQL and dynamic SQL for multi-row INSERT statement:

- Static SQL:

  The FOR n ROWS and ATOMIC/NOT ATOMIC CONTINUE ON SQLEXCEPTION clauses are specified on the INSERT statement.

- Dynamic SQL:

  The FOR n ROWS clause is specified on the EXECUTE statement.

  The ATOMIC/NOT ATOMIC CONTINUE ON SQLEXCEPTION clause is specified on the PREPARE statement.

In the example on the visual, the PREPARE statement uses the INSERT statement string in host variable *stmt* and the attributes in the host variable *attrvar* and places the prepared statement in the DB2 designated area *ins_stmt* for use by the EXECUTE statement.

Multi-Row FETCH / INSERT -- DRDA Considerations

Redbooks

Can be implemented by any requester or server that supports DRDA Version 3

Between DB2 for z/OS V8 systems

- Multi-row INSERT does not affect blocking
  A single rowset is inserted in a single INSERT statement
- A single network multi-row fetch or insert request can maximum be 10 MB
- For remote client one rowset returned in a network request
  - 32K block size is ignored for multi-row fetch

Between DB2 on distributed platforms and DB2 for z/OS V8

- No support for multi-row operations in embedded SQL
- In ODBC/CLI driver
  - Limited support for multi-row FETCH
  - Support for multi-row INSERT

4.7 Multi-row FETCH / INSERT - DRDA considerations

Let us first look at multi-row operations between DB2 for z/OS systems.

Both AR and AS are DB2 for z/OS V8 systems

Between DB2 for z/OS systems, V8 limits the size of user data and control information to 10M (except for LOBs, which are processed in a different data stream) for a single multi-row FETCH statement using host variable arrays.

For remote applications using rowset cursors, a single network request always returns one rowset. Once you specify the rowset size, that becomes the block size of the query block. So, if you specify 10 rows as the rowset size, only 10 rows come back per block. The 32 KB query block size is turned off. This means that if you specify a rowset that is very small (compared to the 32 KB block size), your performance can degrade.

When an error occurs at a remote requester, SQLCODE -353 may be issued on a subsequent fetch. In some of these cases, SQLCODE -353 is issued by the remote requester even though the same FETCH statement when executed as part of a local application would have been successful.

For example, suppose that a row with a null value was returned to an application, but no indicator variable was provided by the application. In this case, SQLCODE -353 is issued on a subsequent fetch by a remote application, but not for a local application.

Multi-row INSERT and FETCH statements can be implemented by any DRDA application requester or server that supports the DRDA Version 3 protocols. SQLCODE -30005, SQLSTATE 56702 is returned if an attempt is made to issue a multi-row INSERT or FETCH statement on a server that does not support DRDA Version 3 protocols. The fact that both the AR and AS support DRDA V3 is of course not enough to make this work. The AR and AS have to implement rowset processing inside the engine or driver as well.

Multi-row fetch and insert operations are fully supported between DB2 for z/OS systems. The support between DB2 distributed clients and DB2 for z/OS is discussed next.

Multi-row operations between the distributed platform and DB2 for z/OS

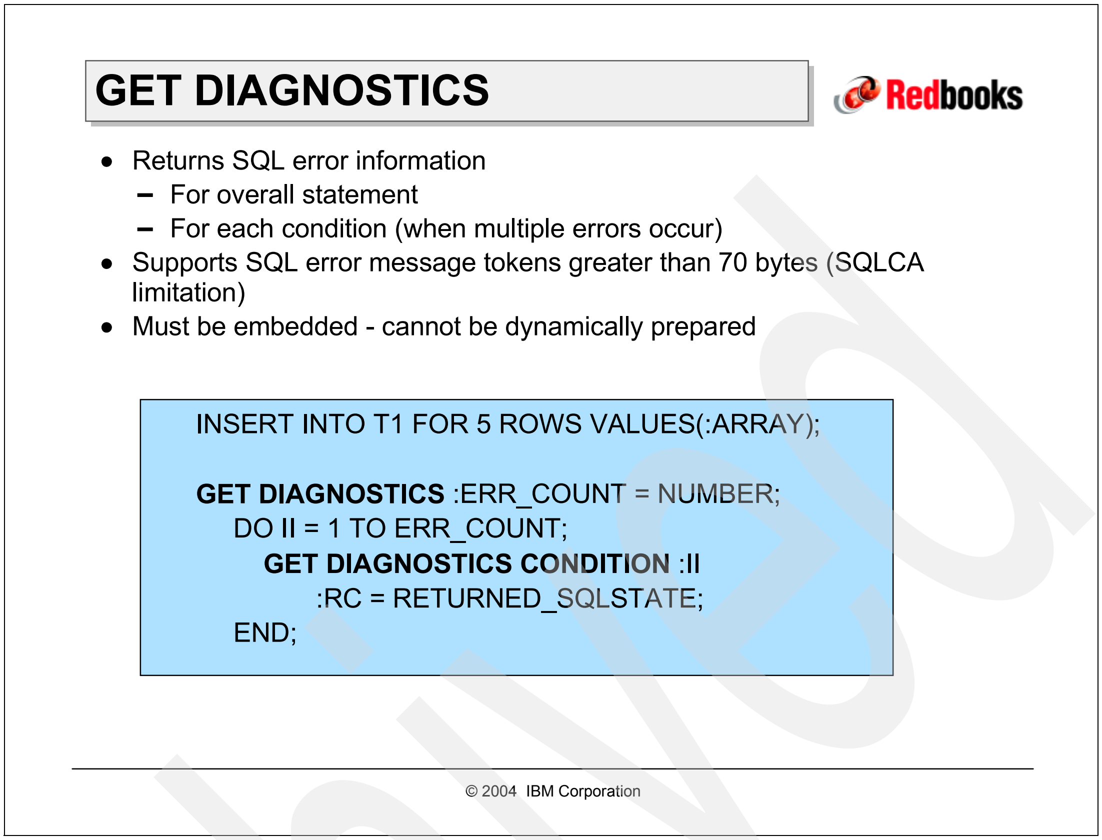
At the time of writing of this publication (DB2 Connect for Linux, UNIX, and Windows Version 8.1 FixPak 4), there is no support for either multi-row fetch or insert with embedded SQL.

The ODBC/CLI driver on the distributed platform supports most of the DB2 for z/OS multi-row INSERT functionality, but there is no support for "WITH ROWSET POSITIONING" cursors beyond the their implicit use by dynamic scrollable cursors.

To use multi-row INSERT, you code an ODBC "array-input" in your application. When the DB2 AR realizes that it is talking to a DB2 for z/OS V8 running in new-function mode, it transforms the flow into a single message containing a single INSERT of n rows (multi-row insert). (When communicating with a DB2 V7 on the mainframe, the ODBC/CLI driver at the DB2 for LUW client would have sent a single message containing n INSERT statements, each for a single row.) You can specify whether this is an atomic operation or not by using the statement attributes SQL_ATTR_PARAMOPT_ATOMIC and SQL_ATOMIC_NO.

DB2 Connect on Linux, UNIX, and Windows V8.1 has implemented dynamic scrollable cursors via ODBC (or DB2 CLI) with FixPak 4. Via calling the SQLSetStmtAttr() function with the SQL_ATTR_CURSOR_TYPE statement attribute set to SQL_CURSOR_DYNAMIC, you can make the statement a dynamic scrollable cursor. These dynamic scrollable cursor always use “WITH ROWSET POSITIONING” cursors when retrieving data for a DB2 for z/OS V8 system. At the time of writing, this is the only implementation of rowset cursors on the distributed platform.

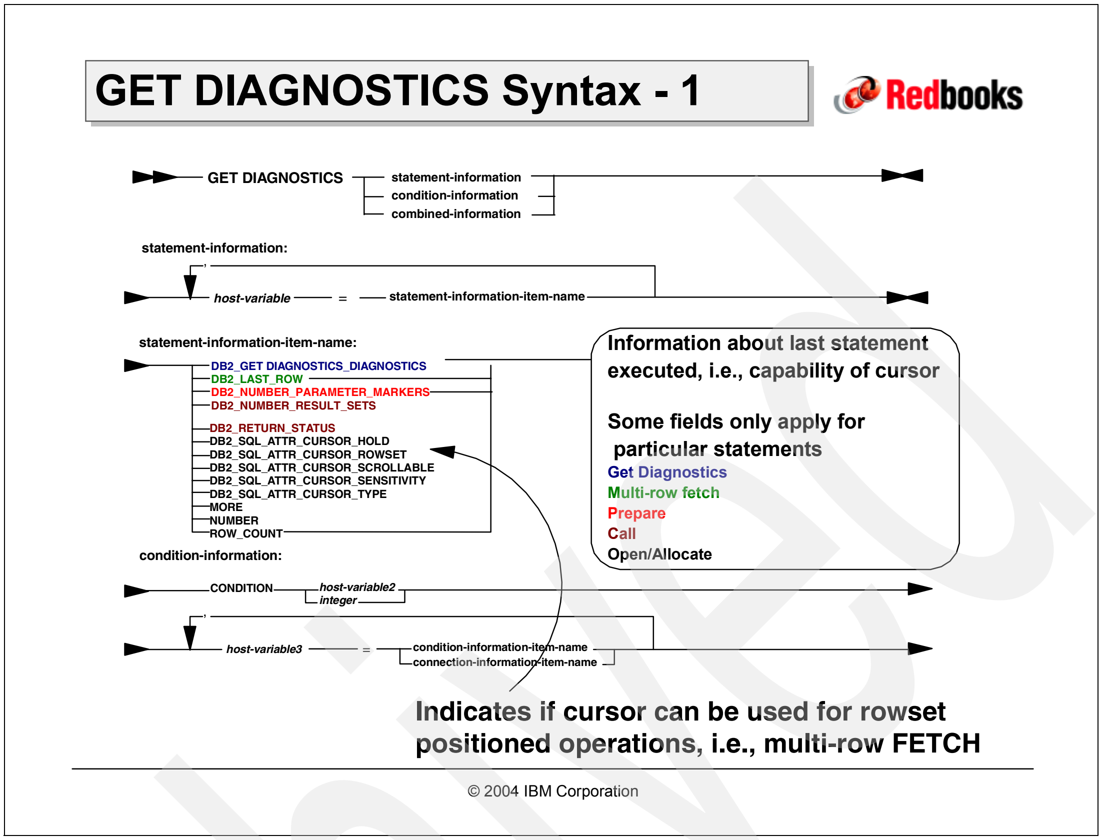
Using “array fetch”, you can code arrays for output column values, and then via SQLSetStmtAttr set the number of rows for an SQLFetch, SQLExtendedFetch, or SQLScroll. However, this only applies to the client application program talking to the ODBC/CLI driver on the workstation. No cursor with rowset positioning is used when going out to obtain data from the database on DB2 for z/OS. (To help out a little bit, the DB2 for z/OS server implements multi-row fetch “under the covers” when DDF is fetching rows from the database, but the query block will be based on the client's RQRIOBLK size, which can only go up to 65535 in the current level of DB2 Connect, and only impacts the API crossings between the DDF and DBM1 address space.)



4.8 GET DIAGNOSTICS statement

The *GET DIAGNOSTICS* statement enables applications to retrieve diagnostic information about statements that have been executed. This statement complements and extends the diagnostics that are available in the SQLCA. This statement can only be embedded in an application program that cannot be dynamically prepared.

GET DIAGNOSTICS can be used in conjunction with and instead of the SQLCA to interrogate the results of all SQL statements. It is especially important when dealing with non-atomic multi-row insert statements, and objects with long names, that potentially no longer fit into the SQLCA message area.



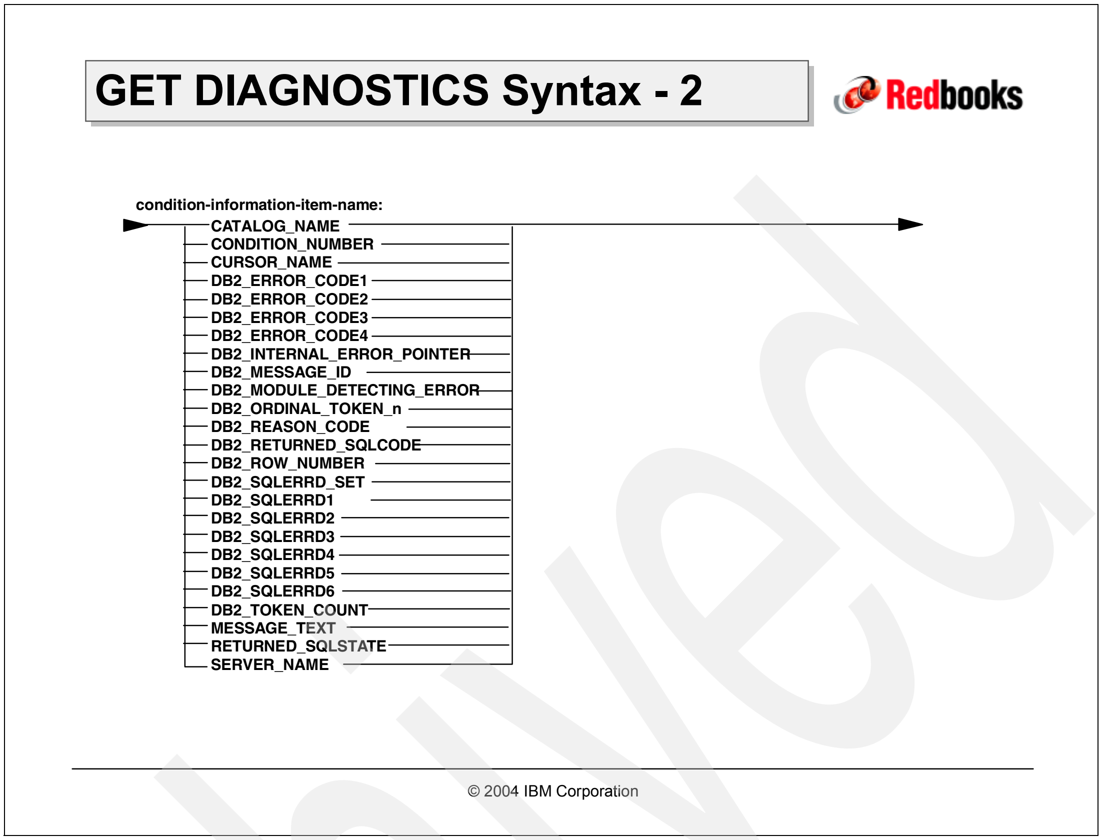
4.8.1 GET DIAGNOSTICS syntax

Diagnostic information is provided in three main areas: the statement-information area, the condition-information area, and the combined-information area. After the execution of an SQL statement, information about the execution of the statement is provided in the statement-information area, and at least one instance of the condition-information area. The number of instances of the condition-information area is indicated by the NUMBER item that is available in the statement-information area. The combined-information area contains a text representation of all the information gathered about the execution of the SQL statement.

For example, the statement-information-item name ROW_COUNT has the following information:

- It identifies the number of rows associated with the previous SQL statement that was executed.
- If the previous SQL statement is a DELETE, INSERT, or UPDATE statement, ROW_COUNT identifies the number of rows deleted, inserted, or updated by that statement, excluding rows affected by either triggers or referential integrity constraints.
- If the previous SQL statement is a multiple-row FETCH, ROW_COUNT identifies the number of rows fetched.
- A value of -1 indicates a mass delete from a table in a segmented table space.
- Otherwise, or if the server only returns an SQLCA, the value zero is returned.

Please refer to the *DB2 SQL Reference,* SC18-7426 for more details on all the items.

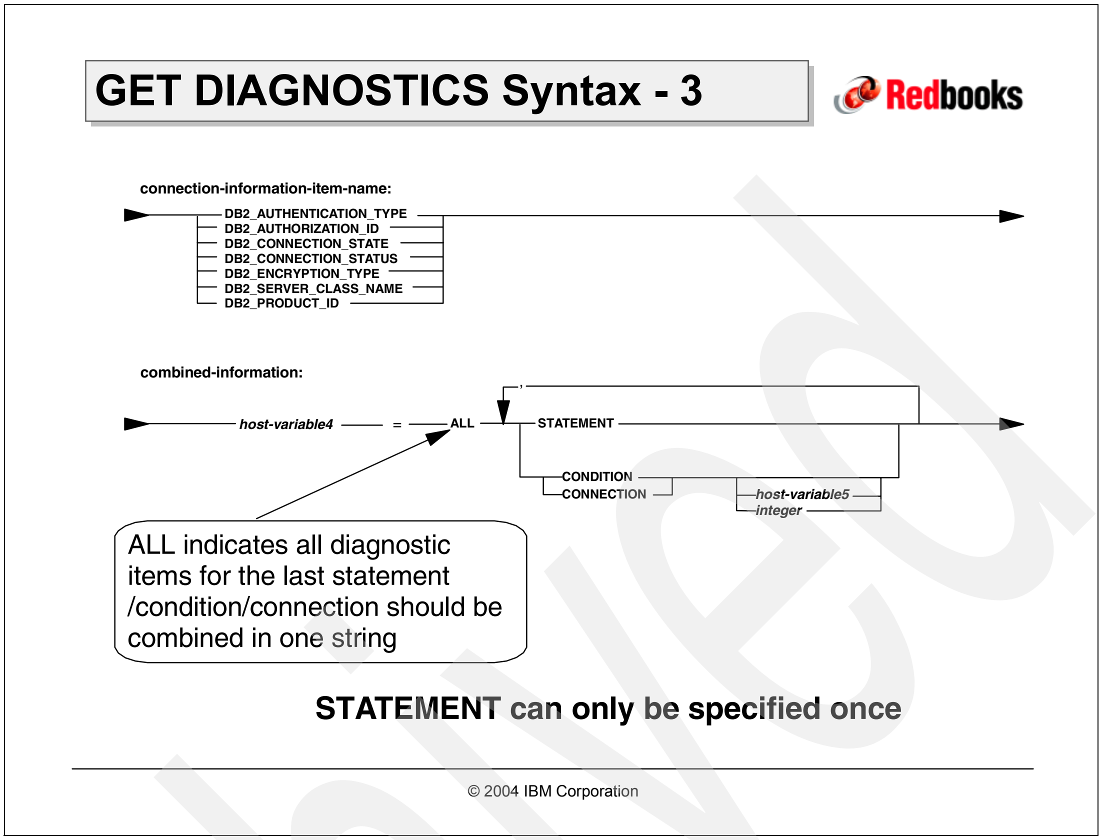


4.8.2 GET DIAGNOSTICS syntax - condition-information-item-name block

The item names in the condition-information-item-name block are shown on the visual.

For example, RETURNED_SQLSTATE contains the SQLSTATE for the specified diagnostic.

Please refer to the *DB2 SQL Reference,* SC18-7426 for more details on all the items.



4.8.3 GET DIAGNOSTICS syntax - connection-information-item-name block

The item names in the connection-information-item-name block are shown on the visual.

For example, DB2_SERVER_CLASS_NAME contains QDB2 for DB2 UDB for z/OS.

Please refer to the *DB2 SQL Reference,* SC18-7426 for more details on all these items.

GET DIAGNOSTICS Examples

Redbooks

To determine how many rows were updated in an UPDATE statement

- GET DIAGNOSTICS :rcount = ROW_COUNT;

To handle multiple SQL errors during a NOT ATOMIC multi-row insert

- GET DIAGNOSTICS :numerrors = NUMBER;
- Then code a loop to execute the following for the number of errors
  - GET DIAGNOSTICS CONDITION :i :retstate = RETURNED_SQLSTATE

To see all diagnostic information for an SQL statement

- GET DIAGNOSTICS :diags = ALL STATEMENT
- Sample output in :diags
  - Number=1; Returned_SQLSTATE=02000; DB2_RETURNED_SQLCODE=+100;
  - Would continue for all applicable items and for all conditions
  - Items are delimited by semicolons

4.8.4 GET DIAGNOSTICS examples

As an example of the use of the GET DIAGNOSTICS statement, we discuss the diagnostic information for multi-row FETCH and multi-row INSERT.

Diagnostic information for multi-row FETCH

The SQLCA is used to return information on errors and warnings found while fetching from a rowset cursor. After each FETCH statement from a rowset cursor, information is returned to the program through the SQLCA as follows:

- SQLCODE contains the SQLCODE.
- SQLSTATE contains the SQLSTATE.
- SQLERRD3 contains the actual number of rows returned. If SQLERRD3 is less than the number of rows requested, then an error or end-of-data condition occurred.
- SQLWARN flags are set to represent all the warnings that were accumulated while processing the FETCH statement.

Additional information may be obtained about the fetch, including information on all exception conditions encountered while processing the fetch statement, from the GET DIAGNOSTICS statement.

Consider the following examples, where we attempt to retrieve 10 rows with a single FETCH statement.

Assume that an error, SQLCODE -802, is detected on the row 5. SQLERRD3 is set to 4 for the four returned rows, SQLSTATE is set to 22003, SQLCODE is set to -802. This information is also available from the GET DIAGNOSTICS statement, for example:

```
GET DIAGNOSTICS :num_rows = ROW_COUNT, :num_cond = NUMBER;
```

This would result in num_rows = 4 and num_cond = 1 (1 condition).

```
GET DIAGNOSTICS CONDITION 1 :sqlstate = RETURNED_SQLSTATE,
:sqlcode = DB2_RETURNED_SQLCODE, :row_num = DB2_ROW_NUMBER;
```

This would result in SQLSTATE = 22003, SQLCODE = -802, and ROW_NUM = 5.

There are some cases where DB2 returns a warning if indicator variables are provided, or an error if indicator variables are not provided. These errors can be thought of as data mapping errors that result in a warning (SQLCODE +802 for instance) if indicator variables are provided. The GET DIAGNOSTICS statement may be used to retrieve information about all the data mapping errors that have occurred as in the case of multi-row INSERT.

Diagnostic information for multi-row INSERT

When NOT ATOMIC CONTINUE ON SQLEXCEPTION is specified, the inserts are processed independently. This means that if one or more errors occur during the execution of an INSERT of a row, then processing continues. The row that was being inserted at the time of the error is not inserted. Execution continues with the next row to be inserted, and any other changes made during the execution of the multi-row INSERT statement are not backed out. When ATOMIC is in effect, if an insert value violates any constraints, or if any other error occurs during the execution of an INSERT of a row, then all changes made during the execution of the multi-row INSERT statement are backed out.

The SQLCA reflects the last warning encountered. The SQLCA is used to return information on errors and warnings found during a multi-row INSERT. If indicator arrays are provided, the indicator variable values are used to determine if the value from the host variable array, or NULL, is used. The SQLSTATE contains the warning from the last data mapping error.

Additionally, when NOT ATOMIC CONTINUE ON SQLEXCEPTION is in effect, then status information is available for each failure or warning that occurred while processing the insert. The status information for each row is available with the GET DIAGNOSTICS statement.

As an example, assume that you are inserting multiple rows using host variable arrays for column values. Table T1 has two columns, C1 is a SMALL INTEGER column, and C2 is an INTEGER column. INSERT 10 rows of data into table T1. The values to be inserted are provided in host variable arrays :hva1 (an array of INTEGERS and :hva2 an array of DECIMAL(15,0) values. The data values for :hva1 and :hva2 are represented in Table 4-2.

Table 4-2 Data values for :hva1 and :hva2

Array entry	:hva1	:hva2
1	1	32768
2	-12	90000
3	79	2
4	32768	19
5	8	36
6	5	24
7	400	36
8	73	4000000000
9	-200	2000000000

Array entry	:hva1	:hva2
10	35	88

```
EXEC SQL
INSERT INTO T1 (C1, C2) FOR 10 ROWS VALUES (:hva1:hvind1, :hva2:hvind2)
NOT ATOMIC CONTINUE ON SQLEXCEPTION;
```

After execution of the INSERT statement, we have the following in SQLCA:

```
SQLCODE = 0
SQLSTATE = 0
SQLERRD3 = 8
```

Although we attempted to insert 10 rows, only 8 rows of data were inserted. Further information can be found by using the GET DIAGNOSTICS statement, for example:

```
GET DIAGNOSTICS :num_rows = ROW_COUNT, :num_cond = NUMBER;
```

This would result in NUM_ROW = 8 and NUM_COND = 2 (2 conditions)

```
GET DIAGNOSTICS CONDITION 1 :sqlstate = RETURNED_SQLSTATE,
:sqlcode = DB2_RETURNED_SQLCODE, :row_num = DB2_ROW_NUMBER;
```

This would result in SQLSTATE = 22003, SQLCODE = -302, and ROW_NUM = 4

```
GET DIAGNOSTICS CONDITION 2 :sqlstate = RETURNED_SQLSTATE,
:sqlcode = DB2_RETURNED_SQLCODE, :row_num = DB2_ROW_NUMBER;
```

This would result in SQLSTATE = 22003, SQLCODE = -302, and ROW_NUM = 8

Nested Table Expressions - Review

```
SELECT E.EMPNO, E.LASTNAME, E.HIREDECADE, E.SALARY, M.MINIMUM_SALARY
       FROM
          (
           SELECT EMPNO, LASTNAME,SALARY,
                  SUBSTR(CHAR(HIREDATE,ISO),1,3) CONCAT '0 - 9'
                    AS HIREDECADE
                  FROM EMPLOYEE
          ) AS E

         INNER JOIN
          (
           SELECT S.HIREDECADE, MIN(S.SALARY) AS MINIMUM_SALARY
                  FROM
                    (
                      SELECT SUBSTR(CHAR(HIREDATE,ISO),1,3)
                               CONCAT '0 - 9' AS HIREDECADE,
                             SALARY
                             FROM EMPLOYEE
                    ) AS S
                  GROUP BY S.HIREDECADE
          ) AS M

          ON E.HIREDECADE = M.HIREDECADE
```


4.9 Nested table expressions in joins - review

This visual shows the SQL statement that uses nested table expressions in joins to determine, for each employee, their employee number, last name, hiring decade, salary, and the minimum salary being paid to employees of their hiring decade. The EMPLOYEE table is used for this purpose.

The nested table expression creates a temporary result table within the FROM clause of an outer query. The nested table expression cannot be referenced elsewhere in the query, although the result columns can be referenced.

The SQL statement is fairly complex because the table expressions are part of the FROM clause. Perhaps you are wondering whether the query can be made more comprehensible. Yes, there is an easier way: common table expressions introduced in DB2 V8 to be compatible with other members of the DB2 family.

Common Table Expressions

```
WITH
  E AS
    (
    SELECT EMPNO, LASTNAME, SALARY,
           SUBSTR(CHAR(HIREDATE,ISO),1,3) CONCAT '0 - 9'
             AS HIREDECADE
           FROM EMPLOYEE
    ),
   M (HIREDECADE, MINIMUM_SALARY) AS
    (
    SELECT HIREDECADE, MIN(SALARY)
           FROM E
           GROUP BY HIREDECADE
    )
SELECT E.EMPNO, E.LASTNAME, E.HIREDECADE,
       E.SALARY, M.MINIMUM_SALARY
       FROM E INNER JOIN M
         ON E.HIREDECADE = M.HIREDECADE
```


4.10 Common table expressions

The example on the visual reformulates the query that we used in the previous example, that is, the example for nested table expressions. It uses common table expressions (CTEs), which are named E and M as the nested table expressions were before.

They are introduced by the keyword WITH and occur at the beginning of the query. They are separated from each other by commas. Every reference to a specific common table expression within the same query uses the same result set. Common table expressions can be referenced elsewhere in the query, even by other common table expressions within the same query.

The first common table expression (this is the SQL statement we used in the previous visual for table expression E) determines employee number, last name, salary, and hiring decade for all employees of the EMPLOYEE table. The first common table expression is again called E. The columns of the associated result table are those named in the SELECT statement.

Although the second table expression looks different, it provides the same result as the SQL statement for nested table expression M of the previous query: For the various decades, it determines the minimum salary being paid to the employees hired during the appropriate decade.

These are the basic differences:

- The SELECT statement now uses common table expression E (defined before in front of common table expression M) instead of table EMPLOYEE.
- The columns of the common table expression have the names specified in parentheses following the name of the common table expression. This is the same technique as naming the columns of a view. No AS clause is required for the calculated columns in the SELECT statement.

The SELECT follows the common table expression. Since it can refer to the common table expressions, the SQL statement is more comprehensible compared to the use of nested table expressions.

Notice that the common table expression M is based on the common table expression E. This is by no means a requirement for using common table expressions, but is certainly possible and useful.

The example shows the solution to an application requirement that is somewhat common: listing aggregate information on the same output line as detailed information.This problem can also be handled with views as follows:

```
CREATE VIEW E (EMPNO, LASTNAME, SALARY, HIREDECADE)
AS SELECT
EMPNO, LASTNAME, SALARY, SUBSTR(CHAR(HIREDATE,ISO),1,3) CONCAT '0 -9'
FROM EMPLOYEE;
CREATE VIEW M (HIREDECADE, MINIMUM_SALARY)
AS SELECT
HIREDECADE, MIN(SALARY)
FROM E
GROUP BY HIREDECADE;
SELECT E.EMPNO, E.LASTNAME, E.DECADE, E.SALARY, M.MINIMUM_SALARY
FROM E INNER JOIN M
ON E.HIREDECADE = M HIREDECADE;
```

Remember that, if views are used, each view needs to be defined, and then access to it has to be granted. This can be a tedious procedure if the solution requires lots of views, and so common table expressions are useful.

Common table expressions are materialized if they are referenced more than once. If a CTE is only referenced once, the CTE is treated like a regular table expression and materialization is avoided whenever possible.

Common table expressions are required if you want to use recursive SQL introduced in DB2 V8 to be compatible with other members of the DB2 family.

Recursive SQL

```
WITH
  RPL (PART, SUBPART, QUANTITY) AS
    (
```

Initialization Select

```
      SELECT ROOT.PART, ROOT.SUBPART, ROOT.QUANTITY
             FROM PARTLIST ROOT
             WHERE ROOT.PART = '01'
```

UNION ALL

Iterative Select

```
      SELECT CHILD.PART, CHILD.SUBPART, CHILD.QUANTITY
             FROM RPL PARENT, PARTLIST CHILD
             WHERE PARENT.SUBPART = CHILD.PART
    )
```

Main Select

```
  SELECT PART, SUBPART, SUM(QUANTITY) AS QUANTITY
         FROM RPL
         GROUP BY PART, SUBPART
```


4.11 Recursive SQL

Recursive SQL is very useful to retrieve data from tables that contain component breakdowns where each component is broken down into subcomponents and each subcomponent is broken down again into sub-subcomponents, etc. Applications involving these kinds of tables are often called "Bill of Materials" applications. A table that represents the parts in a computer would be an example of Bill of Materials: the major components, the monitor, system unit, and printer, all contain subassemblies like the hard drive, the mother board, and the print head, each of which is composed of other subassemblies, etc.

Another example is given: A table of courses containing course codes, course names, and prerequisites, determines all the courses that are a prerequisite to a particular course. Yet another example is given: A table of airline connections containing an originating airport, a destination airport, and distance, determines all the places you can go to and how distant each destination is from the originating point.

Recursive SQL involves defining a common table expression that references itself. The common table expression consists of two distinct components, an initialization SELECT and an iterative SELECT. The initialization SELECT is the first SELECT in the table expression and the iterative SELECT is the second SELECT in the table expression. The iterative SELECT is combined with the initialization SELECT by means of UNION ALL.

The recursive common table expression in the example on the visual is named RPL. The definition is enclosed in parentheses.

The common table expression in a recursive SQL statement is followed by a main SELECT. The main SELECT identifies the columns which are obtained from the result set of the common table expression.

The example on the visual builds a final result set that identifies all the parts and subparts needed to build Part 01 (WHERE clause of initialization SELECT) in a parts table called PARTLIST. We will see the PARTLIST table when stepping through the various "phases" in the subsequent visuals.

Recursive SQL- Initialization SELECT

```
SELECT ROOT.PART, ROOT.SUBPART, ROOT.QUANTITY
       FROM PARTLIST ROOT
       WHERE ROOT.PART = '01'
```

PART	SUBPART	QUANTITY
00	01	5
00	05	3
01	02	2
01	03	3
01	04	4
01	06	3
02	05	7
02	06	6
03	07	6
04	08	10
04	09	11
05	10	10
05	11	10
06	12	10
06	13	10
07	12	8
07	14	8

PARTLIST Table

PART	SUBPART	QUANTITY
01	02	2
01	03	3
01	04	4
01	06	3

RPL

4.11.1 Recursive SQL - initialization SELECT

This visual shows the PARTLIST table and illustrates what happens as the consequence of the initialization SELECT.

The initialization SELECT is executed only once. In the example, it reads the PARTLIST table.

The WHERE clause of the initialization SELECT controls the starting point of the recursion. In the example, the starting point is all rows with a part number of '01'.

The right-hand side of the visual displays the four rows placed in the temporary table RPL as the consequence of the initialization SELECT. Parts 02, 03, 04, and 06 are the assemblies that directly make up Part 01. The first column (PART) of the interim result identifies the major part. The second column (SUBPART) identifies the subparts that make up the major part. The third column (QUANTITY) identifies the quantity of the subpart needed to construct one complete major part. For example, it takes three units of Part 06 to construct Part 01.

Recursive SQL - First Iteration

Redbooks

```
SELECT CHILD.PART, CHILD.SUBPART, CHILD.QUANTITY
       FROM RPL PARENT, PARTLIST CHILD
       WHERE PARENT.SUBPART = CHILD.PART
```

PART	SUBPART	QUANTITY
00	01	5
00	05	3
01	02	2
01	03	3
01	04	4
01	06	3
02	05	7
02	06	6
03	07	6
04	08	10
04	09	11
05	10	10
05	11	10
06	12	10
06	13	10
07	12	8
07	14	8

PARTLIST Table

PART	SUBPART	QUANTITY
01	02	2
01	03	3
01	04	4
01	06	3
02	05	7
02	06	6
03	07	6
04	08	10
04	09	11
06	12	10
06	13	10

RPL

4.11.2 Recursive SQL - first iteration

Unless it is limited by control variables (See “Controlling depth of recursion - example” on page 295), the iterative SELECT is executed until all subparts of all parts have been broken down into their subparts, no matter how many repetitions are required. In our example, there are no control variables so the iteration continues until all parts are completely resolved.

Note that It is very easy to write a recursive SQL statement incorrectly and initiate an infinite loop. Control variables are very useful for limiting the number of iterations and are discussed in a later visual.

The iterative SELECT in the example is the part of the recursive SQL statement between the UNION ALL and the parenthesis that closed the common table expression named RPL. Only the iterative SELECT is repeated on this visual.

During the first iteration, each row from the initialization select is joined to all rows in the PARTLIST table that meet the join criteria. The result rows are added to the temporary table RPL. The rows that are added to RPL indicate that Parts 05 through 09 and 12 through 13 make up the parts returned by the initialization select:

- Part 02 consists of Parts 05 and 06
- Part 03 consists of Part 07
- Part 04 consists of Parts 08 and 09
- Part 06 consists of Parts 12 and 13

The white rows in RPL are rows resulting from the initialization SELECT. The subparts from which they are assembled are shaded in the PARTLIST table. The first iteration adds these subparts to the RPL temporary table. Appropriate rows in the RPL table are shaded as well.

The rows added from the first iteration are used to drive the second iteration.

Recursive SQL - Second Iteration

```
SELECT CHILD.PART, CHILD.SUBPART, CHILD.QUANTITY
       FROM RPL PARENT, PARTLIST CHILD
       WHERE PARENT.SUBPART = CHILD.PART
```

PARTLIST Table

PART	SUBPART	QUANTITY
00	01	5
00	05	3
01	02	2
01	03	3
01	04	4
01	06	3
02	05	7
02	06	6
03	07	6
04	08	10
04	09	11
05	10	10
05	11	10
06	12	10
06	13	10
07	12	8
07	14	8

No correspondence in PARTLIST table

RPL

PART	SUBPART	QUANTITY
01	02	2
01	03	3
01	04	4
01	06	3
02	05	7
02	06	6
03	07	6
04	08	10
04	09	11
06	12	10
06	13	10
05	10	10
05	11	10
06	12	10
06	13	10
07	12	8
07	14	8

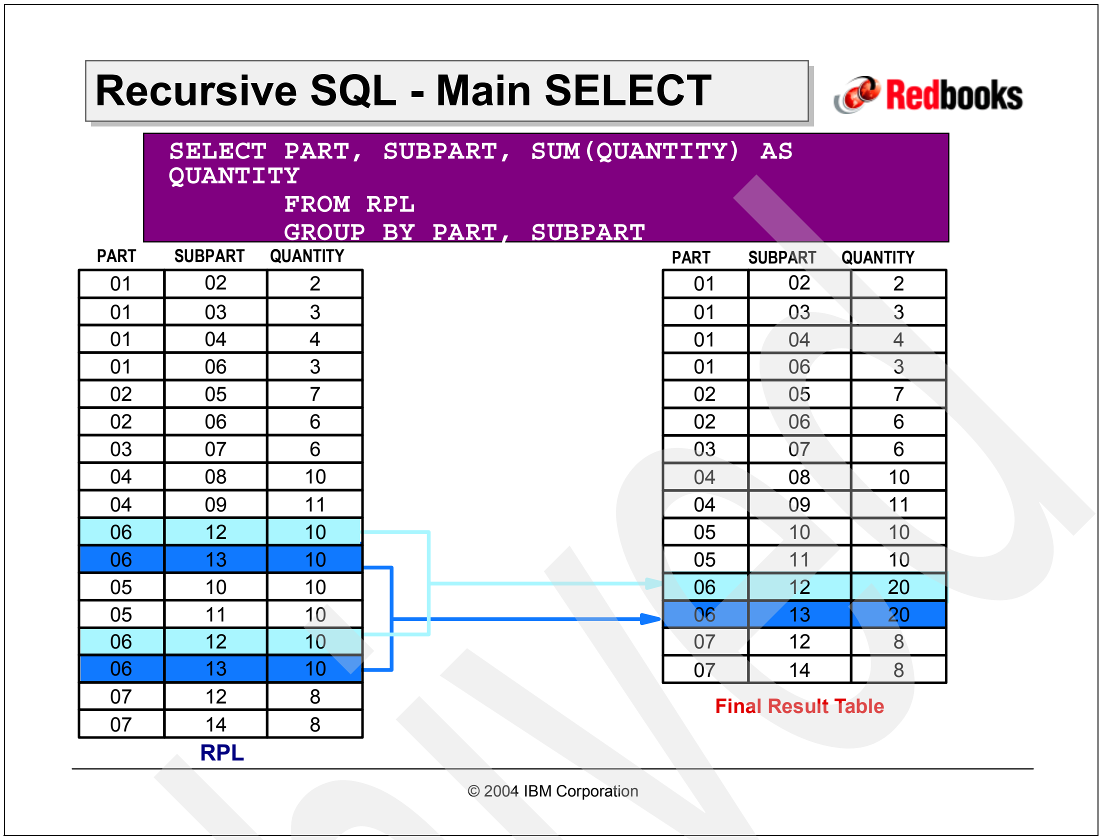
4.11.3 Recursive SQL - second iteration

The second iteration joins the rows added by the first iteration to the PARTLIST table. The result rows of the second iteration are again added to RPL. The second iteration indicates that Part 05 consists of Parts 10 and 11, Part 06 consists of Parts 12 and 13, and Part 07 consists of Parts 12 and 14.

Since there are no correspondences for the subparts of Parts 04 and 06, added by the first iteration, in the PARTLIST table, parts are not added for them to the RPL temporary table.

Note that RPL now contains two occurrences each of the rows that define the subparts of Part 06, namely, Parts 12 and 13. The first occurrence of these rows was contributed by the first iteration and the second occurrence of these rows came from the second iteration. The UNION ALL preceding the iterative select prevents the removal of duplicate rows.

In this example, the recursion does not yield additional rows after the second iteration because there are no further subparts for the parts added by the second iteration. However, if the PARTLIST table contained additional levels of subparts, the recursion would continue, since the current example does not limit the depth of the recursion.



PART	SUBPART	QUANTITY
01	02	2
01	03	3
01	04	4
01	06	3
02	05	7
02	06	6
03	07	6
04	08	10
04	09	11
06	12	10
06	13	10
05	10	10
05	11	10
06	12	10
06	13	10
07	12	8
07	14	8

RPL

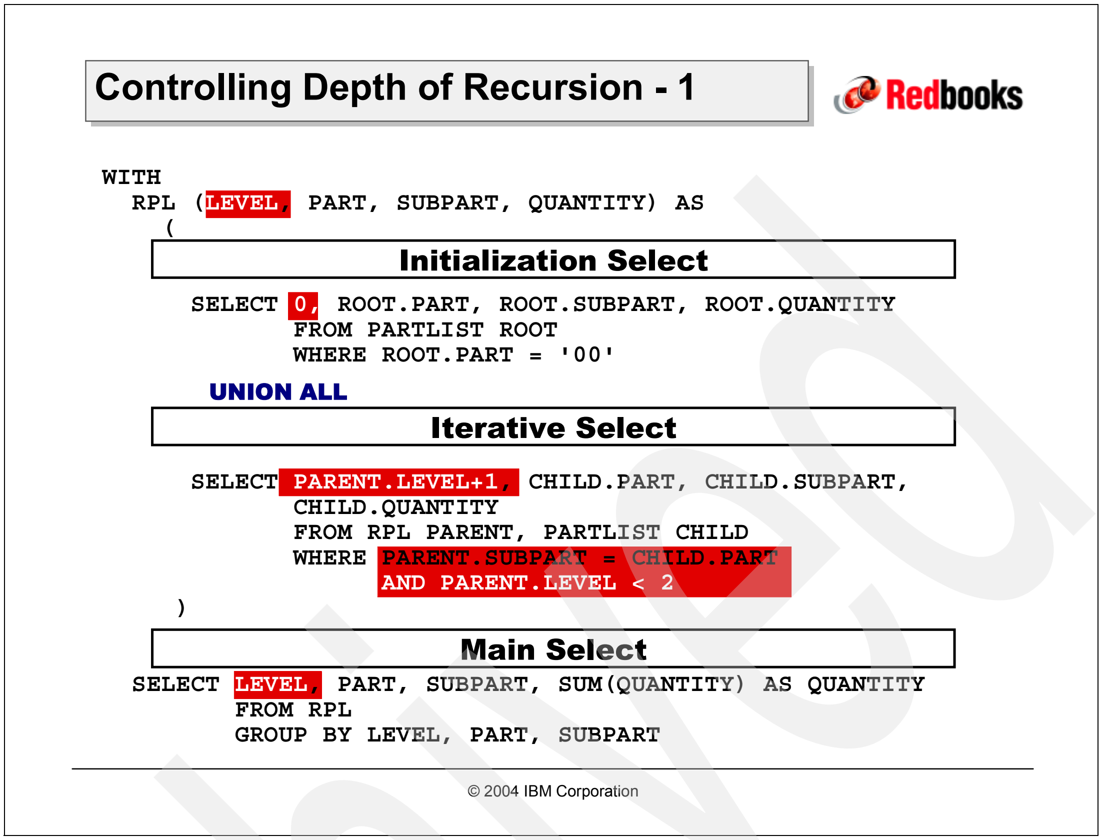
PART	SUBPART	QUANTITY
01	02	2
01	03	3
01	04	4
01	06	3
02	05	7
02	06	6
03	07	6
04	08	10
04	09	11
05	10	10
05	11	10
06	12	20
06	13	20
07	12	8
07	14	8

Final Result Table

4.11.4 Recursive SQL - main SELECT

After the recursive common table expression has been evaluated completely, the main SELECT is evaluated. The main SELECT references the result of RPL, the common table expression.

The main SELECT summarizes the total quantity of all parts needed to build Part 01. The grouping and the SUM() function ensure that the quantities of the respective subparts of Part 06 are added together. In other words, the two rows for Part 06, Subpart 12, are combined to make a single row. So are the two rows for Part 06, Subpart 13. A user who wishes to verify that the warehouse contains enough of each of the components required to make Part 01 can execute this query, and then check existing stocks against the result of the query.



4.11.5 Controlling depth of recursion - example

The nature of recursive SQL makes it prone to infinite cycles, but you can control the number of recursions as illustrated in this visual.

Recursion normally continues until all parts have been resolved into their components. However, the depth of the recursion can be controlled by simulating the addition of a control column to the table.

This control column is initialized arbitrarily to 0 in the initialization SELECT and increased by 1 on every execution of the iterative SELECT. A condition in the WHERE clause of the iterative SELECT is used to ensure that the iteration only continues for a fixed number of levels.

In the example, the control column is named LEVEL. It is set to an integer value of 0 in the initialization SELECT. The WHERE clause of the initialization SELECT determines the value in the PART column with which the table expression begins. This time, we are interested in the breakdown of Part 00, but we could have started with any part number we were interested in. The initial value of LEVEL would still be 0, regardless of the starting part number. The iterative select increments the LEVEL value by adding 1 on each iteration.

The following condition in the WHERE clause of the iterative SELECT is used to limit the number of iterations:

```
PARENT.LEVEL < 2
```

You simply set the constant to the number of iterations that are desired.

The main SELECT displays the result of the table expression. The LEVEL column in the final result makes the origin of each result row clear: rows that came from the initialization SELECT have a level of 0, rows from the first iteration have a level of 1, rows from the second iteration have a level of 2, and so on. The ORDER BY puts the result in a convenient sequence.

Note that LEVEL is not a column of table PARTLIST. It does not have to be added to table PARTLIST using an ALTER TABLE statement. It is a “virtual” column created by the SQL statement.

The actual result of the recursive SQL statement is illustrated on the next visual.

Controlling Depth of Recursion - 2

Redbooks

PART	SUBPART	QUANTITY
00	01	5
00	05	3
01	02	2
01	03	3
01	04	4
01	06	3
02	05	7
02	06	6
03	07	6
04	08	10
04	09	11
05	10	10
05	11	10
06	12	10
06	13	10
07	12	8
07	14	8

PARTLIST Table

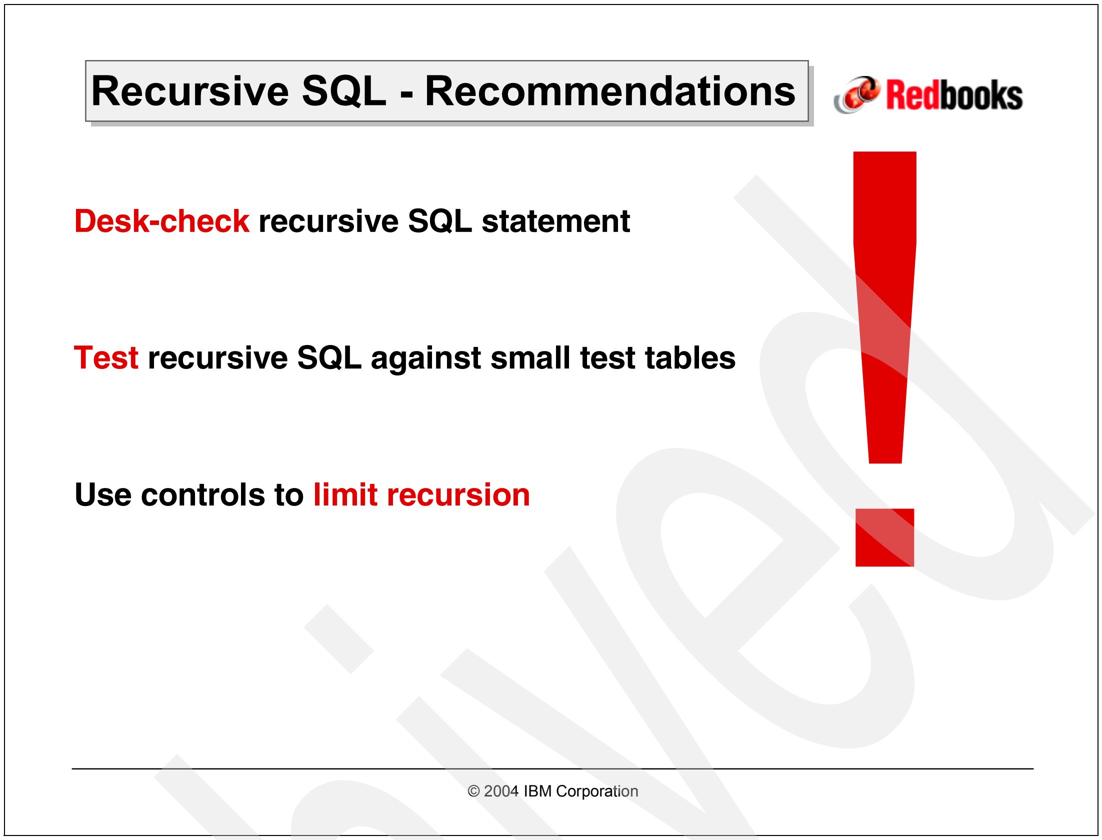
LEVEL	PART	SUBPART	QUANTITY
0	00	01	5
0	00	05	3
1	01	02	2
1	01	03	3
1	01	04	4
1	01	06	3
1	05	10	10
1	05	11	10
2	02	05	7
2	02	06	6
2	03	07	6
2	04	08	10
2	04	09	11
2	06	12	10
2	06	13	10

Final Result Table

4.11.6 Controlling depth of recursion - result

This visual displays the result of the breakdown for Part 00 if the recursion is limited to two iterations.

Subparts 05 and 06 of Part 02 and Subpart 07 of Part 03 could be further decomposed if the number of iterations is not limited to 2.



4.11.7 Recursive SQL - recommendations

Recursive SQL is cyclical by definition. This means that it is easy to cause infinite loops if the SQL is coded incorrectly or if the data itself is cyclical. For example, if the join in the iterative select in the earlier examples was coded as:

PARENT.SUBPART = CHILD.SUBPART

Infinite recursion would occur if there were even one row where the Part and Subpart values were the same. By the same token, a loop can occur if the data were illogical. For example, if the PARTLIST table had a row where the Part was 05 and the Subpart was 01, a loop would occur. To prevent this sort of problem, you should desk-check all recursive SQL. Also, test it against small tables before implementing it in production.

Any recursive SQL statement that does not use a control variable receives an SQL warning (SQLSTATE = 01605 and SQLCODE = +347 WARNING). Although this is not a serious problem, you can use techniques shown on the preceding visuals to avoid it.

Identity Column Enhancements

Redbooks

Benefits .....

- Better performance and concurrency than application generated counters
- Guaranteed uniqueness both within an individual subsystem and a data sharing group
- Recoverability in the event of a DB2 system failure
- Failure of one data sharing member will not impact the other members from generating key values
- Simple implementation

What's new? .....

- Identity Column introduced in Version 6
  - Automatically generated by DB2
  - Unique, sequential, recoverable values
  - Useful for generating unique primary key values
- in Version 8, ALTER TABLE statement is extended to include identity column specifications to allow modifying the attributes of an existing identity column

4.12 Identity column enhancements

In this section we discuss the V8 enhancements related to identity columns.

4.12.1 Identity columns

The column attribute AS IDENTITY was introduced in the DB2 for OS/390 Version 6 refresh through APAR PQ30652 and was delivered as part of DB2 Version 7. DB2 for z/OS Version 8 enhances identity columns by extending the ALTER COLUMN clause of the ALTER TABLE SQL statement to include the identity column specification. In addition, there is a close tie between the enhancements to identity columns and sequences discussed in the next section.

Prior to Version 8, you could not alter the characteristics of an identity or whether the GENERATED keyword used the ALWAYS or BY DEFAULT options. Not being able to alter the GENERATED option was one of the real challenges of using this option when it was initially introduced.

Since sequence objects build on the existing infrastructure of identity columns, we will discuss the identity enhancements first.

Identity Column Considerations

Before Version 8, if you had a requirement to unload and reload your tables, you were forced to specify GENERATED BY DEFAULT for the identity column

- If you had specified GENERATED ALWAYS at table design, the only option to unload/reload would be to:
  - Unload the table
  - DROP the table
  - Re-CREATE the table using GENERATED BY DEFAULT
  - Reload the table
- Otherwise, DB2 will generate new values for the rows during the reload, which is probably not what you want

With Version 8, if you specify GENERATED ALWAYS and later have a requirement to unload/reload your tables, you could:

- ALTER TABLE ALTER COLUMN SET GENERATED BY DEFAULT
- Unload the table
- Reload the table
- ALTER TABLE ALTER COLUMN SET GENERATED ALWAYS

4.12.2 Identity column considerations

Two keywords work in conjunction with identity columns: GENERATED BY DEFAULT and GENERATED ALWAYS. There is a significant difference on how these values affect the generation of values for a column defined with AS IDENTITY. GENERATED ALWAYS will always generate a value for its column. GENERATED BY DEFAULT will only generate a value if a value does not already exist.

The preferred way to use identity columns is GENERATED ALWAYS. However, before Version 8, if you had a requirement to unload and reload your tables, you were forced to specify GENERATED BY DEFAULT for the identity column. If you had specified GENERATED ALWAYS at table design, the only option to unloading and reloading the table would be to unload the table, DROP the table, and then re-CREATE the table using GENERATED BY DEFAULT. Only after all of those steps could you reload the data.

If this procedure was not followed and GENERATED ALWAYS was used, DB2 would generate new values for the rows during the reload. This is not what you would want to occur.

With Version 8, if you specify GENERATED ALWAYS and later have a requirement to unload and reload the table, you could simply use the following procedure:

- ALTER TABLE ALTER COLUMN SET GENERATED BY DEFAULT
- Unload the table
- Reload the table
- ALTER TABLE ALTER COLUMN SET GENERATED ALWAYS

Identity Column Enhancements

Redbooks

Dynamic alter of Identity column attributes

- ALTER TABLE ALTER COLUMN extended to:
  - Enable modification of identity column attributes, and
  - Specify continuation of the sequence associated with identity column from a new point in the range of values that is different from where the column values would otherwise have continued
- Only future values of column affected by change
- Cannot alter data type of identity column
- Unused cache values may be lost when column attributes are altered

New keyword support to aid porting from other vendor platforms

- NO MINVALUE
- NO MAXVALUE
- NO ORDER, ORDER

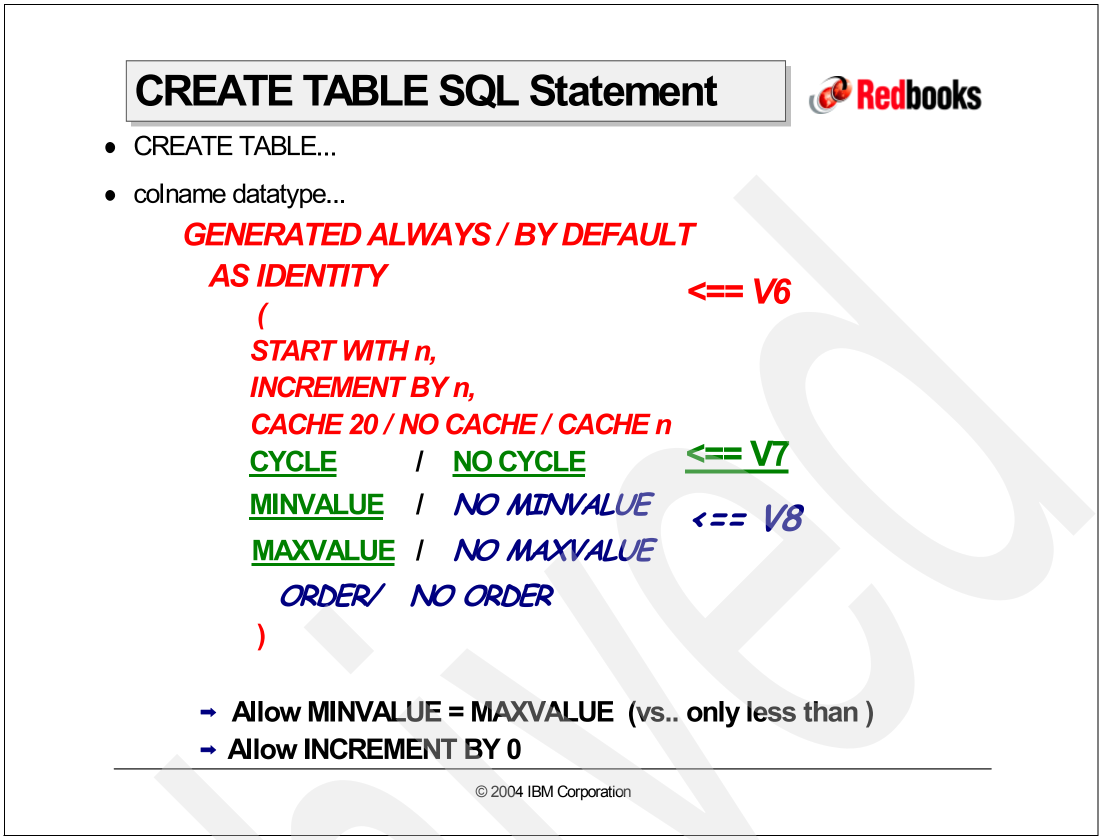
Allows:

- INCREMENT BY to be 0
- MINVALUE = MAXVALUE

4.12.3 Identity column enhancements

In addition to being able to add an identity to a column, the ALTER SQL statement can now be used to modify the attributes of an identity column. ALTER TABLE ALTER COLUMN has been extended to allow you to modify the identity column attributes and specify continuation of the sequence associated with identity column from a new point in the range of values that is different from where the column values would otherwise have continued. Making this change only affects future values of columns. Although the keywords can be changed, the data type of the identity cannot be altered. If caching is in use, values that have already been cached may be lost when the identity column is altered.

Version 8 adds new keywords to the syntax of the AS IDENTITY clause. New keyword support now exist for NO MINVALUE, NO MAXVALUE, NO ORDER and ORDER. The semantics of INCREMENT BY, MINVALUE, and MAXVALUE have also changed. INCREMENT BY allows 0, and MINVALUE and MAXVALUE can be equal. The keyword on the GENERATED clause is also now modifiable via the ALTER SQL statement. You can dynamically switch between GENERATED ALWAYS and GENERATED BY DEFAULT.



4.12.4 CREATE TABLE SQL statement

AS IDENTITY is an attribute of the GENERATED keyword on the CREATE TABLE SQL statement. With each new version of DB2, identity columns were enhanced.

Version 6

Initially, identity columns only allowed three option keywords. START WITH, if used, specified the first, or starting, value for the identity column. This value had a numeric data type and could be any positive or negative value with a zero scale. If not specified, this keyword defaulted to 1.

The second allowed, but optional, keyword was INCREMENT BY. INCREMENT BY defines the interval between subsequent sequentially generated values. This keyword allowed any non-zero positive or negative numeric value with a zero scale within the range of a large integer. If not specified, this keyword defaults to 1.

The final grouping of keywords all have to do with caching the generated value to help improve performance. You have two options when dealing with caching:

- The first option turns caching on by specifying CACHE followed by the number of values that should be cached. Any value greater than 2 within the range of an integer can be used, and if not specified, the default is 20.
- The other option is to turn caching completely off with NO CACHE. Caching sequence values in memory is a performance and tuning option that promotes faster access to the sequence values when the application can handle the behavior. NO CACHE turns off the caching mechanism. A data sharing environment is one example where NO CACHE might be specified to avoid the possibility of out of sequence values.

Version 7

So what happened in Version 7?   What did DB2 do to improve identity columns? Version 7 added CYCLE, MINVALUE and MAXVALUE to the definition of an identity column. CYCLE allows an identity to wrap around to a new beginning when the minimum or maximum value is reached. Cycling through values a second time can create duplicate values. NO CYCLE caused the identity value to stop being generated when the minimum or maximum was reached. NO CYCLE is the default.

MAXVALUE specified the maximum value that can be generated for this identity column. This value can be any negative or positive value that is greater than the MINVALUE. Having the MAXVALUE greater than the MINVALUE was a Version 7 requirement. If MAXVALUE is not specified, the default for an ascending sequence is the greatest value allowed by the data type. For a descending sequence, it is the START WITH value if specified or –1 if no START WITH was used.

MINVALUE specified the minimum value that could be generated for this identity column. This value can be any negative or positive value that is less than MAXVALUE. If not specified, the START WITH value, or 1 if a START WITH value was not specified, is the default for an ascending sequence.   For a descending sequence, the default would be the lowest value for the data type.

MINVALUE, MAXVALUE, and CYCLE have the same semantics as sequence objects and are explained again in more detail in the next section.

All keywords introduced in Version 6 were carried forward into Version 7 with no change in their definitions.

Version 8

Version 8 enhances identity columns by adding additional keywords and changing the semantics of some of the existing keywords.

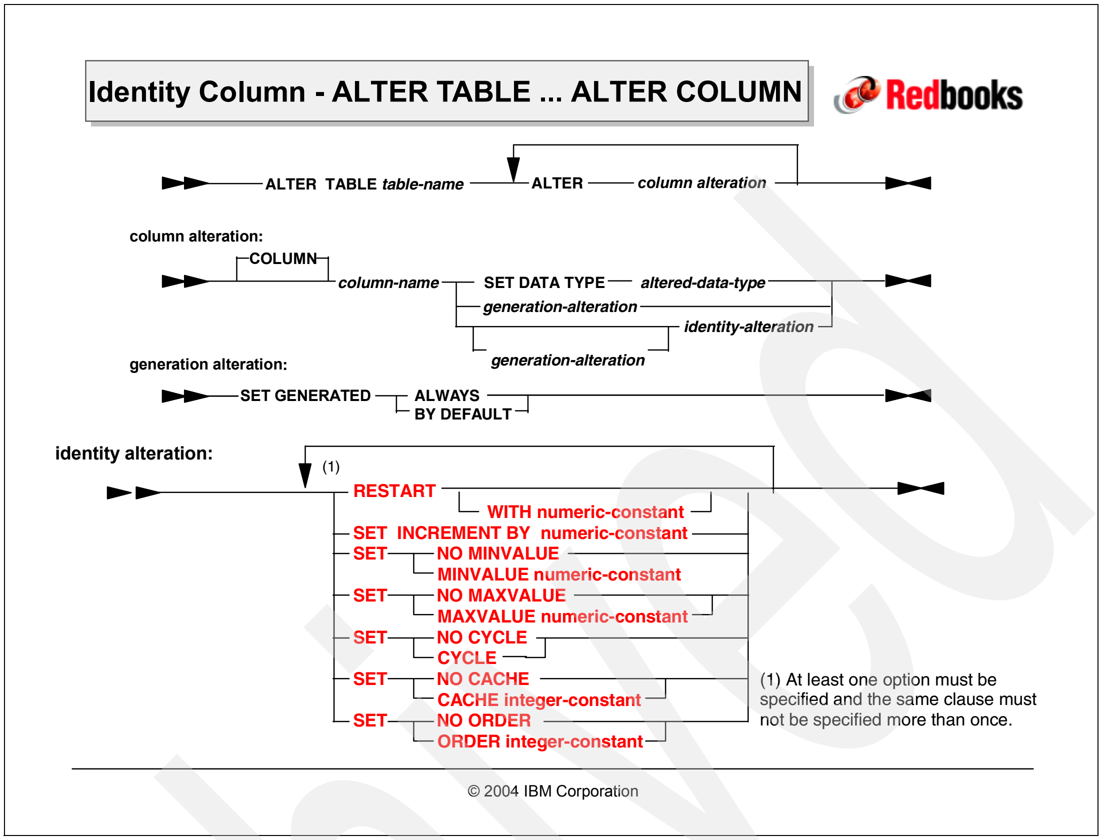
ORDER, NO ORDER, NO MINVALUE and NO MAXVALUE are the keywords added to identity columns in Version 8. NO MINVALUE and NO MAXVALUE also become the defaults if either MINVALUE or MAXVALUE is not specified.

NO MINVALUE specifies that no minimum end point of the range has been set. The minimum value for an ascending sequence becomes the START WITH value, or 1 if a START WITH value was not specified. For a descending sequence, the default would be the lowest value, the largest negative value, for the data type of the column the identity is assigned.

NO MAXVALUE specifies that no maximum end point of the range has been set. The maximum value for an ascending sequence is the greatest value allowed by the data type for that column. For a descending sequence, it is the START WITH value if specified or –1 if no START WITH was specified.

The semantics of MINVALUE and MAXVALUE have also changed. In Version 8, any negative or positive value including zero can be specified. Version 8 also allows INCREMENT BY to be set to zero. In addition, MINVALUE can be equal to MAXVALUE.

ORDER and NO ORDER specifies whether or not the identity values must be generated in the order of the request. NO ORDER is the default. NO ORDER specifies that the values do not need to be generated in order of request, while ORDER specifies that the values are generated in order of request. In data sharing environments where sequence values are cached by multiple DB2 members simultaneously and the CACHE option is used, the value assignments may not be in strict numeric order unless you also specify the ORDER option.



4.12.5 ALTER TABLE … ALTER COLUMN

In DB2 Versions 6 and 7, the only ALTER operation that could be performed that involved an identity column was to add it to a column definition. Once added, or if the table had been created with an identity column, none of the characters of that identity could be changed. This all changes with the arrival of DB2 Version 8.

The ALTER COLUMN portion of the ALTER TABLE SQL statement has been extended in Version 8 to include the ability to modify the characteristics of the GENERATED and AS IDENTITY clauses.

The GENERATED value can be changed by coding SET GENERATED ALWAYS or SET GENERATED WITH DEFAULT as the column-alteration value of the ALTER COLUMN clause. Because GENERATED has been a part of DB2 for some time now, there will be no further discussion here.

The next column-alteration value is RESTART WITH. Altering this value changes the starting point of the next value generated for an identity column. This keyword can be set to any negative or positive value including zero, that could be assigned to column's data type. If RESTART is specified without WITH, the sequence is restarted with the START WITH value the identity column was originally created with. An example of using RESTART WITH is to correct a gap in sequence number possibly cause by the loss of cached sequence values.

SET INCREMENT BY, SET MINVALUE or NO MINVALUE, SET MAXVALUE or NO MAXVALUE, SET CYCLE or SET NO CYCLE, SET CACHE or SET NO CACHE and SET ORDER or SET NO ORDER are additional values that can be specified on the ALTER COLUMN clause. The values to be SET are discussed in detail later in the section of the book describing sequence objects.

Altering an identity column only affects future values of that column and there is limited validation performed by DB2 when an identity column is altered. Validation is performed for the value specified for RESTART WITH to ensure that it conforms to the same rules as for START WITH at the time the column's original definition was created. Keywords and the values specified for those keywords are also validated.

Any values of an identity column that are not specified on the ALTER are left unchanged. No validation is performed to verify the affects of altering an identity column on existing values in that column. For example, if an ascending sequence is changed to descending, no messages are generated to warn you of the possibility that duplicate values could be created.

Generating Sequence Numbers - Before

Common application-level implementation is to maintain a one row table that contains the sequence number

- Each transaction locks this table, increments the number, commits (that is, one transaction at a time can increment the sequence number)
- OR a variation of this theme is to use SELECT MAX( ) + 1...WITH RR followed by INSERT using retrieved key
- Problem:
  - Page containing counter constitutes hot spot in data base resulting in unpredictable transaction delays caused by inter-system P-lock negotiation for that page and by buffer invalidation and refresh.
  - Contention inhibits transaction throughput and application processing power
  - If one member fails, retained locks held by failed member can prevent access to shared counter from surviving members

Identity columns provide a partial solution

4.13 Sequence objects

Now that we have discussed identity columns, it is time to look at a new feature in DB2 V8 that is closely related to identity columns, namely, sequences.

4.13.1 Generating sequence numbers - before

During the design of a new table, it is decided that the primary key should have some kind of business intelligence, some kind of meaning. The choice for the key is an ascending sequential value. At one time this was the easiest part of the process. The difficult task came when the application actually needed to “generate” that next sequential value for the INSERT.

It was common practice to SELECT MAX(seq_key) and add one to the value for use in your INSERT. Of course, the primary key needed to be unique. Somehow, the transaction had to ensure that no other transaction needing the same key, performed any processing against the table until the current transaction had inserted its row. ISOLATION (RR) could be used to prevent other transactions from incrementing the key value until the current transaction could commit.

Another popular method was a single row table that contained the next higher key value. The application would retrieve the key and update the table with the next key to be used. This caused serialization within a commit scope. This led to locking conflicts if there was a high INSERT rate.

The one-row table approach also has shortcomings in a data sharing environment. The page containing the counter can easily become a hot spot in the data base, resulting in unpredictable transaction delays caused by buffer invalidation and refresh. This contention inhibits transaction throughput and the application's processing power. In addition, if one DB2 member fails, retained locks that are held by the failed member can prevent access to the shared counter from the surviving members. As you will see, DB2 has a solution forthcoming.

DB2 Version 6 introduced the column attribute AS IDENTITY. This turned out to be only a partial solution. Identity columns are tied to a specific table and cannot be used independently from that table. In addition, a table can have one and only one column that specifies the identity attribute. Manageability of identity columns becomes an issue because attributes of an identity cannot be altered in Version 7. So, Version 8 introduced improvements to identity columns that were discussed in the 4.12, “Identity column enhancements” on page 299. However, for now, let us focus our attention on something completely new to DB2 of z/OS.

Sequence Object Solution

Redbooks

Avoid the concurrency and performance problems when applications generate their own sequence numbers (hotspots)

DB2 sequences allow multiple transactions to concurrently increment sequence number and guarantee each number will be unique

Sequence can be accessed and incremented by many users without waiting

- DB2 does not wait for a transaction that has incremented a sequence to commit before allowing the sequence to be incremented again by another transaction

Failure of one DB2 member in group will never prevent access (R/W) to sequence from surviving members

- No retained locks to prevent access to sequence

4.13.2 Sequence object solution

What is the significance of sequences and why are they preferred over the identity attribute? There are a couple of reasons. First, compatibility with other major database management systems, including the distributed DB2s, is needed. Sequences became available in DB2 for UNIX, Windows, and OS/2® in Version 7.2, and the support for sequences in DB2 for z/OS V8 is part of an effort to make SQL transparent across all DB2s on all platforms.

Second, sequences are completely stand-alone objects and have no connection to a table. Because they stand by themselves, they can be used by multiple applications in different ways. A sequence is a stored object that simply generates the next ascending or descending value when requested by an application. Sequences provide an excellent way for an application to obtain unique values for use in key structures.

A sequence is defined using SQL DDL statements and the attributes of the sequence can be explicitly defined by the user, can use defaults, or a combination of both. The values generated by a sequence can be SMALLINT, INTEGER or DECIMAL with a zero scale. The user has complete control over the starting value of the sequence using the START WITH keyword and can change the starting value via the ALTER SQL statement by specifying RESTART WITH keyword.

How the sequence is incremented is also under the user control through the use of the INCREMENT BY keyword. A minimum (MINVALUE) and maximum (MAXVALUE) value can be specified creating an upper and lower boundary for the sequence range. When either of the range boundaries is reached, the user can choose to cycle (keyword CYCLE) through the sequence again or terminate (NO CYCLE) the sequence generation with an error.

Sequences also provide some performance relief when an application must deal with generating sequential values for use in key structures. A caching capability (CACHE keyword) is available that makes the next sequence value more readily available. In addition, DB2 does not need to wait for an application to commit after incrementing a sequence to allow another different transaction from using that same sequence. Concurrency, as mentioned earlier, has always been an issue when attempting to generate sequential values for this reason, and sequences can resolve this problem. This also carries forward into a data sharing environment. If one data sharing member fails, there are no locks retained on the sequence preventing the surviving DB2 from using it.

Finally, the user has the ability to change (ALTER) any attribute, other than the data type, of a sequence at any time.

Comparing Identity Columns and Sequences

Sequences	Identity columns
Stand-alone object	Tied to a table
Can use one sequence for many tables or many sequences in one table	One to one relationship between identity and tables
Retrieved via NEXT VALUE FOR / PREVIOUS VALUE FOR expressions	Retrieved via IDENTITY_VAL_LOCAL function - within agents scope only
Can be altered via ALTER SEQUENCE	Can be altered via ALTER TABLE (ALTER COLUMN) Prior to V8 could not be altered

4.13.3 Comparing identity attribute to sequences

DB2 Version 8 introduces sequence objects. Sequence objects build on the concepts introduced with the identity attribute in previous versions of DB2 with some significant enhancements. Sequences are stand-alone objects that generate sequence values when requested by an application. These values can be used by that application for whatever purpose the application chooses. However, identity columns are associated with a specific column in a specific table and can only be used to supply a value for that column.

What is similar and what is different?

For starters, as stated previously, a sequence has nothing to do with a specific table. There is not a one-to-one relationship between a sequence object and any table. They simply give the application the ability to get the next unique sequential value. The application can then do with that value anything it chooses. In fact, a single sequence value could be used multiple times in the same SQL statement to supply a value for multiple columns, or multiple sequence objects could be used. Sequences use the SQL expressions NEXT VALUE FOR and PREVIOUS VALUE FOR to retrieve the next generated or previously generated values from the sequence.

These expressions are not allowed against identity columns. To retrieve a value supplied by an identity, that column should be SELECTed or retrieved with the IDENTITY_VAL_LOCAL function. The sequence object used can also be displayed as it is used in an INSERT SQL statement using a SELECT... FROM FINAL TABLE(INSERT …).

Unlike a sequence, which uses the ALTER DDL statement to modify the sequence attributes, an identity column's characteristics can only be altered using the ALTER TABLE and ALTER COLUMN statements. The ability to alter an identity column's attributes is an option introduced in DB2 Version 8. The ALTER SEQUENCE DDL statement cannot be used with an identity column.

This section discusses the details behind creating and using sequence objects along with their advantages over other techniques.

SQL to Support Sequence Objects

Redbooks

CREATE SEQUENCE

- Creates a sequence at the application server
- Can be embedded in application program or used interactively

ALTER SEQUENCE

- Can be used to change INCREMENT BY, MIN VALUE, MAXVALUE, CACHE, CYCLE and to RESTART WITH different sequence
- Only future values affected and only after COMMIT of ALTER
- Cannot alter data type of sequence
- Unused cache values may be lost

DROP SEQUENCE

COMMENT ON SEQUENCE

GRANT/REVOKE ON SEQUENCE

NEXT VALUE FOR and PREVIOUS VALUE FOR

4.13.4 SQL to support sequence objects

DB2 Version 8 introduces five SQL statements and two expressions in support of sequence objects.

CREATE

The CREATE SEQUENCE SQL statement creates a sequence object at the application server. The sequence object is a user-defined object that generates sequential numeric values. The CREATE is also used to describe the sequence value's specifications.

ALTER

The ALTER SEQUENCE SQL statement changes the attributes of the sequence object such as INCREMENT BY, MIN VALUE, MAX VALUE, CACHE, CYCLE and the point the sequence should be restarted at. Only future values are affected and then only after the ALTER has been committed. Whenever a sequence is ALTERed, there is always the risk of creating duplicate values, so care should be taken. If uniqueness is critical to the application, an unique index can be defined for the column using the sequence value.

DROP SEQUENCE

The DROP SEQUENCE statement drops the sequence object and removes a sequence object description from the DB2 catalog.

COMMENT ON SEQUENCE

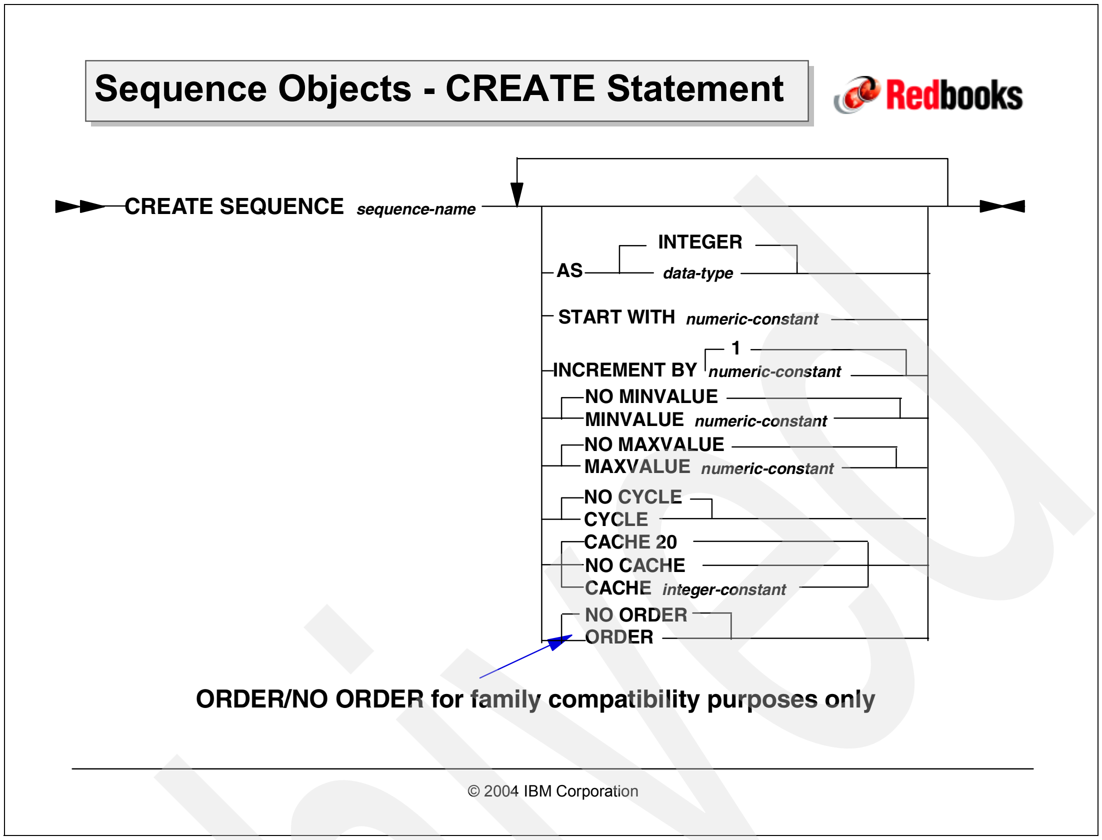
This statement allows a user to supply a comment or description for a sequence in the REMARKS column of the DB2 catalog table SYSIBM.SYSSEQUENCES.

GRANT/REVOKE

This statement is used to grant or revoke the ALTER or USAGE privilege for a user defined sequence or list of user defined sequences from an authorization identifier (auth id, or list of auth ids, or from PUBLIC.

NEXT VALUE FOR and PREVIOUS VALUE FOR

The NEXT and PREVIOUS VALUE FOR statements are used to reference the sequence values by specifying the named sequence.



4.13.5 CREATE SEQUENCE statement

The CREATE SEQUENCE SQL statement creates a sequence object at the application server. The sequence object is a user-defined object that generates sequential numeric values. The CREATE statement also describes the sequence value's specifications. DB2 records the characteristics of a sequence in the catalog tables SYSIBM.SYSSEQUENCES, SYSIBM.SYSSEQUENCEDEP, and SYSIBM.SYSSEQUENCEAUTH.

The catalog table SYSIBM.SYSSEQUENCEAUTH is new in Version 8. The catalog tables SYSIBM.SYSSEQUENCES and SYSIBM.SYSSEQUENCEDEP were initially introduced in Version 6 to support identity columns and extended in Version 8 in support of sequences. The catalog tables that support sequences are discussed in more detail in 4.13.25, "Catalog and directory changes" on page 341.

This statement can be specified in an application program or can be prepared dynamically. It can be prepared dynamically only if DYNAMICRULES run behavior is implicitly or explicitly specified. The following keywords can be specified on the CREATE SEQUENCE statement.

sequence_name

The *sequence_name* identifies the sequence and is a required value. A schema name can implicitly or explicitly qualify the sequence name. The sequence name can be up to 128 characters. If explicitly qualified, the schema can also be up to 128 characters and must be separated by the name portion with a period. The sequence name, when qualified with a schema name, must be unique at the current server. It is the sequence creator's responsibility to ensure that the name chosen for the sequence does not conflict with any sequence names generated for use with identity columns. The keyword *sequence_name* is the only required value. All other keywords are optional. SQL PATH has no affect on a sequence.

AS data type

The AS keyword specifies the data type of the sequence value. The data type must be one of the three DB2 numeric data types: SMALLINT, INTEGER, or DECIMAL with a zero scale. It can also be a user defined distinct type sourced on one of the above three numeric data types. The sourced type must be the exact same numeric type and must have a zero scale.

If SMALLINT is specified, the sequences have a range from –32768 through +32767. If INTEGER is chosen, the range is –2147483648 to +2147483647 and decimal allows 31-digit precision, both negative and positive values.

If the AS keyword is not specified, the default data type used for this sequence is an INTEGER.

START WITH

START WITH is an optional keyword that specifies the sequence's starting, or first, value. Because the START WITH keyword does not have to be explicitly specified, the value specified for MINVALUE or MAXVALUE could be used. For an ascending sequence, the value from the MINVALUE keyword, or the MINVALUE default if MINVALUE is not specified, would be used. If the sequence is descending, the value used for the MAXVALUE keyword, or the MAXVALUE default if MAXVALUE is not specified, is used.

The following rules govern the values that can be specified for the START WITH keyword:

- START WITH can specify any negative or positive value, including zero, that is valid for the data type stated in the AS *data_type* keyword
- START WITH cannot contain any non-zero values to the right of the decimal point.

There is no requirement for a sequence to be started within the range defined by MINVALUE and MAXVALUE when using the START WITH keyword. Any value within the data type range is allowed regardless of the MINVALUE or MAXVALUE values. The first value of a sequence always starts with the START VALUE value. However, once the sequence reaches the end of the logical range of values established by MINVALUE, MAXVALUE, or their defaults, and the CYCLE option is in effect, the sequence wraps around to the first value of the other end of the range. The first value after being cycled may or may not be the same as the original START WITH value if the START WITH value is different than the established MINVALUE or MAXVALUE or their defaults.

INCREMENT BY

INCREMENT BY determines the next value in the sequence. If START WITH is equal to 1 and INCREMENT BY is equal to 2, the sequenced values returned would be 1, 3, 5, 7, etc.… until the MAXVALUE or maximum value for the data type is reached.

INCREMENT BY can be negative, positive, or zero and cannot contain any digits to the right of the decimal point. If a negative value is specified (INCREMENT BY < 0), the current sequence is decremented by the INCREMENT BY value to create the next sequence. If a positive value is specified (INCREMENT BY > 0), the current sequence is incremented by this value to create the next sequence. If a zero is used (INCREMENT BY = 0), the same value as the current value is used for the sequence. This method could be employed to create a constant, non-changing sequence.

If INCREMENT BY is not specified, the default increment is a positive 1.

MINVALUE / NO MINVALUE

MINVALUE defines the minimum end point range. This is the lowest value this sequence can reach or the last value of a descending sequence. The last value will always be equal to or greater than the MINVALUE depending on the value of INCREMENT BY. It is also the value an ascending sequence will restart at if CYCLE is specified and the MAXVALUE is reached.

If the MINVALUE keyword is used, a numeric value must be specified. The MINVALUE can be any positive or negative value, including the value zero, within the data type's range and must have a zero scale. MINVALUE must also be less then or equal to the MAXVALUE.

If MINVALUE is not specified and left to default, the default value will vary depending on whether the sequence is descending or ascending. If the sequence is descending, the default is the minimum value for the sequence's data type. This is also the largest negative value for the sequence's data type. If the sequence is ascending, the default is the START WITH value specified for the sequence. If the START WITH value is not specified, then the default MINVALUE is 1 (one).

NOMINVALUE, as a single word, can be used in place of NO MINVALUE.

MAXVALUE / NO MAXVALUE

MAXVALUE specifies the highest value that can be reached by this sequence. Although this keyword is optional, if used, a value must be specified. DB2 requires the value to be in the range of the data type being used. If the date type is decimal the scale must be zero. In addition, the MAXVALUE must be greater than or equal to the MINVALUE, if the MINVALUE is specified.

If MAXVALUE is not specified and is allowed to default, the default for ascending values is the largest value allowed by the sequence's data type. For a descending value, MAXVALUE will default to the START WITH value or a -1 if START WITH is not specified.

The MAXVALUE does not necessarily have to be reached within a cycle of value. If an INCREMENT BY value great than 1 was used, the MAXVALUE may not be reached depending on the value specified. For example, suppose that MAXVALUE is set to 50 and INCREMENT BY is set to 3. On the 17^{th} sequenced value, 49 is returned. The next increment would take the sequence to 52, a value out of the range for this sequence. This sequence would then recycle at 49 rather than 50.

If NO MAXVALUE, the default if this keyword is not used, is in effect, the maximum value for an ascending sequence is the largest value allowed by the data type. For a descending sequence, the START WITH value is specified. If START WITH is not specified, then –1 is used.

NOMAXVALUE, as a single word, can be used in place of NO MAXVALUE.

CYCLE / NO CYCLE

The CYCLE and NO CYCLE keywords determine what the sequence should do if and when the minimum or maximum value for the range is reached. If CYCLE is chosen, values will continue to be generated for this sequence after the minimum or maximum value for the range is reached:

- If a descending sequence reaches the minimum value for the range, the maximum for the range is generated as the next value in the sequence.
- If an ascending sequence reaches the maximum value for the range, the minimum for the range is generated as the next value in the sequence.

Processing then starts over at the new first value. This generated new value does not have to be equal to the START WITH value. An application using this sequence should be aware that duplicate values can be generated if this sequence is cycled.

NO CYCLE is the default. If NO CYCLE is specified or no cycle keyword is not specified and defaulted to, and the MINVALUE or MAXVALUE is reached, an error occurs and no sequence value is generated. This sequence would have to be altered to continue using this sequence to generate values, or be dropped and recreated with a different data type that would allow more values to be generated.

The first cycle always begins with the START WITH value if specified or the default START WITH value 1 if START WITH is not specified. However, when specifying CYCLE and MAXVALUE is reached, the sequence restarts at the MINVALUE. It is possible then that subsequent sequences could have a different number of values than the first sequence.

NOCYCLE, as a single word, can be used in place of NO CYCLE.

CACHE / NO CACHE

The CACHE keyword specifies whether or not sequenced values will be pre-allocated in memory. This option is used to improve performance in a non-data sharing environment and in a data sharing environment where the application can tolerate the possibility of out of sequence values.

The number of cached sequence values must be a positive integer value greater than one. If nether the CACHE or NO CACHE keywords are specified, the default is CACHE 20. Caching sequence values could improve performance because it reduces synchronous I/O during a sequence request to the DB2 catalog table SYSIBM.SYSSEQUENCES to retrieve the next sequence value.

Caching is performed independent of cycles. If the MAXVALUE is reached during the last cache cycle, only values from the current cycle are cached. For example, if CACHE is set to 20 and only 10 sequence values are created before MAXVALUE is reached, only those 10 values will be cached. The values from the next cycles will be cached on the next cache request.

Data sharing considerations: Strict sequential order cannot be guaranteed in a data sharing environment with sequence numbers being cached on multiple members. In data sharing, each data sharing member has its own cache. However, there is only one SYSIBM.SYSSEQUENCES to request values from. If we have a 2-way data sharing system, for example, an application that is running on both DB2 members at the same time and caching of the sequence is active, each application can alternately request a sequence on each member and the sequence would be satisfied from that member's set of cached sequence numbers. If CACHE is set to 20, each member would have 20 cached values. Member 1 would have cached 1 through 20 and member 2 cached 21 through 40. If the application alternated between members, the sequence order would be 1, 21, 2, 22, 3, 23, 4, etc.… If strict order is necessary, NO CACHE should be specified with the ORDER keyword.

NOCACHE, as a single word, can be used in place of NO CACHE.

ORDER / NO ORDER

The ORDER / NO ORDER keywords specify whether or not the sequence numbers must be generated in order of request.

If ORDER is specified, sequences are generated in order of request. If order is important to the application, then ORDER should be specified.

If NO ORDER, the default, is specified, the sequence values do not need to be generated in order of request.

In data sharing environments where sequence values are cached by multiple DB2 members simultaneously and the CACHE option is used, the sequence value assignments may not be in strict numeric order unless you also specify the ORDER option.

> **Note:** If both the ORDER and CACHE option are used in a *data sharing* environment, ORDER takes precedence and CACHE is ignored, and no caching will take place.

NOORDER, as a single word, can be used in place of NO ORDER.

CREATE Authorizations

To CREATE a sequence object, the privilege set needs:

- CREATEIN for the schema, or all schemas
- SYSADM or SYSCTRL authorizations

Implicit schema match has CREATEIN privilege

4.13.6 CREATE authorizations

To create a sequence object, the privilege set must contain CREATEIN for the schema, or all schemas, SYSADM or SYSCTRL authorizations. An auth id that matches the schema name implicitly has the CREATEIN authority for that schema.

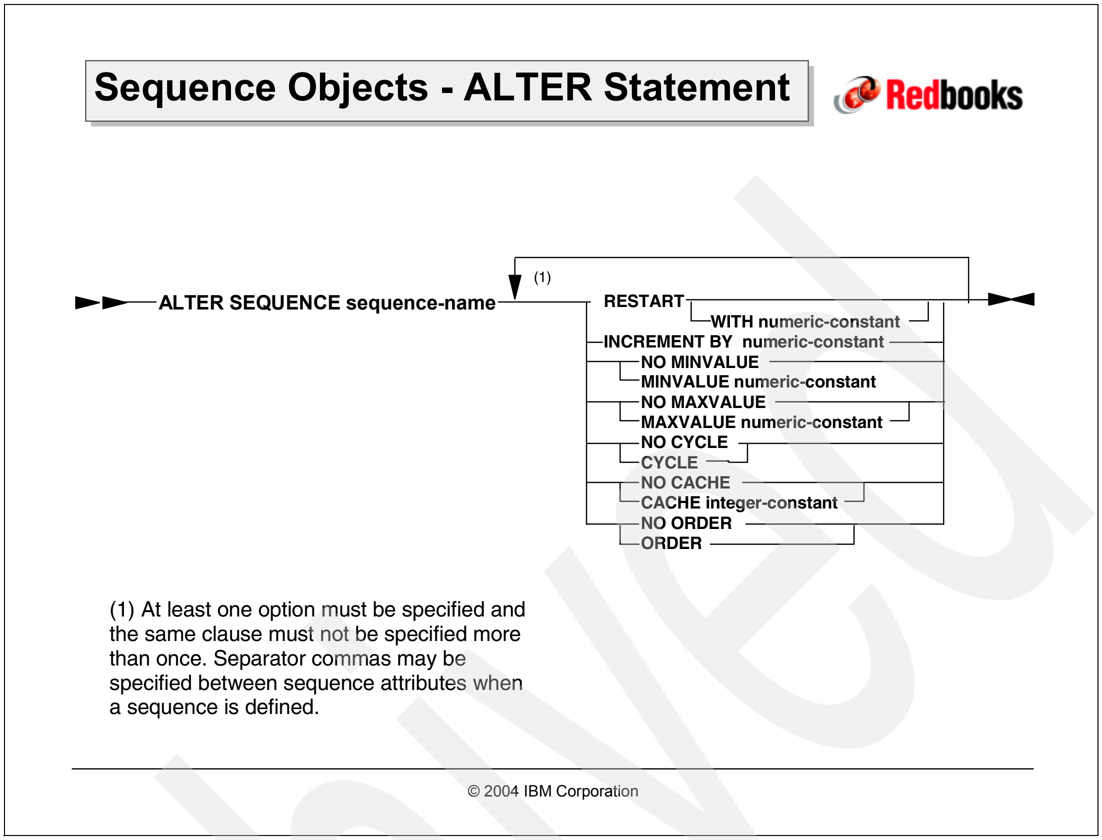
If the data type of the sequence is a distinct type, the privilege set must contain the USAGE privilege for the distinct type.

CREATE Example

```
CREATE SEQUENCE SEQTEST1 AS INTEGER
    START WITH 1
    INCREMENT BY 1
    MINVALUE 1
    MAXVALUE 5
    CYCLE
    CACHE 5
    NO ORDER;
```


4.13.7 CREATE example

This is a simple example showing the format of the CREATE statement with all of the keywords specified. In this example, we have specified INTEGER even though it is the default. The sequence will start at 1 and increment by 1 to a maximum value of 5. When 5 is reached, the sequence will cycle back to the minimum value of 1 (one). Five values will be kept in the cache to assist with performance. On the first pass, that would be all five values generated. However, on subsequent cycles, six values will be generated.



4.13.8 ALTER SEQUENCE statement

The ALTER SEQUENCE SQL statement changes the attributes of the sequence object. The attributes that can be modified by ALTER are: INCREMENT BY, MINVALUE, MAXVALUE, CACHE, CYCLE, and ORDER. At least one sequence attribute must be specified on the ALTER statement. In addition, ALTER can be used to restart the sequence at a different point by specifying the RESTART WITH keyword. ALTER can only affect future values in the sequence and only takes affect after the ALTER has been committed. Altering a sequence could cause unused sequences in cache to be lost.

Although you can alter selected keywords describing a sequence, the data type of the sequence cannot be altered. In order to change the data type of a sequence, the sequence object must be dropped and re-created.

Be cautious when altering the sequence specification. It is easy to create a non-incrementing sequence. For example, if you set MAXVALUE to 50 and INCREMENT BY 100, then every INSERT will exceed the maximum value. DB2 will not give you a warning or error; it will simply insert a zero for the sequence value.

The description of INCREMENT BY, MINVALUE, MAXVALUE, CACHE, CYCLE and ORDER are the same when used by the ALTER SQL statement as their descriptions when used with the CREATE SQL statement. RESTART WITH is the only keyword unique to the ALTER SQL statement. Specifying RESTART WITH *numeric_value* on an ALTER causes the sequence to start at the new specified value on the next sequence request. The MAXASSIGNEDVAL column in SYSIBM.SYSSEQUENCES is set to NULL and the RESTART WITH column is set to the numeric constant on the RESTART WITH keyword if a value is specified.

If RESTART is specified without the WITH numeric_value portion, the sequence is still restarted. However, rather than being restarted at a new sequence value, the sequence is restarted at the START WITH value implicitly or explicitly associated with the original CREATE. Altering a sequence object with a new starting value could result in duplicate sequence values. If an ALTER is performed and the RESTART keyword is not specified at all, then there is no attempt to restart the sequence and it proceeds at the next sequential value.

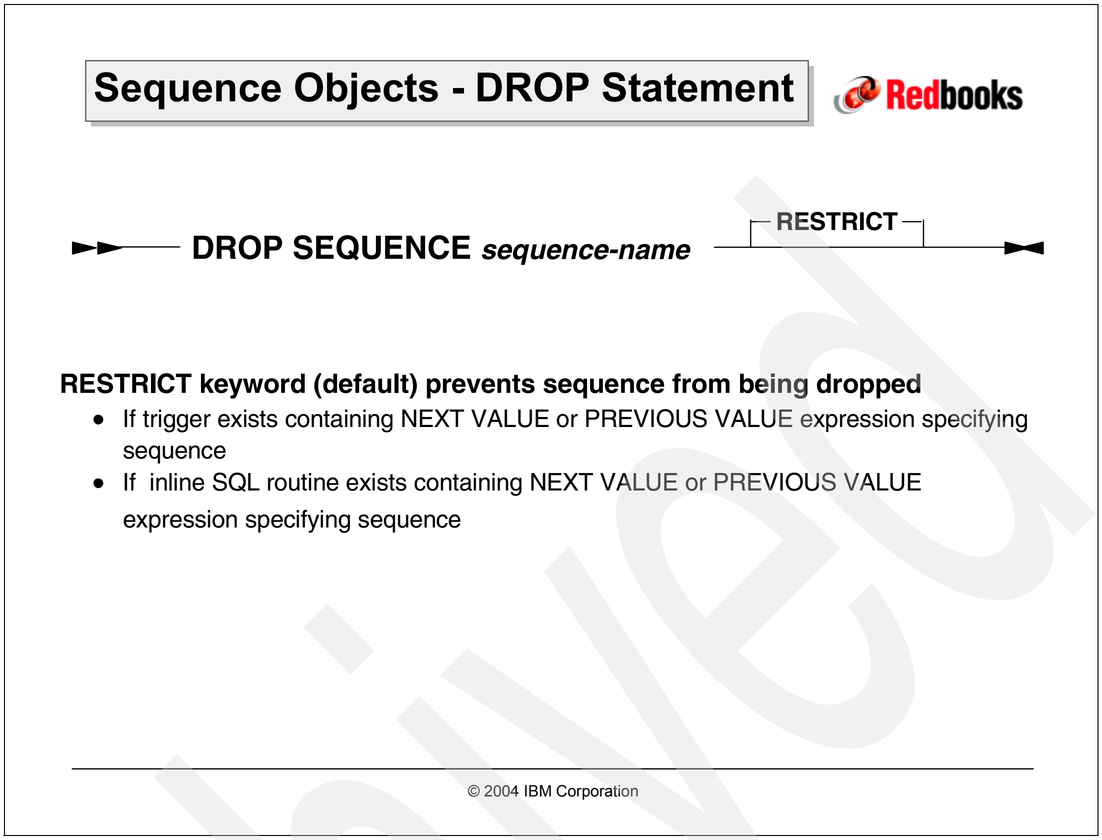
ALTER Example

```
ALTER SEQUENCE SEQTEST1
      RESTART WITH 10
      INCREMENT BY 100
      MAXVALUE 1000000
   ;

COMMIT;
```


4.13.9 ALTER example

In this example, the sequence just created will be modified. It was decided that the original maximum was a little low, so it is being increased to 1,000,000. The sequence will also no longer be incremented by one, but rather by one hundred. When the sequence is restarted, it will restart at 100 rather than the minimum. The minimum, cache size, and cycle values will all remain the same as the values the sequence was created with.

Also note the COMMIT after the ALTER. The ALTER does not take affect until it has been committed.

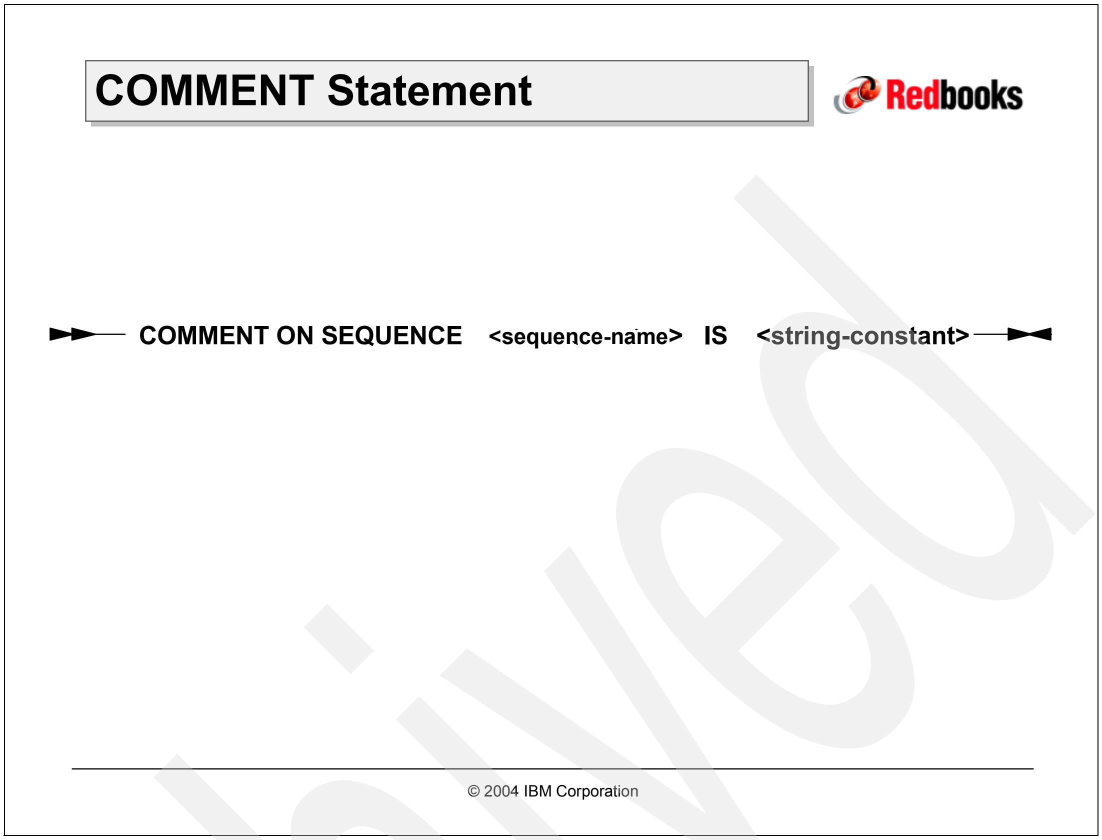


4.13.10 DROP SEQUENCE statement

The DROP SEQUENCE statement drops the sequence object and removes a sequence object description from the DB2 catalog. The sequence combined with an implicit or explicit schema qualifier must identify an existing defined sequence in the catalog. DROP SEQUENCE cannot be used to remove a system generated sequence defined to support an identity column. The default keyword RESTRICT indicates that a sequence cannot be dropped if certain dependencies exist. Those dependencies are:

- A trigger exists that uses the NEXT VALUE FOR or PREVIOUS VALUE FOR expression for the specified sequence.
- An inline SQL routine exists so that a NEXT VALUE FOR or PREVIOUS VALUE FOR expression in the routine body specifies the sequence.

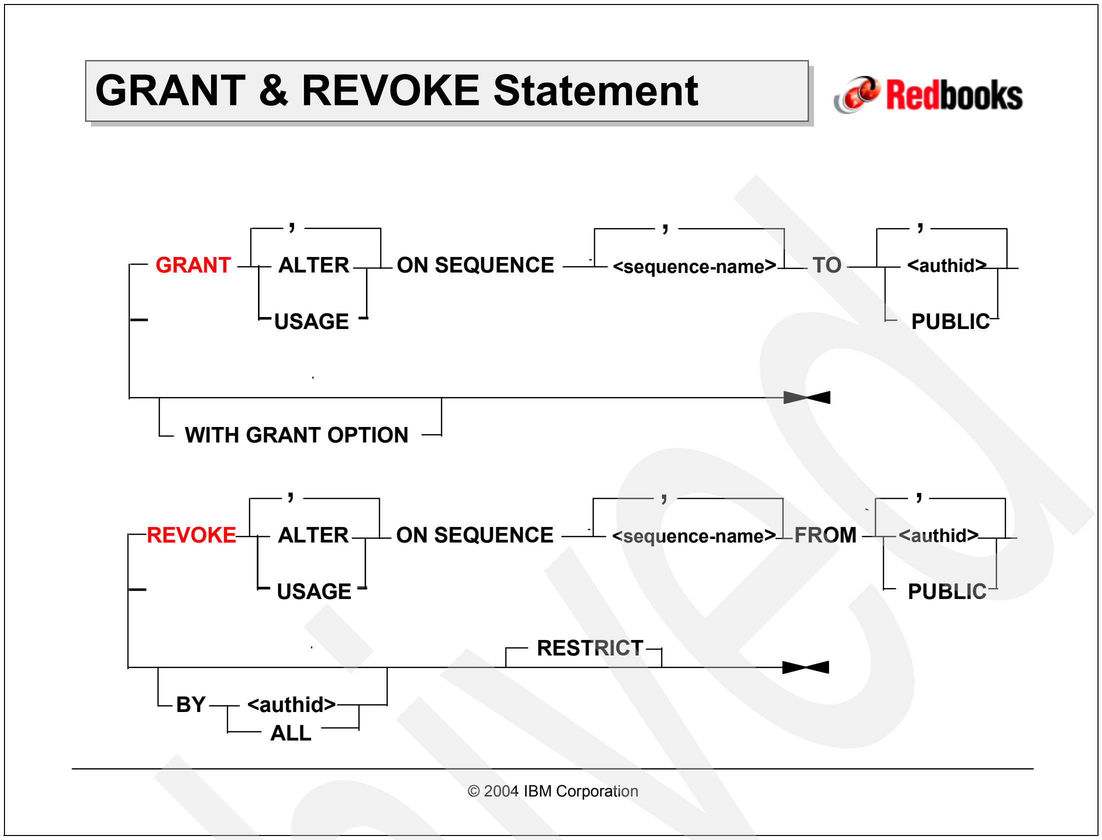
Dropping a sequence object also drops all privileges associated to that object, and all packages or plans with a dependency on that sequence are invalidated. A DROP SEQUENCE SQL statement cannot be used to remove an identity column from DB2. In addition, a DROP TABLE SQL statement has no effect on a sequence object because there is no direct tie between a sequence object and a table.



4.13.11 COMMENT statement

The COMMENT SQL statement allows a user to supply a comment or description for a sequence in the REMARKS column of the DB2 catalog table SYSIBM.SYSSEQUENCES. A comment can be up to 762 characters. The name of an identity column cannot be specified on the COMMENT SEQUENCE SQL statement.

Sequence is one of many object types that can be specified on a COMMENT. The COMMENT SQL statement is not new in Version 8.



4.13.12 GRANT and REVOKE statements

The GRANT and REVOKE SQL statements are used for granting or revoking the ALTER and/or USAGE privilege for a sequence or list of sequences to/from an authorization identifier (auth id) or to/from PUBLIC. Two flavors of the GRANT and REVOKE privileges for sequences exist.

Granting the ALTER privilege over a sequence allows the auth id specified to modify the characteristics of the sequence or add a comment describing that sequence. Granting the USAGE (or SELECT, a synonym for USAGE) privilege over the sequence to an auth id allows the auth id specified to invoke the NEXT VALUE and PREVIOUS VALUE SQL expressions. In both cases, a list of auth ids or PUBLIC can be specified in place of the single auth id.

Revoking ALTER over a sequence removes the ALTER privilege from the auth id. Revoking USAGE (or SELECT, a synonym for USAGE) over the sequence removes the privilege to invoke the NEXT VALUE and PREVIOUS VALUE expressions. REVOKE can also remove the ALTER and USAGE privilege from a list of auth ids or from PUBLIC.

The REVOKE SQL statement has the additional keyword RESTRICT. The REVOKE will default to using the RESTRICT keyword if it is not specified. The keyword RESTRICT prevents the USAGE privilege from being revoked on a sequence if the revokee owns one of the following objects, and does not have the USAGE privilege from another source:

- A trigger that specifies the sequence in a NEXT VALUE or PREVIOUS VALUE expression
- An inline SQL function that specifies the sequence in a NEXT VALUE or PREVIOUS VALUE expression

The sequence name when qualified implicitly or explicitly with a schema qualifier must uniquely identify a sequence that exists at the current server.

Next and Previous Values

Applications can refer to the named sequence object to get its current or next value

- NEXT VALUE FOR < sequence- name >
- PREVIOUS VALUE FOR < sequence-name >
  - Returns most recently generated value for sequence for previous statement within current session
  - NEXT VALUE must have been invoked within current session

Can be invoked during:

- SELECT statement or SELECT INTO statement within select-clause (except if statement contains DISTINCT or UNION keyword, GROUP BY or ORDER BY)
- INSERT statement within VALUES clause
- INSERT statement within select-clause of fullselect
- Update statement within SET clause (except NEXT VALUE cannot be in select-clause of fullselect of expression)
- VALUES or VALUES INTO statement (except within select-clause of fullselect of expression)
- CREATE PROCEDURE, FUNCTION, TRIGGER
- Set :hv = NEXT VALUE FOR sequence

4.13.13 Next and previous values

A sequence is referenced by using the NEXT VALUE FOR or PREVIOUS VALUE FOR SQL expressions and specifying the name of the sequence. The keywords NEXTVAL and PREVVAL can be used alternately for NEXT VALUE and PREVIOUS VALUE respectively.

NEXT VALUE FOR

An application can get the next sequence value using the NEXT VALUE FOR expression with a sequence name. The NEXT VALUE FOR expression generates and retrieves the next (new) sequence value for the specified sequence. If the NEXT VALUE expression is used multiple times within the same SQL statement and specifies the same sequence names on all occurrences of NEXT VALUE, all of the NEXT VALUE expressions will return the same value for the same row in the result.

> **Important:** Once the NEXT VALUE FOR expression is issued, the returned value is considered consumed. If a ROLLBACK occurs, it has no effect on the values already generated. Even though the sequence itself always generates unique values (provided cycling is not allowed) with no gaps, you can still have gaps, for example, because the transaction performs a ROLLBACK after a new sequence value was obtained. In this respect, as in many others, sequences behave exactly the same as identity columns. More information on gaps can be found in 4.13.22, “Gaps” on page 338.

If the next value for a sequence is generated and it exceeds the maximum value for an ascending sequence or the minimum value for a descending sequence, an error (SQLCODE -359) occurs if NO CYCLE was specified. The sequence would need to be modified to enable cycles for the sequence or dropped and recreated with a different data type that allowed more values for the sequence.

The sequence value returned by the NEXT VALUE FOR expression has the same data type as the sequence it was issued against.

PREVIOUS VALUE FOR

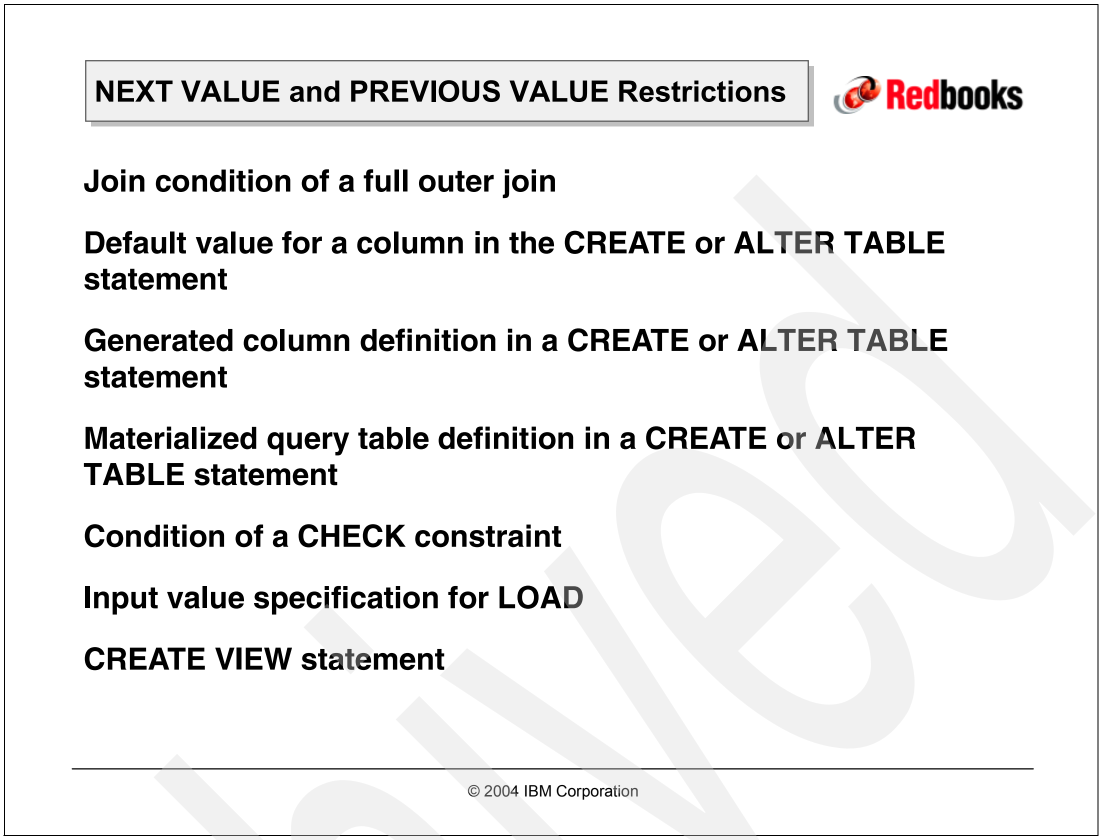
A PREVIOUS VALUE FOR expression returns the most recently generated value for the specified sequence for a previous statement within the current application process. This value can be repeatedly referenced by using PREVIOUS VALUE expressions to specify the name of the sequence. There may be multiple instances of PREVIOUS VALUE expressions specifying the same sequence name within a single statement and they all return the same value.

The PREVIOUS VALUE expression can only be issued if a NEXT VALUE expression has already been issued for the specified sequence. If a NEXT VALUE expression is not issued, an SQLCODE -845 will occur. The previous value persists until the next value is generated for the sequence, the sequence is dropped or the application terminates.

Where can NEXT VALUE and PREVIOUS VALUE be invoked?

The NEXT VALUE FOR and PREVIOUS VALUE FOR expressions can be used in a variety of SQL DML situations:

- They can be used within the select clause of a SELECT statement or SELECT INTO statement. However, those statements cannot contain a DISTINCT or UNION keyword or a GROUP BY or ORDER BY clause.
- They can be used in the VALUES clause of an INSERT statement. This includes multi-row INSERT statements. When a NEXT VALUE FOR expression is specified for a particular sequence name, a sequence value is generated for each row inserted by the multi-row insert process.
- They can be used within the select clause of a fullselect of an INSERT statement.
- An UPDATE statement can specify both expressions on the SET clause. The PREVIOUS VALUE expression can be specified for any UPDATE statement. However, the NEXT VALUE expression cannot be specified within the select clause of a full select of an UPDATE statement.
- A SET host variable statement can use both expressions.
- The VALUES and VALUES INTO statements can use both expressions as long as they are not specified in the select clause of a full select or the expression.
- Both expressions can be used in CREATE PROCEDURE, CREATE FUNCTION, and CREATE TRIGGER statements.



4.13.14 NEXT VALUE and PREVIOUS VALUE restrictions

There are a number of SQL situations where the NEXT VALUE and PREVIOUS VALUE expressions cannot be used. These include:

- Join condition of a full outer join
- Default value for a column in the CREATE or ALTER TABLE statement
- Generated column definition in a CREATE or ALTER TABLE statement
- Materialized query table definition in a CREATE or ALTER TABLE statement
- Condition of a CHECK constraint
- Input value specification for LOAD
- CREATE VIEW statement

In addition, the NEXT VALUE expression cannot be specified in the following places:

- CASE expression
- Parameter list of an aggregate function
- Subquery in a context other than those explicitly allowed
- SELECT statement for which the outer SELECT contains a DISTINCT operator or a GROUP BY clause
- SELECT statement for which the outer SELECT is combined with another
- SELECT statement using the UNION set operator
- Join condition of a join
- Nested table expression
- Parameter list of a table function
- Select-clause of fullselect of an expression in the SET clause of an UPDATE statement
- WHERE clause of the outer-most SELECT statement or a DELETE or UPDATE statement
- ORDER BY clause of the outer-most SELECT statement
- IF, WHILE, DO . . . UNTIL, or CASE statements in an SQL routine

Sequence Examples

Redbooks

1) *Assume sequence created with START WITH 1, INCREMENT BY 1*

```
SELECT NEXT VALUE FOR MYSEQ FROM SYSIBM.SYSDUMMY1;  Generates Value of 1
SELECT NEXT VALUE FOR MYSEQ FROM SYSIBM.SYSDUMMY1;  Generates Value of 2
COMMIT;
SELECT PREVIOUS VALUE FOR MYSEQ FROM SYSIBM.SYSDUMMY1;
                    Returns most recently generated value (2)
```

2) *PREVIOUS/NEXT VALUE FOR specified within SET clause of UPDATE statement*

```
UPDATE T SET C1 = (SELECT PREVIOUS VALUE FOR S1 FROM T);
UPDATE T SET C1 = NEXT VALUE FOR S1;
```

3) *Viewing sequence while inserting*

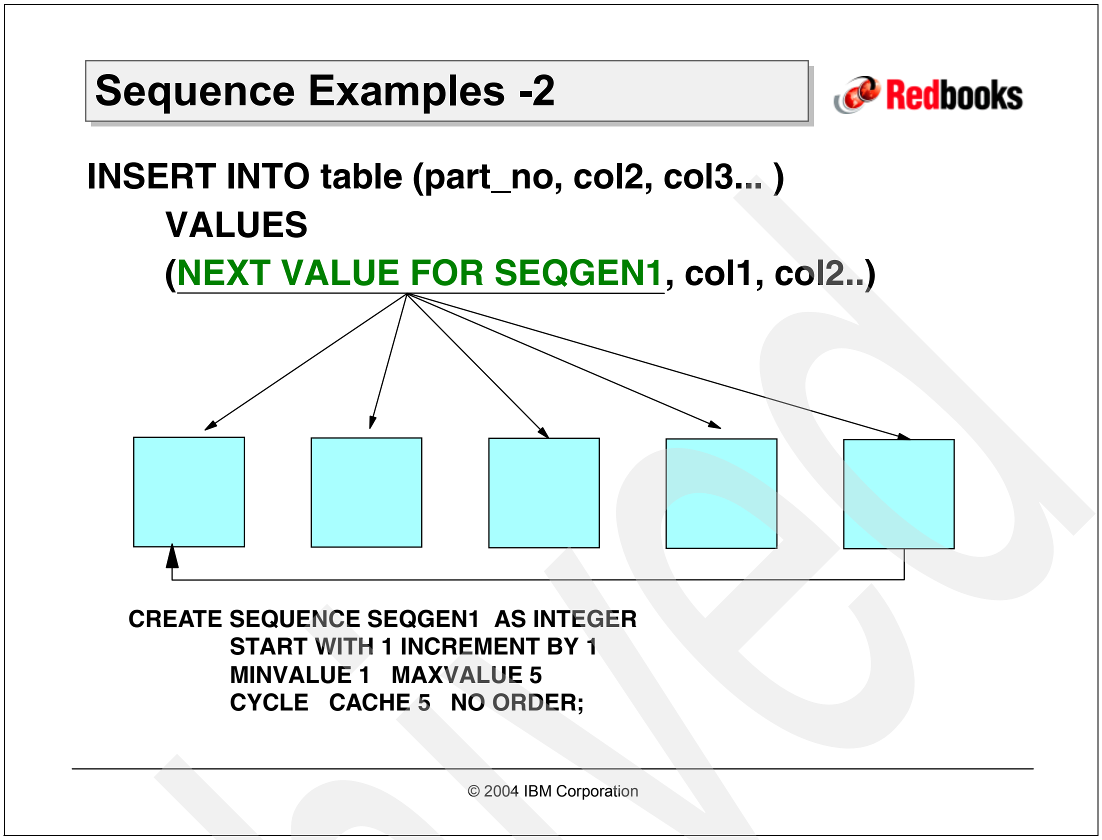
```
SELECT * FROM FINAL TABLE
  (
    INSERT INTO TESTTAB (KEYVALUE, TESTSEQ)
        VALUES ( NEXT VALUE FOR SEQTEST1, NEXT VALUE FOR SEQTEST2 )
  )     ;
```


4.13.15 Sequence examples - 1

In the first example, the sequence is simply being incremented by one. Each subsequent SELECT statement generates the next available sequence. The COMMIT has no affect on the sequence generation. The last SELECT, performing a PREVIOUS VALUE FOR, still returns the last value generated by the previous SELECT.

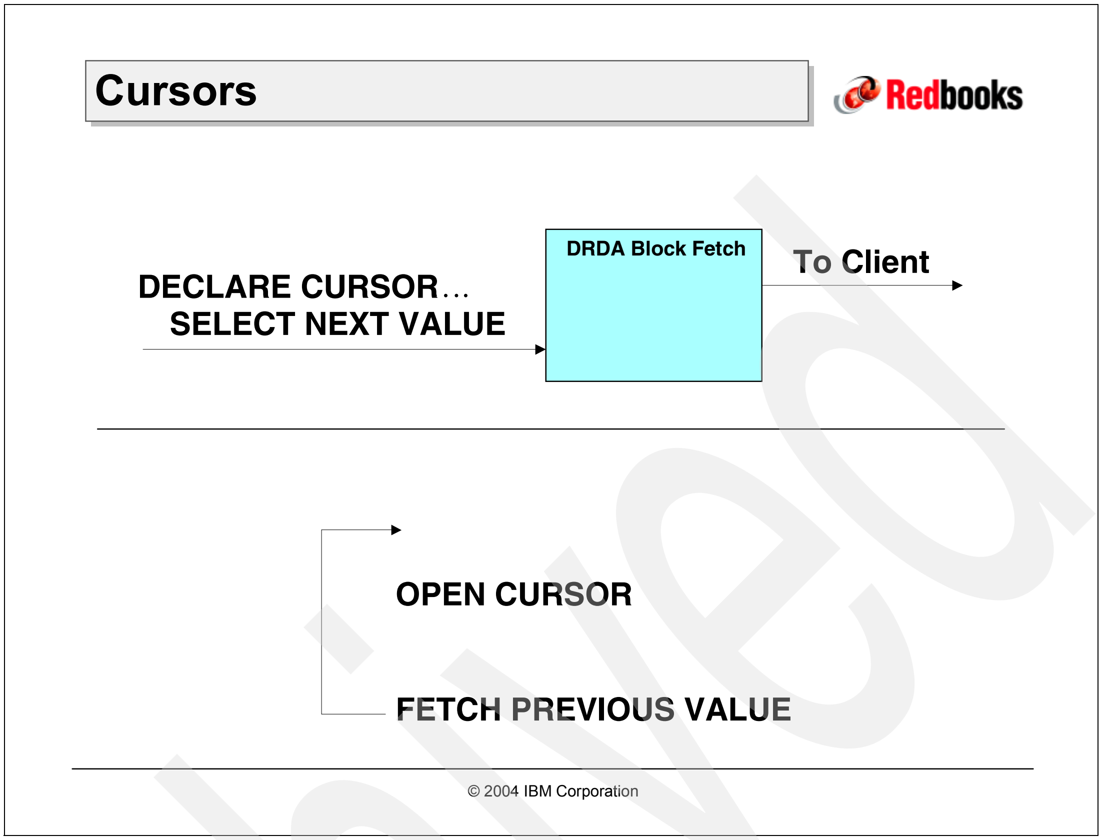
The second example uses the sequence generated as part of the SET clause of an UPDATE. This is valid for both searched and positioned updates.

In the final example on this page, we the Version 8 syntax for a select from insert. This syntax allows us to examine the sequence value generated without having to perform a separate SELECT.



4.13.16 Sequence examples - 2

Sequences can also be used to spread inserted rows across multiple partitions if used to supply the partitioning value. By manipulating the INCREMENT BY and START WITH values and specifying CYCLE, a sequence could be generated that would force each new inserted row to go to a different partition of a partition table space. As an alternative and because a sequence is independent and not associated to any particular column, all or part of the generated sequence value could be used as part of the partition key in an attempt to spread the rows evenly across multiple partitions of a portioned table space.



4.13.17 Cursors

Caution should be the rule of thumb if cursors and sequences are used together.

NEXT VALUE

When working with a SELECT statement, a sequence is generated for every row selected for the number of rows in the answer set when NEXT VALUE is specified. However, if that SELECT with NEXT VALUE is defined as part of a cursor, the sequence is generated as the row is retrieved.

This can cause gaps in the sequence if all the rows retrieved are not accessed. When a client requests rows using a cursor, groups of rows are blocked together and then sent to the client to complete the fetch process. If all of the rows in the block are not processed by the fetch at the client, sequence values could be unused, creating a gap in the sequence. This could be avoided if FETCH FOR 1 ROW ONLY was specified on the cursor. FETCH FOR 1 ROW ONLY will affect performance, though.

So the consequences must be carefully weighed between causing gaps in the sequence and potentially negatively impacting the performance of the fetch. This issue is important only if gaps in the sequence cannot be tolerated. If performance and the prevention of gaps are both important, then NEXT VALUE should not be considered for use with the cursor in a client application.

PREVIOUS VALUE

In almost all cases, PREVIOUS VALUE should be avoided when working with cursors. PREVIOUS VALUE does not function the same with a cursor as it might with other SQL statements. The sequence evaluated to PREVIOUS VALUE is the last sequence generated prior to opening the cursor. The fetch process after cursor open has no affect on the sequence returned by PREVIOUS VALUE on a cursor, the value returned will always be the same regardless of the number of fetches that are executed with a cursor with NEXT VALUE specified.

After the cursor is closed, PREVIOUS VALUE will return the last sequence generated by the NEXT VALUE expression of the previously opened cursor. There is probably little if any reason to take advantage of PREVIOUS VALUE when using cursors.

Ranges and Cycles

Sequence can be optionally defined to CYCLE after reaching its maximum value (or minimum if descending)

- Sequence will wrap around to other end of range and start new cycle of values
- Default is NOCYCLE

First cycle always starts with START WITH value but all subsequent cycles start with MINVALUE (asc sequence) or MAXVALUE (desc sequence)

If NO CYCLE in effect, sequence value generation will stop when sequence reaches end of logical range of values

4.13.18 Ranges and cycles

Cycles are tied to a range. When the CYCLE keyword is specified and the end of a range is reached, the sequence cycles back to the beginning of that range. The range itself can be affected by ALTER/CREATE keywords and by the sequence's data type.

The data type affects the high value end of a range. The range cannot extend beyond the highest allowable value within a data type.

If specified, the keywords MINVALUE and MAXVALUE will take precedence if within a data type's range. MINVALUE establishes the low end of a range and the point at which the sequence will start over if CYCLE is used. MAXVALUE is the highest sequence value that can be generated. When MAXVALUE is reached and CYCLE is specified, the sequence starts over.

INCREMENT BY can also have an affect on what sequences are generated within a range. If INCREMENT BY is a value greater than 1 or less than -1, the full range of sequence values may not be realized. For example, if a sequence is defined with START WITH set to 1, a MINVALUE of 1, MAXVALUE of 10, and an INCREMENT BY of 3, the highest sequence that can be generated is 7. The sequence requested would cause this sequence to cycle back to the MINVALUE of 1.

A variation of this problem could be using an INCREMENT BY value that is too large. Using the same basic setup as above with an INCREMENT BY value set to 10, MAXVALUE would be reached every time a sequence is requested so the only sequence value that would ever be generated would be the MINVALUE of 1 because the sequence would constantly cycle.

Generating Constants

CREATE SEQUENCE consequence AS INTEGER

START WITH 1

INCREMENT BY 0

MINVALUE 0

MAXVALUE 5

CYCLE

CACHE 5

NO ORDER;

4.13.19 Generating constants

A constant, non-changing sequence can be generated by specifying an INCREMENT BY keyword set to 0 (zero). Every sequence generated would be the same as the START WITH value, the first sequence generated. The MAXVALUE or high value of the data type could never be reached and a cycle would never be performed for this sequence if the CYCLE keyword were specified. The START WITH value would have to be within the MINVALUE and MAXVALUE keyword values.

If START WITH were equal to MINVALUE and MINVALUE were equal to MAXVALUE and CYCLE were specified, the START WITH value would be generated repeatedly. However, if NO CYCLE were specified, one sequence value would be generated equal to the START WITH value and a subsequent attempt to generated a sequence would receive an error. This is true even if INCREMENT BY is a non-zero value.

Duplicate Sequences

Sequences are guaranteed to be unique within a cycle

However, duplicates can occur if :

- Sequence cycles while data from previous cycles still exist
- Sequence is restarted with a value that has already been generated
- Ascending/descending direction of sequence is reversed by ALTER statement (i.e., changing INCREMENT BY from positive to negative number, vice versa)
- System crashes followed by COLD START or CONDITIONAL RESTART and skips forward recovery leaving SYSIBM.SYSSEQUENCES table in inconsistent state
- A point-in-time recovery of SYSIBM.SYSSEQ table space regresses SYSIBM.SYSSEQUENCES table to point in time, causing MAXASSIGNEDVAL to become inconsistent with actual current point of the sequence

4.13.20 Duplicate sequences

DB2 will make every attempt to ensure that all sequences generated are unique. However, this is not always possible. If CYCLE is specified for the sequence, the sequence is restarted and the same sequence values are generated. If the previous set of values are still in use, the new set will be duplicates of the previous set.

The same can be true if the RESTART WITH keyword is specified and the sequence is restarted with a value that is already in use or the direction the sequence is generated is reversed by altering the INCREMENT BY keyword from positive to negative or negative to positive. One also must be careful to not recover SYSIBM.SYSSEQUENCES to a prior point-in-time, forcing the MAXASSIGNEDVAL to get out of sync with the actual values being cached.

Finally, conditional restart and cold start of the DB2 subsystem do not process all log records and could leave SYSIBM.SYSSEQUENCES in an inconsistent state. Duplicate values can be avoided if a unique index is created on the column that will contain the sequence value. Prevention is accomplished by generating an error and having the insert process fail for that sequence value.

Caching

CACHE option allows DB2 to cache preallocated values in memory for fast access

- Reduces synchronous I/O to the catalog table SYSIBM.SYSSEQUENCES
- I/O to table only required when cached values are exhausted

Recommended value is 20

- Assigning a higher value gives no benefit and increases size of gap should a failure occur

In data sharing, each member will have its own set of cached values for a single sequence

- Numbers will not be allocated in sequence

Use NOCACHE if values must be assigned in strict numeric order

- Each assignment of sequence value results in update of Catalog

4.13.21 Caching

The performance of allocating the next sequence number can happen faster with caching than without caching because a range of sequence numbers is allocated in DB2 memory when caching is in effect. When the CACHE keyword is specified and the cache is allocated, the first value of the cache is assigned to the sequence and the MAXASSIGNEDVAL column of the SYSIBM.SYSSEQUENCES table will contain the last value in this cache. Subsequent values assigned to the sequence come from the set of cached values that have not been assigned yet, until all the cached values are exhausted.

There is no update of the SYSIBM.SYSSEQUENCES table while the values from a cache, except the very first value, are being assigned. When caching is not in effect, each assignment of a sequence value results in an update of the catalog as compared to having caching in effect. With caching in effect, the catalog is only updated when the cache is refreshed and will minimize catalog update activity. Catalog locking should not be an issue when SYSIBM.SYSSEQUENCES is being updated.

A cache value should be chosen that allows access to more successive sequence values while minimizing the number of I/Os to the catalog table. Care should be taken though, to not chose a number too large. The larger the cache, the greater the number of unused values could be lost in the event of a system failure. These lost values represent a gap in the sequence.

The recommended value for the CACHE keyword is 20. Assigning a higher value gives no benefit and increases the size of the gap should a failure occur. Remember also, that in a data sharing environment, each data sharing member will use its own cache and set of cached values for a single sequence.

Gaps

Gaps are possible if:

- Transaction advances sequence and then rolls back
- SQL statement leading to generation of next value fails after value generated
- NEXTVAL used in SELECT statement of cursor in DRDA where client uses block-fetch and not all retrieved rows are FETCHed
- Sequence or identity column associated with sequence is altered and then ALTER rolled back
- Sequence or identity column table DROPped and then DROP rolled back
- SYSIBM.SYSSEQ table space is stopped, leading to loss of unused cache values
- DB2 system failure or shut-down leading to loss of unassigned cache values lost causing gap in sequence

Note that a transaction incrementing a sequence twice may not be assigned consecutive values

Big gaps could be removed by Altering sequence using RESTART WITH parameter

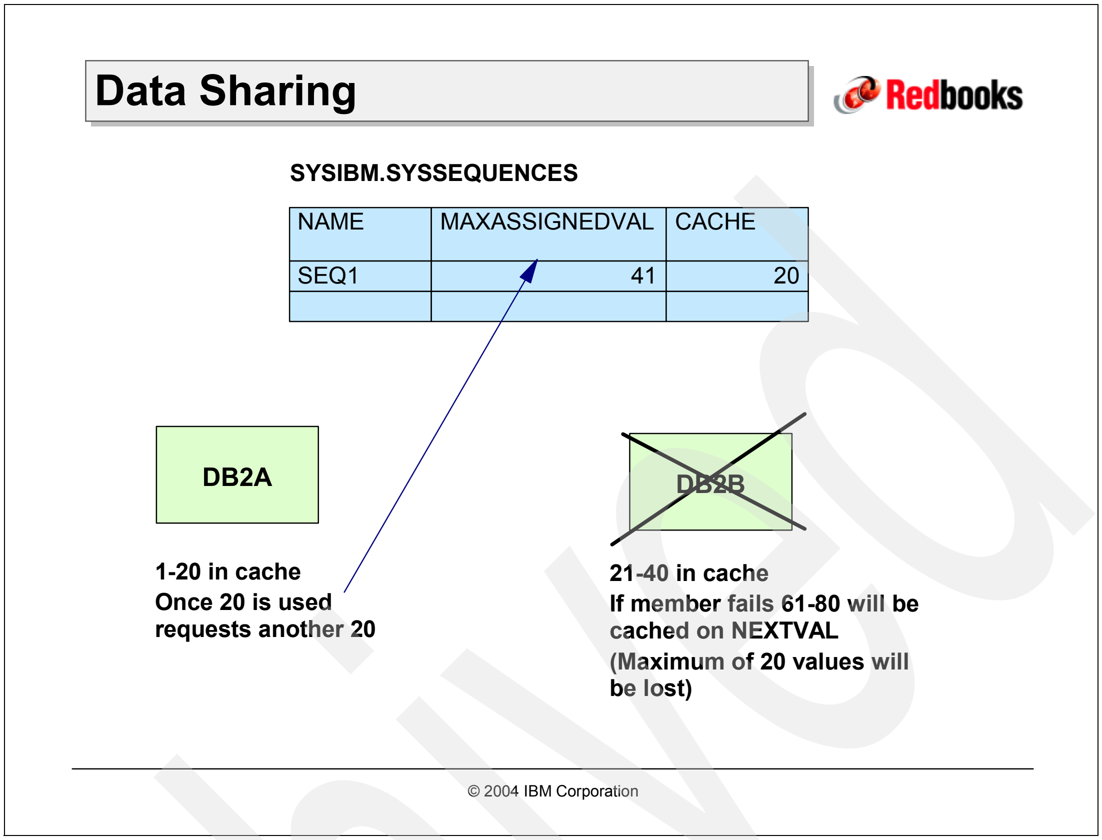
4.13.22 Gaps

A gap is any unintentional break in a sequence. Unintentional here means the gap was not the result of using the INCREMENT BY keyword of the CREATE or ALTER SQL statement. If INCREMENT BY specifies a value greater than 1 or less than -1, the generated sequences, although sequential, will not be without gaps. The gaps that are of concern to us are the unintentional gaps caused by anything other than the INCREMENT BY keyword.

There are a number of ways gaps might be caused in a sequence. If a transaction is rolled back after the sequence has been generated, that generated sequence is lost and a gap will exist. The SQL statement using the sequence is all that is rolled back; the sequence itself continues to increment forward. Similarly, if an SQL statement fails after the sequence for that SQL statement has already been generated, the generated sequence is once again lost and a gap will exist. Another example of how a gap might be created is given in 4.13.17, “Cursors” on page 332, where the use of NEXT VALUE and cursors was discussed.

Generated sequences can also be lost if something happens to the cache. If the cache is lost, all sequences not used in the cache are also lost, causing a gap. The cache can be lost by stopping the SYSIBM.SYSSEQ table space, or if the DB2 subsystem is stopped or the DB2 subsystem crashes. DDL can also negatively affect a sequence. If the SEQUENCE is dropped or altered and the drop or alter is rolled back, gaps could be left in the sequence.

Intentional gaps could also result if multiple transactions are processing using the same sequence name.



4.13.23 Data sharing

Caution should be taken using sequence cache in a data sharing environment. If caching is in effect and multiple tasks cause a sequence to be generated in multiple DB2 members, each member would cache some number of sequence values. For example, if CACHE were set to 20, member 1 would cache values 1-20 and member 2 would cache values 21-40. However, the task running would simply take the next value from their particular cache. So if task A running on member 1 requested the next sequence, it would get the value 1 from that member's cache. Then, if another task running on a different member is next to request a sequence, it would get 21 from its cache. Task A would then get 2, and so on. The generated sequence values being assigned are not in sequential order.

For data sharing systems, if sequence numbers must be assigned in strict numeric order, then the NOCACHE option must be used. This consideration does not apply for non-data sharing subsystems where the assigned numbers will always be in strict numeric order, since there is only one active cache at any given time.

Recoverability

If DB2 fails, sequence status is recovered from catalog and logs, thereby guaranteeing unique sequence values continue to be generated

Unassigned sequence values kept in cache of failing members are lost

- With efficient use of sequence, gaps can be minimized

DB2 may generate duplicate sequence numbers after restart if no log records are processed in forward recovery

If there is a gap between first value assigned before system crash and value assigned after system crash

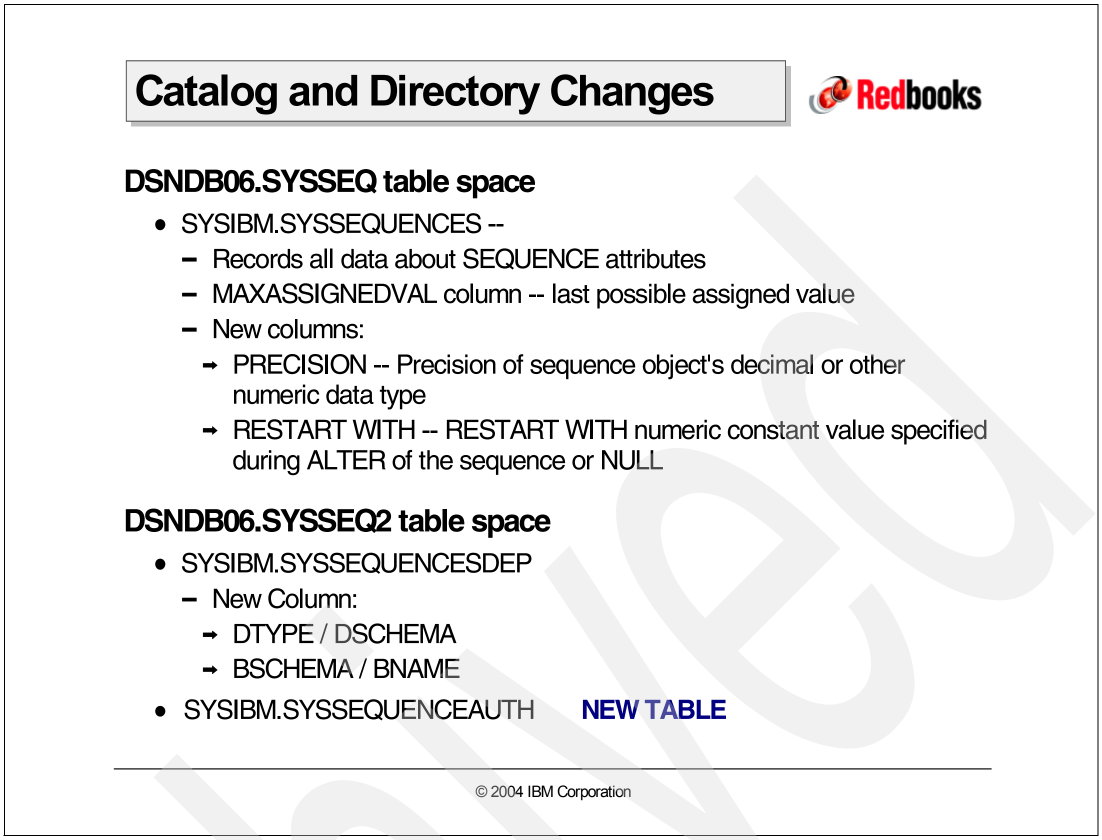
- ALTER sequence to RESTART WITH value that is next value in sequence to be assigned
- DROP and then re-CREATE sequence specifying a START WITH value that is next value in sequence to be assigned
- SELECT MAX(colname) or SELECT MIN(colname) may give actual last assigned value (colname is column to which sequence numbers were being assigned) -- works only if every value generated goes into one table -- won't work if CYCLE used

4.13.24 Recoverability

If DB2 fails, sequence status is recovered from catalog and logs, thereby guaranteeing that unique sequence values continue to be generated. However, this does not preclude the chance that some sequence values can be lost. For example, unassigned sequence values kept in the sequence cache of failing member(s) are lost. If this happens and you are able to determine the last sequence used, an ALTER sequence specifying the RESTART WITH keyword could reset the sequence back to the correct next value. You can also DROP the sequence and recreate the sequence specifying the START WITH keyword with a value equal to the next sequence that should be generated. In some instances, it is possible that duplicate sequences could be generated after a restart if no log records are processed in a forward recovery. With efficient use of sequence, gaps and duplicates can be minimized.

In some instances it may be possible to determine the last sequence used with a SELECT MAX(colname) or SELECT MIN(colname) where colname is the column the sequence was assigned. This test will only work if the sequence is not used in multiple tables and the sequence is not defined to cycle. If the sequence is used by multiple tables, the test above would have to be applied to every table and the results compared to determine the actual last sequence used. If the CYCLE keyword was specified, it is possible that the sequence has cycled and all values would appear to be used.

Selecting the column MAXASSIGNEDVAL from SYSIBM.SYSSEQUENCES may also give you inaccurate results. Although it will reflect the last sequence value assigned, that value may have been assigned to a cache. If a failure occurred and the values still in the cache were lost, MAXASSIGNEDVAL may be higher than the actual last value used. The best practice of course, is not to skip any of the recovery steps during a restart, thus minimizing the opportunities to corrupt the sequence.



4.13.25 Catalog and directory changes

Here we discuss the use of several catalog tables involved when using sequences.

SYSIBM.SYSSEQUENCES

SYSIBM.SYSSEQUENCES is an existing catalog table in DSNDB06.SYSSEQ. It became a DB2 catalog in Version 6 in support of identity columns. In Version 8 this table is still used to track information about identity columns. However, it now also tracks all information describing a sequence. SYSIBM.SYSSEQUENCES contains rows for both identity columns and sequences.

Two new columns, PRECISION and RESTARTWITH, have been added to the SYSIBM.SYSSSEQUENCES for Version 8.

The PRECISION column records the precision of the numeric data type chosen. For SMALLINT the precision is 5, for INTEGER the precision is 10, and the actual value is coded by the user for decimal. This column contains a value only for rows created in Version 8. Any rows that existed prior to Version 8 will contain a zero.

RESTARTWITH contains the RESTART WITH value specified on an ALTER SEQUENCE DDL statement. This column is set to null prior to altering the sequence object and is set back to null after the first value is generated after the sequence object has been altered.

A number of other columns in SYSIBM.SYSSEQUENCES have taken on new or additional meanings and uses after migrating to Version 8. REMARKS has been increased to VARCHAR (762) and contains the comment specified by the user with the COMMENT ON SEQUENCE SQL statement. Prior to Version 8 this column was only VARCHAR (254) and was blank.

The already existing column SEQTYPE has an additional new value, an “S”, to signify that this row is associated with a sequence object. This column only represented an identity column in DB2 Version 7.

SYSIBM.SYSSEQUENCESDEP

Currently, the relationship between an identity column and the associated DB2-generated sequence is recorded in the existing catalog table SYSIBM.SYSSEQUENCESDEP in the SYSSEQ2 table space. Some columns have been added as well. DTYPE describes the type of object that is dependent on this sequence. The object can be an identity column, an inline SQL function, or blank for entries created prior to Version 8.

Columns describing the sequence name and the schema associated with sequence have also been added. The qualifier for the object dependent on this sequence has likewise been added. In addition, the meaning of DNAME and DCREATOR has changed. DNAME is now the name of the object that is dependent on this sequence, and DCREATOR is the owner of that dependent object.

SYSIBM.SYSSEQUENCEAUTH

This new DB2 catalog table added in Version 8 records the privileges ALTER and USAGE held by an user over a sequence. This table is created in the new table space DSNDB06.SYSSEQ2.

Scalar Fullselect

What is it? .....

- A scalar fullselect is a **fullselect**, enclosed in parentheses, that returns a single value
- Allows scalar fullselect where expressions were previously supported
- Example:

```
SELECT PRODUCT, PRICE
       FROM PRODUCTS
                 WHERE PRICE <= 0.7 * (SELECT AVG(PRICE)
                                          FROM   PRODUCTS);
```

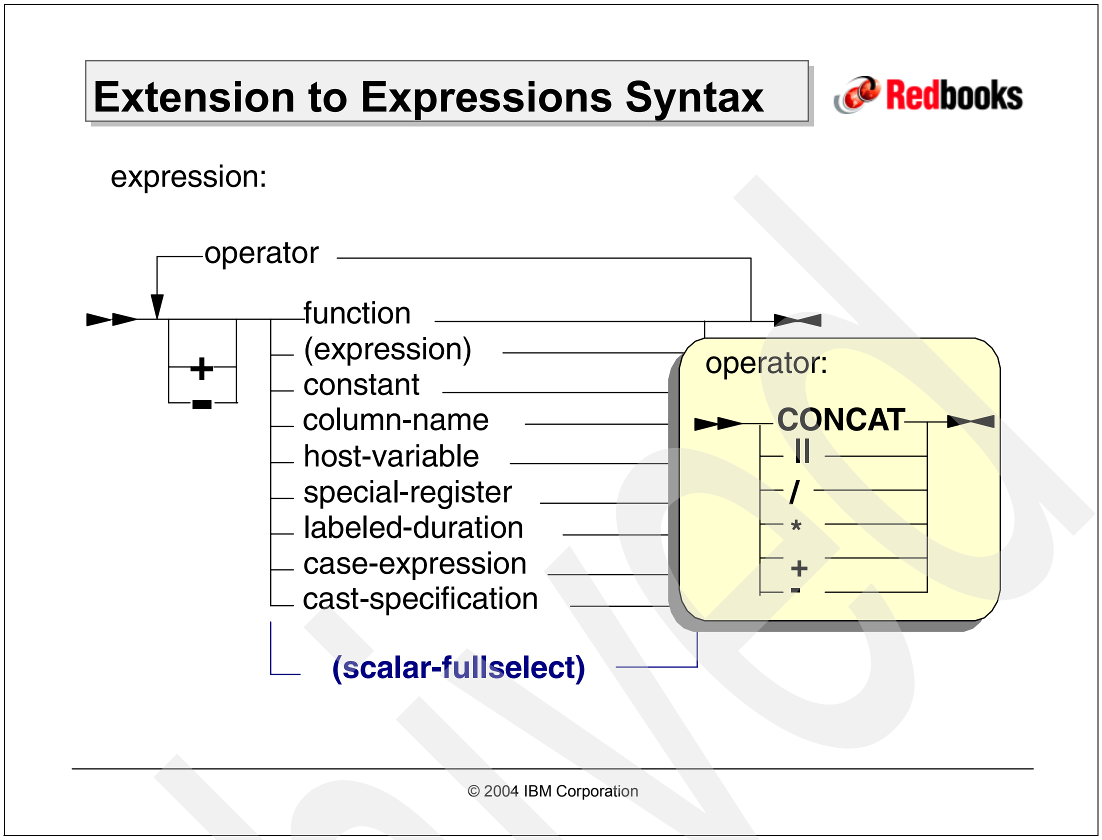
Benefits .....

- Enhances usability and power of SQL
- Facilitates portability
- Conforms with SQL standards

4.14 Scalar fullselect

DB2 UDB for Linux, UNIX, and Windows provides support for scalar fullselect. The introduction of this support in DB2 UDB for z/OS enhances the usability and power of SQL, allows applications that use scalar fullselects to be portable without any changes and also conform with the SQL standards.

A scalar fullselect is a fullselect enclosed in parentheses that can be specified in an expression and returns either a null or a single value. If more than one value is retrieved, it results in SQLSTATE 21000, SQLCODE -811, and no value is returned.

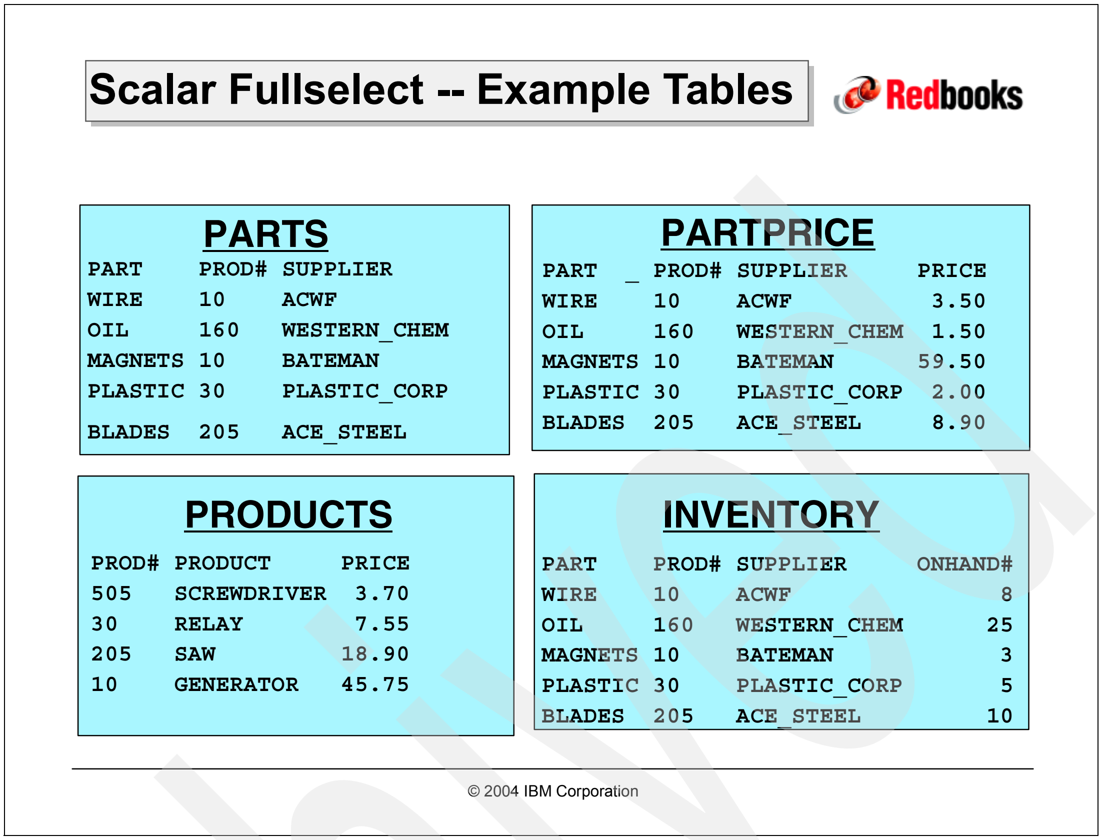


4.14.1 Extension to expressions syntax

SQL Syntax has been enhanced to allow for scalar fullselects where only expressions were allowed prior to V8. An example follows.

For each part, find its price and its inventory:

```
SELECT PART,
      (SELECT PRICE  FROM PARTPRICE WHERE PART=A.PART),
      (SELECT ONHAND#FROM INVENTORY WHERE PART=A.PART)
FROM PARTS A;
```



PARTS

PART	PROD#	SUPPLIER
WIRE	10	ACWF
OIL	160	WESTERN_CHEM
MAGNETS	10	BATEMAN
PLASTIC	30	PLASTIC_CORP
BLADES	205	ACE_STEEL

PARTPRICE

PART	PROD#	SUPPLIER	PRICE
WIRE	10	ACWF	3.50
OIL	160	WESTERN_CHEM	1.50
MAGNETS	10	BATEMAN	59.50
PLASTIC	30	PLASTIC_CORP	2.00
BLADES	205	ACE_STEEL	8.90

PRODUCTS

PROD#	PRODUCT	PRICE
505	SCREWDRIVER	3.70
30	RELAY	7.55
205	SAW	18.90
10	GENERATOR	45.75

INVENTORY

PART	PROD#	SUPPLIER	ONHAND#
WIRE	10	ACWF	8
OIL	160	WESTERN_CHEM	25
MAGNETS	10	BATEMAN	3
PLASTIC	30	PLASTIC_CORP	5
BLADES	205	ACE_STEEL	10

4.14.2 Scalar fullselect - example tables

The visual shows four tables, PARTS, PRODUCTS, PARTPRICE, and PARTINVENTORY used in the examples that we use to demonstrate the enhancement. The column PROD# links all the tables.

Scalar Fullselects in a WHERE Clause

Find which products have the prices of at least twice the lowest price of all the products.

```
SELECT PRODUCT, PRICE
  FROM PRODUCTS A
 WHERE PRICE >=  2 * (SELECT MIN(PRICE) FROM PRODUCTS);
```

PRODUCT	PRICE
RELAY	7.55
SAW	18.90
GENERATOR	45.75

4.14.3 Scalar fullselects in a WHERE clause

The example on the visual demonstrates the use of scalar fullselect in the WHERE clause.

Here the scalar fullselect is used as part of a predicate.

Nested Scalar Fullselects in a SELECT List

Redbooks

Find the cost of inventory for each product by calculating the sum of (price * onhand#) for each part in the product.

```
SELECT PRODUCT,
  (SELECT COALESCE(SUM(X.COST),0) AS INV_COST
    FROM (SELECT (
           (SELECT PRICE FROM PARTPRICE P WHERE P.PART = B.PART)
           *(SELECT ONHAND# FROM INVENTORY I WHERE I.PART =B.PART)
           ) AS COST
           FROM PARTS B
           WHERE B.PROD# = A.PROD#
         ) X(COST)
  )
  FROM PRODUCTS A;
```

PRODUCT	INV_COST
SCREWDRIVER	.00
RELAY	10.00
SAW	89.00
GENERATOR	206.50

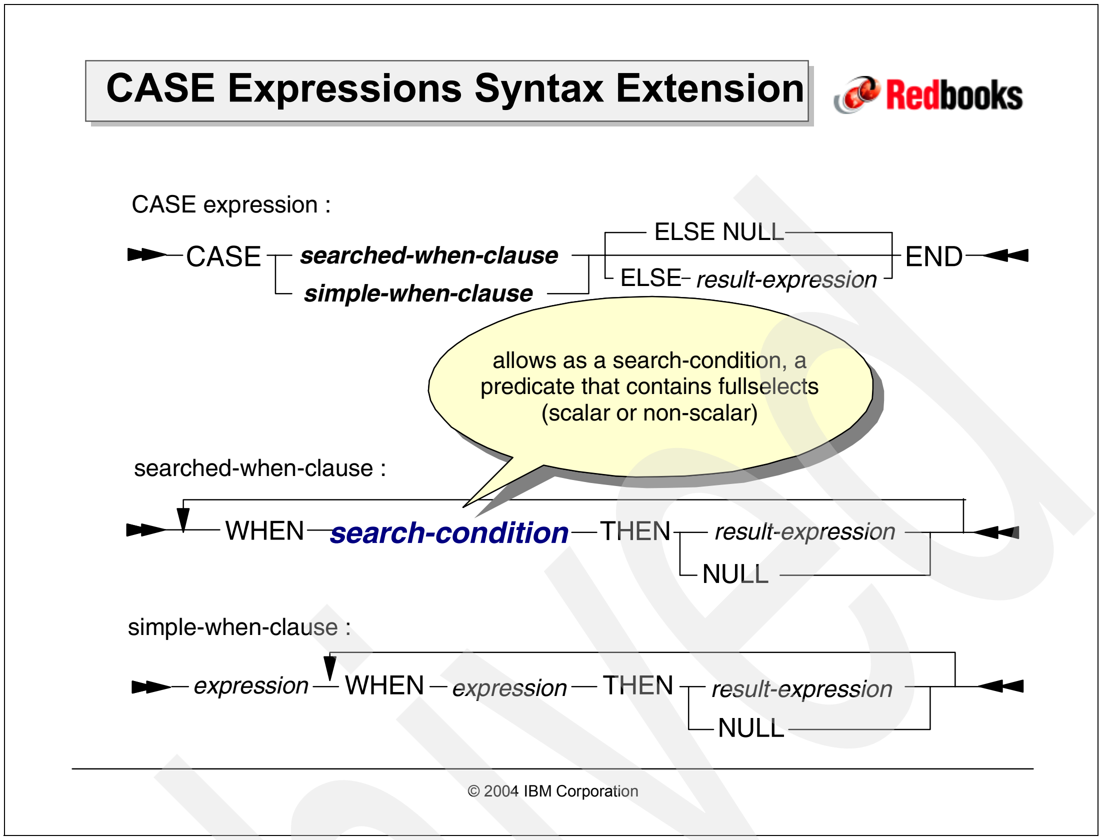
4.14.4 Nested scalar fullselects in a SELECT list

The example on the visual demonstrates the use of nested scalar fullselects in a SELECT list. Since the SQL construct is not very easy to follow at a glance, we provide some explanation. If the AS clause is specified, then the name of the result column is the name specified on the AS clause. Therefore, in this example, the name of the result column is COST and is derived by multiplying the values in two columns.

These columns are PRICE retrieved from table PARTPRICE and ONHAND# retrieved from INVENTORY table. Since the scalar fullselects for these two tables are within the scope of the SELECT statement for the PARTS table, the columns in the PARTS table can be referred to in these statements. Since the scalar fullselect for the PARTS table is within the scope of the SELECT statement for PRODUCTS table, the columns in the PRODUCTS table can be referred to in the SELECT statement for the PARTS table. The column COST is passed through the derived table X.

Note, however, that it is not essential to specify the column name COST with the derived table name X, that is, X(COST) as shown in the example. Merely specifying the name of the derived table X would also work.

The query works OK because either a null value in the case of product SCREWDRIVER or, at the most, one value for other products (because of the column function SUM) is retrieved by the scalar fullselect. Instead, if the scalar fullselect specifies just the column name COST, it would work OK for the first three products and then result in SQLSTATE 21000 and SQLCODE -811 since more than one value would be retrieved for product GENERATOR.



4.14.5 CASE expressions syntax extension

The visual shows the extended syntax of scalar fullselect in CASE expressions. The search-condition can be specified as a predicate that contains fullselect.

We now look at an example.

Scalar Fullselects in CASE Expression

Give discount to the parts that have the large inventory and raise price on the parts that have the small inventory.

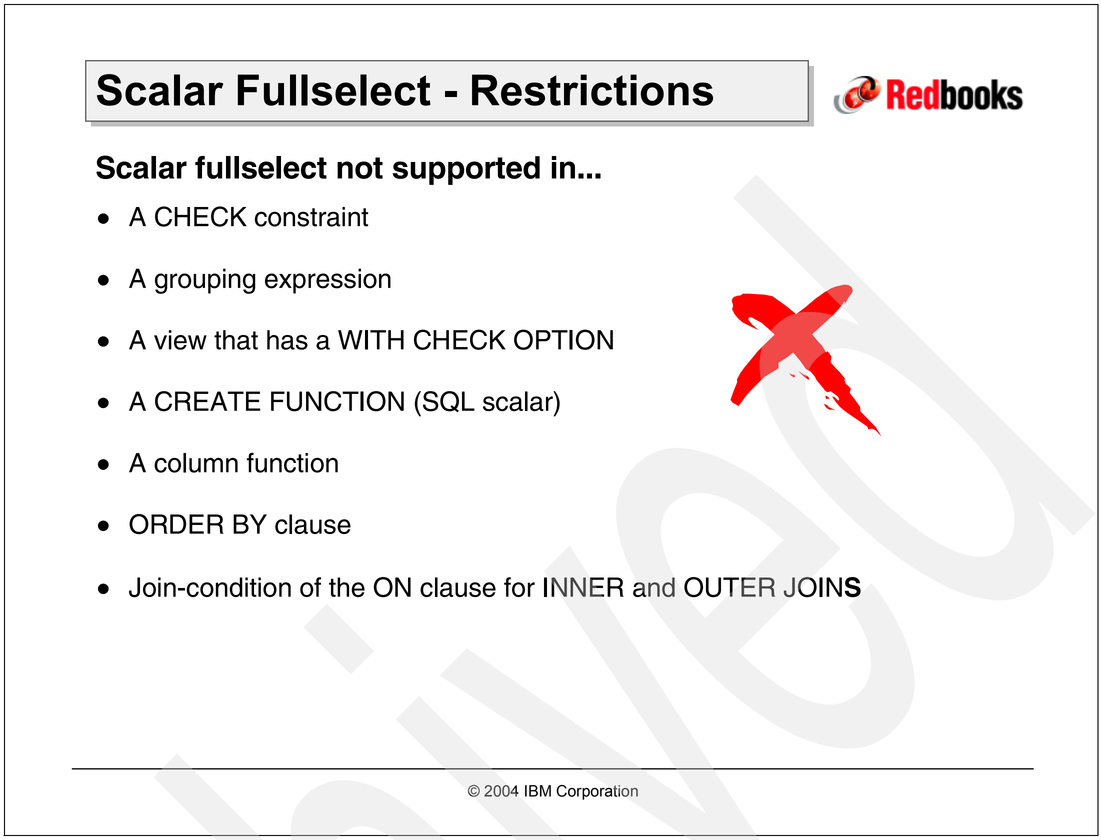
```
CREATE TABLE NEW_PARTPRICE LIKE PARTPRICE;
INSERT INTO  NEW_PARTPRICE SELECT * FROM PARTPRICE;
UPDATE NEW_PARTPRICE N SET PRICE =
  CASE
    WHEN( (SELECT ONHAND# FROM INVENTORY WHERE PART=N.PART) < 7)
      THEN 1.1 * PRICE
    WHEN( (SELECT ONHAND# FROM INVENTORY WHERE PART=N.PART) > 20)
      THEN  .8 * PRICE
    ELSE PRICE
  END;

SELECT * FROM NEW_PARTPRICE;
```

PART	PROD#	SUPPLIER	PRICE
WIRE	10	ACWF	3.50
OIL	160	WESTERN_CHEM	1.20
MAGNETS	10	BATEMAN	65.45
PLASTIC	30	PLASTIC_CORP	2.20
BLADES	205	ACE_STEEL	8.90

4.14.6 Scalar fullselects in CASE expression of UPDATE

Here the scalar select participates as part of the search condition of the WHEN clause.

Note that the two SELECT statements make use of the correlation variable N to refer to the table NEW_PARTPRICE.



4.14.7 Scalar fullselect - restrictions

A scalar fullselect cannot be used in the following instances:

- A CHECK constraint in CREATE TABLE and ALTER TABLE statements.
- A grouping expression, which is limited to a list of column names. Even though you can now (in V8) code an expression in a GROUP BY (see 4.17, "GROUP BY expression" on page 364), you cannot specify a scalar fullselect in a grouping expression.
- A view definition that has a WITH CHECK OPTION.
- A CREATE FUNCTION (SQL) statement (subselect already restricted from the expression in the RETURN clause).
- An aggregate function.
- An ORDER BY clause.
- A join-condition of the ON clause for INNER and OUTER JOINs.

Multiple DISTINCT Clauses

What is it? .....

- Allows more than one DISTINCT keyword on the SELECT or HAVING clause for a query

Benefits .....

- Enhances usability and power of SQL
- DB2 Family compatibility
- Previously you would get an SQLCODE -127

4.15 Multiple DISTINCT clauses

In general, DB2 V7 allows only one DISTINCT keyword on the SELECT or HAVING clause of any given query.

With DB2 V7 you can have multiple DISTINCT keywords on the same column only, as in:

```
SELECT COUNT(DISTINCT(A1)), SUM(DISTINCT A1) FROM T1
```

However, if you specify multiple DISTINCT keywords on different columns as shown below:

```
SELECT COUNT(DISTINCT A1), SUM(DISTINCT A2) FROM T1
```

In this case, you receive an SQLCODE -127 for the reason "DISTINCT IS SPECIFIED MORE THAN ONCE IN A SUBSELECT".

This restriction causes inefficient execution of a query when multiple distinct operations must be performed for multiple columns before any column functions, such as AVG, COUNT, and SUM are applied. Instead of one query retrieving multiple distinct column values, multiple queries must be executed in Version 7, causing performance degradation.

DB2 V8 removes this restriction, and allows more than one DISTINCT keyword on the SELECT clause or the HAVING clause for a query. This enhancement is accomplished by performing multiple sorts on multiple distinct columns. As a result, when multiple distinct column values need to be processed, only one query is required. This enhancement supports DB2 family compatibility.

Multiple DISTINCT Clauses - 2

Redbooks

Prior to Version 8 .....

- SELECT DISTINCT C1, C2 FROM T1;
- SELECT COUNT(DISTINCT C1) FROM T1;
- SELECT C1, COUNT(DISTINCT C2) FROM T1 GROUP BY C1;
- SELECT COUNT(DISTINCT(C1)),SUM(DISTINCT C1)FROM T1; -- same col

With Version 8 .....

- SELECT DISTINCT COUNT(DISTINCT C1), SUM(DISTINCT C2) FROM T1;
- SELECT COUNT(DISTINCT C1), AVG(DISTINCT C2)
  FROM T1 GROUP BY C1;
- SELECT SUM(DISTINCT C1), COUNT(DISTINCT C1), AVG(DISTINCT C2)
  FROM T1 GROUP BY C1 HAVING SUM(DISTINCT C1) = 1;

Not Supported in Version 8 .....

- SELECT COUNT(**DISTINCT A1,A2**)
  FROM T1 GROUP BY A2;
- SELECT COUNT**(DISTINCT(A1,A2))**
  FROM T1 GROUP BY A2;

4.15.1 Multiple distinct clauses examples

The DISTINCT keyword can be used at the statement level, as in:

```
SELECT DISTINCT C1,C2,C3 FROM T1
```

Or, it can be used at the column level, as in:

```
SELECT AVG(C1),COUNT(DISTINCT C2) FROM T1
```

Executing a query with multiple distinct columns can be costly in terms of the number of sorts performed and work files created. Therefore, whenever possible, some optimization may be done to the query by eliminating unnecessary DISTINCT keywords if it is semantically correct to do so. For instance, the following two SELECT statements are semantically the same after DISTINCT is removed from the first statement:

```
SELECT DISTINCT COUNT(DISTINCT(A1)), COUNT(A2)
SELECT COUNT(DISTINCT(A1)), COUNT(A2)
```

The use of this enhancement in DB2 V8 is shown in the following list of queries:

```
SELECT DISTINCT COUNT(DISTINCT(A1)), COUNT(A2) FROM T1
SELECT DISTINCT SUM(DISTINCT(A1)), COUNT(A2) FROM T1
SELECT DISTINCT AVG(DISTINCT(A1)), COUNT(A2) FROM T1
SELECT COUNT(DISTINCT(A1)), COUNT(DISTINCT(A2)) FROM T1
SELECT COUNT(DISTINCT(A1)), AVG(DISTINCT(A2)) FROM T1
SELECT DISTINCT COUNT(DISTINCT(A1)), COUNT(A2) FROM T1 GROUP BY A3
SELECT COUNT(DISTINCT(A1)), SUM(DISTINCT(A2)) FROM T1 GROUP BY A3
SELECT COUNT(DISTINCT(A1)) FROM T1 HAVING AVG(DISTINCT(A4)) > 1
SELECT COUNT(DISTINCT(A1)) FROM T1 WHERE A3 > 0 GROUP BY A2 HAVING AVG(DISTINCT(A4)) > 1
```

If executed under DB2 V7, all the queries in the examples would result in SQLCODE -127.

INSERT within SELECT Statement

What is it? .....

- Users can automatically retrieve column values inserted in tables by DB2 such as:
  - Identity columns, sequence values
  - User-defined defaults, expressions
  - Columns modified by BEFORE INSERT triggers
  - ROWIDs

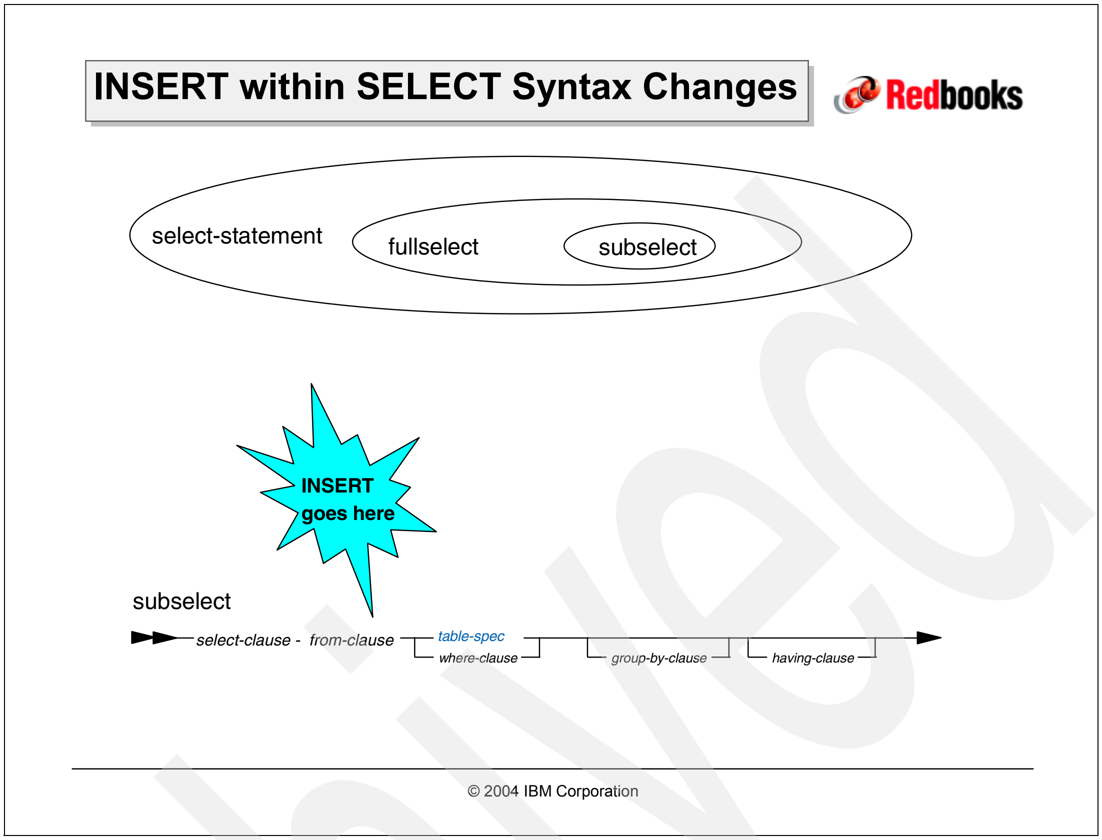
Benefits .....

- Enhances usability and power of SQL
- Cuts down on network cost in application programs
- Cuts down on procedural logic in stored procedures

4.16 INSERT within SELECT statement

Enhancements such as identity columns, ROWID columns, sequences, and triggers have resulted in more data being inserted into DB2 tables that is not directly inserted by applications. Instead, either DB2 or the trigger inserts the data. In cases like this, users need a way to immediately determine the values that have been inserted into a table for them.

INSERT within the SELECT statement provides this capability, enhancing the usability and power of SQL. The associated benefits include reduced network costs and simplified procedural logic in stored procedures.



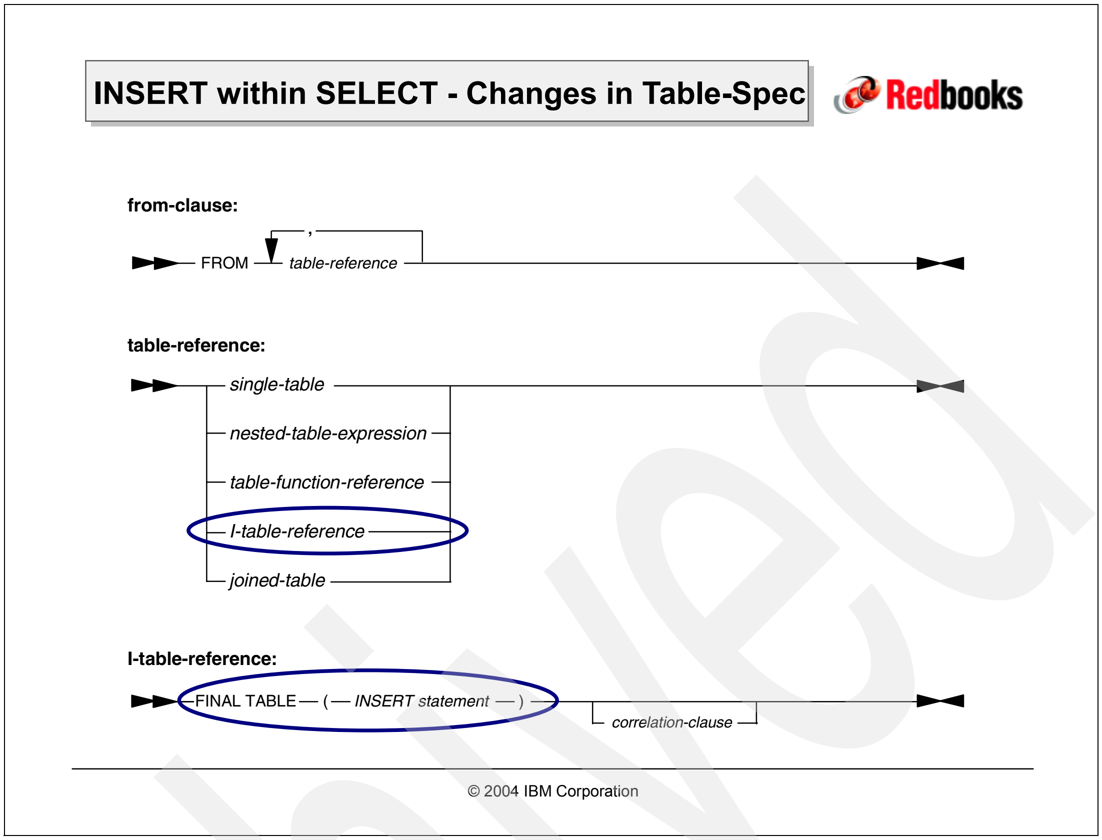
4.16.1 INSERT within SELECT syntax changes

The INSERT statement is now allowed in the FROM clause of:

- A SELECT statement that is a subselect
- A SELECT INTO statement

You should be aware of the following considerations:

- Authorization to INSERT and SELECT from the target object is required. That is, if a user only has INSERT privileges on a table, the user would not be able to use the INSERT within SELECT statement in a query to retrieve the rows that are inserted in the table.
- The INSERT statement can only appear in the FROM clause of the top-level SELECT statement; that is a subselect or a SELECT INTO statement.
- A fullselect in the INSERT statement cannot contain correlated references to columns outside the fullselect of the INSERT statement.
- If a table-spec includes an INSERT, then exactly one table-spec can be specified in the FROM clause. That is, joins are not allowed.
- The INSERT statement in a SELECT statement makes the cursor read-only. That is, you cannot use UPDATE WHERE CURRENT OF or DELETE WHERE CURRENT OF against the cursor.
- If the application declares a non-scrollable cursor and the same application performs a searched update or searched delete against the target object of the INSERT statement within the SELECT statement, the searched update or searched delete does not affect the result table rows of the cursor. For example, suppose that the application declares a cursor, opens the cursor, performs a fetch, updates the table, and then fetches additional rows. The fetches after the update statement always return those values that are determined at the time the cursor is opened.



4.16.2 INSERT within SELECT - changes in table-reference

The SELECT FROM INSERT statement enhancement provides the applications with the following facilities:

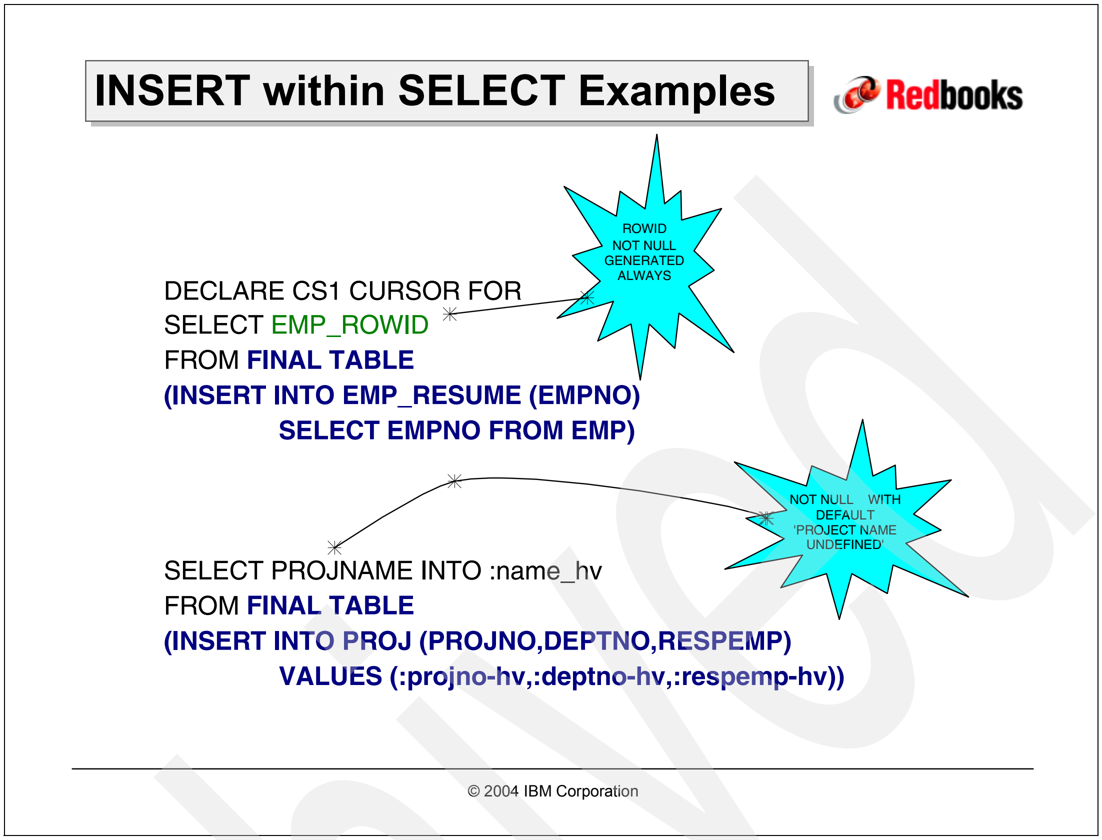
- Find the value of an automatically generated column.
- Retrieve default values for columns.
- Retrieve column values changed by a BEFORE INSERT trigger.
- Retrieve all values for an inserted row without specifying individual column names.
- Retrieve all values inserted through a multiple-row INSERT.

With the new syntax of having an INSERT statement on the SELECT statement, the rows inserted into the table are considered to be a result table. Therefore, all of the columns in this result table can be referenced by name in the select list of the query.

The keywords FINAL TABLE refer to the result table of the INSERT statement.

The result table contains all of the rows inserted and includes all of the columns requested in the SELECT list. Triggers, constraints, and DB2 generated values affect the result table in the following ways:

- If the INSERT activates a BEFORE trigger, the values in the result table include any changes that are made by the trigger. AFTER triggers cannot affect the values in the result table.
- DB2 enforces check constraints, unique index constraints, and referential integrity constraints before it generates the result table.
- The result table includes generated values for identity columns, ROWID columns, and columns based on expressions.



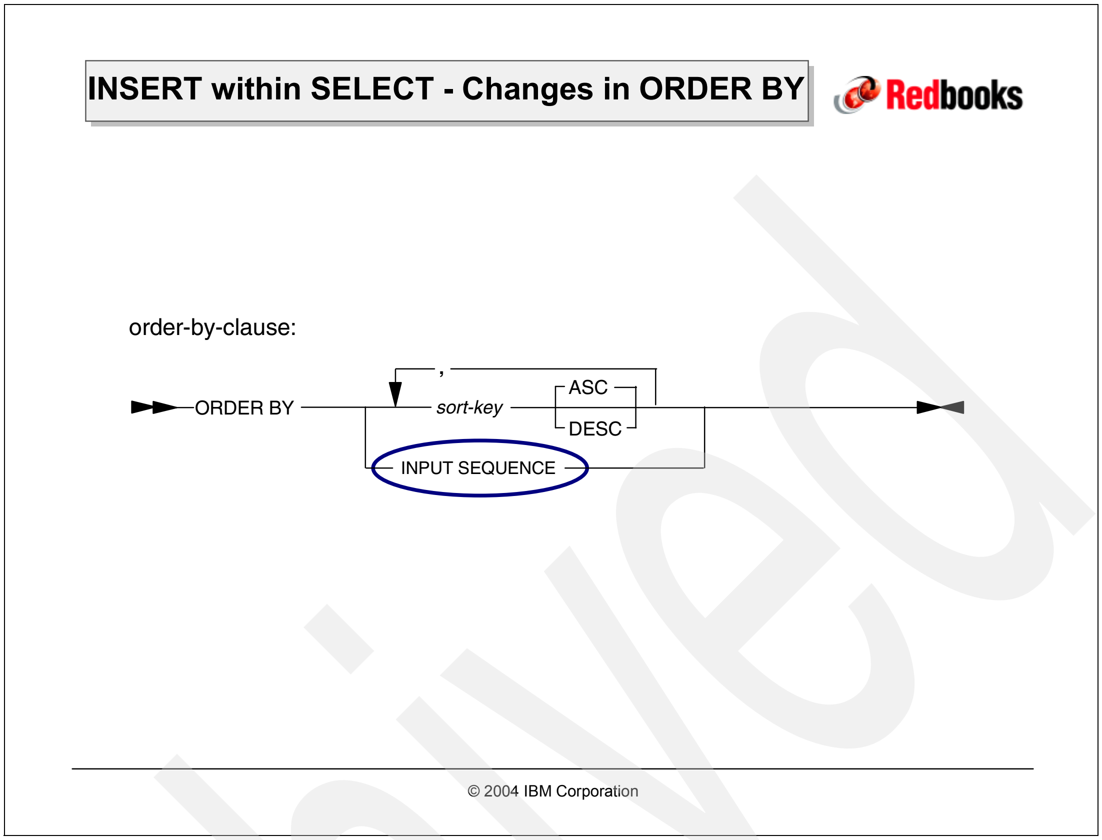
4.16.3 INSERT within SELECT examples

Retrieving the values for multiple rows being inserted could be done by having the INSERT statement within a SELECT statement as a subquery. You can use the result table of that subquery as input to the INSERT statement and then use the outer query to retrieve the rows that you inserted.

In the first example on the visual, column EMP_ROWID is defined as ROWID NOT NULL GENERATED ALWAYS in the EMP_RESUME table. In the program, corresponding to the EMP_ROWID column, the host variable is set up as USAGE SQL TYPE IS ROWID.

The second example on the visual uses the VALUES clause to insert one row using the values in the host variables into the PROJ table. Column PROJNAME is a NOT NULL character data type column with the default value of “PROJECT NAME UNDEFINED’.

The DECLARE CURSOR statement sets up the cursor using the new SQL construct. The program logic includes, as usual, the OPEN CURSOR statement followed by the FETCH to retrieve the ROWID value into the host variable set up for the purpose.

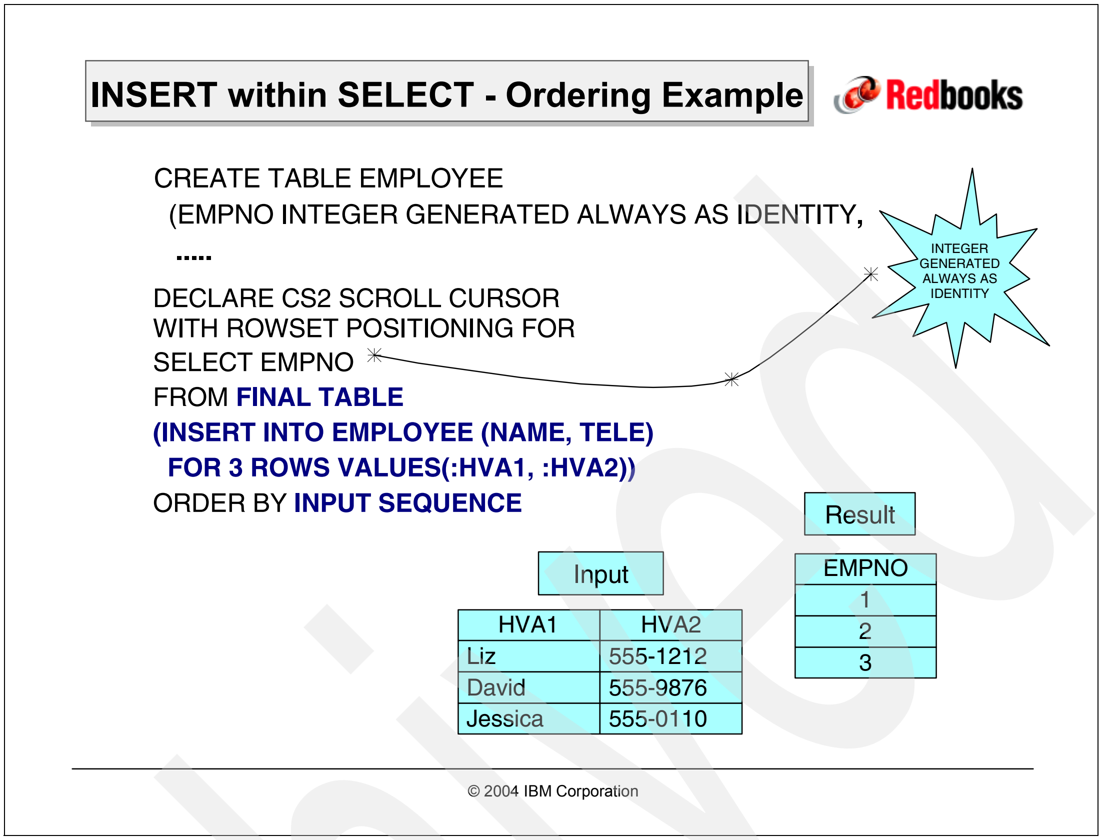


4.16.4 INSERT within SELECT - changes in ORDER BY

If there is a requirement in the applications to retrieve the rows in the same sequence as they are inserted, the application may use the INPUT SEQUENCE keywords with the ORDER BY clause of SELECT.

The INPUT SEQUENCE clause can only be specified if the table-spec is included in a SELECT statement that contains an INSERT statement.

The example on the next visual illustrates this situation.

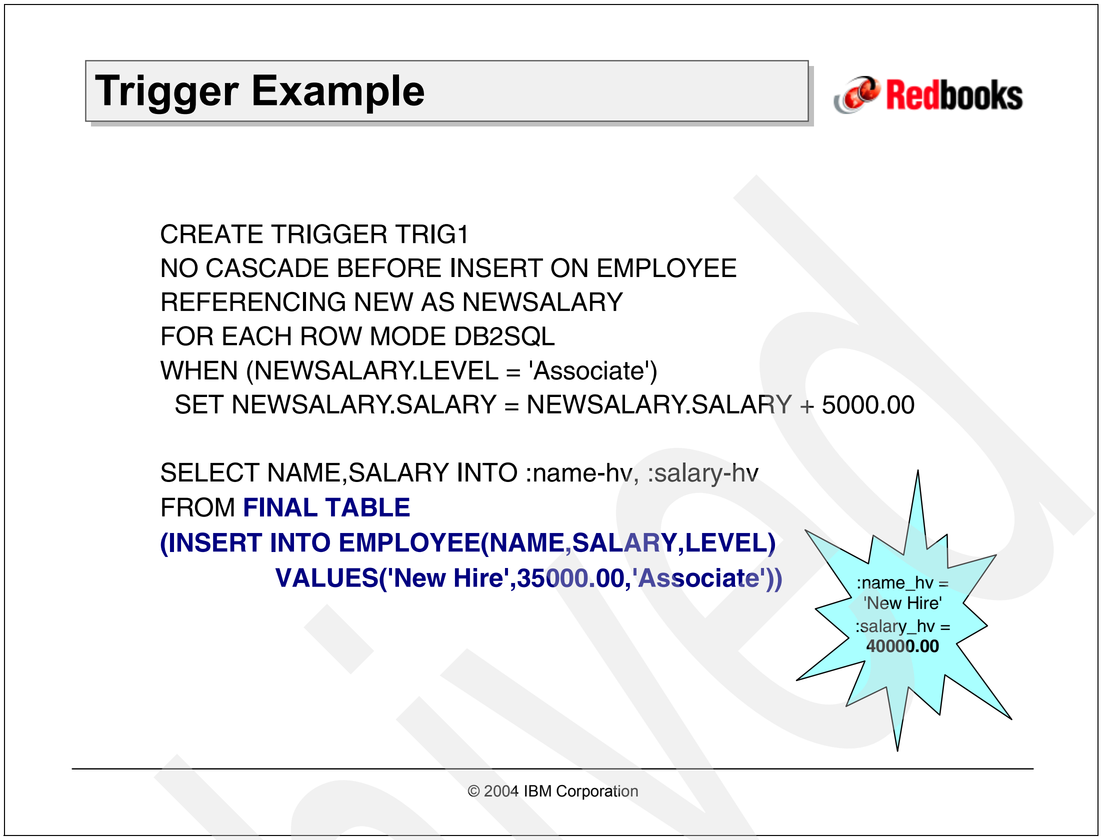


4.16.5 INSERT within SELECT - ordering example

The visual shows an example of a multi-row INSERT. HVA1 and HVA2 are host variable arrays each representing multiple rows, in this case 3 rows, with each entry for each array representing a single row.

The program logic includes the OPEN CURSOR CS2 statement followed by the FETCH FIRST ROWSET FROM CS2 FOR 3 ROWS INTO ... and FETCH NEXT ROWSET FROM CS2 FOR 3 ROWS INTO :HC1 statements.

The ORDER BY INPUT SEQUENCE ensures that the rows are retrieved in the sequence they are inserted.



4.16.6 Trigger example

For example, consider an EMPLOYEE table defined with columns EMPNO, NAME, SALARY, DEPTNO, TELE, and LEVEL. The column EMPNO holds integer data and is defined as GENERATED ALWAYS AS IDENTITY.

A BEFORE INSERT trigger is created on this table to give all new employees at level 'Associate' a $5000 salary raise.

The example on the visual shows the BEFORE INSERT trigger statement followed by the SELECT FROM INSERT statement.

Since the value specified for column LEVEL is 'Associate', the INSERT trigger is activated and the salary is raised by $5000 before the row is inserted. Thus, the SELECT statement returns a salary of $40000.00 for employee 'New Hire'.

What happens if the INSERT statement or the SELECT fails in the SELECT INTO statement? No row is inserted into the target table and so no row is returned.

What happens if the INSERT statement fails during OPEN CURSOR processing time? If any row being inserted fails, then all the rows that were successfully inserted before the failure occurred are undone, and the result table is empty.

Result table columns from the INSERT statement

The target of an INSERT statement can be either a table or a view. When the target is a table, the columns of the result table include all the columns of the target table. If the target is a view, the columns of the result table include all the columns of the target view. This is true even when the INSERT statement only assigns values for a subset of the columns of the target table or view.

For example, suppose you want to determine what value DB2 generates for the EMPNO column when you insert a row into EMPLOYEE table. Recall that EMPNO is defined to hold integer data and is defined as GENERATED ALWAYS AS IDENTITY. You can use the following SQL statements to achieve this:

```
SELECT EMPNO INTO :empno_hv
FROM FINAL TABLE
(INSERT INTO EMPLOYEE (NAME, SALARY, LEVEL)
VALUES('New Hire',35000.00,'Associate')
```

Using WITH HOLD Option

Using WITH HOLD option on cursor

- **Example:** User declares cursor, opens cursor, fetches 2 rows, performs commit, then fetches 3rd row successfully

```
DECLARE CS1 CURSOR WITH HOLD FOR
 SELECT EMP_ROWID FROM FINAL TABLE
   (INSERT INTO EMP_RESUME(EMPNO)
    SELECT EMPNO FROM EMP)
OPEN CS1                      <------------------- inserts 5 rows
FETCH  CS1 INTO :hv1          <------------------  receives row id for 1st row
FETCH  CS1 INTO :hv2          <------------------  receives row id for 2nd row
COMMIT                        <------------------  all 5 inserts are committed
FETCH CS1 INTO :hv3           <-----------------   receives row id for 3rd row
```


4.16.7 Using WITH HOLD option

In the example on the visual, assume that when the cursor is opened, five rows are inserted. The first two FETCHEs retrieve the first two rows. COMMIT causes all the five rows inserted to be committed. However, since the cursor is declared as WITH HOLD, the cursor is still positioned at the second row. When the next FETCH is executed, the third row is retrieved.

Using SAVEPOINT and ROLLBACK

Redbooks

If application sets a savepoint prior to opening cursor and then rolls back to that savepoint, all inserts are undone

Example:

```
DECLARE CS2 CURSOR FOR
 SELECT EMP_ROWID FROM FINAL TABLE
  (INSERT INTO EMP_RESUME(EMPNO)
  SELECT EMPNO FROM  EMP)
SAVEPOINT A ON ROLLBACK RETAIN CURSORS  <--- sets 1st savepoint
OPEN CS2                                <--- inserts the rows
SAVEPOINT B ON ROLLBACK RETAIN CURSORS  <--- sets 2nd savepoint
...
ROLLBACK TO SAVEPOINT B                  <--- rows still exist in table
                                              EMP_RESUME
ROLLBACK TO SAVEPOINT A                  <--- all inserted rows are undone
```


4.16.8 Using SAVEPOINT and ROLLBACK

If an application sets a savepoint prior to opening the cursor and then rolls back to that savepoint, all of the inserts are undone. The visual shows an example of this situation.

Some Considerations

FETCH FIRST clause

- Does not affect which rows are inserted
- All of the rows from the INSERT statement will be inserted into the target object
- The result table will only contain those rows for which the FETCH FIRST clause satisfies

DECLARE CURSOR

- The cursor is always read-only

OPEN CURSOR

- SQLERRD3 is set to reflect the effects of the INSERT statement (number of rows inserted)

4.16.9 Some considerations

If the INSERT is used within the SELECT statement of a cursor and the cursor definition contains the FETCH FIRST clause, all of the rows from the INSERT statement are inserted, and the outer SELECT result table contains only those rows for which the FETCH FIRST clause satisfies.

The INSERT statement may be defined within the SELECT statement of a scrollable cursor. If the cursor is defined as ASENSITIVE, a warning is returned indicating that it is being treated as INSENSITIVE. If SENSITIVE DYNAMIC or SENSITIVE STATIC is specified, an error is returned. This is because the result table is generated at OPEN CURSOR time and no further changes are reflected in the result table.

When a cursor associated with an INSERT within SELECT statement is opened, after executing the INSERT INTO statement that contains the SELECT statement, the SQLERRD3 field in SQLCA is set to indicate the number of rows inserted into the target table of the INSERT statement.

GROUP BY Expression

EMPNO	LASTNAME	WORKDEPT	SALARY	HIREDATE
000010	HAAS	A00	52750.00	1965-01-01
000030	KWAN	C01	38250.00	1975-04-05
000120	O'CONNELL	A00	29250.00	1963-12-05
000130	QUINTANA	C01	23800.00	1971-07-28
000140	NICHOLLS	C01	28420.00	1976-12-15

EMPLOYEE

```
SELECT     SUBSTR(CHAR(HIREDATE,ISO),1,3)
             CONCAT '0 - 9' AS HIREDECADE,
           MIN(SALARY) AS MINIMUM_SALARY
FROM       EMPLOYEE
GROUP BY   SUBSTR(CHAR(HIREDATE,ISO),1,3)CONCAT '0 - 9'
```

HIREDECADE	MINIMUM_SALARY
1960 - 9	29250.00
1970 - 9	23800.00

4.17 GROUP BY expression

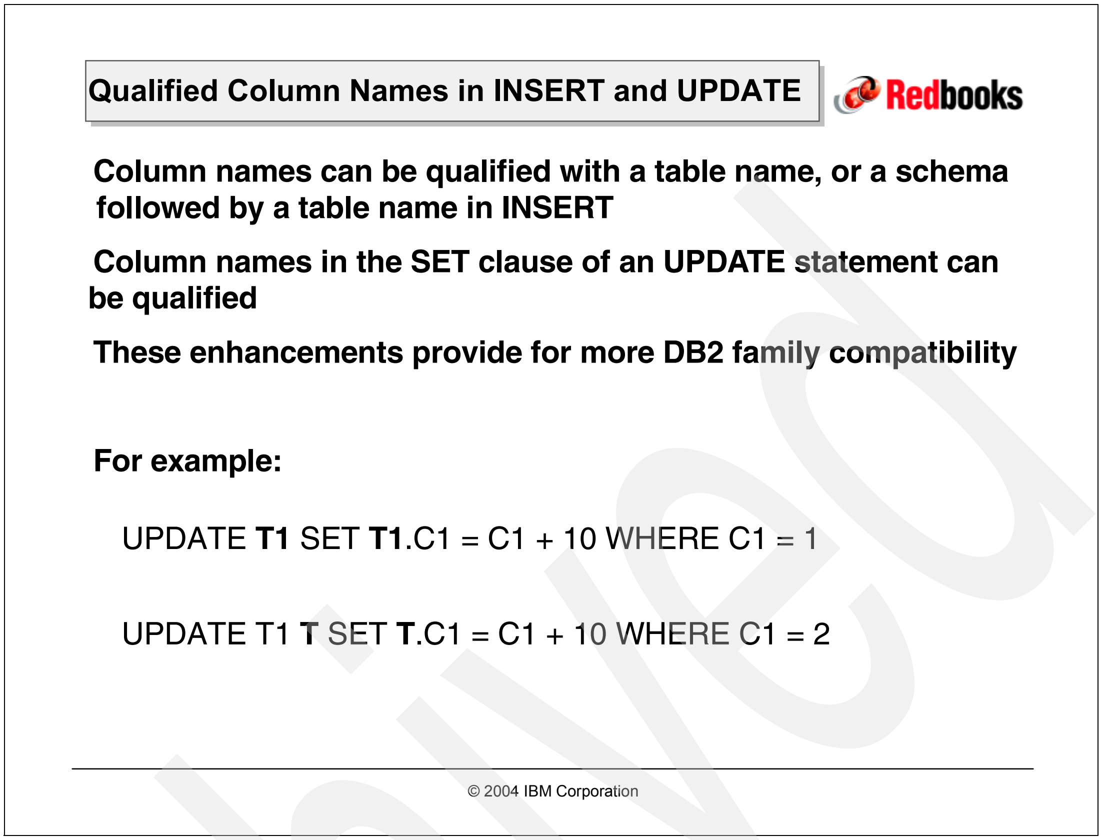
Use of expressions in the GROUP BY clause is possible in DB2 V8 and thus provides compatibility with the support existing in DB2 on the UNIX, Windows, and iSeries (AS/400®) platforms.

The example on the visual demonstrates the use of expression in GROUP BY. The requirement is to find the minimum salary paid to any employee during the decade when the employees were hired. The hiring decade is determined by extracting the first three digits from the ISO format of the hiring date and appending the character string '0 - 9'. For example, 1970 - 9 means that the employee was hired between January 1, 1970 and December 31, 1979.

Prior to DB2 V8, since use of expressions in the GROUP BY clause is not permitted, the query has to be written using the nested table expression as shown below:

```
SELECT S.HIREDECADE, MIN(S.SALARY) AS MINIMUM_SALARY
FROM
     (SELECT SUBSTR(CHAR(HIREDATE,ISO),1,3) CONCAT ‘0-9’ AS HIREDECADE, SALARY
      FROM   EMPLOYEE
     ) AS S
GROUP BY S.HIREDECADE
```

The alternative to using a nested table expression is to create a view to first provide a result table with the expression as a column of the result, then specify the column in a GROUP BY clause.



4.18 Qualified column names in INSERT and UPDATE

DB2 V8 allows column names to be qualified with a table name, or a schema followed by a table name in INSERT statements. DB2 V8 also allows column names in the SET clause of an UPDATE statement to be qualified. This facilitates application portability as DB2 on UNIX and Windows platforms provide for this support.

Consider the following examples:

- A correlation name is not specified for T1 (table or view), and T1 is used as the qualifier. This is allowed:

```
UPDATE T1 SET T1.C1 = C1 + 10 WHERE C1 = 1
```

- A correlation name 'T' is specified for T1, and it is used to qualify the column name. This is allowed:

```
UPDATE T1 T SET T.C1 = C1 + 10 WHERE C1 = 2
```

- A correlation name 'T' is specified for T1, but it is not used to qualify the column name. Instead, T1 is used as the qualifier, but it is not exposed because of the correlation name. This results in SQLCODE -206 being returned:

```
UPDATE T1 T SET T1.C1 = C1 + 10 WHERE C1 = 3
```

- A correlation name which is not specified is used as the qualifier for the column name. This results in SQLCODE -206 being returned:

```
UPDATE T1 SET T.C1 = C1 + 10 WHERE C1 = 4
```

- The table name following the UPDATE verb is qualified, and the qualifier is also used to qualify the column name. This is allowed:

```
UPDATE MY.T1 SET MY.T1.C1 = C1 + 10 WHERE C1 = 4
```

- The table name following the UPDATE verb is qualified, and the qualifier is not used to qualify the column name. This is allowed:

```
UPDATE MY.T1 SET T1.C1 = C1 + 10 WHERE C1 = 4
```

- The table name following the UPDATE verb is not qualified, but the implicit qualifier is used as an explicit qualifier for the column name. In this expression, myid is the auth id of the statement. This is allowed:

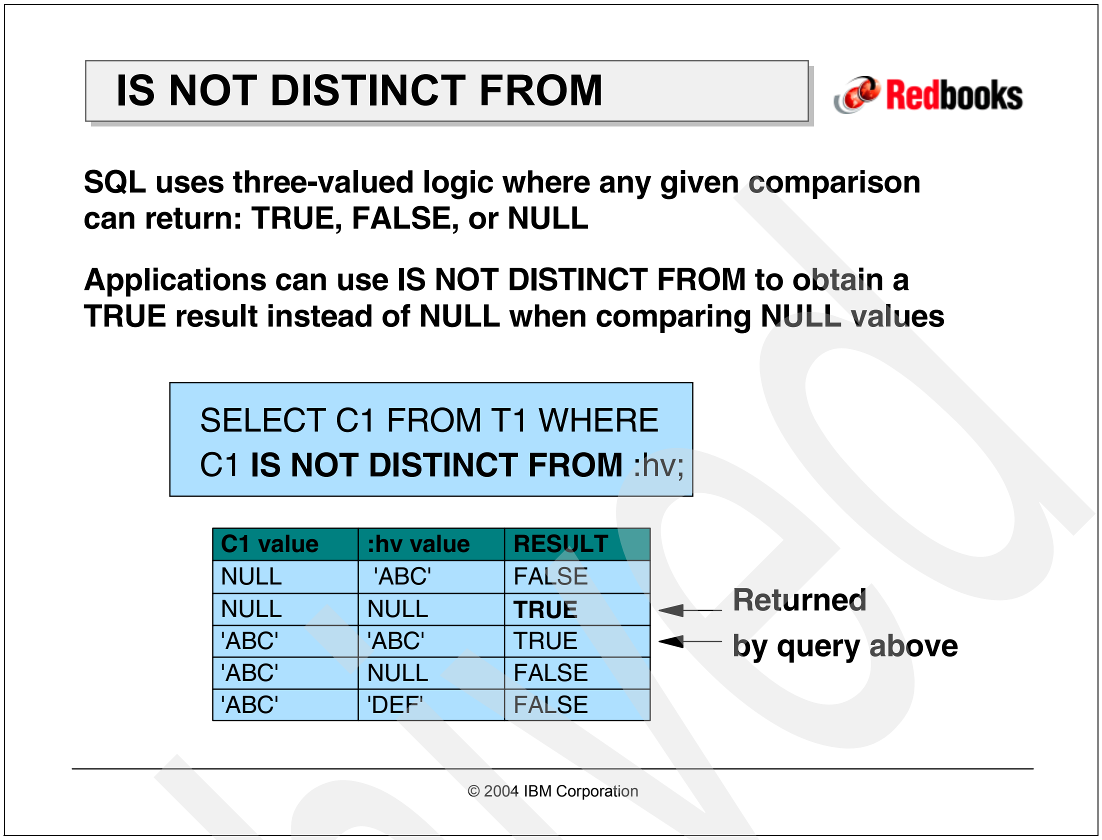
```
UPDATE T1 SET myid.T1.C1 = C1 + 10 WHERE C1 = 4
```

- The table name is qualified with the location, but the location is not used in qualifying the column name. This is allowed:

```
UPDATE SVL.MY.T1 SET T1.C1 = C1 + 10 WHERE C1 = 4
```

- The table name is qualified with the location, and the location is used in qualifying the column name:

```
UPDATE SVL.MY.T1 SET SVL.MY.T1.C1 = C1 + 10 WHERE C1 = 4
```



4.19 IS NOT DISTINCT FROM

By definition, a null value is unknown and this makes it unequal to all other values, including other null values. The only way to test for null values is to use the IS NULL predicate, as in "WHERE col IS NULL". A predicate of the form "WHERE col = :hv :nullind" never matches a null value in "col", even if the host variable "nullind" contains a null indicator. This is not intuitively obvious. For example: Assume that you are trying to select rows where the value of a column P1 is the null value. You code WHERE P1 IS NULL.

Taking it a step further, to compare two expressions to see if they are equivalent or both null, an application would currently have to use a compound search condition as follows:

```
( expr1 = expr2 ) OR ( expr1 IS NULL AND expr2 IS NULL )
```

For example, if you want to select rows for which the city value in one table is the same as the value in a host variable and null values are considered the same, you might code it like this:

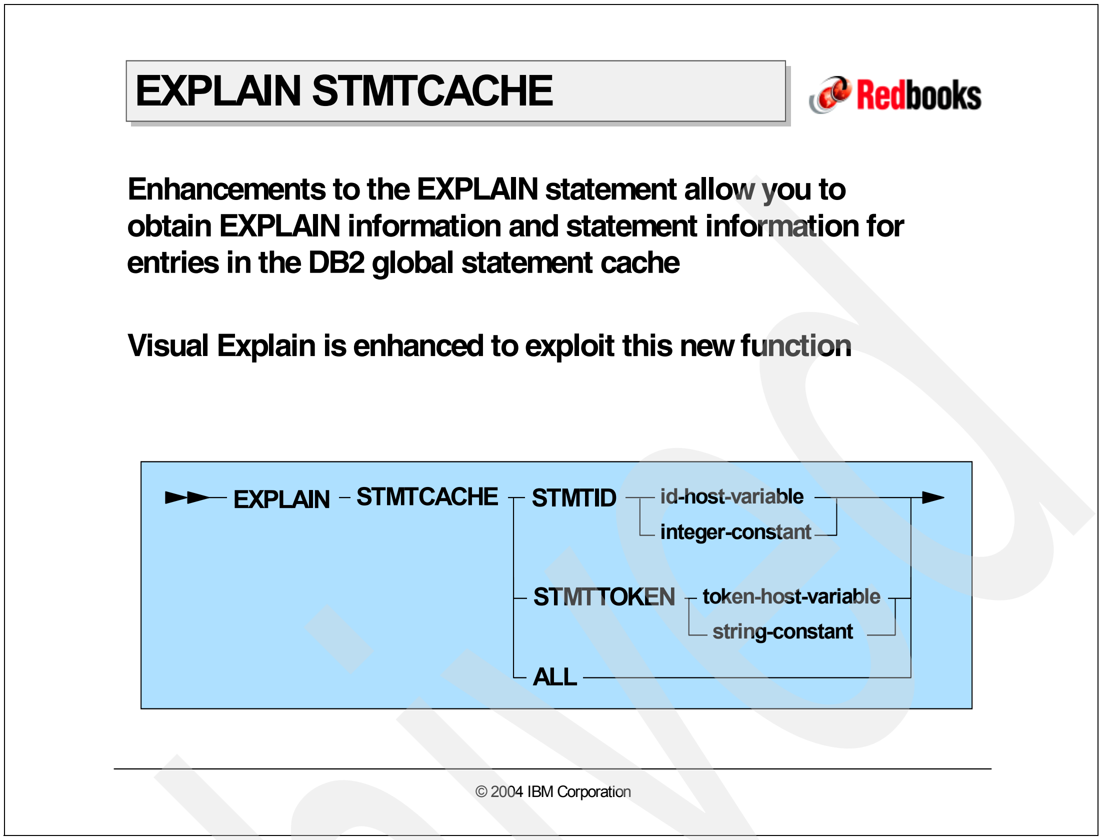
```
CITY = :CT :ctind
```

However, the search condition above would never be true when the value in :CT is null, even if the host variable "ctind" contains a null indicator. This is because one null value does not equal another null value. Instead of the simple predicate above, two predicates would need to be coded, one to handle the non-null values and another to handle the null values:

```
CITY = :CT OR ( CITY IS NULL AND :CT :ctind IS NULL )
```

With the introduction of the IS NOT DISTINCT FROM predicate, the search condition could be simplified as follows:

```
CITY IS NOT DISTINCT FROM :CT :ctind
```



4.20 EXPLAIN STMTCACHE

A new clause is added in the EXPLAIN statement in order to get the information about cached statements. The visual shows the syntax.

- STMTCACHE ALL

  Specifies that information about the cached statements that you are allowed to access in the global dynamic statement cache, is to be extracted. STMTCACHE ALL returns one row for each cached statement to the new DSN_STATEMENT_CACHE_TABLE. These rows contain identifying information about the statements in the cache, as well as statistics that reflect the execution of the statements by all processes that have executed the statement.

- STMTCACHE STMTID id-host-variable or integer-constant

  Specifies that the cached statement associated with the statement ID contained in host variable id-host- variable or specified by integer-constant is to be explained. The statement ID is an integer that uniquely determines a statement that has been cached in dynamic statement cache. The statement ID can be retrieved through IFI monitor facilities from IFCID 316 or 124 and is shown in some diagnostic IFCID trace records such as 172, 196, and 337.

  The column QUERYNO is given the value of the statement ID in every row inserted into the plan table, statement table, or function table by the EXPLAIN statement.

- STMTCACHE STMTTOKEN token-host-variable or string-constant

Specifies that all cached statements associated with a statement token contained in host variable token-host-variable or specified by string-constant are to be explained. The statement token must be a character string that is not longer than 240. This string is associated with the cached statement by the application program that originally prepares and inserts the statement into the cache. The application can do this by using the RRSAF SET_ID function, or by using the sqleseti API from a remotely-connected program.

For every row inserted into the plan table, statement table, or function table by the EXPLAIN statement, the column STMTTOKEN (which is newly added column) is given the value of the statement token, and the column QUERYNO is given the value of the statement ID for the cached statement with the statement token.

Since column QUERYNO in the plan table, statement table, or function table can now contain values of statement IDs, it may not be unique for each statement in the table. The user can tell if a row in the plan table, statement table, or function table is for a cached statement by checking the value of the PROGNAME column: If PROGNAME has the value "DSNDCACH", then the row is for a cached statement.

When EXPLAIN is executed, the statement with specified ID/token must still be in the cache. Otherwise, this EXPLAIN statement fails, returning SQLCODE -20248.

Important: When DB2 explains a statement from the statement cache, it writes the current access path that is used by that statement in the PLAN_TABLE. The statement is NOT going through access path selection again when the STMTCACHE option is used on the EXPLAIN statement (This is different from normal EXPLAIN processing of dynamic SQL statements). Going through access path selection again would defeat the purpose, as we try to find out what the *current access path* of a statement in the cache is. Determining the access path again may result in a different access path, as statistics for example may have changed in the meantime.

Authorization

When an EXPLAIN STMTCACHE statement is used to explain a cached statement, the application process must have the authority that is required to share the cached statement or the process must have SYSADM authority.

If an application process tries to explain a cached statement but it is not authorized to explain the cached statement, no row is added in the EXPLAIN tables.

Examples

1. Explain the cached statement with statement ID 124:

```
SID = 124;
EXEC SQL EXPLAIN STMTCACHE STMTID :SID;
```

2. Explain the cached statement with statement token 'SELECTEMP':

```
EXEC SQL EXPLAIN STMTCACHE STMTTOKEN 'SELECTEMP';
```

PLAN_TABLE changes

The PROGNAME column in the plan table (PLAN_TABLE), the statement table (DSN_STATEMNT_TABLE), and the function table (DSN_FUNCTION_TABLE), has a new value, "DSNDCACH", indicating that the row is the EXPLAIN result for a cached statement.

A new column is added in the plan table: STMTTOKEN VARCHAR(240). This new column contains the statement token that is associated with the statement that has been explained.

Old EXPLAIN tables (from V6 or V7) are still accepted by the new EXPLAIN STMTCACHE statements. But STMTTOKEN information is not available in this case.

Creating a statement cache table

The statement cache table must be created before the EXPLAIN statement with the STMTCACHE ALL keyword is executed. The statement cache table is called:

userid.DSN_STATEMENT_CACHE_TABLE

where *userid* is:

- The owner of the plan or package if the EXPLAIN statement is embedded in a plan or package.
- The SQL authorization ID of the process if the statement is dynamically prepared.

The DDL to create such a statement cache table is shown in Example 4-1.

Example 4-1 DSN_STATEMENT_CACHE_TABLE DDL

```
CREATE DATABASE DSNSTMTC;

 CREATE TABLESPACE DSNSUMTS IN DSNSTMTC;

 CREATE LOB TABLESPACE DSNLOBTS IN DSNSTMTC
  BUFFERPOOL BP32K1;

 CREATE TABLE DSN_STATEMENT_CACHE_TABLE
 (
 STMT_ID        INTEGER      NOT NULL,
 STMT_TOKEN     VARCHAR(240)         ,
 COLLID         VARCHAR(128) NOT NULL,
 PROGRAM_NAME   VARCHAR(128) NOT NULL,
 INV_DROPALT    CHAR(1)      NOT NULL,
 INV_REVOKE     CHAR(1)      NOT NULL,
 INV_LRU        CHAR(1)      NOT NULL,
 INV_RUNSTATS   CHAR(1)      NOT NULL,
 CACHED_TS      TIMESTAMP    NOT NULL,
 USERS          INTEGER      NOT NULL,
 COPIES         INTEGER      NOT NULL,
 LINES          INTEGER      NOT NULL,
 PRIMAUTH       VARCHAR(128) NOT NULL,
 CURSQLID       VARCHAR(128) NOT NULL,
 BIND_QUALIFIER VARCHAR(128) NOT NULL,
 BIND_ISO       CHAR(2)      NOT NULL,
 BIND_CDATA     CHAR(1)      NOT NULL,
 BIND_DYNRL     CHAR(1)      NOT NULL,
 BIND_DEGRE     CHAR(1)      NOT NULL,
 BIND_SQLRL     CHAR(1)      NOT NULL,
 BIND_CHOLD     CHAR(1)      NOT NULL,
 STAT_TS        TIMESTAMP    NOT NULL,
 STAT_EXEC      INTEGER      NOT NULL,
 STAT_GPAG      INTEGER      NOT NULL,
 STAT_SYNR      INTEGER      NOT NULL,
 STAT_WRIT      INTEGER      NOT NULL,
 STAT_EROW      INTEGER      NOT NULL,
 STAT_PROW      INTEGER      NOT NULL,
 STAT_SORT      INTEGER      NOT NULL,
 STAT_INDX      INTEGER      NOT NULL,
 STAT_RSCN      INTEGER      NOT NULL,
 STAT_PGRP      INTEGER      NOT NULL,
```

```
STAT_ELAP      FLOAT        NOT NULL,
STAT_CPU       FLOAT        NOT NULL,
STAT_SUS_SYNIO FLOAT        NOT NULL,
STAT_SUS_LOCK  FLOAT        NOT NULL,
STAT_SUS_SWIT  FLOAT        NOT NULL,
STAT_SUS_GLCK  FLOAT        NOT NULL,
STAT_SUS_OTHR  FLOAT        NOT NULL,
STAT_SUS_OTHW  FLOAT        NOT NULL,
STAT_RIDLIMT   INTEGER      NOT NULL,
STAT_RIDSTOR   INTEGER      NOT NULL,
EXPLAIN_TS     TIMESTAMP    NOT NULL,
SCHEMA         VARCHAR(128) NOT NULL,
STMT_TEXT      CLOB(2M)     NOT NULL,
STMT_ROWID     ROWID        NOT NULL  GENERATED ALWAYS
)
IN DSNSTMTC.DSNSUMTS CCSID EBCDIC;

CREATE TYPE 2 INDEX DSN_STATEMENT_CACHE_IDX1
       ON DSN_STATEMENT_CACHE_TABLE (STMT_ID ASC) ;

CREATE TYPE 2 INDEX DSN_STATEMENT_CACHE_IDX2
       ON DSN_STATEMENT_CACHE_TABLE (STMT_TOKEN ASC)
       CLUSTER;

CREATE TYPE 2 INDEX DSN_STATEMENT_CACHE_IDX3
       ON DSN_STATEMENT_CACHE_TABLE (EXPLAIN_TS DESC) ;

CREATE AUX TABLE DSN_STATEMENT_CACHE_AUX IN
       DSNSTMTC.DSNLOBTS
       STORES DSN_STATEMENT_CACHE_TABLE COLUMN STMT_TEXT;

CREATE TYPE 2 INDEX DSN_STATEMENT_CACHE_AUXINX
       ON DSN_STATEMENT_CACHE_AUX;
```

The contents of these rows show identifying information about the cache entries, as well as an accumulation of statistics reflecting the executions of the statements by all processes that have executed the statement. This information is nearly identical to the information returned from the IFI monitor READS API for IFCID 0316 and 0317. Note that the collection and reset of the statistics in these records is similarly controlled by starting and stopping of IFCID 318. For information about the contents of the dynamic statement cache and how to control the statistics gathered, see the IBM Redbook *DB2 for z/OS and OS/390 : Squeezing the Most Out of Dynamic SQL*, SG24-6418.
The EXPLAIN STMTCACHE ALL enhancement was introduced by APAR/PTF PQ88073/UQ89372.

Visual Explain exploits these enhancements.

READ ONLY USING UPDATE LOCKS

Redbooks

Allows WebSphere persistence layer to minimize network traffic when using searched update and pessimistic locking

```
PREPARE stmt 1 FROM
   "SELECT C1, C2, C3 FROM T1 WHERE C1=? AND C2=?
        FOR READ ONLY KEEP UPDATE LOCKS"
```


4.21 Read-only using update locks

Depending on the WebSphere deployment options, the persistence layer (which interacts with the database on behalf of an entity bean) currently uses ISOLATION(RS) to retrieve one or more rows with the FOR UPDATE KEEP UPDATE LOCKS clause when loading the WebSphere entity beans. The Java application is then allowed to perform updates on those beans, and the updates are subsequently sent to DB2 as searched UPDATE statements. WebSphere cannot do a positioned update because after the row is read in, the cursor is closed. WebSphere uses this approach to minimize the number of open cursors at runtime.

In Version 7, you can only specify the KEEP UPDATE (or EXCLUSIVE) LOCKS clause in combination with the FOR UPDATE clause. Specifying the FOR UPDATE clause causes DRDA to use a separate network flow for each operation (OPEN, FETCH,... CLOSE) because the cursor may appear in an UPDATE or DELETE WHERE CURRENT OF (positioned delete).

DB2 V8 is able to obtain exclusive locks with a FOR READ ONLY query. This allows the JDBC driver and DDF to use block fetch for the SELECT (eliminating the extra network messages required with a FOR UPDATE query), while still obtaining and holding the locks WebSphere needs for the searched UPDATE statement. This will provide significant CPU and elapsed time improvements.

Although your “hand-coded” application can immediately take advantage of this enhancement once you migrate to DB2 V8, at the time of writing, WebSphere has not yet implemented this enhancement for the persistence code they generate.

New / Enhanced Built-in Functions

New encryption related functions:
ENCRYPT_TDES: Encrypt a column in a table with a user-provided encryption password

- ENCRYPTION PASSWORD special register

DECRYPT_BIT, DECRYPT_CHAR, DECRYPT_DB

GETHINT: obtain hint to help remember the ENCRYPTION password

Generate unique values:
New GENERATE_UNIQUE creates a CHAR(13) FOR BIT DATA value, unique across a Sysplex

Obtaining the values of session variables:
New GETVARIABLE (discussed later in session variables)

New and enhanced character based string functions

Enhanced Unicode support for UPPER / LOWER / TRANSLATE built-in functions

4.22 New built-in functions

As in previous versions, DB2 Version 8 comes with a number of additional built-in functions. They are described in the following sections.

4.22.1 Encryption functions

Functions ENCRYPT_TDES (or ENCRYPT), DECRYPT_BIN, DECRYPT_CHAR, and GETHINT are added. The SET ENCRYPTION PASSWORD statement allows the application to specify a password.

These functions allow you to encrypt data at the column level. Because you can specify a different password for every row that you insert, you can really encrypt data at the “cell” level in your tables. Make sure to have a mechanism in place to manage the passwords that are used to encrypt the data. Without the password, there is absolutely no way to decrypt the data.

However, to facilitate remembering the password, you have an option to specify a hint (for the password) at the time you encrypt the data. Example 4-2 shows how to insert data that needs to be encrypted (a social security number in this case) using a password and a hint. The row is then retrieved, first using the wrong password. As you can see, the data is not readable. Then we obtain the hint that we used when we inserted the data to help us remember the real password, and then retrieve the row using the correct password.

The hint string is stored with the data in the column (ssn in the example). In addition to the length of the data that you are storing, encryption adds an additional 24 bytes; using a hint adds another 32 bytes. Then the result is rounded up to the next 8 byte boundary. Make sure to take this into account when specifying the length of your encrypted columns.

Example 4-2 Using encryption functions

```
CREATE TABLE EMP (SSN VARCHAR(124)FOR BIT DATA);

-- Insert a row providing a password and a hint

INSERT INTO EMP (SSN)
    VALUES(ENCRYPT('289-46-8832','TARZAN','? AND JANE'));

-- Selecting a row using the wrong password

SELECT DECRYPT_BIT(SSN,'MOWGLY') AS SSN FROM EMP;
---------+---------+---------+---------+---------+---------+-
SSN
---------+---------+---------+---------+---------+---------+-
ÔÂYN.ũ^\. .

-- Retrieving the hint for a row

SELECT GETHINT(SSN) AS HINT FROM EMP;
---------+---------+---------+---------+---------+---------+-
HINT
---------+---------+---------+---------+---------+---------+--
? AND JANE
---------+---------+---------+---------+---------+---------+--

-- Retrieving data using the correct password

SELECT DECRYPT_BIT(SSN,'TARZAN')AS SSN FROM EMP;
---------+---------+---------+---------+---------+---------+--
SSN
---------+---------+---------+---------+---------+---------+--
289-46-8832
```

Creating indexes on encrypted data is a good idea in some cases. Exact matches and joins of encrypted data (if both tables used the same encryption key to encrypt the same data) can use the indexes you create. Since encrypted data is essentially binary data, range checking of encrypted data will require table space scans. Range checking will require decrypting all the row values for a column, so it should be avoided, or at least tuned appropriately.

If an index exists on the ssn column in Example 4-3, DB2 would be able to use the index (provided that the optimizer decides to use it). In this case the provided string is encrypted and can then be evaluated against the index. As mentioned before, this only works for equal predicates, not range predicates; they require decryption before they can be evaluated.

Example 4-3 Index usage

```
set encryption password = a1b2c3';

select projectName
  from empProject
  where ssn = encrypt('480-93-7558');
```

For the example above to work, all rows in the ssn column must be encrypted using the same encryption key. In addition, SQL statements that use the encryption functions are considered multiple CCSID set statements. For more information, see 6.7, "Multiple CCSID sets per SQL statement" on page 546.

Encryption, by its nature, will slow down most SQL statements. If some care and discretion are used, the amount of extra overhead can be minimized. Encrypted data can have a significant impact on your database design. In general, you want to encrypt a few very sensitive data elements in a schema, like social security numbers, credit card numbers, patient names, etc.

Note: Built-in functions for encryption and decryption require cryptographic hardware in a cryptographic coprocessor, cryptographic accelerator, or cryptographic instructions. You must also have the z/OS Cryptographic Services Integrated Cryptographic Service Facility (ICSF) software installed.

4.22.2 Generating unique values

DB2 V8 brings yet another way for you to generate unique values. The GENERATE_UNIQUE built-in function gives you the ability to generate a unique value. These function is also supported in DB2 for Linux, UNIX and Windows V8, so this enhancement improves DB2 Family consistency.

The GENERATE_UNIQUE function returns a bit data character string (CHAR(13) FOR BIT DATA) that is unique compared to any other execution of the same function. The result of the function is a unique value that includes the internal form of the Universal Time, Coordinated (UTC) and, if in a sysplex environment, the sysplex member where the function was processed.

The result of this function can be used to provide unique values in a table. Each successive value will be greater than the previous value. The timestamp value that is part of the result of this function can be determined using the TIMESTAMP function with the result of GENERATE_UNIQUE as an argument. Example 4-4 gives an example of the usage of this built-in function.

This function differs from using the CURRENT TIMESTAMP special register in that a unique value is generated for each row of a multiple row insert statement or an insert statement with a fullselect.

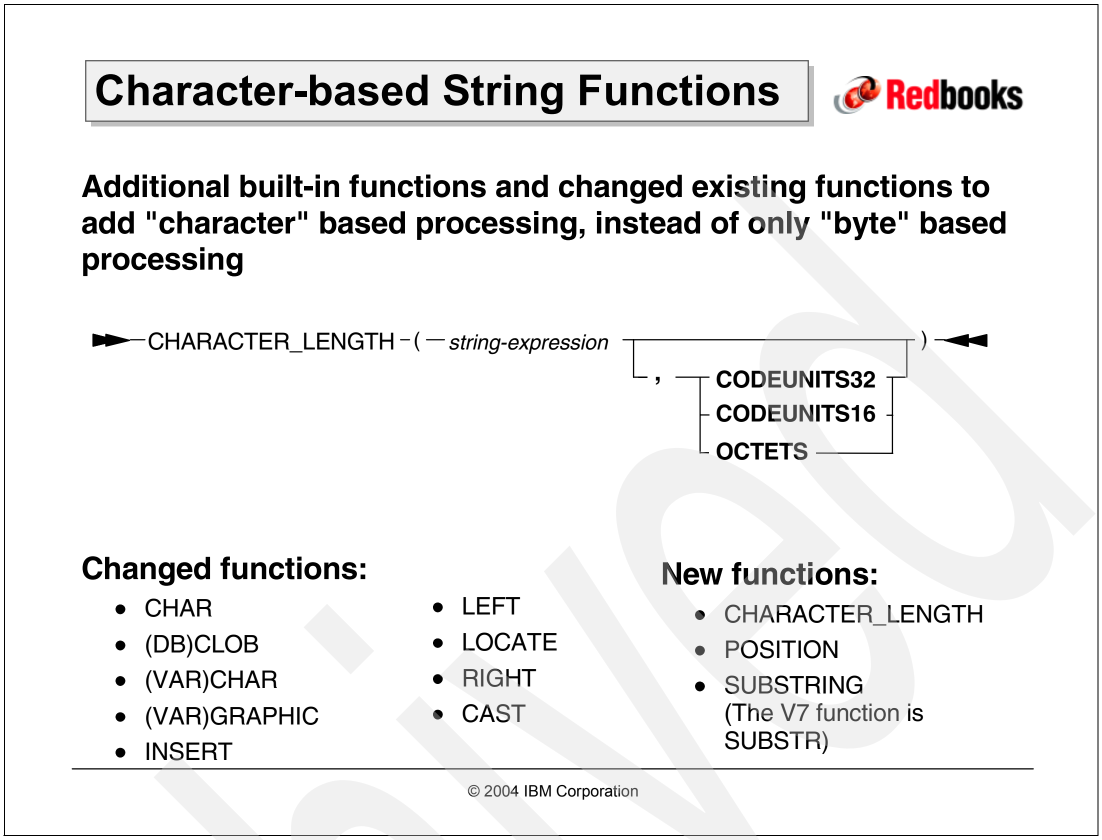
The GENERATE_UNIQUE function has been retrofitted back to DB2 Version 7 via the PTF for APAR PQ70901.

Example 4-4 Using GENERATE_UNIQUE()

```
-- Using GENEREAT_UNIQUE()
--   The result is unique but not very readable
SELECT GENERATE_UNIQUE() FROM SYSIBM.SYSDUMMY1;
---------+---------+---------+---------+---------+---------+--
.[ZE¾>ì.-....
DSNE610I NUMBER OF ROWS DISPLAYED IS 1
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100

-- Show the value in hex

SELECT HEX(GENERATE_UNIQUE()) FROM SYSIBM.SYSDUMMY1;
---------+---------+---------+---------+---------+---------+--
00BAE9C5B971D0712000010A02
DSNE610I NUMBER OF ROWS DISPLAYED IS 1
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100

-- Extract the timestamp value to make it more meaningful

SELECT TIMESTAMP(GENERATE_UNIQUE()) FROM SYSIBM.SYSDUMMY1;
---------+---------+---------+---------+---------+---------+--
2004-03-14-00.01.58.584731
```

4.22.3 Obtaining session variable information

DB2 Version 8 introduces a new type of variables called session variables. DB2 provides several built-in session variables that contain information about the server and application process, and you also have the ability to create user-defined session variables. The value of a (built-in or user-defined) session variable can be obtained by invoking the GETVARIABLE function with the name of the built-in session variable. Session variables are discussed in more detail in 4.23, “Session variables” on page 380.



4.22.4 Character-based string functions

Unicode data stored in DB2 can be multiple bytes of a single character. Actually, a single character in UTF-8 can be one, two, three or four bytes, depending on the character that you are trying to represent. Today's (V7) string functions are byte oriented. Therefore, using existing functions on multiple byte character set data can be challenging.

An example to illustrate this is shown in Example 4-5.

Example 4-5 Byte-based SUBSTR function

```
CREATE TABLE  TESTB (COLB CHAR(20) NOT NULL) CCSID UNICODE;

INSERT INTO TESTB VALUES(X'C2A7C2A7C2A7C2A7');

SELECT * FROM TESTB;
---------+---------+---------+---------+---------+---------+---------+-
COLB
---------+---------+---------+---------+---------+---------+---------+-
§§§§
SELECT SUBSTR(COLB,1,3) FROM TESTB;

DSNT408I SQLCODE = -331, ERROR:  A STRING CANNOT BE ASSIGNED TO A HOST VARIABLE
         BECAUSE IT CANNOT BE CONVERTED.  REASON 1, CHARACTER 1208, POSITION 37
DSNT418I SQLSTATE   = 22021 SQLSTATE RETURN CODE
DSNT415I SQLERRP    = DSNXROHB SQL PROCEDURE DETECTING ERROR
DSNT416I SQLERRD    = -117  0  0  -1  0  0 SQL DIAGNOSTIC INFORMATION
DSNT416I SQLERRD    = X'FFFFFF8B'  X'00000000'  X'00000000'  X'FFFFFFFF'
         X'00000000'  X'00000000' SQL DIAGNOSTIC INFORMATION
```

In our Unicode sample table we insert four “§” paragraph signs. To do so we use the Unicode hex representation of that character X'C2A7’. After inserting, we retrieve what we inserted, and all is well. Now, we want to use the SUBSTR function to retrieve the first there characters.

Unfortunately the SUBSTR function does not understand “characters”; it only understands “bytes”. That is OK in an EBCDIC or ASCII world where all characters have the same length, and bytes and characters mean the same thing. In UTF-8 that is not the case, hence the SQCLODE -331 because we tried to retrieve an incomplete character. SUBSTR(colb,1,3) retrieves the first three bytes, and ends up in the middle of a Unicode character.

DB2 Version 8 introduces a number of new character-based built-in functions, and enhances a number of existing built-in functions to allow for the specification of character-based processing.

The new functions are:

- CHARACTER_LENGTH
- POSITION
- SUBSTRING (note that in V7 only a SUBSTR function exists)

The changed existing functions are:

- (VAR)CHAR
- (DB)CLOB
- (VAR)GRAPHIC
- INSERT
- LEFT
- LOCATE
- RIGHT

All these functions allow for the specification of a “string measurement keyword”, or codeunits. You can specify:

CODEUNITS32 Working in CODEUNITS32 and CODEUNITS16 normally give the same result, unless surrogate characters are involved.

CODEUNITS16 When surrogate characters are involved, a surrogate will be counted as two characters (two byte UTF-16 code points CODEUNITS16).

OCTETS “Octet” is the French word for “byte”, so when working with the OCTETS option, we work in “byte-mode”, as in the past.

Note that there is no special option to work with UTF-8. Counting in CODEUNITS32, and counting UTF-8 multi-byte character set code points, both return the same result. Therefore you can use the CODEPOINTS32 option when dealing with UTF-8.

Note: It is good to know that UTF-8 data does NOT have to be converted to UTF-32 to count in CODEUNITS32. However, it does have to be converted to UTF-16 to count in CODEUNITS16.

Example 4-6 shows how to use the new SUBSTRING function to obtain the correct result on the same data that we used in Example 4-5.

Example 4-6 Using SUBSTRING with CODEUNITS32

```
SELECT SUBSTRING(COLB,1,3,CODEUNITS32) AS RESULT FROM TESTB;
---------+---------+---------+---------+---------+---------+---------+-
RESULT
---------+---------+---------+---------+---------+---------+---------+-
§§§
```

Note that the query now correctly returns the three requested paragraph signs.

The specification of CODEUNITS on a built-in function does not affect the result data type of the built-in function. However, the specification of CODEUNITS may affect the result as shown in the examples above.

Implicit in the specification of CODEUNITS32 or CODEUNITS16 is conversion to Unicode, if necessary, to evaluate the function. This is not the case when using OCTETS. In that case the evaluation of the built-in function is done in the encoding of the input string. If you want to evaluate a string in Unicode UTF-8 octets, and the string is not already Unicode UTF-8, you need to cast the string to Unicode explicitly.

The CAST function is also enhanced. It now also allows the specification of CODEUNITS32, CODEUNITS16, or OCTETS. The specification of this keyword indicates how DB2 is to count the resulting string. If CCSID and length are specified, the conversion to the correct CCSID is done first, before evaluation the length.

Assume an application needs to cast an EBCDIC string to Unicode UTF-8. The string contains the word Jürgen, which is 6 bytes in EBCDIC, but 7 bytes in Unicode UTF-8. To make sure the data is not truncated, we specify CODEUNITS32 in the length clause of the cast function:

```
SELECT CAST (‘Jürgen’ AS VARCHAR( 6 CODEUNITS32)) FROM SYSIBM.SYSDUMMY1;
```

4.22.5 Enhanced Unicode support for UPPER, LOWER and TRANSLATE

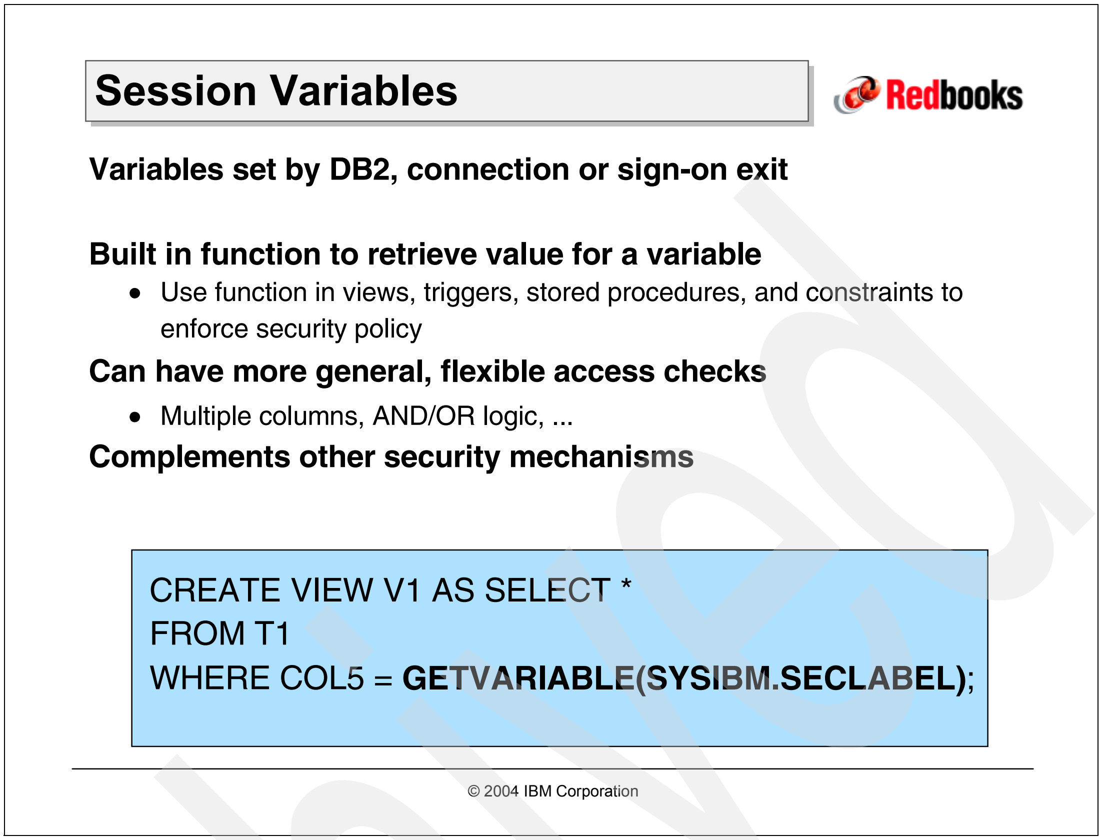
The UPPER, LOWER, and TRANSLATE built-in functions, as well as the LOCALE LC_CTYPE special register are being enhanced to allow DB2 to perform casing on Unicode characters according to the Unicode 4.0 standard. (The Unicode 4.0 standard has incorporated the former Unicode Technical Report 21, that treated case mappings.)

Prior to this enhancement, the UPPER and LOWER functions work with LE Locales for EBCDIC data, but not for Unicode data.

You must set the LOCALE LC_CTYPE special register to get the enhanced behavior.

For Unicode data, usage of the LOWER, UPPER, or TRANSLATE function can result in expansion if certain characters are processed. The user is responsible for ensuring that the result string is large enough to contain the result of the expression.

This enhancement was introduced by the PTF for APAR PQ88759.



4.23 Session variables

Session variables are similar to special registers in many respects. Session variables are an additional way to provide certain information to applications. Session variables can be referenced by SQL statements just like special registers. They can be used in view, triggers, stored procedures, or constraints to enforce security policies, for example.

You have to use the GETVARIABLE built-in function to retrieve the values of a session variable. Version 8 supports two types of session variables, which we now describe.

4.23.1 DB2-defined session variables

These session variables are defined by DB2 itself. They all use the SYSIBM schema name. The DB2-defined session variables are:

- SYSIBM.DATA_SHARING_GROUP_NAME
- SYSIBM.PACKAGE_NAME
- SYSIBM.PACKAGE_SCHEMA
- SYSIBM.PACKAGE_VERSION
- SYSIBM.PLAN_NAME
- SYSIBM.SECLABEL
- SYSIBM.SYSTEM_NAME
- SYSIBM.SYSTEM_ASCII_CCSID
- SYSIBM.SYSTEM_EBCDIC_CCSID
- SYSIBM.SYSTEM_UNICODE_CCSID
- SYSIBM.VERSION

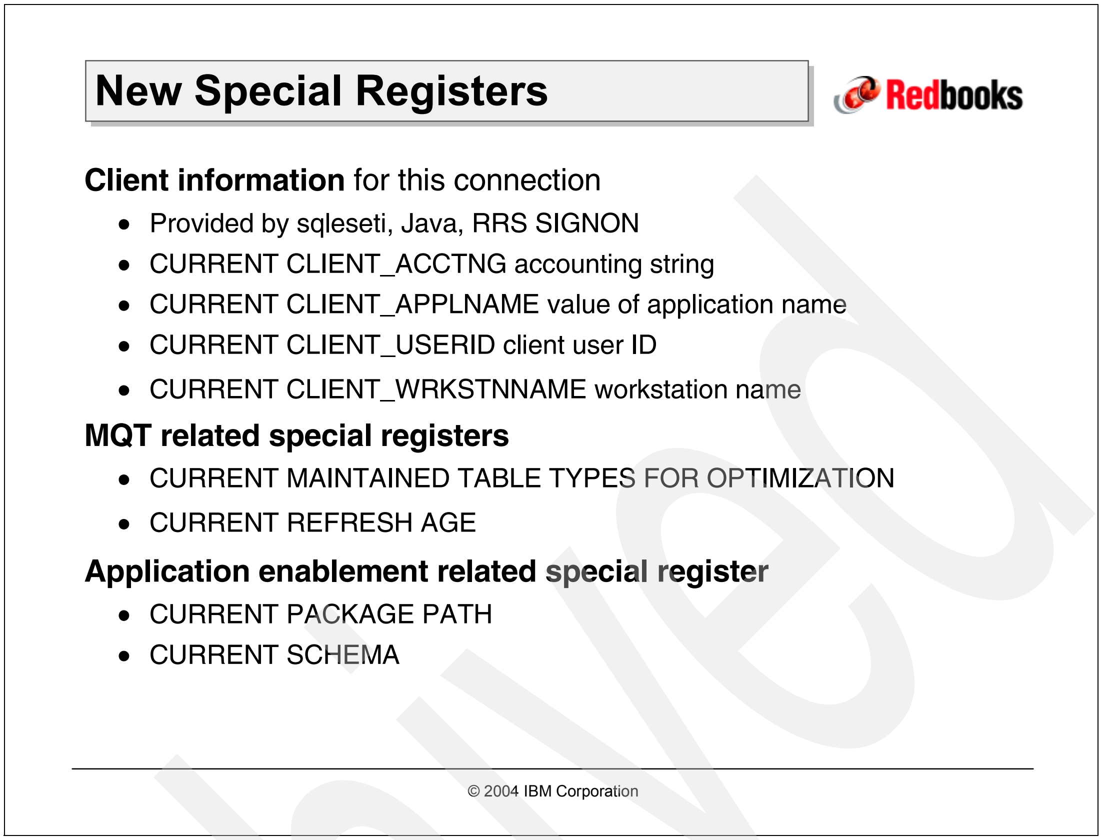
4.23.2 User-defined session variables

These session variables can defined by the installation itself. They all use the SESSION schema name.

You can establish up to 10 user-defined session variables in the connection and sign-on exit routines. To be able to do so, the parameter list passed to the connection and sign-on exit is extended with a third parameter (a 64-bit pointer) to a new DSECT called DSNDSVS (session variable structure). DSNDSVS has some control information, such as the number of session variables defined, and a pointer to the actual session variable array (mapped by DSNDSVA). In the session variable array, you define the session variables as follows:

- Unqualified session variable name length (2 bytes)
- Unqualified session variable name (128 bytes)
- Value length (2 bytes) of the next field
- Value you want to set the session variable to (255 bytes)

The unqualified session variable names and the values of the session variables have to be in Unicode UTF-8 (CCSID 1208) format.



4.24 New special registers

Four new special registers are added. These special registers are CURRENT CLIENT_ACCTNG, CURRENT CLIENT_APPLNAME, CURRENT CLIENT_USERID, and CURRENT CLIENT_WRKSTNNAME. The information is provided through a number of application programming interfaces.

These special registers were also added to DB2 for Linux, UNIX, and Windows V8, and so provide for DB2 family compatibility.

For more information about the CURRENT PACKAGE PATH special register, see 8.7, “CURRENT PACKAGE PATH special register” on page 624.

Another new special register is CURRENT SCHEMA. The CURRENT SCHEMA, or equivalently CURRENT_SCHEMA, special register specifies the schema name used to qualify unqualified database object references in dynamically prepared SQL statements. The data type is VARCHAR(128). The usage of CURRENT SCHEMA is described in more detail in 8.6, “SET [CURRENT] SCHEMA” on page 622.

Transparent ROWID

Redbooks

Eliminates the need to explicitly declare a ROWID column in tables that include LOBs

DB2 generates a "hidden" ROWID column, which is not visible on SELECT *

Simplifies porting of LOB applications from other platforms

4.25 Transparent rowid

DB2 currently requires a ROWID column to be included in tables which have a LOB column (one or more). In V8, DB2 will generate a “hidden” ROWID column if a ROWID column is not present. This column will not be included in a SELECT * (but you can select its content by specifying the name explicitly). These changes make porting of applications with LOBs simpler.

Example 4-7 illustrates this concept. As you can see, there is no ROWID column in the definition of the LOB_TEST table, even though it contains a LOB column (resume).

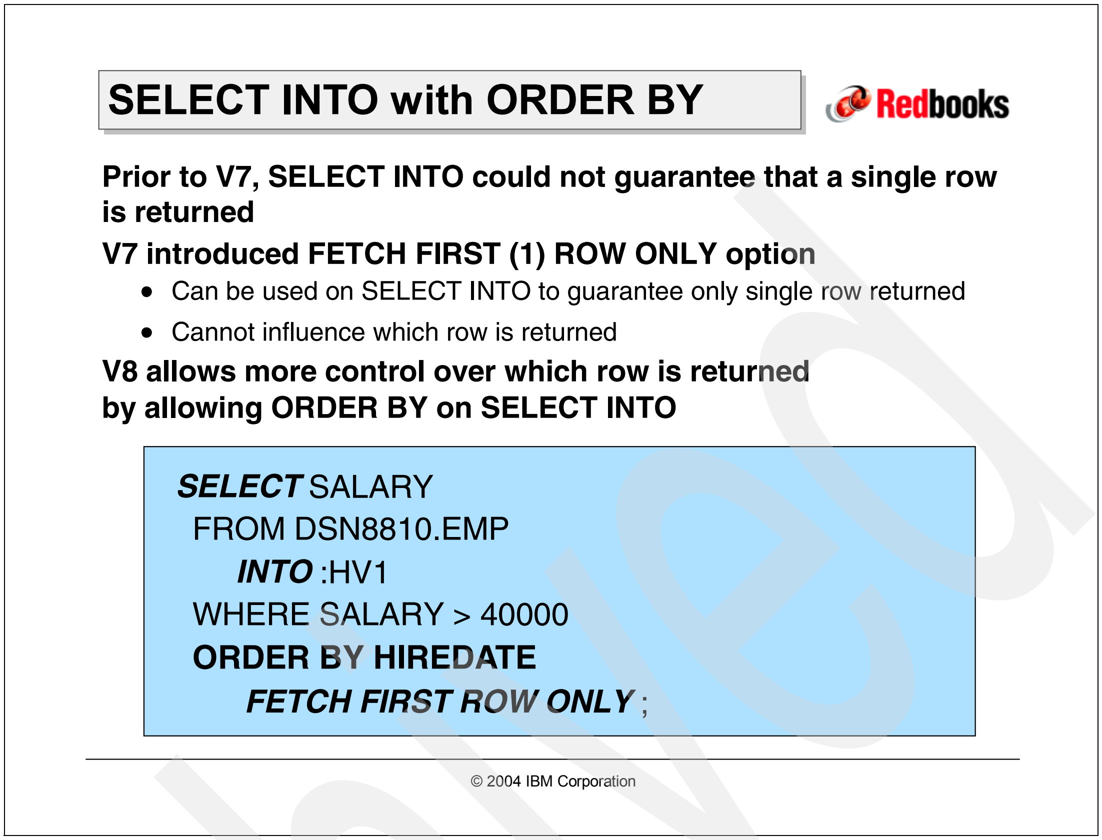
Example 4-7 Using hidden ROWID

```
CREATE DATABASE BSDBLOB ;
CREATE TABLESPACE BSTSLOB IN BSDBLOB;
CREATE TABLE  LOB_TEST
  (      EMPNO  CHAR( 06 )   NOT NULL,
         RESUME  CLOB(   1K ) )
            IN  BSDBLOB.BSTSLOB
         CCSID  EBCDIC;
CREATE LOB TABLESPACE  BSTSLOBC
                     IN  BSDBLOB
                    LOG  NO;
CREATE AUX TABLE  AUX_LOB_TEST
                     IN  BSDBLOB.BSTSLOBC
                 STORES  LOB_TEST
                 COLUMN  RESUME;
```

```
CREATE UNIQUE INDEX  XAUX_LOB_TEST
                     ON  AUX_LOB_TEST  ;
INSERT INTO LOB_TEST VALUES ('1234', 'MY RESUME IS NOT LONG');

SELECT * FROM LOB_TEST;
---------+---------+---------+---------+---------+---------+
EMPNO   RESUME
---------+---------+---------+---------+---------+---------+
1234    MY RESUME IS NOT LONG
---------+---------+---------+---------+---------+---------+

SELECT
     SUBSTR(NAME,1,30) AS NAME
    ,COLTYPE
    ,HIDDEN
   FROM SYSIBM.SYSCOLUMNS
   WHERE TBNAME = 'LOB_TEST';
---------+---------+---------+---------+---------+---------+
NAME                            COLTYPE   HIDDEN
---------+---------+---------+---------+---------+---------+
EMPNO                           CHAR      N
RESUME                          CLOB      N
DB2_GENERATED_ROWID_FOR_LOBS    ROWID     P

SELECT DB2_GENERATED_ROWID_FOR_LOBS , EMPNO, RESUME FROM LOB_TEST;
---------+---------+---------+---------+---------+----------------+---------+-
DB2_GENERATED_ROWID_FOR_LOBS                EMPNO   RESUME
---------+---------+-----------------+---------+---------+---------+---------+-
F9D986D90575D3162104015C5630010000000000201    1234    MY RESUME IS NOT LONG
```

Note also that when inserting, we do not specify a rowid column. In addition, when running a select *, only two columns are returned, even though the rowid column exists, as shown in the the select against SYSIBM.SYSCOLUMNS. The `DB2_GENERATED_ROWID_FOR_LOBS` column is marked as “partially hidden” HIDDEN =’P’. However, you can select this rowid column by explicitly coding the name in your select clause as shown in the last SQL statement, in case you need to.

The important thing here is that applications ported from another platform, for example, do not have to be changed to include a rowid column when inserting or retrieving data.



4.26 SELECT INTO with ORDER BY

The SELECT INTO statement must produce a result that contains a single row; else an SQLCODE -811 is returned. Prior to DB2 V7, there was no way of ensuring that only a single row could be returned when using a SELECT INTO statement. You had to use a cursor, and the program itself would have to read in only the first row that matched the predicates and throw away all other rows. With the addition of the FETCH FIRST n ROWS ONLY clause in V7, this situation can be avoided. V7 allows you to code the FETCH FIRST ROWS ONLY clause on a SELECT INTO statement to indicate that only one row is to be returned to the program, even if multiple rows match the WHERE criteria. However, you could not specify the ORDER BY clause to affect which row was returned.

With DB2 for z/OS Version 8, you can now specify the ORDER BY clause as well. When you use both the FETCH FIRST 1 ROW ONLY and ORDER BY clauses, the result set is retrieved and ordered first, and then the first row is returned. For example, using the sample employee table, for all employees with a salary of more than $40000, put the salary of the employee who has been employed the longest in host variable :HV1. This query can be coded as shown in Example 4-8.

Example 4-8 SELECT INTO using ORDER BY

```
SELECT SALARY
  FROM DSN8810.EMP
       INTO :HV1
  WHERE SALARY > 40000
  ORDER BY HIREDATE FETCH FIRST ROW ONLY;
```

This functionality is only available in New-function mode.

5

e-Business

List of Topics

IBM DB2 Universal Driver for SQLJ and JDBC

SQLJ support in the DB2 Universal Driver

z/OS Application Connectivity to DB2 for z/OS and OS/390

Integrated SQL/XML publishing functions

DB2 Multilevel security with/without row granularity

In this chapter we describe enhancements related to e-business applications. This general term covers a lot of different areas, and DB2 V8 includes many enhancements that can make e-business applications better.

The e-business world is very much a Java world, and DB2 makes great strides in this area with the introduction of a new JDBC (and SQLJ) driver for the DB2 Family; the so-called IBM DB2 Universal Driver for SQLJ and JDBC.

On the same note, SQLJ is very important to DB2 for z/OS, because it brings the static SQL model, that has brought us so much over the years, in terms of performance and security, to the Java world. With the tooling and run-time support in place, SQLJ becomes a very attractive alternative to JDBC on the mainframe platform.

In December 2003, IBM introduced a new no-charge feature of DB2 V7 (and V8) called z/OS Application Connectivity to DB2 for z/OS and OS/390. Its goal is to remove the prerequisite of having a DB2 on the same LPAR as WebSphere for z/OS (even when that DB2 only serves as a gateway to remote DB2 data) but it can do more than just that, as we describe in this chapter.

With XML becoming more and more prevalent in enterprises, especially when exchanging information between enterprises, DB2 Version 8 integrates so-called XML publishing functions into the DB2 engine. These functions are based on the emerging SQL/XML standard. This is the first step DB2 for z/OS takes to more tightly integrate XML into the database engine.

And last but not least, we describe the security enhancements in DB2 V8. With an ever more complex interconnected e-business society, the need for proper and more granular security mechanisms are a very valuable DBMS asset. DB2 V8 introduces multilevel security at the row level.

Universal Driver for SQLJ and JDBC

Summary of JDBC driver types

DB2 Universal Driver for SQLJ and JDBC

DB2 Universal Driver objectives

DB2 JDBC Universal Driver architecture

Benefits for DB2 UDB for z/OS

Migration to the Universal Driver

Supported platforms and connectivity

Licensing

Getting a connection

Java API enhancements

5.1 DB2 Universal Driver for SQLJ and JDBC

In this section we describe the DB2 UDB Universal Driver for SQLJ and JDBC.

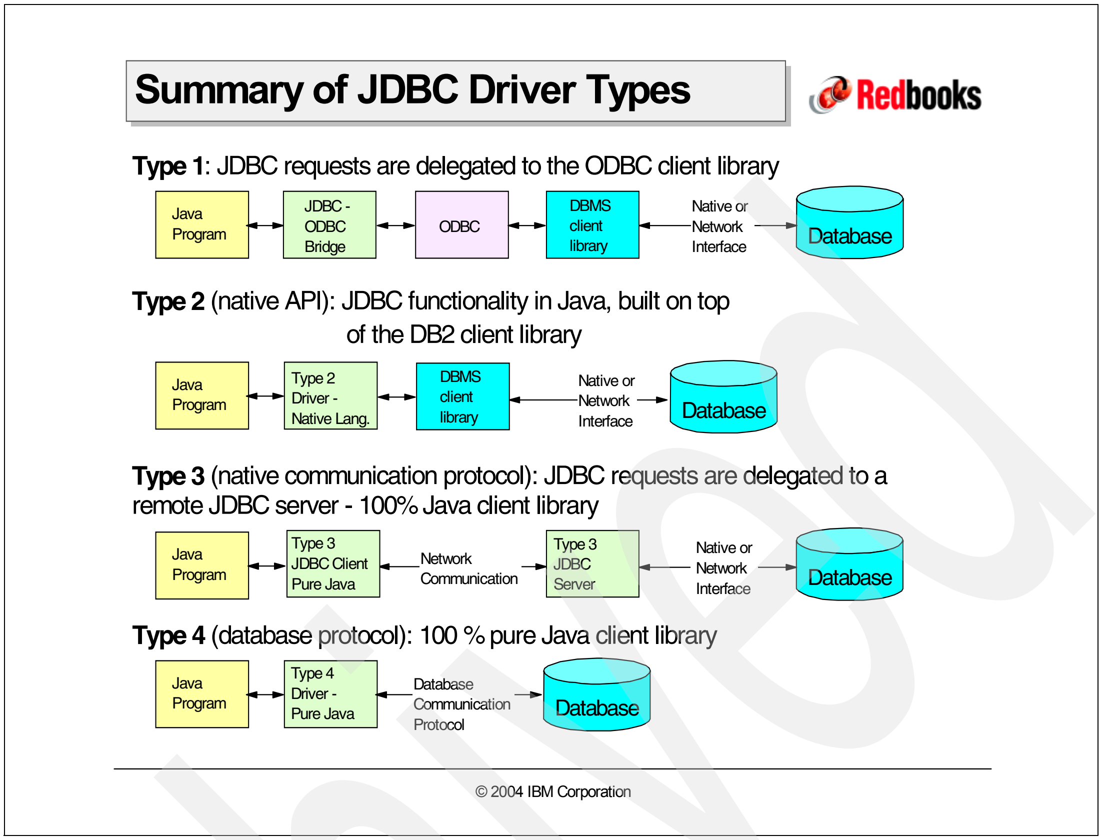
We start out with a short review of the different JDBC driver types defined in the JDBC standard.

Then we introduce the new DB2 Universal Driver for SQLJ and JDBC, why it is introduced, its architecture, and the benefits it brings to DB2 for z/OS.

We also describe the supported platforms, the licensing, and a few of the migration considerations.

Lastly, we describe some of the enhanced functionality that comes with the new DB2 Universal Driver.

For more information on JDBC and the DB2 Universal Driver, also see *DB2 for z/OS and OS/390: Ready for Java*, SG24 6435 and *DB2 Application Programming Guide and Reference for Java,* SC18-7414.



5.1.1 Summary of JDBC driver types

Before we dive into what is new and enhanced regarding JDBC and SQLJ support in DB2 V8, we provide you with an overview of the existing JDBC driver types based upon the JDBC 3.0 specification.

Type 1: Drivers that implement the JDBC API as a mapping to another data access API, such as ODBC. Drivers of this type are generally dependent on a native library, which limits their portability. The JDBC-ODBC bridge driver is an example of a Type 1 driver. This is usually a transition solution, and requires an ODBC driver to work.

Type 2: Drivers that are written partly in the Java programming language, and partly in native code. Part of the JDBC driver is implemented in Java and uses the Java Native Interface (JNI) to call a database specific API. Type 2 drivers use a native client library specific to the data source to which they connect. Because of the native code, their portability is limited.

Type 3: Drivers that use a pure Java client (100% Java) and communicate with a middleware server using a database independent protocol, often via TCP/IP socket calls. The middleware server then communicates the client's requests to the data source.

Type 4: Drivers that are pure Java and implement the network protocol for a specific data source. The client connects directly to the data source. In case of DB2, the DRDA protocol is used to talk directly to the data source.

Note: The number in the driver type has no meaning whatsoever in terms of capability. Do not assume that because 4 is greater than 2, that a Type 4 driver is better than a Type 2 driver. In fact, a Type 2 driver is almost certain to outperform a Type 3 or Type 4 driver, because it does not have to route through a network layer. Normally, a Type 2 driver is the best suitable driver from the point of view of performance and scalability.

IBM DB2 Universal Driver for SQLJ and JDBC

What is it ?

Architected as an abstract JDBC processor

- Independent of driver-type connectivity or target platform.
- As an abstract machine, driver types become connectivity types
- Architecture-neutral JDBC driver for:
  - Distributed DB2 access (Type 4 connectivity) and
  - Local DB2 access (Type 2 connectivity)

5.1.2 DB2 Universal Driver for SQLJ and JDBC

The clients (prior to DB2 V8) for Linux, UNIX, Windows and OS/2 platforms provide the basis for three distinct products: DB2 Run-Time Client, also known as the Client Application Enabler (CAE); the DB2 Application Development Client, formerly known as the Software Development Kit (SDK); and DB2 Connect Personal Edition. DB2 Connect Enterprise Edition is based on a combination of the UNIX, Windows, OS/2 engine infrastructure and DB2 Connect Personal Edition. Each product is positioned as either the client for a Linux, UNIX, or Windows application server, or the client for a z/OS database server.

Prior to V8, access to a Linux, UNIX, and Windows (LUW) server and a z/OS server used different database connection protocols, for example, DB2RA, DRDA, “net driver”. Each protocol defines a different set of methods to implement the same functions. To provide transparent access across the DB2 Family, the database connection protocols are now standardized.

All of them use the Open Group’s DRDA Version 3 standard, which provides an open, published architecture that enables communication between applications, application servers and database servers on platforms with the same or different hardware and software architectures. This new architecture is called the Universal Client. A deliverable of this new architecture for Java applications is the *IBM DB2 Universal Driver for SQLJ and JDBC*, also known as Java Common Connectivity (JCC).

The Universal Driver is architected as an abstract JDBC processor that is independent of driver-type connectivity or target platform. The IBM DB2 Universal Driver is an architecture-neutral JDBC driver for distributed and local DB2 access.

Since the Universal Driver has a unique architecture as an abstract JDBC state machine, it does not fall into the conventional driver type categories as described in 5.1.1, “Summary of JDBC driver types” on page 390.

For the Universal Driver as an abstract machine, driver types become connectivity types. This abstract JDBC machine architecture allows for both all-Java connectivity (Type 4) or Java Native Interface (JNI)-based connectivity (Type 2) in a single driver. A single Universal Driver instance is loaded by the driver manager for both Type 4 and Type 2 implementations. Type 2 and 4 connections may be made (simultaneously if desired) using this single driver instance.

DB2 Universal Driver Objectives

Provided functions of DB2 UDB for Linux, UNIX, and Windows and those for z/OS are the same

- Improves Family compatibility - true portability

Functionality enhancements for Type 2 and Type 4 drivers

- Fully compliant with JDBC 3.0 standard

Provides a full Java application development process for SQLJ

Ease of installation and deployment for Type 4 driver

- No dependencies on a runtime of DLL
- Installation merely a copy of a .jar and .zip file

Reduces the client footprint for Linux, Unix and Windows platforms

Delivers functional and performance enhancements quicker

Improves trace capabilities

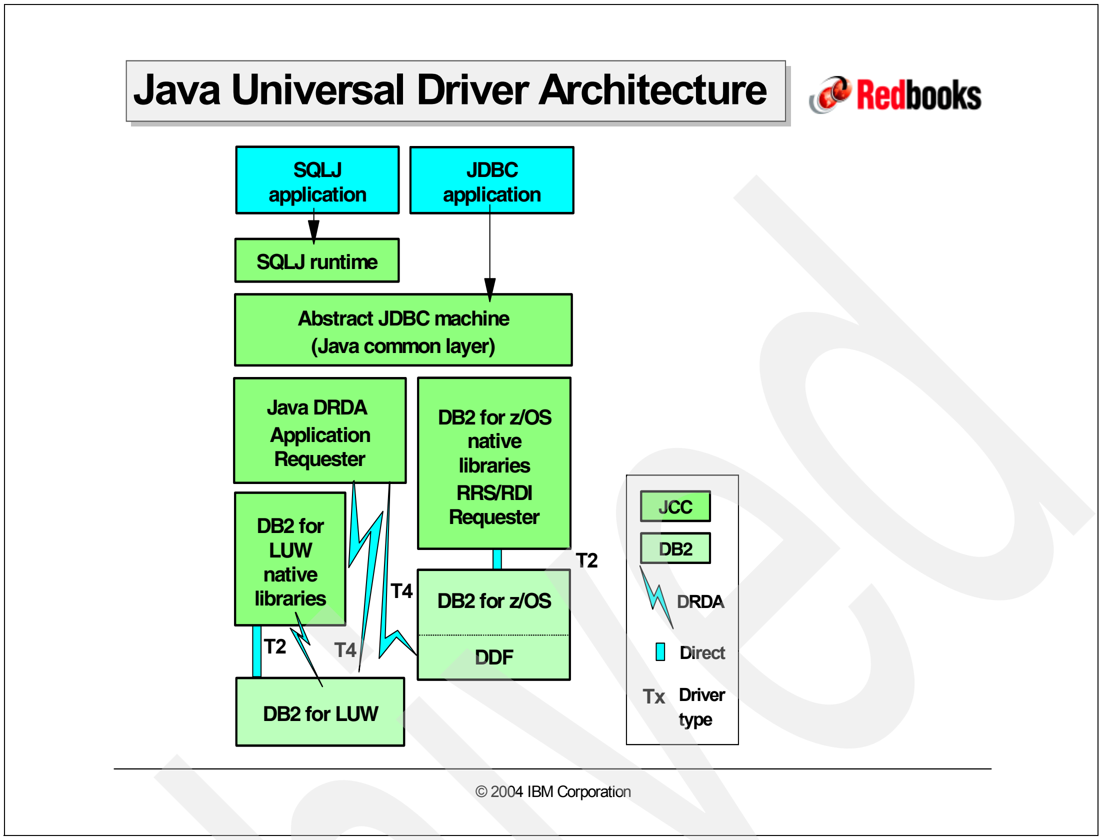
5.1.3 DB2 Universal Driver objectives

The new common runtime environment fulfills the following key requirements:

- Has a single Java driver with a common code base for Linux, UNIX, Windows and z/OS. The functions provided on DB2 UDB for Linux, UNIX, and Windows, and DB2 UDB for z/OS are the same, not just similar. This largely improves DB2 Family compatibility. For example, it enables users to develop on Linux, UNIX, and Windows, and deploy on z/OS without having to make any change, and eliminates the major cause of today's Java porting problems.
- Enhances the current API to provide a fully compliant JDBC 3.0 driver, for both a Type 2 and Type 4 JDBC driver. The functionality of the V7 (legacy) SQLJ/JDBC (Type 2) driver for z/OS will not be enhanced. JDBC 3.0 compliance will only by made available in the new Universal Driver. (However, the legacy driver will be shipped with DB2 for z/OS V8 for compatibility reasons.)
  DB2 for Linux, UNIX, and Windows V8 FixPak 3 was the first to ship with a full JDBC 3.0 compliant Universal Driver.
- Provides a full Java application development process for SQLJ, by:
  - Providing a fully portable customized SQLJ profile
  - Enabling the (remote) bind of DB2 packages from the client (using the Type 4 driver)
- Reduces the client footprint. Footprint reduction is achieved by eliminating the multiple layers of processing, which reduces both disk and memory consumption on the client. This mainly applies to DB2 for Linux, UNIX, and Windows (where the previous drivers were CLI based).

- Ease of installation and deployment. The Java type 4 driver is 100% Java code, without dependencies on a runtime or DLL. Installation is merely a copy operation of a .jar and .zip file. Deployment on z/OS can now be completely done from the workstation.
- Having a large common code base also enables IBM to deliver performance and functional enhancements quicker (since it only needs to be developed once).
- Trace improvements, by allowing:
  - Turning traces on and off dynamically
  - Allowing multiple levels of tracing, with different levels of detail
- With the elimination of the “private protocols” used by the Linux, UNIX and Windows clients in previous versions, using DRDA will render better performance.

Note: The DB2 Universal Driver for SQLJ and JDBC will be made available for DB2 for z/OS and OS/390 Version 7 through the PTF for APAR PQ80841 and its prerequisite PTFs.



5.1.4 DB2 JDBC Universal Driver architecture

Whereas before the DB2 Universal Driver, each DB2 client platform came with its own driver files for JDBC and SQLJ support, the new driver is a single set of Java Archive (.jar) and .zip files, which can be used on UNIX, Windows, and z/OS. It is a combined driver providing both JDBC Type 2 and Type 4 connectivity, depending on the URL used in the `getConnection()` call (or the underlying data source definition).

Of course, selecting Type 2 functionality requires that native code has been installed on the client.

Migration to the Universal Driver

New Java Universal Driver will eventually replace existing legacy implementations of DB2 JDBC drivers

Subtle behavioral differences from legacy drivers are to be expected

- Existing drivers do not have the same behavior
- Impossible to match JDBC behavior of LUW and z/OS simultaneously in all cases

Migration should be done gradually as there will be differences

db2sqljupgrade **utility for DB2 for z/OS serialized profiles**

5.1.5 Migration to the Universal Driver

The IBM DB2 JDBC Universal Driver is written from the ground up. It is an entirely new architecture, design, and implementation, and should not be viewed as a follow-on release of existing JDBC/CLI drivers, nor as the legacy type 2 driver for DB2 for OS/390 and z/OS. You should plan to migrate to the new Universal JDBC Driver gradually, as there may be subtle behavioral differences from the legacy drivers.

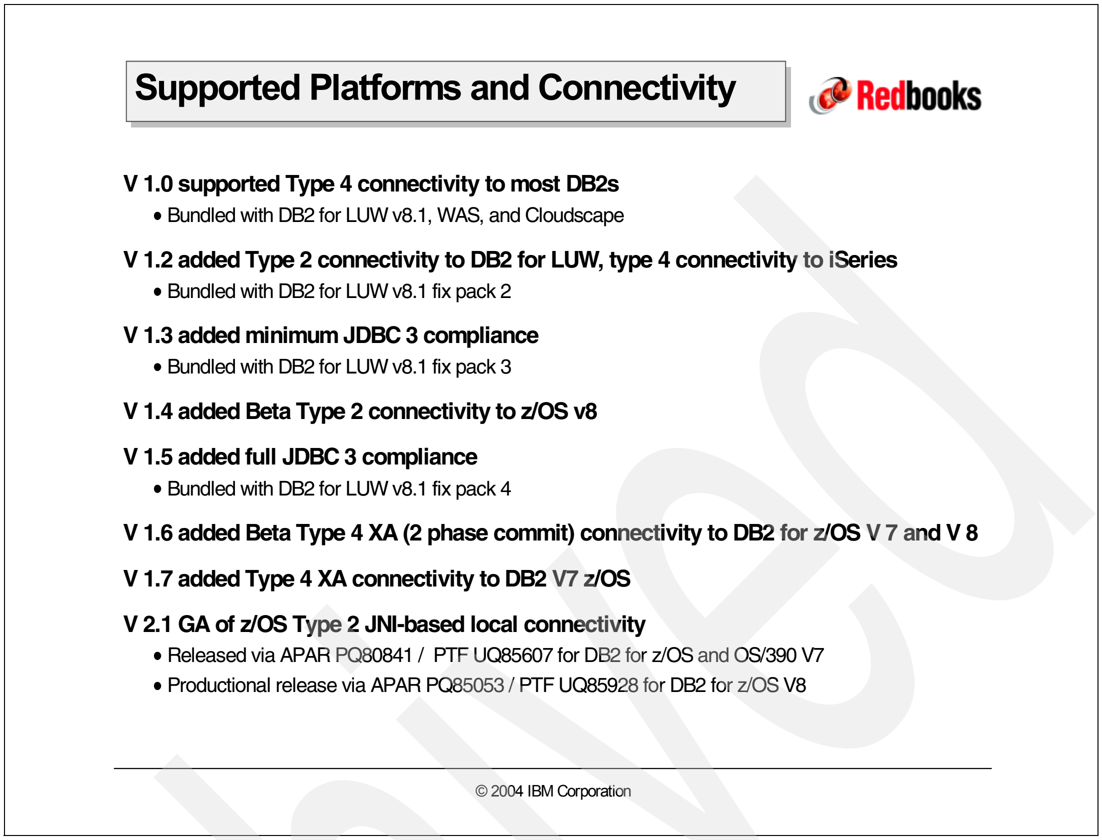
For your legacy applications, it may not always be possible to migrate in a plug-and-play manner. Those of your applications that are written to be portable according to the JDBC specification can continue to run under the Universal JDBC driver. However, currently running JDBC applications that are not written in a portable way, and only run under one particular driver, may require changes to run under the Universal JDBC driver.

For more information about behavioral differences between the legacy T2 driver for OS/390 and the new JCC driver, refer to the sections “JDBC differences between the DB2 Universal JDBC Driver and other DB2 JDBC drivers” and “SQLJ differences between the DB2 Universal JDBC Driver and other DB2 JDBC drivers” in the *DB2 Application Programming Guide and Reference for Java*, SC18-7414, as well as 12.4.2, “DB2 Universal Driver for SQLJ and JDBC” on page 899.

However, one of the areas where changes are required is related to SQLJ program preparation. The SQLJ program preparation process is different with the new JCC driver. For more information, see 5.2.4, “Application development using SQLJ” on page 426. The result of this is that the customization of the serialized profile is different. In order not to force everybody to rebind all the packages of their existing SQLJ applications, existing serialized profiles can be “upgraded” to the new format.

db2sqljupgrade utility

The purpose of the upgrade utility is to upgrade serialized profiles that were customized by DB2 for z/OS legacy driver, to work with JCC without having to bind a new packages. This will prevent the optimizer from choosing a new access path. This upgrade utility cannot be used to upgrade a serialized profile that was customized with any other driver.

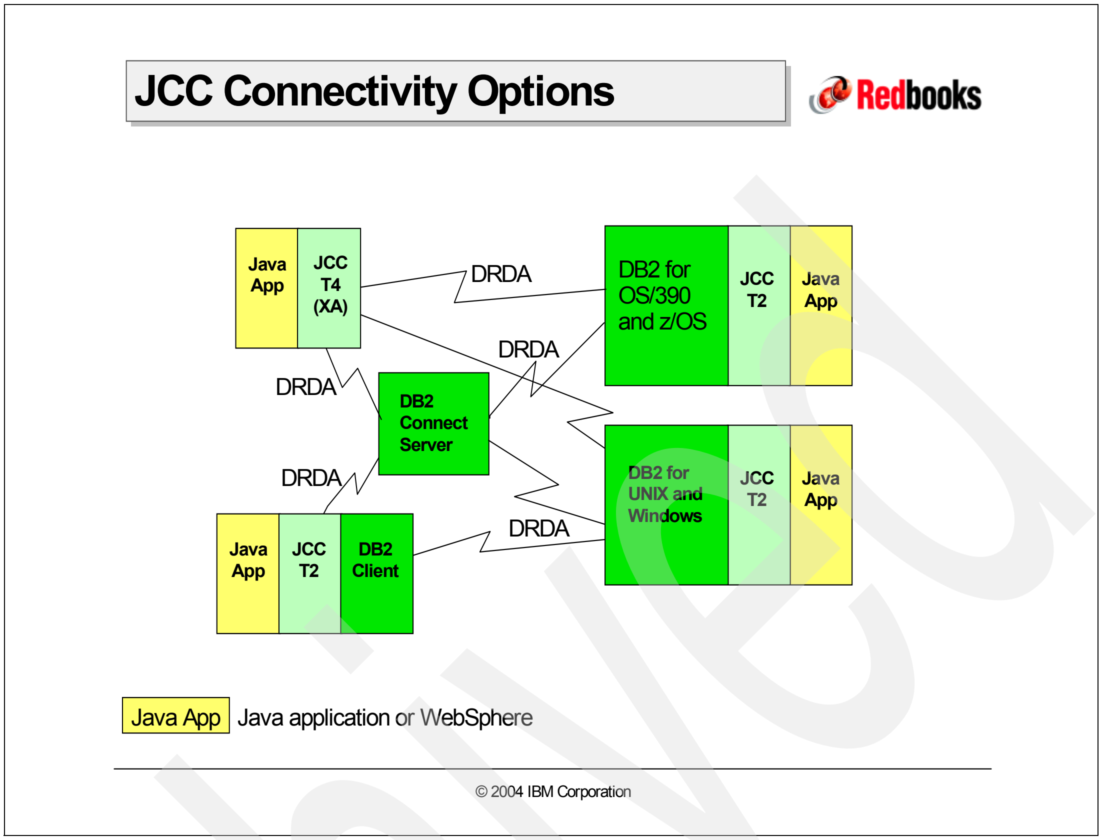


5.1.6 Supported platforms and connectivity

The visual above shows, for DB2 UDB for Linux, UNIX, Windows, and DB2 for OS/390 and z/OS, which JCC release supports which version of those DB2 systems. We omitted other DB2 family members here in order to keep it simple.

There is no downlevel support available for UNIX and Windows platforms for Type 4 connectivity. Only DB2 for Linux, UNIX, and Windows V8 systems are supported by the Universal Driver.

As shown in the figure above, the Universal Driver that ships with DB2 for z/OS Version 8 has been out in the field for quite some time now, and has gone through a number of versions and test cycles to make sure it is ready for prime time.



5.1.7 JCC connectivity options

This figure describes the different options available for you to connect your Java application to a DB2 for z/OS, or DB2 for Linux, UNIX and Windows server, using the IBM DB2 Universal Driver for SQLJ and JDBC.

Licensing

Redbooks

License file	Description
db2jcc_license_c.jar	Permits JDBC connectivity to Cloudscape servers only. Cloudscape is bundled with WebSphere Application Server (WAS) along with this license.
db2jcc_license_cu.jar	Permits JDBC/SQLJ connectivity to all DB2 LUW servers and Cloudscape. This is the standard license provided with UDB on UNIX/Windows. db2jcc_license_cu.jar is included with all of the DB2 LUW products, including Personal Edition(PE), WebSphere Edition (WSE), and DB2 Express.
db2jcc_license_cisuz.jar	Permits JDBC/SQLJ connectivity to all DB2 servers, including z/OS, iSeries, VM/VSE DB2 products, UDB for UNIX/Windows, and Cloudscape. This license is provided to DB2 Connect licensees only. db2jcc_license_cisuz is included with all DB2 Connect products including Personal Edition, Enterprise Edition (CUE/CASE), and DB2 ESE.

5.1.8 Licensing

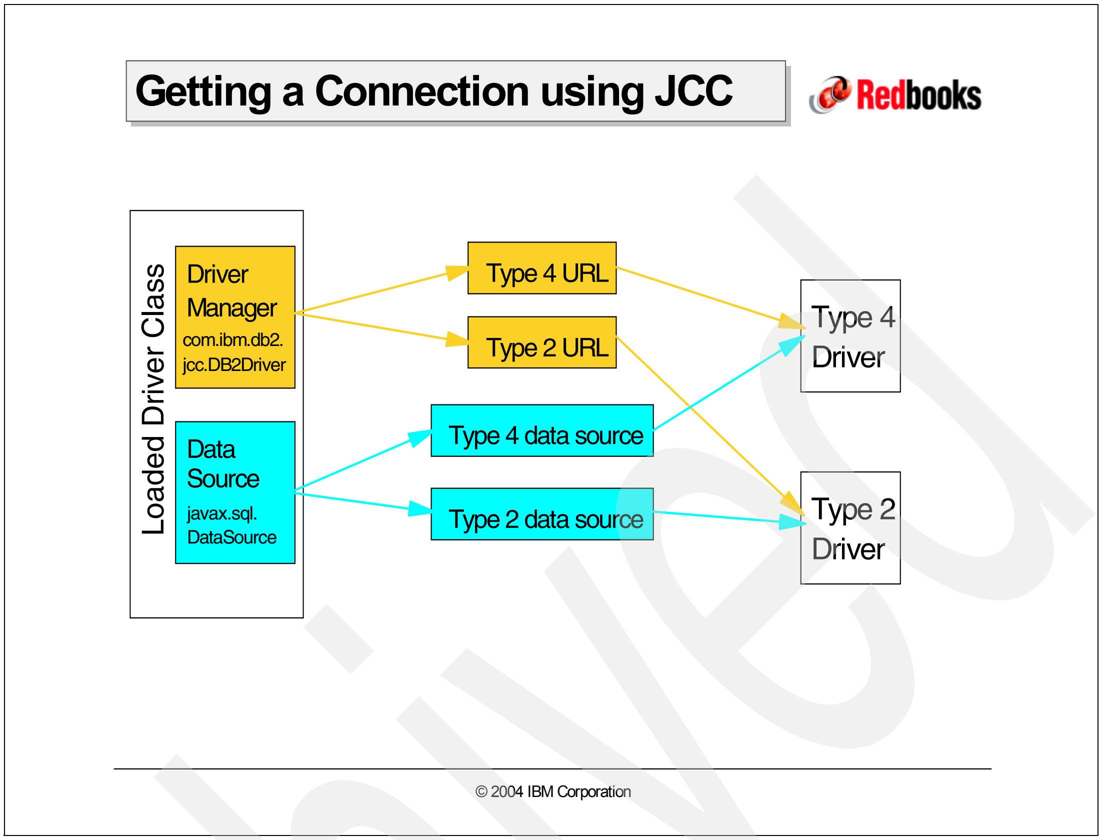
Although technically speaking, you do not need DB2 Connect to use the Universal Type 4 driver to connect your Java application to a DB2 for z/OS, you need a DB2 Connect license to be able to use the Type 4 driver to connect to a DB2 for z/OS system.

Beginning in release 1.2 of JCC (DB2 for Linux, UNIX and Windows 8.1.2), JCC requires an auxiliary license jar file installed to the application classpath in order to enable connectivity to target servers. The names of the jar files are shown on the visual above.

The meaning of the suffix letters in the license file names, shown in the visual above, is as follows:

- c = Cloudscape
- i = iSeries
- z = z/OS
- s = SQL/DS™
- u = Linux/UNIX/Windows

The availability of the license jar files is checked by the driver at driver load time.



5.1.9 Getting a connection

In a Java program, you can establish a connection to a database, either by:

- The DriverManager interface (JDBC 1 java.sql.DriverManager API). This API requires the application to load the com.ibm.db2.jcc.DB2Driver (in case of the Universal Driver) and hard code a database URL description in the application code.
- The Datasource interface (JDBC 2 javax.sql.DataSource API). This API is preferred over the DriverManager interface because the database target connectivity descriptions are contained in the data source object itself, separate from the application code. The same application code may be used to connect to any datasource object using its associated data source properties.

Since the IBM Universal Driver supports both Type 4 and Type 2 connectivity in a single driver, only a single driver instance is loaded by the driver manager for both Type 4 and Type 2 implementation. This means that distinguishing between Type 2 and Type 4 connectivity cannot be determined by driver name or data source name. Instead, the connectivity type (T2 or T4) is determined by the URL syntax under JDBC1, or a proprietary data source property named driverType under JDBC 2. This means that regardless of the underlying connectivity type, the same JDBC 2 DataSource implementation is used for obtaining connections.

When you obtain connections from the DriverManager, you select a Type 4 connectivity specifying:

```
jdbc:db2://server:port/database or
jdbc:db2://server/database
```

You select Type 2 connectivity if you specify:

```
jdbc:db2:database
```

When using the DataSource interface, a concrete data source class `com.ibm.db2.jcc.DB2SimpleDataSource` is provided that extends the abstract class `com.ibm.db2.jcc.DB2BaseDataSource`.

With the exception of a password, all data source properties are defined on the abstract `com.ibm.db2.jcc.DB2BaseDataSource` class.

A base DataSource property named driverType is used to determine the connectivity type:

- If `driverType == 4`, then both a server and database property are expected, and the Type 4 implementation of JCC is used:
  - The `databaseName` property is then the actual database name for T4, and is not the locally cataloged database name. The Universal JDBC driver does not rely on information cataloged in the DB2 database directory for Type 4 connectivity. For connectivity to a DB2 for z/OS system, you must specify the location name.
  - The `serverName` property specifies the TCP/IP address or name of the server you are connecting to.
  - The `portNumber` property indicates the port number the target server is listening on. You only need to specify this property when the system is not listening to the default port number (446).
- If `driverType == 2`, then server and port properties are ignored and the Type 2 implementation of JCC is used. The databaseName is the actual name only if the serverName is null; otherwise it is a cataloged alias. As before, for DB2 for z/OS connectivity, use the DB2 location name (even for this type of local connection).

The simplest data source implementation for the new Universal Driver is `com.ibm.db2.jcc.DB2SimpleDataSource` and is not enabled for XA or pooling.

When you use both the new Universal JDBC driver and the legacy Type 2 OS/390 driver, you must take care to adjust the classpath properly, as both drivers share the same data source class name `com.ibm.db2.jcc.DB2SimpleDataSource` and `com.ibm.db2.jcc.DB2DataSource`.

To support XA transactions, the Type 4 Universal Driver in the z/OS environment also provides the `com.ibm.db2.jcc.DB2XADataSource`. You may want to use `javax.sql.XADataSource` instead of `com.ibm.db2.jcc.DB2XADataSource`. The latter is a DB2 specific driver implementation and can only be used if you are using a DB2 connection.

DriverManager URL Syntax

Database URL syntax	Connectivity used
jdbc:db2:databaseName[:propertyKey=value;...]	Type 2
jdbc:db2://server[port]/databaseName[:propertyKey=value; ...]	Type 4
jdbc:db2j:net://server[port]/databaseName[:propertyKey=value; ...]	Type 4 to a Cloudscape Network Server

5.1.10 DriverManager database URL syntax

The Universal Driver URL syntax differs between the platforms. The following JDBC database URL syntax is accepted:

- jdbc:db2:databaseName [:propertyKey=value;...]
- jdbc:db2://server[:port]/databaseName[:propertyKey=value;...]
- jdbc:db2j:net://server[:port]/databaseName[:propertyKey=value;...] to Cloudscape Network Server

You can also pass connection properties as part of the URL, for example:

```
jdbc:db2://wtscpok.ibm.com:12345/DB8A:user=itsousr;password=itsopwd;deferPrepares=false;
```

> **Note:** The databaseName is case-sensitive, and must always be upper case for DB2 for z/OS.

Data Source Classes

DataSource name	Description
com.ibm.db2.jcc.DB2SimpleDataSource	The simplest data source not enabled for connection pooling or distributed transactions
com.ibm.db2.jcc.DB2DataSource	A data source with a rudimentary pooling manager
com.ibm.db2.jcc.DB2ConnectionPoolDataSource	A factory for pooled connections. The DB2ConnectionPoolDataSource and DB2PooledConnection interfaces provide all the necessary hooks to implement a connection pooling module. This datasource is used by the WAS pooling module
com.ibm.db2.jcc.DB2XADataSource	A factory for XA connections. The DB2XADataSource, DB2XAConnection, and DB2Xid interfaces provide all that is needed by a JTA implementation. These interfaces are currently supported for type 4 connectivity.

5.1.11 DataSource class implementations

The *com.ibm.db2.jcc.DB2SimpleDataSource* and *com.ibm.db2.jcc.DB2BaseDataSource* classes have also been shipped for the legacy Type 2 driver on DB2 for OS/390 and z/OS. However, those lack certain additions and refinements that exist in the versions that ship with the Universal Driver.

Because of that, an interface *com.ibm.db2.jcc.DB2JCCDatasource* is provided by the Universal Driver to distinguish between the two differing *com.ibm.db2.jcc.DB2DataSource* instances. You can use this interface for your applications to distinguish between instances of *com.ibm.db2.jcc.DB2BaseDataSource* as shipped with the IBM DB2 JDBC Universal Driver from data sources that are shipped with the legacy OS/390 Type 2 driver. If the data source instance that you use implements this interface, the data source is a “true” Universal Driver data source instance.

To support XA transactions, the Type 4 Universal Driver in the z/OS environment provides the required javax.sql.transaction.xa implementations via:

- `com.ibm.db2.jcc.DB2XADataSource`
- `com.ibm.db2.jcc.DB2Xid`
- `XAConnection`
- `XAResource`

You may want to use `javax.sql.XADataSource` instead of `com.ibm.db2.jcc.DB2XADataSource`. The latter is a DB2 specific driver implementation and can only be used if you are using a DB2 connection.

5.1.12 Specifying properties for your database connection

The JCC driver allows you to specify additional connection properties after the instance-name part, separated by colons and taking the form `propertyKey=value`. For the list of supported property keys, see Table 5-1 below. These apply to driver manager and datasource connections.

Table 5-1 Property keys for the DB2 Universal Driver

Property key	Description
driverType	Determines the JDBC connectivity type to a data source. If driverType is not set, type 2 connectivity is selected by default. This property is a data source property, and not a connection property. JDBC 1 connectivity selection is based on the URL syntax.
user	Specifies the user id when connecting using the DriverManager.getConnection(String url, Properties properties) method.
password	Specifies the password when connecting using the DriverManager.getConnection (String url, Properties properties) method.
serverName	See "Improved security for DB2 authentication" on page 410.
portNumber	The TCP/IP port number where the DRDA server listens for connection requests to this data source. The default value is 446.
databaseName	For the type 4 driver directly accessing a DB2 for z/OS system, or a type 2 connection directly accessing a DB2 for z/OS system, you specify the DB2 location name.
logWriter	Used for tracing.
traceLevel	Specifies the level of detail you want to trace at.
traceFile	Used for tracing.
traceFileAppend	If set to `true`, causes the trace to be appended to the trace file; otherwise, the file is overwritten.
fullyMaterializeLobData	See the Large Object (LOB) Considerations when Using the Universal JDBC Driver topic. FIXME
resultSetHoldability	For DB2 targets, the default is HOLD_CURSORS_OVER_COMMIT. For Cloudscape Network Server, the default is CLOSE_CURSORS_AT_COMMIT.
currentPackageSet	This property is used in conjunction with the db2jdbcbind -collection option, which is given when the JDBC/CLI package is bound during DB2 installation.
securityMechanism	See "Improved security for DB2 authentication" on page 410.

Property key	Description
kerberosServerPrincipal	See “Improved security for DB2 authentication” on page 410.
gssCredential	See “Improved security for DB2 authentication” on page 410.
readOnly	Creates a read-only connection. By default this is false. This property is flowed at connect time for type 4 connectivity.
deferPrepares	See the Deferred Prepares with the Universal JDBC Driver topic.
currentSchema	Sets the CURRENT SCHEMA special register of the DB2 server. See the CURRENT SCHEMA special register topic for details. This property is currently not supported by DB2 Universal Database™ for OS/390 and z/OS, and will result in either an error or an exception when targeting DB2 Universal Database for OS/390 and z/OS servers. (This includes V8 systems at the time of writing. However, this property is expected to be supported in the near future for DB2 for z/OS V8 systems.)
currentSQLID	Sets the CURRENT SQLID special register on DB2 Universal Database for OS/390 and z/OS. See the DB2 Universal Database for OS/390 and z/OS publications for more information. Setting this property will fail when targeting DB2 Universal Database on Windows or UNIX-based platforms.
cliSchema	Indicates the schema of the DB2 shadow catalog tables or views to search when you issue a database metadata catalog query.
retrieveMessagesFromServerOnGetMessage	Enable this property to direct all calls to the standard JDBC SQLException.getMessage() method to invoke a server-side stored procedure that retrieves the formatted message text for the error (the invocation of the stored procedure starts a separate unit of work). By default, this property is disabled and the full message text is not returned to the client when a server-side error occurs, because the proprietary method DB2Sqlca.getMessage() can be called to retrieve the formatted message text.
clientUser	See “Java API for Set Client Information (SQLESETI)” on page 411. The clientUser property establishes the current client user name for this connection. The current client user name is not the JDBC connection user name, but is for client accounting purposes. Unlike the JDBC connection user name, the current client user name, and any associated client information, may change during the life of this connection.

Property key	Description
clientWorkstation	Establishes the work station name for the current client on this connection.
clientApplicationInformation	Establishes generic application information for the current client on this connection.
clientAccountingInformation	Establishes generic accounting information for the current client on this connection.

You can supply values for the individual keys by either using the DriverManager.getConnection (String url, Properties properties) method (Example 5-1), or by encoding them in the URL (for example, in a DataSource definition, or on the application's command line). When using WebSphere Application Server, you can specify all these properties in the WebSphere Administrative console's setup windows.

Example 5-1 Setting properties using DriverManager.getConnection(String, Properties)

```
import java.net.InetAddress;
import java.sql.Connection;
import java.sql.DriverManager;
import java.util.Properties;

...

Properties properties = new Properties();
properties.put("user", "bartr1");
properties.put("password", "secret");
properties.put("clientWorkstation", InetAddress.getLocalHost().getHostName());
properties.put("clientUser", System.getProperty("user.name"));
Connection conn = DriverManager.getConnection(url, properties);
```

Java API Enhancements

Redbooks

Scrollable cursor support

Batched updates support

Improved Java SQL error information

- Native DB2 server SQL error messages

Java API for set client information (SQLESETI)

Built-in DB2SystemMonitor class (application-driver-network-server)

Multiple open result sets

SAVEPOINT support

Auto-generated keys

Enhanced LOB support

5.1.13 Java API enhancements

The DB2 Universal Driver provides more functionality than the older drivers, for example, it supports scrollable cursors, batch updates, and savepoints, as discussed in the following sections. The Universal Driver delivers almost full JDBC 3.0 functionality. Next we list some of the important functional enhancements.

Scrollable cursor support

Scrollable cursor support refers to the possibility to move a result set's cursor backward as well as forward. This is especially useful for GUI applications, when users want to browse a result set backward and forward. Without scrollable cursors, the application has to either cache the result set in memory, or submit the query again when the user scrolled backward.

This feature, introduced in JDBC 2.0, is now supported in the DB2 Universal Driver. The driver relies on the underlying DB2 infrastructure to support scrollable cursors. The Universal Driver will not mimic the behavior of a scrollable cursor for downlevel servers if the engine does not have support for scrollable cursors.

Batched updates

A batched update is a set of multiple update statements that are submitted to the database for processing as a batch. Sending multiple update statements to the database together as a unit can, in some situations, be much more efficient than sending each update statement separately. It reduces the number of times the application has to cross over to the JDBC driver. This ability to send updates as a unit, referred to as the batched update facility, is one of the features of the JDBC 2.0 API, and is now supported with the DB2 Universal Driver.

Example 5-2 demonstrates how to use batch updates. Note that autocommit should be turned off when using batch updates. Also, each of the statements in the batch must be one that returns an update count (for example, a SELECT statement in the batch is not allowed, and will cause an SQLException to be thrown).

The Universal Driver uses non-atomic batching. Each statement is treated independently. It is up to you to decide what to do in case one operation in the batch fails.

Example 5-2 Creating and executing batched INSERT statements

```
// Turn off autocommit
conn.setAutoCommit(false);
Statement stmt = conn.createStatement();
stmt.addBatch("INSERT INTO employees VALUES (1000, 'Joe Jones')");
stmt.addBatch("INSERT INTO departments VALUES (260, 'Shoe')");
stmt.addBatch("INSERT INTO emp_dept VALUES (1000, 260)");

// Submit a batch of update commands for execution
int[] updateCounts = stmt.executeBatch();
```

Improved security for DB2 authentication

Authentication security has been improved significantly with the DB2 Universal Driver. It supports the following authentication techniques:

- Userid and password in plain text
- Userid and password encrypted
- Kerberos security

For encrypted and Kerberos security, the DB2 Universal Driver uses the following Java services:

- IBM Java Generic Security Service (JGSS)
- IBM Java Authentication and Authorization Service (JAAS)
- IBM Java Cryptography Extension (JCE)

You can refer to the following URL on the Web for more information on how to enable these components on z/OS:

http://www.ibm.com/servers/eserver/zseries/software/java/aboutj2.html

The IBM Java Cryptography Extension (JCE) is required for using encryption. The installed JCE jar files are:

- ibmjceprovider.jar
- ibmjcefw.jar
- ibmjlog.jar
- US_export_policy.jar
- Local_policy.jar

These jar files are typically installed to jdk\jre\lib\ext.

IBM JGSS, JAAS and JCE are required for using Kerberos. The installed jar files are:

- ibmjgssprovider.jar
- jaas.jar
- ibmjceprovider.jar
- ibmjcefw.jar
- ibmjlog.jar

- US_export_policy.jar
- Local_policy.jar

The authentication technique can be specified by either setting a Java property in the application, or by recording the required technique in the DB2 DataSource definition.

These jar files are typically installed to jdk\jre\lib\ext.

Improved Java SQL error information

The standard JDBC SQLException class does not provide a way to retrieve vendor-specific error information (other than the getVendorCode() method which, for DB2, returns the SQLSTATE reported by the DB2 server). For example, if an INSERT into a table failed because you tried to insert a NULL value into a NOT NULL column, there is no standard way to find out which column DB2 was complaining about (other than to parse the error message).

Again, the DB2 Universal Driver provides a proprietary API to retrieve detailed error information (in DB2 terms, to access the SQLCA). Each DB2 server, including DB2 for OS/390 V6 and V7, now provides stored procedures which allow applications to retrieve the “native” error message text for a given error or warning.

DB2 typically returns an SQLCA structure containing an SQLCODE, and associated message tokens. To format the message text for the SQLCODE requires the application to invoke a local API (such as DSNTIAR). To provide a more uniform method for remote clients (that cannot invoke DSNTIAR) to accomplish this, a new stored procedure is provided.

Stored Procedure SYSIBM.SQLCAMESSAGE, which is also available for DB2 V6 (UQ72081/UQ72082) and V7 (UQ72083) formats SQLCODE message text, given input fields from a DB2-generated SQLCA. The procedure provides a method to invoke the DSNTIAR message formatter via an SQL CALL statement. No result set is returned. The output message is returned via a VARCHAR output parameter. This procedure is not used by ODBC.

For more information, see the *DB2 for z/OS and OS/390: Ready for Java*, SG24 6435 Redbook.

Java API for Set Client Information (SQLESETI)

As a DB2 specific extension, the JCC driver provides an interface to supply extended client information to the DB2 server. This is especially useful in a client/server or application server environment. Using this interface, the client program, or the application server, can supply information that identifies the specific user issuing a request. Otherwise, only the application server’s information is passed. Unlike the user information supplied when creating the JDBC connection, the extended information can be changed at any time (for example, when the application server processes a request from another user).

Also, the extended client information can be used for accounting or workload management purposes.

The client information will be sent to the server at the next opportunity, that is, along with the next SQL call on that connection.

To use this interface, cast the `java.sql.Connection` object to a `com.ibm.db2.jcc.DB2Connection`. In addition to the standard `java.sql.Connection` methods, the `DB2Connection` class provides the following methods to set extended client information.

setDB2ClientUser

```
public abstract void setDB2ClientUser(String s)
  throws SQLException;
```

Sets a user name for the connection. Unlike the user ID supplied when creating the connection, this can be a full user name.

setDB2ClientWorkstation

```
public abstract void setDB2ClientWorkstation(String s)
  throws SQLException;
```

Sets a client workstation name for the connection, for example, the workstation's TCP/IP hostname.

setDB2ClientApplicationInformation

```
public abstract void setDB2ClientApplicationInformation(String s)
  throws SQLException;
```

Sets an application name. For example, you can specify the main class name of the Java application working with the connection.

setDB2ClientAccountingInformation

```
public abstract void setDB2ClientAccountingInformation(String s)
  throws SQLException;
```

Sets accounting information.

Example 5-3 demonstrates the use of the extended client information API.

Example 5-3 Using the extended client information API

```
public class ClientInfoTest {

  public static void main(String[] args) {
    String url = "jdbc:db2://wtsc63.itso.ibm.com:33756/DB7Y";
    try {
      Class.forName("com.ibm.db2.jcc.DB2Driver");
      Connection conn = DriverManager.getConnection(url, "bartr1", "secret");
      if (conn instanceof DB2Connection) {
        DB2Connection db2conn = (DB2Connection) conn;
        db2conn.setDB2ClientUser("Ulrich Gehlert");
        db2conn.setDB2ClientWorkstation("a23wpm64");
        db2conn.setDB2ClientApplicationInformation("ClientInfoTest");

        // Dummy call to force extended client information to be sent
        conn.prepareStatement("SELECT * FROM SYSIBM.SYSDUMMY1"
                              + "WHERE 0 = 1").executeQuery();

        // Sleep for a while 30 secs) so we can display DB2 thread information
        Thread.sleep(30000);
      }
    } catch (Throwable e) {
      e.printStackTrace();
    }
```

```
    }
  }
```

Without setting client information, the DB2 DISPLAY THREAD command only shows the authorization ID and location name for the thread associated with the JDBC connection. However, after setting these client strings, you get the following information (see Example 5-1).

```
 -DISPLAY THREAD(*) DETAIL

DSNV401I  -DB7Y DISPLAY THREAD REPORT FOLLOWS -
 DSNV402I  -DB7Y ACTIVE THREADS -
 NAME     ST A   REQ ID           AUTHID   PLAN     ASID TOKEN
 SERVER   RA *     4 db2jccmain   BARTR1   DISTSERV 0050  1166
  V437-WORKSTATION=a23wpm64, USERID=Ulrich Gehlert,
       APPLICATION NAME=ClientInfoTest
  V445-G9012728.G5A4.00F1996EB1A2=1166 ACCESSING DATA FOR 9.1.39.40
  V447--LOCATION         SESSID           A ST TIME
  V448--9.1.39.40        33756:1444       W R2 0232218144291
```

Figure 5-1   DISPLAY THREAD command showing extended client information

Java API for application monitoring

To help you isolate performance problems with your applications, the DB2 Universal Driver provides a proprietary API (DB2SystemMonitor class) to enable application monitoring.

The driver collects the following timing information, as depicted in Figure 5-2:

- Server time (the time spent in DB2 itself)
- Network I/O time (the time used to flow the DRDA protocol stream across the network)
- Core driver time (the time spent in the driver; this includes network I/O time and server time)
- Application time (the time between the start() and stop() calls)

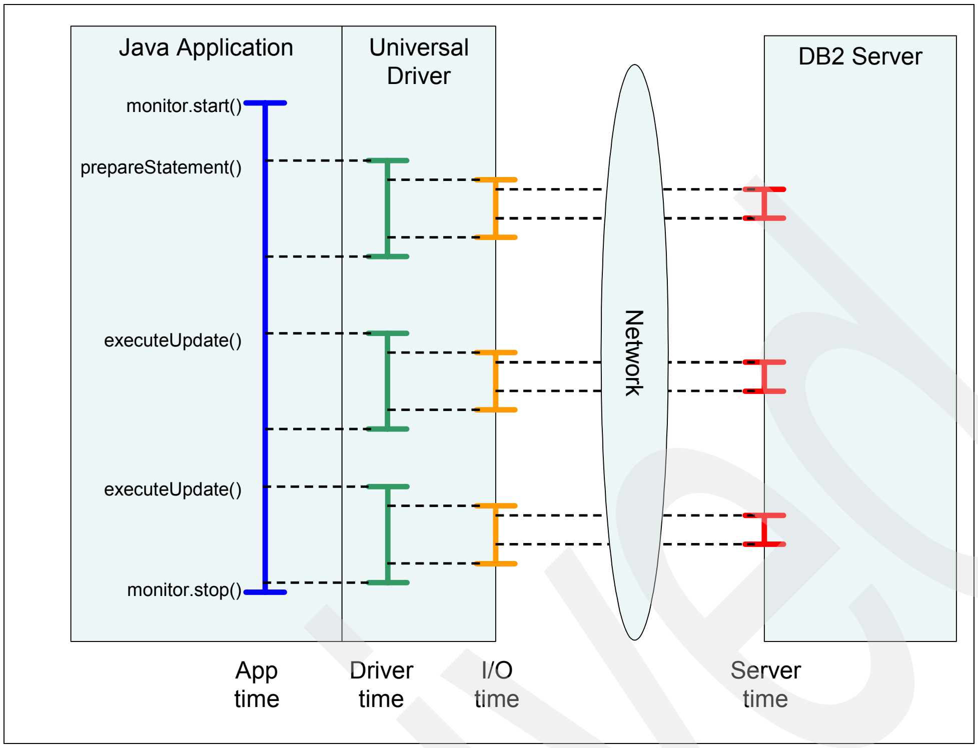


Figure 5-2 Times collected by DB2 Universal Driver monitoring

The use of the API is demonstrated in Example 5-4.

Example 5-4 Using the DB2 Universal Driver monitoring API

```
import com.ibm.db2.jcc.DB2Connection;
import com.ibm.db2.jcc.DB2SystemMonitor;

...
DB2SystemMonitor monitor = conn.getDB2SystemMonitor();
monitor.enable(true);
monitor.start(DB2SystemMonitor.RESET_TIMES);

// ... SQL statements ...

monitor.stop();
   // ServerTimeMicros is red line in figure above
System.out.println("Server time:      " + monitor.getServerTimeMicros());
   // NetworkIOTimeMicros is orange line in figure above
System.out.println("Network I/O time: " + monitor.getNetworkIOTimeMicros());
   // CoreDriverTimeMicros is green line in figure above
System.out.println("Core driver time: " + monitor.getCoreDriverTimeMicros());
   // ApplicationTimeMillis is blue line in figure above
System.out.println("Application time (ms): " + monitor.getApplicationTimeMillis());
```

You can choose to either reset or to accumulate times when starting the monitoring, using `DB2SystemMonitor.RESET_TIMES`, or `DB2SystemMonitor.ACCUMULATE_TIMES` on the `monitor.start()` method, respectively.

Note that the various getTime() methods may throw an `SQLException` if the driver cannot provide the information requested.

DB2SystemMonitor prerequisites

Although already shipped in an earlier FixPak, the DB2SystemMonitor is only fully operational with DB2 for LUW Version 8 FixPak 4 or later.

In addition, to be able to get accurate times for the core driver and network I/O times, your JVM has to be at a certain code level as well. Currently these JVM enhancements have only been implemented as a feature of IBM's JVMs, more specifically in:

- JDK 131 SR 5, available now, for non-z/OS platforms
- JDK131 (cm131s), and JDK 141 SR1(cm141) for the z/OS platform

At the time of writing of this publication, the Sun JVM 1.4.1 does not have this feature. If the `DB2SystemMonitor` does not find the accurate timer functionality in the JVM, it throws an `SQLException`:

```
com.ibm.db2.jcc.a.SqlException: Network IO time in microseconds is not available
```

Or:

```
com.ibm.db2.jcc.a.SqlException: Core driver time in microseconds is not available
```

The ServerTime is based on the DRDA server elapsed time feature that was introduced in DB2 for z/OS and OS/390 Version 7. As a consequence, DRDA has to be involved in the transaction to get the ServerTime. If you are running your application local to DB2, you always get 0 (zero) from the getServerTimeMicros method. This is true both for DB2 LUW and DB2 for z/OS when running the JCC Driver as a Type 2 driver. When run as a Type 4 driver, JCC always uses DRDA to connect to the server.

Please also note that the ApplicationTime is in milliseconds (using System.currentTimeMillis), whereas the other times are presented in microseconds.

Multiple open stored procedure result sets

Up to DB2 for z/OS and OS/390 V7 it is not possible for an (JDBC or a CLI) application to have more than one instance of the same open (result set) cursor, open at the same time.

With the new Universal Java Client, as well as using ODBC/CLI applications on Linux, UNIX Windows, accessing z/OS, you can now have multiple instances of the same result set cursor open concurrently.

This enhancement is also applicable to SQLJ iterators (the SQLJ equivalent of a cursor).

The SQLJ API allows an application to issue a "new" operation that makes a new copy of the static cursor that can be used with different host variable input on OPEN. Since a cursor has to be unique, based on the fully-qualified package name, consistency token, and section number, it is currently (V7) not possible for an SQLJ application to have more than one instance of an open cursor. In V7, a second instance or OPEN of the same cursor would not be allowed in DB2, and would result in SQLCODE -502. Allowing multiple opens for the same cursor in V8 eases this problem.

SAVEPOINT support

The DB2 Universal Driver supports the SAVEPOINT mechanism specified in the JDBC 3.0 specification.

If JDBC statement Connection.setSavepoint() is issued, the Universal Driver immediately executes the SQL statement 'SAVEPOINT *savepoint_name* ON ROLLBACK RETAIN CURSORS'. 'RETAIN CURSORS' is used as default and there is no JDBC or proprietary API available to drop cursors on rollback to savepoint. This behavior is identical to the regular usage of savepoints in DB2 for z/OS.

There are, however two differences:

- You cannot specify the UNIQUE keyword.
- Even though the RETAIN CURSORS attribute is used when rolling back to a savepoint, a cursor will be invalidated by the server under some circumstances regardless. This can occur when the cursor relies on DDL that was created after the savepoint and subsequently rolled back. Currently, such a "ROLLBACK TO SAVEPOINT" request may not immediately invalidate a cursor that is cached in the driver. If an application is scrolling through cached cursor data, the driver is not able to determine that the cursor has been invalidated due to a rollback to savepoint until a server request is made for more data.

To release a savepoint, you can use the JDBC Connection.releaseSavepoint() implementation, which immediately executes the SQL statement 'RELEASE SAVEPOINT *savepoint_name*'.

Auto-generated keys

Like many other database servers, DB2 does have a mechanism that automatically generates a new, unique key value whenever a row is inserted. In the case of DB2, you declare a column to be an `IDENTITY` column.

Of course, after inserting a new row into a table containing an `IDENTITY` column, you probably want to retrieve the value that DB2 generated for that column (you might need it, for example, as a foreign key value in a dependant table). You can use the DB2-specific function `IDENTITY_VAL_LOCAL()` to retrieve the last value generated.

Beginning with JDBC 3.0, however, there is a mechanism that allows an application to retrieve the value without using vendor-specific extensions, and this API is supported by the DB2 Universal Driver. An example of using `Statement.RETURN_GENERATED_KEYS` is shown in Example 5-5. The generated result set contains 1 row and 1 column of data type DECIMAL(31,0) regardless of the actual data type of the identity column retrieved. Auto-generated keys only apply to INSERT statements (or SELECT ... FROM FINAL TABLE (INSERT ...) statements).

Example 5-5 Retrieving auto-generated keys

```
Statement stmt = conn.createStatement();
// indicate that the key generated is going to be returned
int rows = stmt.executeUpdate("INSERT INTO ORDERS (ISBN, CUSTOMERID) "
                            + "VALUES (195123018, 'BILLG')",
                             Statement.RETURN_GENERATED_KEYS);
ResultSet rs = stmt.getGeneratedKeys();
if (rs.next()) {
// retrieve the new key value
  int orderID = rs.getInt(1);
}
```

Note: Unfortunately, there is no equivalent construct for SQLJ; you have to use the IDENTITY_VAL_LOCAL() function:

```
#sql orderID = { VALUES(IDENTITY_VAL_LOCAL()) };
```

In DB2 for z/OS Version 8, you are also able to use the 'SELECT FROM INSERT' construct to retrieve generated values of a table:

```
#sql orderID = { SELECT order-identity-col
                   FROM FINAL TABLE (INSERT INTO my-order-table(col-list)
               };
```

Some other restrictions apply if you want to work with auto-generated keys in your JDBC application:

- An identity column defined as "GENERATED ALWAYS AS IDENTITY" can only be updated using the keyword DEFAULT. You cannot update such columns using the JDBC 2 updateable result set APIs. However, you can work around this problem, using the JDBC1 positioned update syntax or define the column as "GENERATED BY DEFAULT AS IDENTITY".
- Consider the following statements:
  - PreparedStatement.execute (String sql, String[] columnNames)
  - PreparedStatement.executeUpdate (String sql, String[] columnNames)

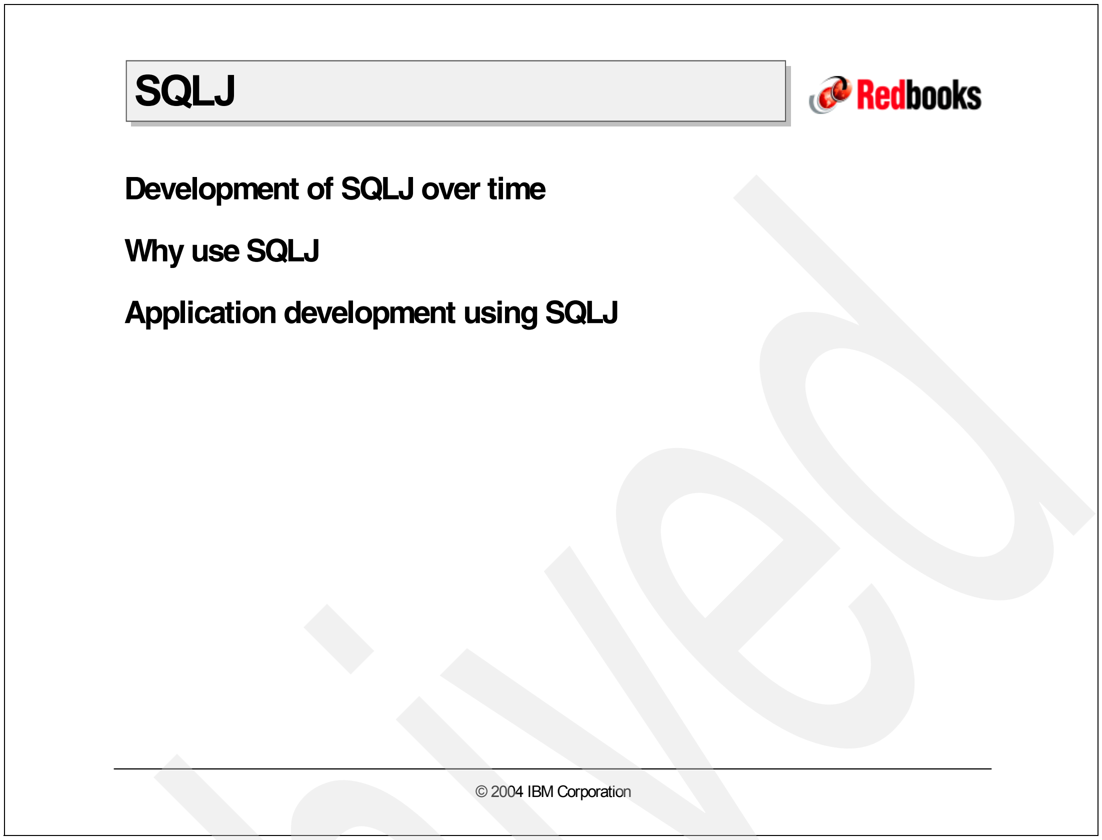
  You can use these with a DB2 for z/OS V8 system on a table with identity columns, because DB2 for z/OS supports the SELECT .. FROM FINAL TABLE (INSERT ...) construct.
- The Universal Driver throws an exception if you invoke either of the following methods:
  - PreparedStatement.execute (String sql, int autoGeneratedKeys)
  - PreparedStatement.execute (String sql, int[] columnIndexes)
  - PreparedStatement.executeUpdate (String sql, int autoGeneratedKeys)
  - PreparedStatement.executeUpdate (String sql, int[] columnIndexes)

Enhanced LOB support

When you retrieve LOB data with legacy JDBC drivers, the sent data is always blasted (the entire LOB is always sent). This can result in poor performance or even out-of-memory exceptions if the LOB data is large and too big for the memory.

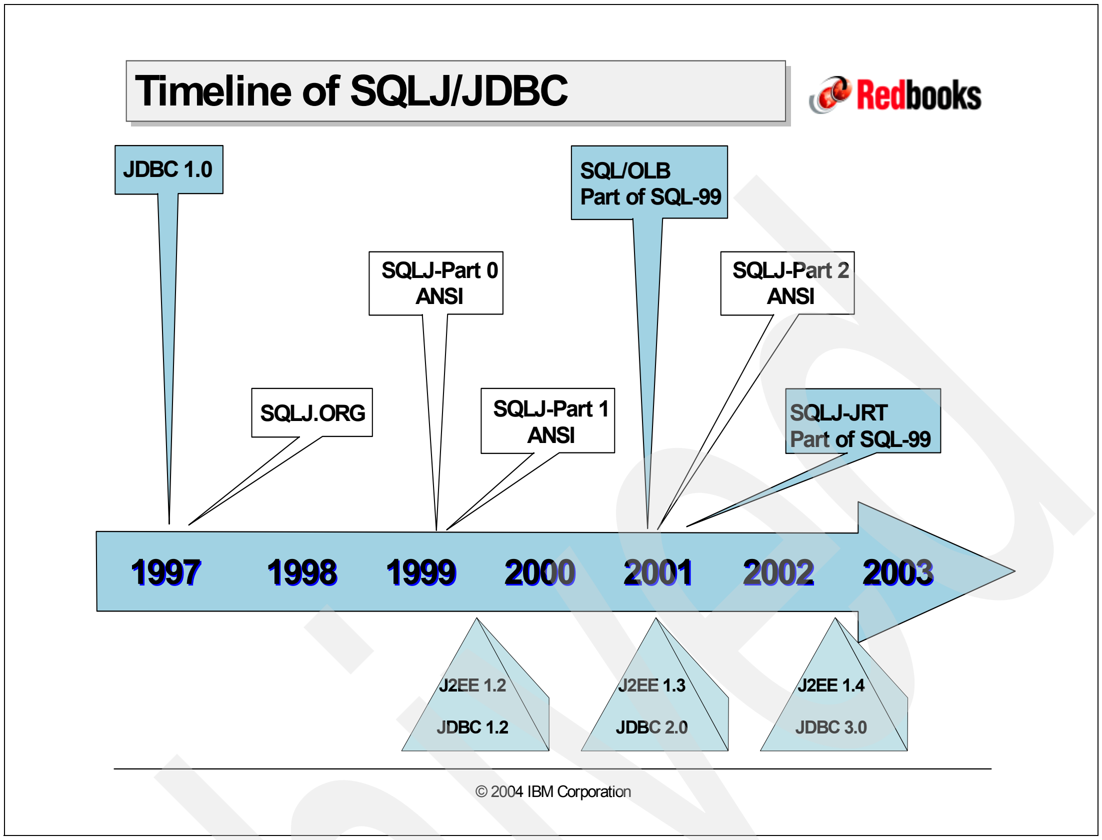
The Universal Driver provides a proprietary JDBC 2 data source and JDBC 1 connection property fullyMaterializeLobData. Using this setting, you can influence the way LOB data is sent to the Client:

- If *fullyMaterializeLobData* is enabled, that is set to TRUE, LOB data is not streamed, but fully materialized within the JDBC driver at the time the row data is fetched. Apart from poor performance, this setting might cause out of memory conditions to occur. *fullMaterializedLobData=TRUE* is currently the default.
- If *fullyMaterializeLobData* is disabled, that is set to FALSE, then the LOB data is streamed. Streaming means that the driver itself uses locators to retrieve LOB data in chunks on an as-needed basis. This setting avoids out of memory exceptions on the one hand, but can lead to invalidation of LOB data across transaction boundaries.



5.2 SQLJ

In this section, we explain the DB2 for z/OS support for applications using SQLJ interfaces to access DB2 data. We give you an overview of SQLJ, followed by more detailed information about the DB2 for z/OS implementations of SQLJ, and provide guidelines for writing an SQLJ program.



5.2.1 Development of SQLJ over time

SQLJ is a series of specifications for ways to use the Java programming language with SQL. It was developed by IBM, Oracle, and Tandem to provide an alternative to the dynamic JDBC specification. SQLJ is not part of J2EE, but is part of the SQL-1999 ISO/ANSI standard.

The SQLJ specification consists of two parts:

- ISO/IEC 9075 Part 10: Object Language Bindings (SQL/OLB). (This specification sometimes also referred to as SQLJ Part 0.)
- ISO/IEC 9075 Part 13: Routines and Types Using the Java Programming Language (SQL/JRT). This part is the specification for SQL routines using Java.

Although JDBC came up prior to SQLJ, SQLJ support has been around for a while now. It provides superior performance, because it uses static SQL (in contrast to JDBC, which uses dynamic SQL), and it uses a powerful authorization model (like static SQL in other programming languages). But prior to the Universal Driver, the development and deployment of SQLJ applications was somewhat cumbersome.

Why use SQLJ ?

Redbooks

Static SQL performance for Java applications

- Significant performance advantage over JDBC

Static SQL authorization model

- Provides Java with a stronger authorization model
- Errors detected earlier

Productivity/manageability

- Less code written by the application programmer
- Resulting code is easier to maintain
- Static SQL packages for accounting/monitoring
- Static SQL locks in access path, so that access path changes don't occur without a conscious choice

5.2.2 Why use SQLJ?

There are some major differences between SQLJ and JDBC, with a lot of good reasons for using SQLJ over JDBC for Java application development.

Static SQL performance for JAVA applications

Most of the time, static SQL is faster than dynamic SQL, because at runtime only the authorization for packages and plans must be checked prior to the execution of the program. In contrast to that, dynamic SQL statements require the SQL statements to be parsed, table/view authorization to be checked, and the optimization path to be determined.

Static SQL authorization model

Apart from the performance point of view, SQLJ also provides the advantages of static SQL authorization checking. With SQLJ, the owner of the application grants EXECUTE authority on the plan or package, and the recipient of that GRANT must run the application as written. With JDBC, the owner of the application grants privileges on all the underlying tables that are used by the application. The recipient of those privileges can do anything that is allowed by those privileges, for example, using them outside the application the authorizations were originally granted for. The application cannot control what the user can do.

Productivity/manageability

SQLJ code is generally more compact and error-free than JDBC code.

SQLJ is easier to code

The first advantage of SQLJ over JDBC is that SQLJ is easier to code, to read and to maintain. This is an effect of SQLJ being not an API, but a language extension, providing for better integration of the SQL code with the Java code. The developer can concentrate on the logic of individual SQL statements without having to worry about wrapping them in API calls. This simplicity is helped by the ease by which host variables are defined, maintained, and accessed within an SQLJ program.

As SQLJ statements are coded in purely SQL syntax, without the need to wrap them in a Java method, the programs themselves are easier to read, making them easier to maintain. Also, since some of the boilerplate code which has to be coded explicitly in JDBC is generated automatically in SQLJ, programs written in SQLJ tend to be shorter than equivalent JDBC programs.

We give some examples on the next couple of pages ("Comparing coding SQLJ and JDBC" on page 423).

SQLJ catches errors sooner

Not only is SQLJ typically more concise and easier to read than JDBC, it also helps you to detect errors in your SQL statements earlier in the program development process.

JDBC is a pure call-level API. This means that the Java compiler does not know anything about SQL statements at all — they only appear as arguments to method calls. If one of your statements is in error, you will not catch that error until runtime when the database complains about it.

SQLJ, on the other hand, is not an API but a language extension. This means that the SQLJ tooling is aware of SQL statements in your program, and checks them for correct syntax and authorization during the program development process.

It also enforces *strong typing* between iterator columns and host variables. In other words, it prevents you, for example, from assigning a numeric column to a String host variable.

Common errors that will be caught earlier with SQLJ, but will only be detected at runtime by JDBC, include these:

- **Misspelled SQL statements (for example, INERT instead of INSERT)**
  The SQLJ translator will catch and report this error. However, the translator does not parse the entire SQL statement, so most syntax errors will only be detected by the profile customizer.
- **No parameter supplied for parameter marker**
  Consider Example 5-7 on page 424. If you forgot to supply a value for one of the two parameter markers in JDBC, a runtime error will occur. In SQLJ, there are no parameter markers, rather, the host variables are embedded in the statement.
- **Misspelled table name**
  A misspelled table name will not be caught by the SQLJ translator (after all, it cannot know if a table does exist or not); however, the profile customizer will complain unless it was invoked with online checking disabled.
- **Not all columns retrieved**
  Assume that you add another column to the SELECT list, but forget to retrieve the corresponding column in the loop which processes the result set. This cannot happen with SQLJ, since the number of host variables in the FETCH statement (and the number of columns in the SELECT list) must match the number of columns in the iterator declaration.

SQLJ is more predictable and reliable

The key here is that you have static SQL statements that are bound into packages. The access paths for the statements are locked in during package bind. They will not change on your production system unless you recompile the program or rebind.

This means you can use normal change control procedures to manage the Java programs, and can reasonably expect that the programs will run and perform the exact same way until the next time you make a conscious change (rebind or recompile).

SQLJ allows for better monitoring

With SQLJ, you get much better system monitoring and performance reporting. Static SQL packages give you the names of the programs that are running at any given point in time. This is extremely useful for studying CPU consumption by the various applications, locking issues (such as deadlock or time-out), etc.

For more information, see also:

`ftp://ftp.software.ibm.com/software/data/pubs/papers/javazos.pdf`

Comparing SQLJ and JDBC Coding

Redbooks

Retrieving a single row from DB2

SQLJ

```
String addr; String name;
....
#sql [ctx] {SELECT address INTO :addr FROM emp WHERE name =:name};
```

JDBC

```
String addr; String name;
....
PreparedStatement pstmt = con.prepareStatement(
      "SELECT address FROM     emp WHERE name = ?");
pstmt.setString(1, name);
ResultSet names = pstmt.executeQuery();
names.next();
addr = names.getString(1);
names.close();
pstmt.close();
```


5.2.3 Comparing coding SQLJ and JDBC

The visual above, as well as Example 5-6 below, compare how to code a single-row query, that is, a query returning exactly one row of data.

In JDBC, we have to open a result set, advance it to the next (and only) row, and retrieve the values using getxx methods. Also, we have to check if exactly one row has been found.

In SQLJ, on the other hand, we can use the SELECT INTO syntax; an SQLException will be thrown if more than one row was found.

By the way, the SQLJ version is more efficient as well. JDBC has to make four calls into DB2 (prepare statement, fetch row, fetch row, close statement), whereas the SQLJ version only has to do one single SELECT INTO call.

> **Note:** The SQLJ version will only be more efficient when the program has been customized and bound. If it is running uncustomized, it will emulate SELECT INTO by using result sets under the covers, just like the JDBC version.

Another example is shown in Example 5-6.

Example 5-6   JDBC vs. SQLJ: Single-row query

```
PreparedStatement stmt =
  conn.prepareStatement(
    "SELECT MAX(SALARY), AVG(SALARY)"
  + "  FROM DSN8810.EMP");
rs = statement.executeQuery();
if (!rs.next()) {
  // Error -- no rows found
}
maxSalary = rs.getBigDecimal(1);
avgSalary = rs.getBigDecimal(2);
if (rs.next()) {
  // Error -- more than one row found
}
rs.close();
stmt.close();
```

```
#sql {
  SELECT MAX(SALARY), AVG(SALARY)
    INTO :maxSalary, :avgSalary
    FROM DSN8810.EMP
};
```

As you can see, coding applications in SQLJ requires less code than coding in JDBC.

As a result, SQLJ statements that are directly embedded in Java programs show more concise and easy-to-read code than JDBC. This makes coding and code maintenance easier, especially when done by somebody that did not write the original code.

Below are some additional coding examples.

Example 5-7 shows a multi-row query. The amount of coding is similar with JDBC and SQLJ. Note, however, that the binding between statement and host variables in SQLJ is much tighter than between parameter markers and the setBigDecimal methods in JDBC. Also, JDBC uses statement handles that must be explicitly closed when you are done with the statement. In SQLJ, the translator automatically generates the cleanup code for you. (Iterators must still be closed explicitly, of course.)

Example 5-7   JDBC vs. SQLJ: Multi-row query

```
PreparedStatement stmt =
  conn.prepareStatement(
    "SELECT LASTNAME"
  + "     , FIRSTNME"
  + "     , SALARY"
  + "  FROM DSN8810.EMP"
  + " WHERE SALARY BETWEEN ? AND ?");

stmt.setBigDecimal(1, min);
stmt.setBigDecimal(2, max);

ResultSet rs = stmt.executeQuery();
while (rs.next()) {
   lastname = rs.getString(1);
   firstname = rs.getString(2);
   salary = rs.getBigDecimal(3);
  // Print row...
}
rs.close();
stmt.close();
```

```
EmployeeIterator iter;
#sql iter = {
  SELECT LASTNAME
       , FIRSTNME
       , SALARY
    FROM DSN8810.EMP
   WHERE SALARY BETWEEN :min AND :max
};

while (true) {
  #sql {
    FETCH :iter
     INTO :lastname, :firstname, :salary
  };
  if (iter.endFetch()) break;
  // Print row...
}
iter.close();
```

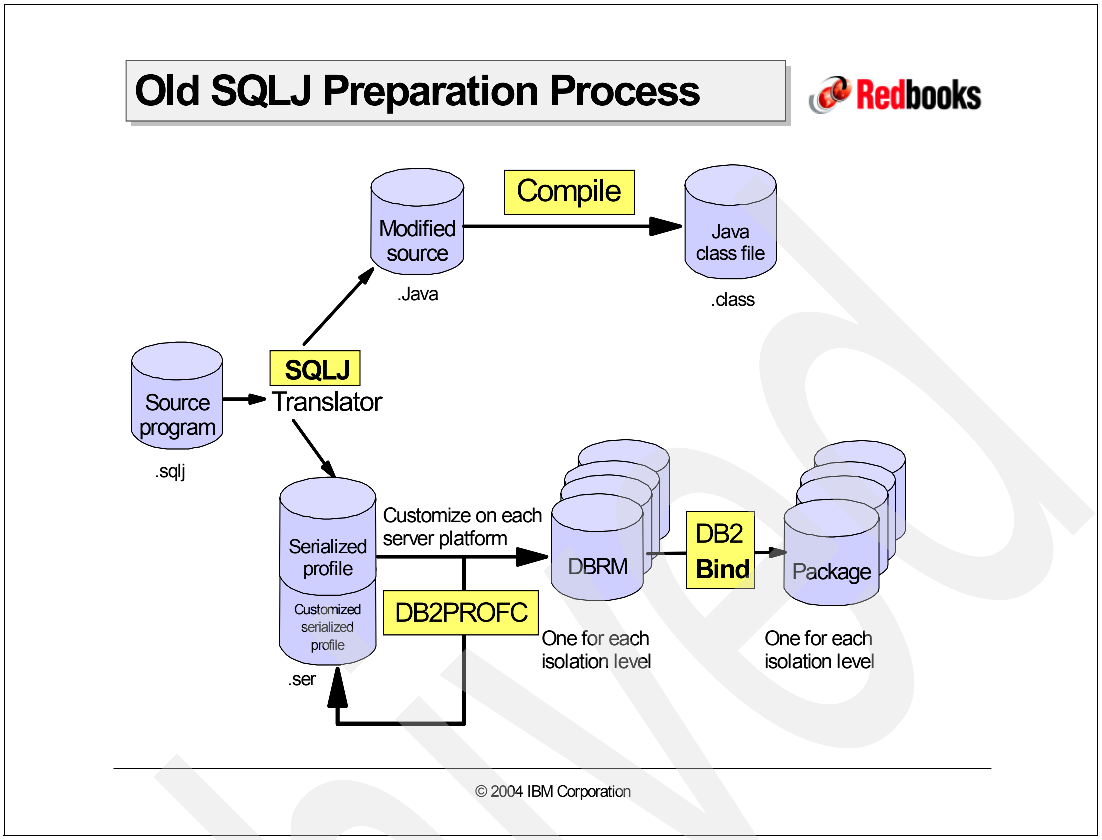
As our last example, consider the INSERT statement in Example 5-8. Again, the SQLJ code is easier to read and to maintain.

Example 5-8 JDBC vs. SQLJ: INSERT statement

```
stmt = conn.prepareStatement(
   "INSERT INTO DSN8810.EMP ("
  + "  EMPNO, FIRSTNME, MIDINIT"
  + ", LASTNAME, HIREDATE, SALARY) "
  + "VALUES (?, ?, ?, ?, CURRENT DATE, ?)"
);
stmt.setString(1, empno);
stmt.setString(2, firstname);
stmt.setString(3, midinit);
stmt.setString(4, lastname);
stmt.setBigDecimal(5, salary);
stmt.close();
```

```
#sql {
  INSERT INTO DSN8810.EMP (
      EMPNO, FIRSTNME, MIDINIT
    , LASTNAME, HIREDATE, SALARY)
  VALUES
    ( :empno, :firstname, :midinit
    , :lastname, CURRENT DATE, :salary)
};
```

Also note that people with a (static) embedded SQL programming background, for example, in COBOL, will find it very easy to start using SQLJ, as iterators and SELECT INTO constructs look very much like those in embedded SQL.



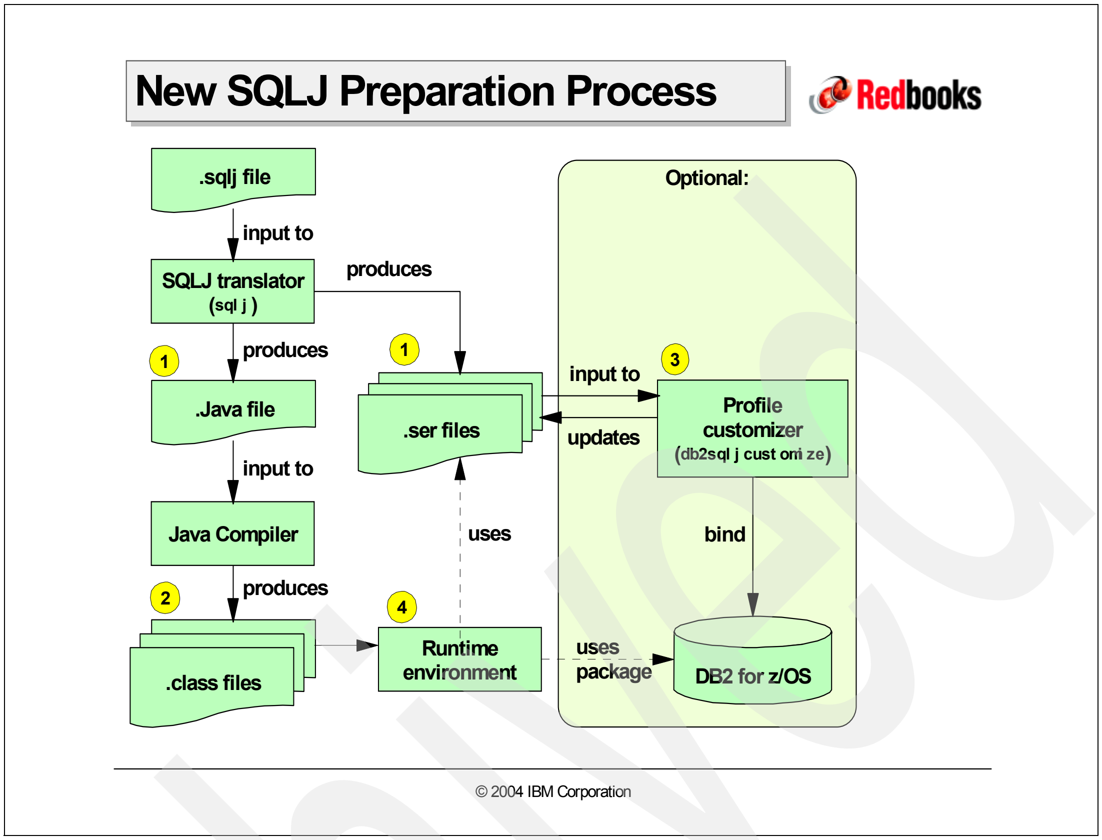
5.2.4 Application development using SQLJ

In this section we compare the current (pre-Universal Driver) program preparation process for SQLJ applications on z/OS with the Universal Driver's program preparation process.

SQLJ preparation process using the legacy DB2 for z/OS JDBC driver

The visual shows the Non-Universal Driver SQLJ program preparation process. After creating the serialized profile by means of the SQLJ translator, you have to execute the db2profc utility to create a DBRM, and then bind the DBRM into a set of packages (one package for each isolation level, UR, CS, RS, and RR). Even if you prefer to develop your Java applications on a workstation, the (uncustomized) serialized profile has to be shipped to the host before you can run the db2profc utility. This is because db2profc creates DBRMs, which are a unique feature of DB2 UDB for z/OS and OS/390 only.

The db2profc utility also customizes the serialized profile (by updating it). Unfortunately, after customization, the profile is no longer portable.



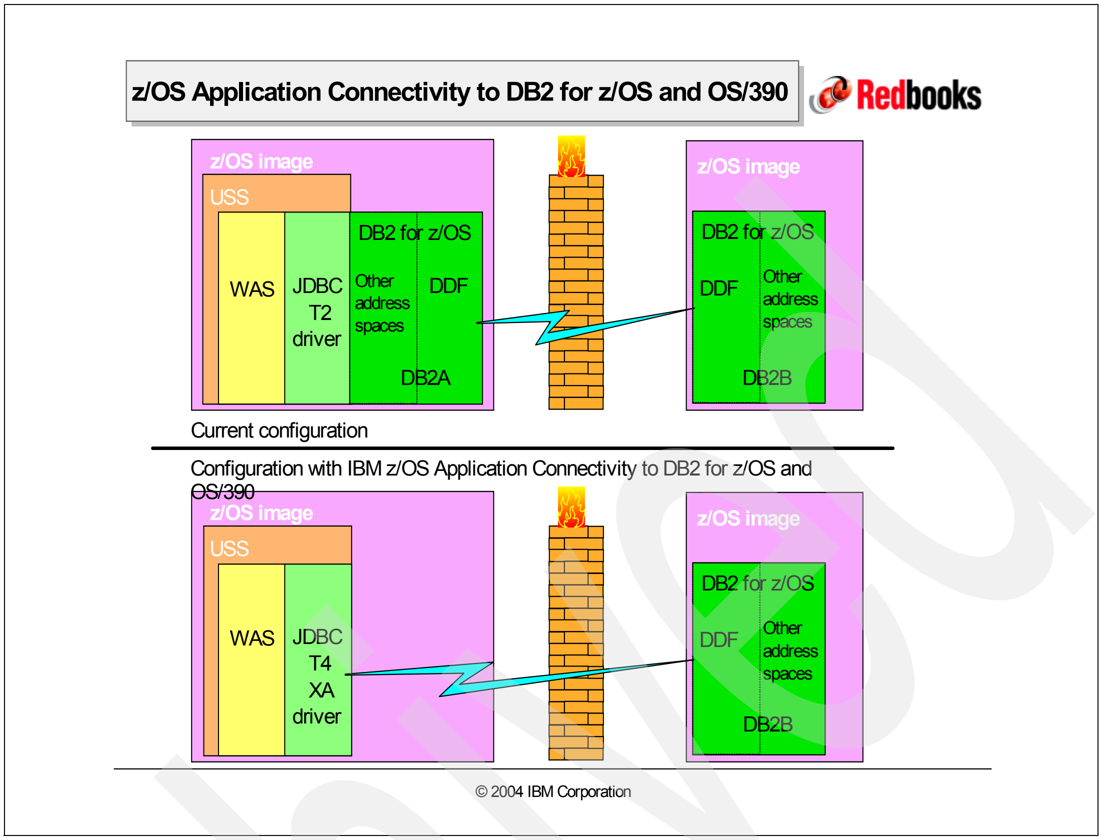
SQLJ preparation process using the Universal Driver

With the new Universal Driver, DBRMs (or .bnd files) are no longer used, as shown in the visual. Using the **db2sqljcustomize** command, you can customize the serialized profile and bind the packages at the same time against the target DB2 system. With the Type 4 driver, we connect from any platform directly to the target DB2 system, do the online checking (highly recommended), and bind the packages on the target DB2 system.

When you develop on the workstation, for example, using WebSphere Studio Application Developer (WSAD), you may now use the Type 4 driver to bind the packages against the DB2 UDB for z/OS system. You no longer have to ship the uncustomized profile to the z/OS system for customization.

In addition, the new Universal Driver customizes the serialized profile in such a way that it remains portable. You can execute using the same customized program files against any platform, as long as the db2sqlbind utility was used to connect to the new location and bind the correct program packages.

WSAD Version 5.1 provides support for this new application development scheme used by the Universal Driver for SQLJ and JDBC. Again, for more information, see *DB2 for z/OS and OS/390: Ready for Java*, SG24 6435.



5.3 z/OS Application Connectivity to DB2 for z/OS and OS/390

z/OS Application Connectivity to DB2 for z/OS and OS/390 is a no-charge, optional feature of DB2 Universal Database Server for z/OS and OS/390 Version 7, as well as DB2 for z/OS Version 8. The FMID is HDDA210 for both DB2 versions.

What is z/OS Application Connectivity to DB2 for z/OS and OS/390?

This feature consists of a component known as the DB2 Universal Database Driver for z/OS, Java Edition. This is a pure Java, Type 4 JDBC driver, designed to deliver high performance and scalable remote connectivity for Java-based enterprise applications on z/OS to a remote DB2 for z/OS database server. This flavor of the DB2 Universal Driver for SQLJ and JDBC is a stand-alone version of the T4 driver, especially packaged to work on the z/OS platform, with the following features:

- It supports JDBC 2.0 and 3.0 specification and JDK V1.4 to deliver the maximum flexibility and performance required for enterprise applications.
- It provides distributed transaction support (two-phase commit support).
- It delivers robust connectivity to the latest DB2 for z/OS and WebSphere Application Server for z/OS (V5.0.2).
- It enables custom Java applications that do not require an application server to run in a remote partition and connect to DB2 z/OS.

Why do I want to use it?

People who need very high availability run their WebSphere Application Servers (WAS) on zSeries hardware in a Parallel Sysplex® configuration.

Many customers also have security guidelines to separate their Web server and database server. Therefore, both servers are not allowed to run on the same machine and must be separated by a firewall (as shown in the top part of the figure above).

In this case a WebSphere Application Server runs your Java applications and talks to a Type 2 JDBC driver. The JDBC driver talks through RRS to a DB2 system (DB2A) that is local to the machine running the WebSphere Application Server. This local DB2 then routes all requests to a remote database server (DB2B) using DRDA. In this configuration, DB2A does not have any databases that are accessed by the applications running on the WebSphere Application Server. All SQL requests are routed through the local DB2A, but are accessing data on the remote DB2B through DRDA.

In this configuration, we need the DRDA Application Requester functionality of DB2A to access the data residing on DB2B. This is because the current (non-Universal) JDBC driver for z/OS and OS/390 is a Type 2 driver and cannot directly access a remote DB2 system (DB2B).

With IBM z/OS Application Connectivity to DB2 for z/OS and OS/390 (see the bottom part of the figure above), you no longer need a local DB2 (DB2A) on the same machine as your WebSphere Application Server. This no-charge feature of DB2 for z/OS and OS/390 V7 or DB2 for z/OS Version 8 provides a Type 4 JDBC driver that supports two-phase commit. This driver is sometimes (unofficially) called the Type 4 XA driver.

Your Java applications running inside the WebSphere Application Server talk to the (Universal) Type 4 JDBC driver that supports two-phase commit, and the driver talks directly to the remote database server (DB2B) through DRDA. The Universal Type 4 driver implements DRDA Application Requester functionality. You no longer need to buy a DB2 license for DB2A when using the DB2 Universal Database Driver for z/OS, Java Edition. It is sufficient to have a DB2 license for DB2B.

The z/OS Application Connectivity to DB2 for z/OS and OS/390 feature provides connectivity from a z/OS or OS/390 remote partition or system only. For access from any other operating system or platform, including z/Linux, to DB2 for z/OS and OS/390, you must obtain a separate license of the edition of DB2 Connect that is appropriate for your environment.

This feature can only be ordered for DB2 UDB for z/OS and OS/390 V7 or DB2 for z/OS V8.

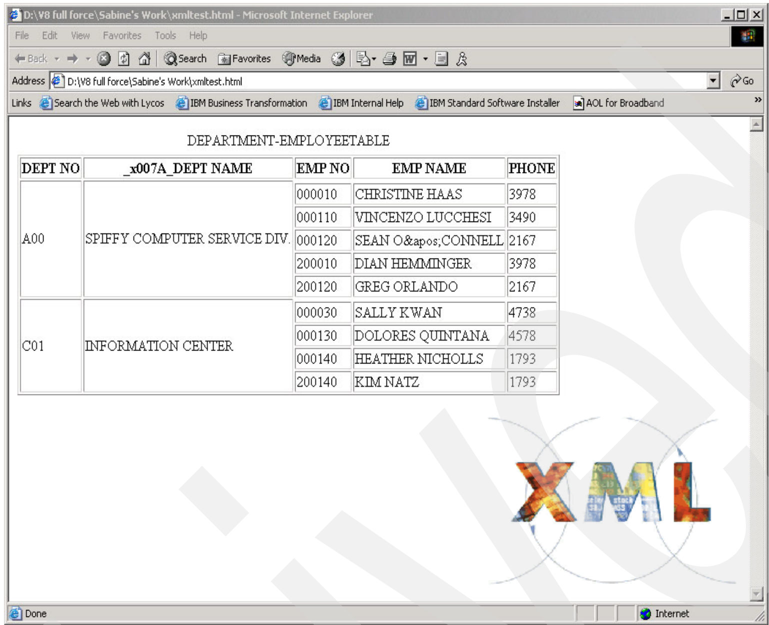
DB2 UDB for OS/390 and z/OS V7 servers also do not have built-in support for distributed transactions that implement the XA specification. In this case, the DB2 Universal JDBC Driver supports distributed transactions (two-phase commit) through emulation.

DB2 UDB for z/OS Version 8 has native XA two-phase commit support in DRDA.

Early measurements have shown that using the z/OS Application Connectivity to DB2 for z/OS and OS/390 (T4 XA driver for z/OS and OS/390) provides better performance than having to go through a local T2 driver attached to a local DB2 subsystem that routes all SQL requests to the remote DB2 database server.

Note that the 2-phase commit support is also provided with the Universal Driver that ships with DB2 for z/OS Version 8 (FMID JDB8812).

XML Publishing Functions



DEPT NO	_x007A_DEPT NAME	EMP NO	EMP NAME	PHONE
A00	SPIFFY COMPUTER SERVICE DIV.	000010	CHRISTINE HAAS	3978
		000110	VINCENZO LUCCHESI	3490
		000120	SEAN O'CONNELL	2167
		200010	DIAN HEMMINGER	3978
		200120	GREG ORLANDO	2167
C01	INFORMATION CENTER	000030	SALLY KWAN	4738
		000130	DOLORES QUINTANA	4578
		000140	HEATHER NICHOLLS	1793
		200140	KIM NATZ	1793

5.4 New integrated XML publishing functions

For the past few years, XML has been increasingly become the de facto data format on the Internet, on corporate intranets, and for data exchange. In DB2 UDB for OS/390 and z/OS V7, if you need to create XML data from traditional relational databases (this is called *XML publishing* or *XML composition*), you must create your own application that converts the DB2 data to the XML format, or use DB2 XML Extender. DB2 V8 provides you with an additional option. DB2 V8 contains a set of brand new built-in functions to help with XML publishing. These DB2 built-in functions reduce your application development efforts in generating XML data from relational data with high performance, and enable the development of lightweight applications.

This set of built-in functions are part of a set of extensions to the SQL language, called SQL/XML. SQL/XML is an emerging standard, and is part of the ANSI and ISO SQL standard, describing the ways the database language SQL can be used in conjunction with XML. The definition of SQL/XML is driven in part by the SQLX Group, of which IBM is a very active member.

The SQL/XML publishing functions are built-in DB2 functions. They run inside the DB2 address spaces, unlike external user-defined functions (UDFs) that run in a WLM managed address space outside of DB2, like the ones used by DB2 XML Extender. The fact that the XML publishing functions are built into the DB2 engine gives them better performance. In addition, much extra work has been done inside the DB2 engine, for example, to make the tagging as efficient as possible.

Seven new built-in functions related to XML publishing can be used with DB2 V8:

- Cast function:
  – XML2CLOB
- Scalar functions:
  – XMLELEMENT
  – XMLATTRIBUTES
  – XMLFOREST
  – XMLCONCAT
  – XMLNAMESPACES
- Aggregate function:
  – XMLAGG

Refer to the subsequent pages within this chapter for more information about how to use the new XML functions.

The visual above shows the power of the SQL/XML functions. The XML query statement to generate this HTML table is shown in Example 5-9.

Example 5-9 Complex XML query example

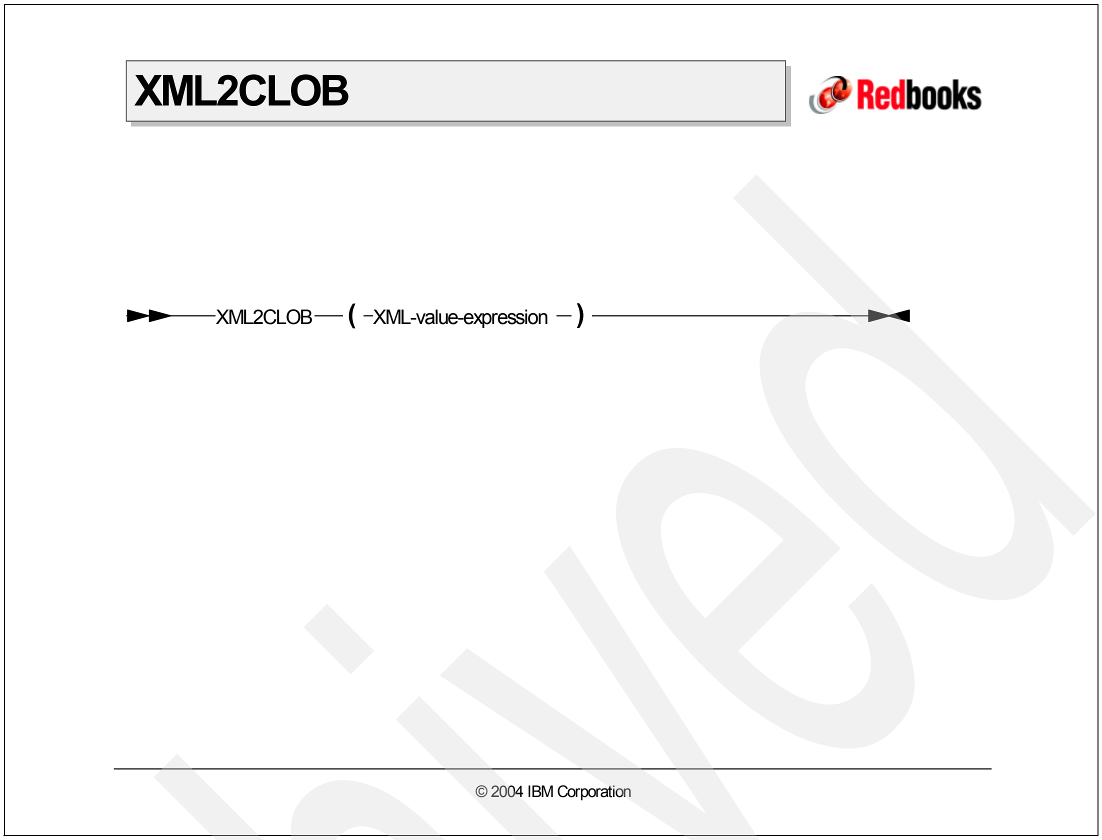
```
SELECT VARCHAR( XML2CLOB( XMLElement(NAME "TABLE",
                     XMLATTRIBUTES('1' as "border"),
                     XMLElement(NAME CAPTION, 'Department-Employee Table'),
                     XMLElement(NAME TR, XMLFOREST('Dept No' as TH, 'Dept Name' as TH,
                                        'Emp No' as TH, 'Emp Name' as TH, 'Phone' as TH) ),
                     XMLAGG(
                     XMLCONCAT(
                     XMLELEMENT(NAME TR, XMLELEMENT(NAME TD,
                                         XMLATTRIBUTES( X.CNT+1 as "rowspan"),
                                            D.DEPTNO),
                                    XMLELEMENT(NAME TD,
                                              XMLATTRIBUTES( X.CNT+1 as "rowspan"),
                                                     D.DEPTNAME)
                                            ),
                     ( SELECT  XMLAGG(XMLElement(NAME TR,
                                             XMLForest(EMPNO as TD,
                                                       FIRSTNME || ' ' || LASTNAME as TD,
                                                       PHONENO as TD)  )  )
                       FROM DSN8810.EMP E
                       WHERE  E.WORKDEPT = D.DEPTNO )
                         ) ) ) ) )
FROM DSN8810.DEPT D, (SELECT WORKDEPT, COUNT(*)
                      FROM DSN8810.EMP GROUP BY WORKDEPT) X(DEPTNO, CNT)
WHERE D.DEPTNO = X.DEPTNO AND
              D.DEPTNO IN ('A00', 'C01')
```

Note that we use SQL/XML to generate the HTML tags necessary to be able to display the result of the query as an HTML table.

These SQL/XML functions can be used instead of the XML publishing functions that are provided by DB2 XML Extender. The DB2 XML Extender publishing functions will still be available for you to use, but the SQL/XML functions provide more flexibility, and as they are built into the DB2 engine, they are likely to provide better performance.

SQL/XML publishing functions can only be used to generate XML documents (or fragments) from data that is stored in relational tables. They cannot be used to store XML data in DB2 tables. For those functions, you need to use DB2 XML Extender's XML collection and XML column functionality. The same is true for XML transformation functions. SQL/XML is not intended to be used for transformation. Again, you can use DB2 XML Extender for that purpose. As before, DB2 XML Extender ships as part of DB2 at no extra charge.

Note: These SQL/XML functions do not require the XML Toolkit for z/OS.



5.4.1 XML2CLOB

The XML data type is a new data type introduced by DB2 V8. However, it is not like any other existing data type. It is a so-called *transient* data type. Transient means that this data type only exists during query processing. There is no persistent data of this type and it is not an external data type that can be declared in application programs. In other words, the XML data type cannot be stored in a database or returned to an application.

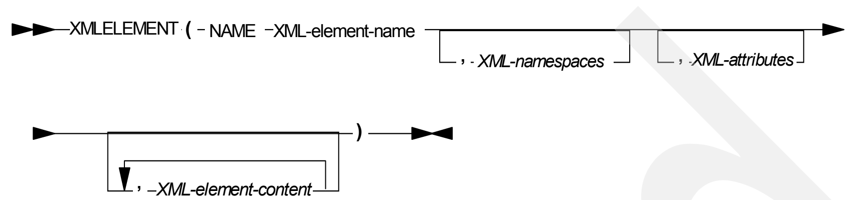
To allow an application to deal with the result of a SQL/XML function (that results in a value with an XML data type), DB2 supplies a new conversion function XML2CLOB, which converts an XML value into a CLOB.

There are some restrictions that apply to the transient XML data type:

- A query result cannot contain this type.
- The columns of a view cannot be of this type.
- XML data cannot be used in SORT (GROUP BY and ORDER BY).
- XML data cannot be used in predicates.
- The XML data type is not compatible with any other data types. The only cast function that may be used is XML2CLOB.

The resulting CLOB is MIXED character data and the CCSID is the mixed CCSID for UNICODE encoding scheme UTF-8 (CCSID 1208). The maximum length of the resulting CLOB is 2 GB -1.

XMLELEMENT




```
SELECT E.EMPNO,XML2CLOB(
    XMLELEMENT ( NAME "EMP", E.FIRSTNME||' '||E.LASTNAME
    )) AS "RESULT"
FROM DSN8810.EMP E;
---------+---------+---------+---------+---------+---------
EMPNO    RESULT
---------+---------+---------+---------+---------+---------
000010   <EMP>CHRISTINE HAAS</EMP>
000020   <EMP>MICHAEL THOMPSON</EMP>
000030   <EMP>SALLY KWAN</EMP>
000050   <EMP>JOHN GEYER</EMP>
```


5.4.2 XMLELEMENT

The visual shows the syntax and a usage example of the XMLELEMENT built-in function.

The XMLELEMENT function returns an XML element from one or more arguments. The arguments can be:

- An element name
- An optional collection of attributes
- Zero or more arguments that make up the element’s content.

The result type is the transient XML data type.

Let us now take a look at the components of the XMLELEMENT function:

- NAME:

  NAME keyword marks the identifier that is supplied to XMLELEMENT for the element name.

- *XML-element-name:*

  Specifies an identifier that is used as the XML element name. (No mapping is applied to this identifier.)

- *XML-namespaces:*

  Specifies the XML namespace for the XML element. See the XMLNAMESPACES function later in the chapter for details.

- *XML-attributes:*

  Specifies the attributes for the XML element. See the XMLATTRIBUTES function below.

- *XML-element-content:*

  Specifies an expression making up the XML element content. The expression *cannot be*:

  – A ROWID
  – A character string defined with the FOR BIT DATA attribute
  – A BLOB
  – A distinct type sourced on these types

  If the result of the expression is an SQL value, it is mapped to the XML value according to the mapping rules from an SQL value to an XML value (see 5.4.8, “Mappings from SQL to XML” on page 447 for details).

  If multiple XML-element-contents are specified, their XML values are concatenated to form the content of the XML element. If the result of an expression is a null value, it is not included in the concatenation result. If all the results of the arguments are the null value, then the result of XMLELEMENT is an element with empty content.

The result of the XMLELEMENT function cannot be null.

Refer to the SELECT statement shown on the visual for a short and simple example of the use of the XML2CLOB and XMLELEMENT function. As you can see in the SQL statement above, the XMLELEMENT function is used to create an element called EMP, which contains the concatenation of the contents of columns FIRSTNME and LASTNAME.

Example 5-10 shows a more complex SELECT statement using multiple elements.

Example 5-10 Nested elements

```
SELECT e.empno, XML2CLOB(
      XMLELEMENT( NAME "Emp",
         XMLELEMENT ( NAME "name", e.firstnme ||' ' ||e.lastname ),
         XMLELEMENT ( NAME "hiredate", e.hiredate )
                )
      ) AS "Result"
FROM dsn8810.emp e;
```

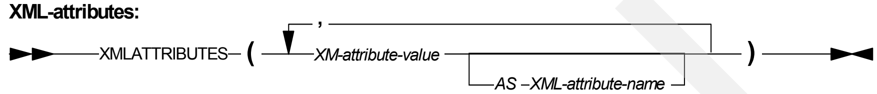
As you can see, element `<Emp>` itself contains two nested elements `<name>` and `<hiredate>`. Example 5-11 shows the result of this SELECT statement:

Example 5-11 Output of nested elements example

```
---------+---------+---------+---------+---------+---------+---------+---------+
EMPNO   RESULT
---------+---------+---------+---------+---------+---------+---------+---------+
000010  <EMP><NAME>CHRISTINE HAAS</NAME><HIREDATE>1965-01-01</HIREDATE></EMP>
000020  <EMP><NAME>MICHAEL THOMPSON</NAME><HIREDATE>1973-10-10</HIREDATE></EMP>
000030  <EMP><NAME>SALLY KWAN</NAME><HIREDATE>1975-04-05</HIREDATE></EMP>
```

XMLATTRIBUTES

XML-attributes:




```
SELECT E.EMPNO,XML2CLOB(
    XMLELEMENT ( NAME "EMP",
          XMLATTRIBUTES( E.EMPNO, E.FIRSTNME||' '||E.LASTNAME AS "NAME")
                )        )
                 AS "RESULT"
FROM DSN8810.EMP E;
---------+---------+---------+---------+---------+---------+---------+--
EMPNO   RESULT
---------+---------+---------+---------+---------+---------+---------+--
000010  <EMP EMPNO="000010" NAME="CHRISTINE HAAS"></EMP>
000020  <EMP EMPNO="000020" NAME="MICHAEL THOMPSON"></EMP>
000030  <EMP EMPNO="000030" NAME="SALLY KWAN"></EMP>
```


5.4.3 XMLATTRIBUTES

This function constructs XML attributes from the arguments. It can only be used as the second argument to the XMLELEMENT function.

- *XML-attribute-value:*

  Specifies the value of the attribute. This expression *cannot be*:

  – A ROWID
  – A character string defined with the FOR BIT DATA attribute
  – A BLOB
  – A distinct type sourced on these types, or XML

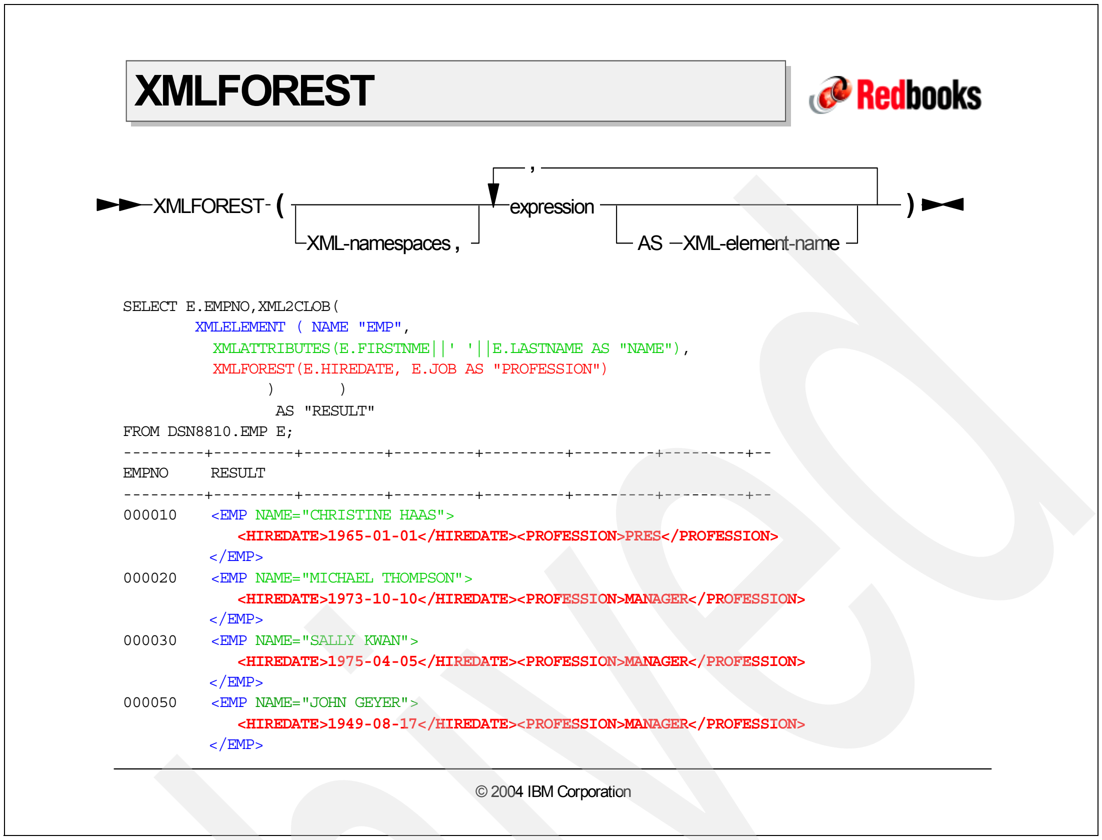
  The result of the expression is mapped to an XML value according to the mapping rules from an SQL value to an XML value. If the value is null, the corresponding XML attribute is not included in the XML element.

- *AS XML-attribute-name:*

  Specifies an identifier that is used as the attribute name. (No mapping is applied to map the identifier to an XML name.)

  If XML-attribute-name is not specified, the expression for XML-attribute-value must be a column name, and the attribute name will be created from the column name using the fully escaped mapping from a column name to an XML attribute name.

Refer to the visual above for the syntax diagram and a usage sample. As you can see, XMLATTRIBUTES adds the attributes EMPNO and NAME (which is a combination of FIRSTNAME and LASTNAME) to the XML element <EMP>



5.4.4 XMLFOREST

The XMLFOREST function returns a bunch of XML elements that all share a specific pattern from a list of expressions, one element for each argument.

- *Expression:*

  Specifies an expression that is used as an XML element content. The result of the expression is mapped to an XML value according to the mapping rules from an SQL value to an XML value. The expression *cannot be*:

  – A ROWID
  – A character string defined with the FOR BIT DATA attribute
  – A BLOB
  – A distinct type sourced on these types

  If the result of an expression is null, then it is not included in the concatenation result for XMLFOREST.

- *AS XML-element-name:*

  Specifies an identifier that is used for the XML element name. (No mapping is applied to map the identifier to an XML name.)

  If XML-element-name is not specified, the expression must be a column name, and the element name will be created from the column name. The fully escaped mapping is used to map the column name to an XML element name.

- *XML-namespaces:*

  Specifies the XML namespace for the XML element. See the XMLNAMESPACES function later in the chapter for details.

The figure above shows the syntax diagram and a usage example. This sample generates an EMP element for each employee. It uses the employee name as its attribute and two subelements that are generated from columns HIREDATE and JOB by using XMLFOREST as its content. The element names for the two subelements are ‘HIREDATE’ and ‘PROFESSION’.

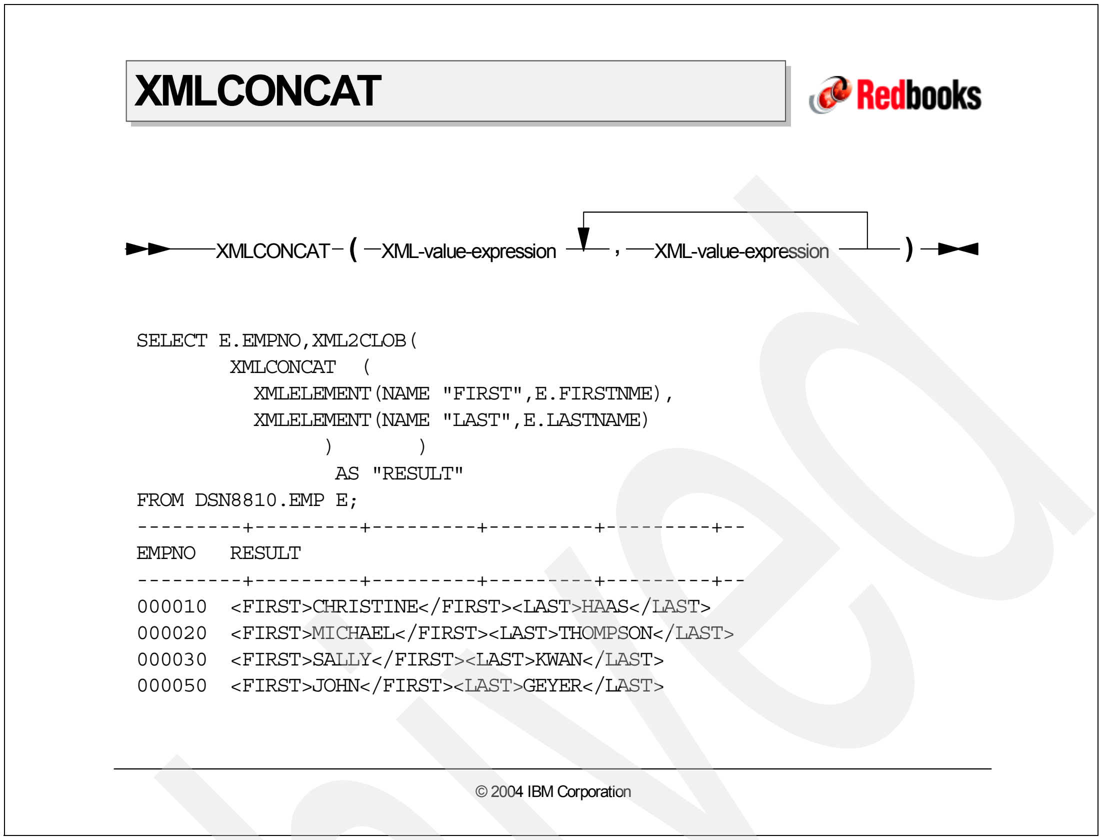
You can also produce the same result if you use two additional nested XMLELEMENT statements instead of XMLFOREST. Example 5-12 shows a coding example for that.

Example 5-12 Nested elements instead of XMLFOREST

```
SELECT E.EMPNO,XML2CLOB(
        XMLELEMENT ( NAME "EMP",
          XMLATTRIBUTES(E.FIRSTNME||' '||E.LASTNAME AS "NAME"),
            XMLELEMENT(NAME "HIREDATE", E.HIREDATE),
            XMLELEMENT(NAME "PROFESSION", E.JOB)
                )       )
                 AS "RESULT"
FROM DSN8810.EMP E;
```

Note: The generated element names are folded to uppercase. If you want them to be lowercase or mixed, you must use quotes (“department”)

Note: In the examples used in this section, there would be a difference between XMLFOREST and XMLELEMENT if there were NULL values in HIREDATE and JOB. XMLFOREST ignores the NULL value (not included in the result) and XMLELEMENT results in an empty element.



5.4.5 XMLCONCAT

The XMLCONCAT function returns a forest of XML elements that are generated from a concatenation of two or more arguments. A syntax diagram is shown in the figure above. In this coding example:

- *XML-value-expression:*

  Specifies an expression whose value is the XML data type. If the value of XML-value-expression is null, it is not included in the concatenation.

The result type of XMLCONCAT is the transient XML data type. If all of the arguments are null, then the null value is returned.

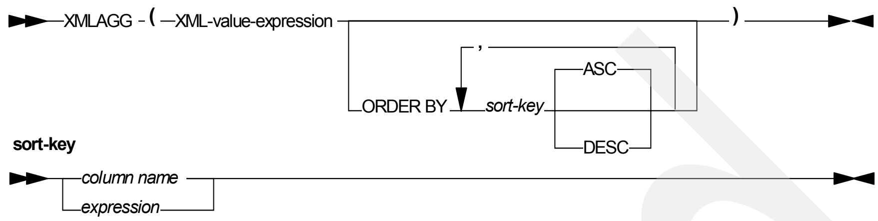
Example 5-13 shows an example of XMLFOREST, which produces the same result as the XMLCONCAT example shown on the visual.

Example 5-13 XMLFOREST instead of XMLCONCAT

```
SELECT E.EMPNO,XML2CLOB(
          XMLFOREST(E.FIRSTNME AS "FIRST", E.LASTNAME AS "LAST")
                        )
                 AS "RESULT"
FROM DSN8810.EMP E WHERE LASTNAME = 'HAAS';
```

Note: One reason for using XMLCONCAT instead of XMLFOREST is that XMLFOREST cannot generate XML elements with attributes. For this purpose, use XMLELEMENT.

XMLAGG




```
SELECT XML2CLOB(
          XMLELEMENT(NAME "PROFESSION",
                    XMLATTRIBUTES(E.JOB AS "NAME"),
                    XMLAGG(XMLELEMENT (NAME "EMP", E.LASTNAME)
                           ORDER BY E.LASTNAME
                          )))
                 AS "PROFESSIONLIST"
FROM DSN8810.EMP E
GROUP BY JOB;
---------+---------+---------+---------+---------+---------+---------+---------+---------+-----------
PROFESSIONLIST
---------+---------+---------+---------+---------+---------+---------+---------+---------+-----------
<PROFESSION NAME="ANALYST "><EMP>NATZ</EMP><EMP>NICHOLLS</EMP><EMP>QUINTANA</EMP></PROFESSION>
<PROFESSION NAME="CLERK   "><EMP>JEFFERSON</EMP><EMP>JOHNSON</EMP><EMP>MARINO</EMP><EMP>MONTEVERDE</EMP>
                           <EMP>O&apos;CONNELL</EMP><EMP>ORLANDO</EMP></PROFESSION>
<PROFESSION NAME="DESIGNER"><EMP>ADAMSON</EMP><EMP>BROWN</EMP><EMP>JOHN</EMP><EMP>JONES</EMP><EMP>LUTZ
                           </EMP><EMP>PIANKA</EMP><EMP>SCOUTTEN</EMP><EMP>WALKER</EMP><EMP>YAMAMOTO
                           </EMP><EMP>YOSHIMURA</EMP></PROFESSION>
```


5.4.6 XMLAGG

The visual shows a syntax diagram and a usage example for the XMLAGG function, which returns a concatenation of XML elements from a collection of XLM elements.

The XMLAGG function has one argument with an optional ORDER BY clause. The ORDER BY clause specifies the ordering of the rows from the same grouping set to be processed in the aggregation. If the ORDER BY clause is not specified, or the ORDER BY clause cannot differentiate the order of the sort key value, the order of rows from the same group to be processed in the aggregation is arbitrary.

- *XML-value-expression:*

  Specifies an expression whose value is the transient XML data type. Different from other column functions, a scalar fullselect is allowed as an argument to XMLAGG. The function is applied to the set of values derived from the argument values by the elimination of null values. If all inputs are null, or there are no rows, then the result of XMLAGG is null.

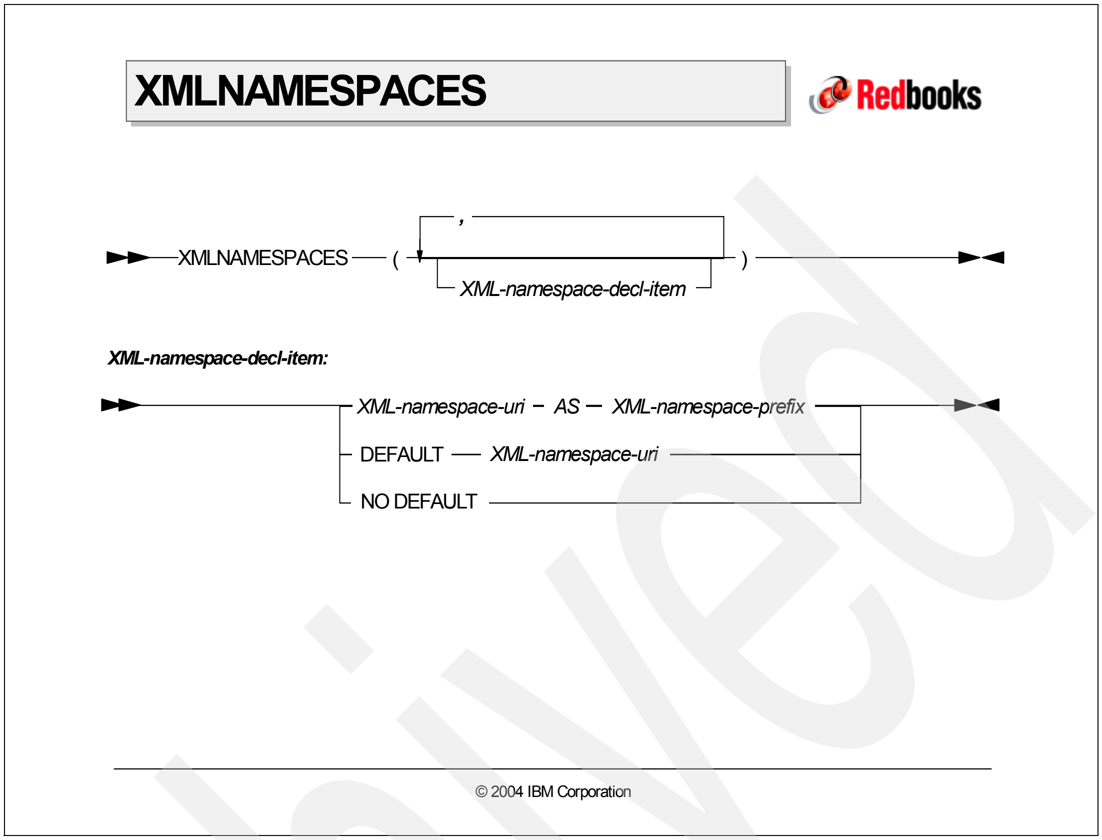
- *sort-key:*

  Specifies a sort-key that is either a column name or an expression. The ordering is based on the SQL values of the sort keys, which may or may not be used in the XML value expression. If the sort-key is a constant, it does not refer to the position of the output column as in the ORDER BY clause of a SELECT statement, and it has no impact on the ordering. You cannot use a CLOB value as a sort key. A character string expression cannot have a length greater than 4000 bytes.

If the sort key is a character string that uses an encoding scheme other than Unicode, the ordering might be different. For example, a column PRODCODE uses EBCDIC. For two values "P001" and "PA01", the relationship "P001" > "PA01" is true in EBCDIC, whereas in Unicode UTF-8 "P001" < "PA01" is true. If the same sort key values are used in the XML value expression, use the CAST function to convert the sort key to Unicode to keep the ordering of XML values consistent with that of the sort key.

The result type is XML. The result can be NULL.

In this example, the employees are grouped by their department. We generate a department element for each department and nest all the emp elements for employees in each department. In addition, all emp elements are ordered by lname.



5.4.7 XMLNAMESPACES

An *XML namespace* is a collection of element type and attribute names. The namespace is identified by a unique name, which is a Uniform Resource Identifier (URI). If you use XML namespaces, any element type or attribute name can be uniquely identified by a two-part name (also known as the *expanded name*): the name of its XML namespace and its local name.

URI references are often inconveniently long, so expanded names are not used directly to name elements and attributes in XML documents. Instead, qualified names are used. A *qualified name* is a name subject to namespace interpretation. In documents conforming to this specification, element and attribute names appear as qualified names. Syntactically, they are either prefixed names or unprefixed names.

QName(Qualified Names):

- PrefixedName: prefix:local part
- UnprefixedName: local part
- Local part: local name (NCName or “non-colonized” name)

The URIs used as XML namespace names are just identifiers, which are not guaranteed to point to schemas, information about the namespace, or anything else. That is, the XML namespaces recommendation does not define anything except a two-part naming system for element types and attributes.

Note: Do not mix up URIs and URLs. URIs are Uniform Resource Identifiers, which are generally not used for lookup, but as reference. In contrast to that, URLs are Uniform Resource Locators, which are, for example, used to locate Web addresses.

The usage of XML namespaces allows people to do several things, such as:

- Combine fragments from different documents without any naming conflicts.
- Write reusable code modules that can be invoked for specific elements and attributes.
- Define elements and attributes that can be reused in other schemas or instance documents without fear of name collisions.

Example 5-14 and Example 5-15 show two XML documents both using Address as XML element name.

Example 5-14 Element Address meaning #1

```
<?xml version="1.0" ?>
  <Address>
     <Street>Theodor-Althoff-Str. 1</Street>
     <City>Essen</City>
     <State>Nordrhein-Westfalen</State>
     <Country>Germany</Country>
     <PostalCode>D-45133</PostalCode>
  </Address>
```

Example 5-15 Element Address meaning #2

```
<?xml version="1.0" ?>
  <Server>
     <Name>OurWebServer</Name>
     <Address>123.45.67.8</Address>
  </Server>
```

Both XML documents use the same element type Address. However, the meaning of both element types is different and should be interpreted differently by an application. This construct is fine, as long as both element types are used in separate documents. Once they are combined in one document, an application would not know which Address element is to be processed.

One solution for this problem could be renaming one of the element types. If you refer to our example above, the Address element type of document one could be renamed to STREETAddress. This, however, is not very satisfactory. As a long term solution, a much better way to deal with this problem is to assign different XML namespaces to each document.

The XML namespace declaration attribute uses the following syntax:

- xmlns:prefix=“URI”

To use a default namespace, use the following syntax:

- xmlns=“URI”

When you do not allow for a default namespace, use:

- xmlns=“”

In these expressions, the URI reference is the namespace name.

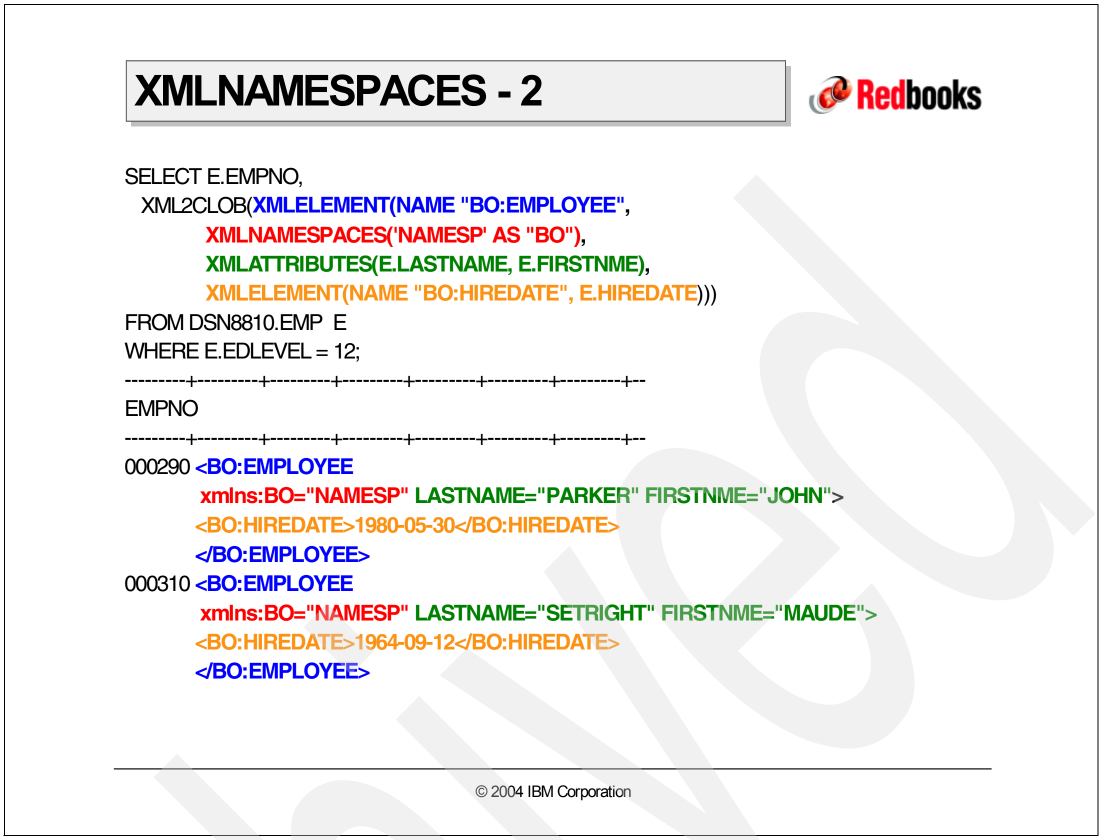
As a result, the combination of both documents of Example 5-14 and Example 5-15 shown above could look as shown in Example 5-16:

Example 5-16 XML namespace usage

```
<Department>
     <Name>DVS1</Name>
     <addr:Address xmlns:addr="http://www.ibm.de/Essen">
        <addr:Street>Theodor-Althoff-Str. 1</addr:Street>
        <addr:City>Essen</addr:City>
        <addr:State>Nordrhein-Westfalen</addr:State>
        <addr:Country>Germany</addr:Country>
        <addr:PostalCode>D-45133</addr:PostalCode>
     </addr:Address>
     <serv:Server xmlns:serv="http://www.ibm.de/servers">
        <serv:Name>OurWebServer</serv:Name>
        <serv:Address>123.45.67.8</serv:Address>
     </serv:Server>
  </Department>
```

The previous visual shows the syntax of the XMLNAMESPACES publishing function in DB2 for z/OS.

XMLNAMESPACES can only be used within the XMLELEMENT or XMLFOREST function. In XMLELEMENT, XMLNAMESPACES can only be used as the second argument. While in XMLFOREST, it can only be the first argument.



The visual above shows a simple example of one of the two possibilities to use the XMLNAMESPACES function, that is, as the second argument of XMLELEMENT. The name of the element is <BO:EMPLOYEE>. The namespace, which is called BO, contains two attributes, FIRSTNAME and LASTNAME, and one additional element, BO:HIREDATE.

The usage of XMLNAMESPACES in conjunction with XMLFOREST is shown in Example 5-17:

Example 5-17 XMLNAMESPACES in XMLFOREST

```
SELECT empno,
XML2CLOB(XMLFOREST(
XMLNAMESPACES(DEFAULT 'http://hr.org', 'http://fed.gov' AS "d"),
lastname, job AS "d:job"))
FROM employee where edlevel = 12;
```

The result of the query would be similar to the result shown in Example 5-18:

Example 5-18 RESULT of XMLNAMESPACES in XMLFOREST

```
00029 <LASTNAME
        xmlns="http://hr.org"
        xmlns:d="http://fed.gov">PARKER
    </LASTNAME>
    <d:job
        xmlns="http://hr.org"
        xmlns:d="http://fed.gov"> OPERATOR
    </d:job>
00031 <LASTNAME
        xmlns="http://hr.org"
        xmlns:d="http://fed.gov"> SETRIGHT
    </LASTNAME>
    <d:job
        xmlns="http://hr.org"
        xmlns:d="http://fed.gov"> OPERATOR
</d:job>
```

These XML publishing functions complement the functionality delivered by the XML Extender product. For more information, see *DB2 UDB for z/OS Version 8 XML Extender Administration and Programming*, SC18-7431.

Mapping Character Sets

Redbooks

SQL character data is mapped to UTF-8 XML data:

SQL data	hexadecimal representation in EBCDIC 37	hexadecimal representation in UTF-8
JAMES	x'D1C1D4C5E2'	x'4A414D4553'
EMP_NO	x'C5D4D76DD5D6'	x'454D505F4E4F'
§_123	x'B56DF1F2F3'	x'C2575F313233' where C257 represents the § sign

Characters shown on 3270/37 without conversion services:

hexadecimal representation in UTF-8	What do you see?
x'4A414D4553	JAMES
x'454D505F4E4F'	EMP_NO
x'C2575F313233',	_123 (read: blank underscore one two three)

5.4.8 Mappings from SQL to XML

To construct XML data from SQL data, the following mappings are performed:

- SQL character sets to XML character sets
- SQL identifiers to XML names
- SQL data values to XML data values

These mappings are based on industry standards. For complete information, see Information technology - Database languages - SQL- Part 14: XML-Related Specifications (SQL/XML) ISO/IEC 9075-14:2003.

Mapping SQL character sets to XML character sets

In DB2 UDB for z/OS, the character set used for XML data is Unicode UTF-8. Therefore, every time you extract SQL data and construct XML data out of it, the SQL character data (when not in UTF-8) format is converted into Unicode UTF-8 while executing the SQL/XML functions.

In Example 5-19, we use a DB2 subsystem with SCCSID 37, and the CCSID of the host emulator is also set to 37. The EBCDIC table we use is called BSXML. The table has a single character column COL1, and contains a single row with the value ‘§_123’.

Example 5-19 Create table

```
CREATE TABLE BSXML (COL1 CHAR(20));
INSERT INTO BSXML VALUES X'B56DF1F2F3'; -- use hex to be sure
SELECT * FROM BSXML;
---------+---------+---------+---------+---------+---------+
```

```
COL1
---------+---------+---------+---------+---------+---------+
§_123
DSNE610I NUMBER OF ROWS DISPLAYED IS 1
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
```

The top query in Example 5-20 shows the result of using XMLELEMENT on this column. As you can see, even though the data was in UTF-8 format while processing the SQL/XML functions, the result gets converted back to application encoding scheme (EBCDIC), and the result is readable.

Example 5-20 Using XMLELEMENT

```
SELECT XML2CLOB( XMLELEMENT ( NAME "EMP", COL1) ) AS "RESULT"
   FROM BSXML;
---------+---------+---------+---------+--
RESULT
---------+---------+---------+---------+--
<EMP>§_123               </EMP>

or in hex edit format

-------------------------------
<EMP>§_123               </EMP>
4CDD6B6FFF444444444444444446CDD6
C547E5D123000000000000000C1547E
 ------------------------------



SELECT HEX ( CHAR (XML2CLOB ( XMLELEMENT ( NAME "EMP", COL1 ))))AS "RESULT"
   FROM BSXML
---------+---------+---------+---------+---------+---------+----
RESULT
---------+---------+---------+---------+---------+---------+----
3C454D503EC2A75F3132332020202020202020202020202020203C2F454D503E
DSNE610I NUMBER OF ROWS DISPLAYED IS 1
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
---------+---------+---------+---------+---------+---------+----
```

The bottom query shows the result of the SQL/XML function before it is translated back to the application encoding scheme (by converting the CLOB to CHAR, and converting the character string to hexadecimal). There you can see that the data is in UTF-8 format, with:

- x'C2A7' = '§'
- x'5F' = '_'
- x'31' = '1'
- x'32' = '2'
- x'33' = '3'

This demonstrates that the data (stored in the table in EBCDIC), is converted to UTF-8 while executing SQL/XML functions, and converted back to the application encoding scheme to display the result.

For more information about Unicode, see Chapter 6., "Unicode in DB2 for z/OS" on page 483.

SQL Identifiers to XML Names

Redbooks

Fully escaped mapping for SQL identifiers applies to column names only

Fully escaped mapping rules for column names:

- A colon within a column name is always mapped to _x003A_
- Column names starting with XML (any case combination):
  - if x is lowercase, it is mapped to _x0058_
  - if X is uppercase, it is mapped to _x0078_
  - M and L remain unchanged
- A column name starting with _x is mapped to _x005F_

Mapping SQL identifiers to XML names

The SQL/XML standard specifies some rules for XML names, which do not always conform to those for the SQL identifier standards. Therefore, this section describes these rules and explains the effects and impacts for your applications.

SQL/XML and therefore DB2 V8 only uses a so-called “Fully Escaped Mapping”, which applies only to column names. Other SQL identifiers are not escaped.

These are the rules that apply for full escaping and therefore for SQL identifiers that are column names:

- A colon (:) is always mapped to string _x003A_, independently of the position this column has within the SQL column name.

  Refer to Example 5-21 to see how this escaping looks in the result table. (XMLTESTTAB3 is a table with two columns, ’SGN:ME’ and CREATOR.)

Example 5-21 Escaping of column names

```
SELECT XML2CLOB(
    XMLELEMENT ( NAME "VOLUMES",
          XMLATTRIBUTES( "SGN:ME",CREATOR)
                )        )
                 AS "VOLUMES TO SGNAME"
FROM XMLTESTTAB3;
---------+---------+---------+---------+---------+---------+----
VOLUMES TO SGNAME
---------+---------+---------+---------+---------+---------+----
<VOLUMES SGN_x003A_ME="SYSDEFLT" CREATOR="HAIMO   "></VOLUMES>
<VOLUMES SGN_x003A_ME="DSN8G810" CREATOR="HAIMO   "></VOLUMES>
<VOLUMES SGN_x003A_ME="SABIGRP1" CREATOR="BARTR2  "></VOLUMES>
<VOLUMES SGN_x003A_ME="SABI    " CREATOR="BARTR2  "></VOLUMES>
```

This mapping only occurs for XMLATTRIBUTES and XMLFOREST, because the actual column names are only used in these two XML publishing functions.

- If column names are starting with “XML”, where the series of the three letters can be in any case combination, x or X is escaped as follows:
  - If x is lowercase, it is mapped to _x0058_
  - If X is uppercase, it is mapped to _x0078_
  - M and L remain unchanged
- If the first two characters of a column name are _x, then this combination is mapped to _x005F_.
- There are some more characters, which are valid in SQL identifiers, but must be escaped in XML names. Table 5-2 shows a list of these characters and tells you how they are escaped in XML names.

Table 5-2 Escape value for valid SQL characters

Character	Replacement string
(blank)	_x0020_
“	_x0022_
#	_x0023_
&	_x0026_
’	_x0027_
<	_x003C_
>	_x003E_
{	_x007A_
}	_x007D_
[	_x005B_
]	_x005D_

If you want to display these escaped characters properly, for example, using a Web browser, you must add some additional application logic to your program, which converts them from UTF-16 big endian format to UTF-8.

See Chapter 6., “Unicode in DB2 for z/OS” on page 483 for details on the different Unicode encodings.

> **Note:** As mentioned above, fully escaped mapping only applies to SQL column names. If an SQL identifier, which is supplied in the AS clause of the SELECT statement, contains any of the characters described above, SQLCODE -20275 occurs. Refer to Example 5-22 for more details.

Example 5-22 Fully escaped mapping and SQL identifiers

```
SELECT XML2CLOB(
    XMLELEMENT ( NAME "VOLUMES",
          XMLATTRIBUTES(SGNAME AS "COL:SG",SGCREATOR)
                )       )
                 AS "VOLUMES TO SGNAME"
FROM SYSIBM.SYSVOLUMES
---------+---------+---------+---------+---------+---------+---------+---------+
DSNT408I SQLCODE = -20275, ERROR:  The XML NAME COL:SG IS NOT VALID. REASON
         CODE = 2
```

SQL Data Value to XML Data Value

Restriction for: ROWID, FOR BIT DATA, BLOB

All other data types are converted to Unicode

Exceptions:

- TIMESTAMP yyyy-mm-dd-hh.mm.ss.nnnnnn converted to
  yyyy-mm-ddThh:mm:ss.nnnnnn

SQL Character	Escaped XML string
<	<
>	>
&	&
"	"
'	'

Mapping SQL data values to XML data values

The third kind of mapping that needs to be performed when you invoke XML publishing functions is the mapping of SQL data values to XML data values. This mapping is based on SQL data types.

The following data types are not supported and therefore cannot be used as arguments to XML value constructors:

- ROWID
- Character string defined with the FOR BIT DATA attribute
- BLOB
- Distinct types sourced on ROWID, FOR BIT DATA character string or BLOB

If you try to use one of the above-mentioned data types, SQLCODE -171, SQLSTATE 42815 is issued.

There are some more rules that apply for the mapping of SQL data values to XML data values:

- CHAR, VARCHAR, GRAPHIC, VARGRAPHIC, CLOB, and DBCLOB strings are converted to Unicode. If one of the following characters is part of this string, it is escaped as shown in Table 5-3.

Table 5-3 Escaping of SQL character data

SQL character	Escaped XML string
<	<
>	>
&	&
“	"
‘	'

> **Note:** The characters shown in Table 5-3 are always escaped. That is, if “&” is in a string, it will result in “&”

- Numeric data types are mapped to UTF-8.
- The DATE data type is mapped to UTF-8.
- The TIME data type is mapped to UTF-8.
- The TIMESTAMP data type is mapped from yyyy-mm-dd-hh.mm.ss.nnnnnn to yyyy-mm-ddThh:mm:ss:nnnnnn.
- For DISTINCT types, the mapping follows its source data type.

Security Enhancements

The need for more granular security

Introduction to multilevel security

Multilevel security for access control

- Multilevel security for object level access
- Multilevel security with row granularity

DB2 Command authorization enhancements

Allow more than 245 secondary authorization IDs

Session variables

Encryption built-in functions

5.5 Security enhancements

Security has become much more important in the past few years. DB2 for z/OS Version 8 makes very substantial changes related to security, with new options with multilevel security and row level security to meet the new e-business security demands.

The PTF for APAR PQ90147 allows for 1012 secondary authorization IDs. The DB2 connection and signon exits, DSN3@ATH and DSN3@SGN, can now supply 1012 secondary authorization IDs in the AIDL, which will be used in DB2 authorization checking, instead of the current maximum of 245.

Other new V8 features, such as session variables, encryption and enhanced authorization for DB2 commands, improve security as well. Session variables are discussed in 4.23, “Session variables” on page 380. Encryption is discussed in section 4.22.1, “Encryption functions” on page 373.

Database Security and Granularity

Low level access control is increasingly critical

- Web hosting
- Privacy of data

Need row level granularity

- Individual users can be restricted to a specific set of rows

Need for mandatory security

- Not easily bypassed by high database authorization levels

Today, you can use views to limit access

- Can be cumbersome
- Not as effective for UPDATE, INSERT, DELETE and utilities

5.5.1 The need for more security options

Everyone seems to be more aware of security today. Improving integration and making security more robust and easier to manage are very important. New options for higher security, more granularity, and more information for additional flexibility in applications and SQL are needed.

Low level access control is increasingly critical. Good examples are Web hosting companies that need to store data from multiple customers into a single subsystem, database, or table. Also, security requirements and laws on privacy demand security at the lowest (row) level.

Many customers need to extend the granularity from table level to row level, so that an individual user is restricted to a specific set of rows.

In addition, there is an increasing need for mandatory access control. This means that subjects (such as users and programs) cannot control or bypass security mechanisms (such as people with install SYSADM authorization in DB2 today).

Views can limit access to selected rows and columns, but they may be cumbersome to construct with the desired level of granularity. Views are not very effective for security when dealing with UPDATE, INSERT, DELETE statements as well as DB2 utilities. Database constraints, triggers, UDFs, and stored procedures are often needed for update control.

New concepts

Redbooks

Subjects and objects

- Objects: "things" you try to protect
- Subjects: "things" that need to access objects

Multilevel security (MLS)

Mandatory access control (MAC)

- Governed by SECLABELs

Discretionary access control (DAC)

- Governed by access lists

Security labels (SECLABEL)

5.5.2 New concepts for DB2 people

Two central concepts of security are security policy and accountability. A *security policy* is a set of laws, rules, and practices that regulate how an organization manages, protects, and distributes its sensitive data. It is the set of rules that the system uses to decide whether a particular subject can access a particular object. *Accountability* requires that each security-relevant event must be able to be associated with a subject. Accountability ensures that every action can be traced to the user who caused the action.

Let us first describe subjects and objects in a bit more detail.

Object

An object is a system resource to which access must be controlled. Examples of objects are: data sets, z/OS UNIX files and directories, commands, terminals, printers, disk volumes, tapes, DB2 objects such as plans and tables, and rows in a DB2 table.

Subject

A subject is an entity that requires access to system resources. Examples of subjects are: human users, started procedures, batch jobs, or z/OS UNIX daemons. The term user usually has the same meaning as the term subject, but sometimes implies a human subject. In this publication, unless stated otherwise, the terms user and subject are used interchangeably.

Multilevel security

Multilevel security (MLS) is a security policy that allows the classification of data and users based on a system of hierarchical security levels, combined with a system of non-hierarchical security categories. A multilevel-secure security policy has two primary goals. First, the controls must prevent unauthorized individuals from accessing information at a higher classification than their authorization (*read up*). Second, the controls must prevent individuals from declassifying information (*write down*).

In the following sections we describe some of the concepts of multilevel security. Multilevel security is a complex matter, and describing the details of it is beyond the scope of this publication. For more information, please refer to the z/OS Security Server publications. An introduction can also be found in *z/OS Planning for Multilevel Security*, GA22-7509.

Mandatory access control (MAC)

Mandatory access control is the principle of restricting access to objects based on the sensitivity of the information that the object contains and the authorization of the subject to access information with that level of sensitivity. This type of access control is mandatory in the sense that subjects cannot control or bypass it.

The sensitivity of each object is defined by means of a security label. This security label indicates the hierarchical level or classification of the information (such as top secret, confidential, general-use), and indicates to which non-hierarchical category the information belongs within that level (such as group ABC, group XYZ).

The security administrator also controls each subject's access to information by specifying which security labels the subject can use. A subject can access information in an object only when the subject's security label entitles the access. If the subject's security label does not have enough authority, the subject cannot access the information in the object. For more information about security labels, see 5.5.3, "Security labels" on page 458.

Mandatory access control is governed by security labels.

Discretionary access control (DAC)

Discretionary access control is the principle of restricting access to objects based on the identity of the subject (the user or the group to which the user belongs). Discretionary access control is implemented using access control lists. A resource profile contains an access control list that identifies the users who can access the resource and the authority (such as read or update) the user is allowed in referencing the resource.

The security administrator defines a profile for each object (a resource or group of resources), and updates the access control list for the profile. This type of control is discretionary in the sense that subjects can manipulate it, because the owner of a resource, in addition to the security administrator, can identify who can access the resource and with what authority. This is what we have in today's RACF® systems.

Discretionary access control is governed by access lists.

Security Labels

Security levels

- RDEFINE SECDATA SECLEVEL UACC(READ)
- RALTER SECDATA SECLEVEL ADDMEM(L0/10 L1/30 L2/50)

Security categories

- RDEFINE SECDATA CATEGORY UACC(READ)
- RALTER SECDATA CATEGORY ADDMEM(C1 C2 C3 C4 C5)

Security labels (SECLABEL)

- RDEF SECLABEL L1C12 SECLEVEL(L1) ADDCATEGORY(C1 C2) U(NONE)
- Subject (for example, the user) can have authorization to more than 1 SECLABEL, but only one can be used at any one time
  - PERMIT L1C12 CLASS(SECLABEL) ACCESS(READ) ID(USRT068)
- Resource can have only one SECLABEL

5.5.3 Security labels

A *security label* enables an installation to classify subjects and objects according to a data classification policy, identify objects to audit based on their classification, and protect objects such that only appropriately-classified subjects can access them:

- *Objects* in a multilevel-secure system have a security label that indicates the sensitivity of the object's data.
- *Subjects* in a multilevel-secure system also have a security label. This label determines whether the subject is allowed to access a particular object.

A security label is used as the basis for mandatory access control decisions. By assigning security labels, the security administrator can ensure that data of a certain classification is protected from access by a user of a lesser security classification. Security labels provide the capability to maintain multiple levels of security within a system.

Defining security labels

A security label establishes an association between a *RACF security level* and a set of zero or more *RACF security categories.* For example, a system might have three security levels, unclassified, sensitive, and secret; and three security categories, group A, group B, and group C.

You must define two profiles in the RACF SECDATA resource class; one to define the security levels and the other to define the security categories for the system.

Security level: SECLEVEL

The SECLEVEL profile contains a member for each hierarchical security level in the system:

```
RDEFINE SECDATA SECLEVEL UACC(READ)
RALTER  SECDATA SECLEVEL ADDMEM(L0/10 L1/30 L2/50 L3/70 L4/90)
```

The hierarchical security level (SECLEVEL) defines the degree of sensitivity of the data. In the example about security level, “L0” is defined to be a security level 10. The security administrator can define up to 254 security levels.

Security category: CATEGORY

The CATEGORY profile contains a member for each non-hierarchical category in the system.

```
RDEFINE SECDATA CATEGORY UACC(READ)
RALTER  SECDATA CATEGORY ADDMEM(C1 C2 C3 C4 C5)
```

The non-hierarchical categories (CATEGORY) further qualify the access capability. The security administrator can define zero or more categories that correspond to some grouping arrangement in the installation. C1 through C5 are security categories.

Guideline: Although the system allows the definition of several thousand categories, define only the security categories you need. A large number of security categories can decrease performance, particularly at IPL time and for the SETROPTS RACLIST(REFRESH) command.

Security labels: SECLABEL

After defining the SECLEVEL and CATEGORY profiles, the security administrator defines a profile in the SECLABEL resource class for each security label. (Note that here we are talking about SEC*LABEL*, not SEC*LEVEL*). The security label is a name of up to eight uppercase alphanumeric or national characters.

Each SECLABEL profile specifies the particular combination of:

- A SECLEVEL member, and
- Zero or more members of the CATEGORY profile that applies to the security label

For example:

```
RDEF SECLABEL L1C12  SECLEVEL(L1) ADDCATEGORY(C1 C2) U(NONE)
```

You do not need to define a security label for every possible combination of level and category. There is no limit on the number of security labels that can be defined.

Assigning security labels

A *subject* can have more than one security label, but can only use one security label at a given time. To authorize a subject to use a security label, the security administrator permits that subject’s user ID to the profile in the RACF SECLABEL resource class for the security label:

```
PERMIT L1C12 CLASS(SECLABEL) ACCESS(READ) ID(USRT068)
```

The security administrator can also assign a default security label for the subject in the user profile:

```
ALTUSER USRT068 SECLABEL(L1C12)
```

The security label a subject uses at a given time can be assigned in a number of ways; for example, a TSO/E user can specify a security label on the logon panel, and a batch user can specify a security label on the JOB statement.

A *resource* can have only one security label. For most types of resources, the security administrator assigns a security label to each resource in the system that is to be protected by ensuring that the resource is protected by a profile, and by specifying the security label in the profile.

Built-in security labels in RACF

The following security labels are automatically defined by RACF:

SYSHIGH: This label is equivalent to the highest security level defined by the security administrator, and all categories defined by the security administrator. It dominates all other security labels in the system. SYSHIGH should be restricted to special system-level address spaces such as consoles, and to system programmers, system operators, and system administrators.

SYSLOW: This label is equivalent to the lowest security level defined by the security administrator, and has no categories assigned. It is dominated by all other security labels.

SYSNONE: SYSNONE is treated as equivalent to any security label to which it is compared. SYSNONE, like SYSLOW, should be used only for resources that have no classified data content.

SYSMULTI: This label is considered to be equivalent to any defined security label. It is intended for use by multilevel security servers and tables that have multilevel data.

5.5.4 Using security labels

After security labels have been created and assigned, the security administrator can activate the RACF SECLABEL resource class to cause the system to use the security labels for authorization checks. Then, when a user tries to access a resource, RACF checks whether the resource has a security label. If it does, RACF compares the security label of the user with that of the resource (mandatory access control). If the security labels allow access, RACF then it checks the access list of the profile that protects the resource (discretionary access control). The decision as to whether or not to allow the access is based on both mandatory access control (MAC), and discretionary access control (DAC).

SECLABEL verification

When verifying whether a user with a SECLABEL is allowed to access a resource with a SECLABEL, different checks can be done.

Dominance

One security label dominates another SECLABEL when both of the following conditions are true:

- The security level that defines the first security label is greater than or equal to the security level that defines the second security label.
- The set of security categories that define the first security label includes the set of security categories that defines the second security label.

You can also look at dominance in a simplistic way as one SECLABEL being “greater than” another.

Two security labels are *disjoint* when each of them has at least one category that the other does not have. Neither of the labels dominates the other. These are also called *incompatible security labels*.

Reverse dominance

With reverse dominance access checking, the access rules are the reverse of the access rules for dominance access checking. This type of checking is not used by DB2.

In loose terms, it can be looked at “less than or equal to” checking.

Equivalence

Equivalence of security labels means that either the security labels have the same name, or they have different names but are defined with the same security level and identical security categories. The security label SYSMULTI is considered equivalent to any security label.

You can look at this type of checking as “equal to” checking. (One way to check is if both dominance and reverse dominance are true.)

Read-up

Multilevel security controls prevent unauthorized individuals from accessing information at a higher classification than their authorization. It does not allow users to “read-up” or read above their authorization level. Read-up is enforced through dominance checking.

Write-down

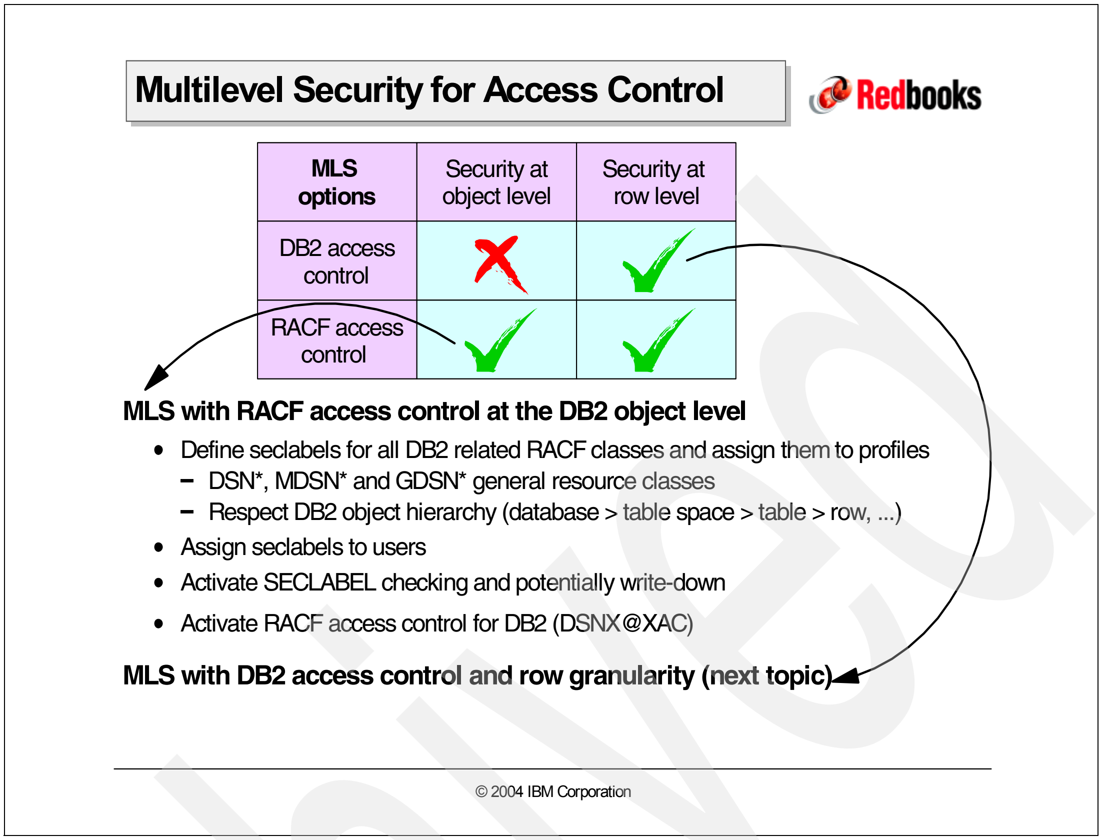
Multilevel security also prevents individuals from declassifying information. This is also known as write-down, that is, writing information back at a lower level (down-level) than its current classification. This property is sometimes called the star property, or *-property. Write-down is prevented by doing equivalence checking.

However, there may be cases where you want to allow write-down by selected individuals.

The security administrator controls whether write-down is allowed at the system level by activating and deactivating the RACF MLS option (using the SETROPTS command).

In addition, to allow for controlled situations of write-down, z/OS allows the security administrator to assign a “write-down by user” privilege to individual users that allows those users to select the ability to write down. To do so, a user has to have at least read authority on the IRR.WRITEDOWN.BYUSER profile in the FACILITY class.

The current value of your user’s SECLABEL is available as a DB2 session variable. For more information on session variables, see 4.23, “Session variables” on page 380.



5.5.5 Multilevel security for access control

You can use multilevel security for multiple purposes in conjunction with DB2, as described in the following sections.

Multilevel security with row-level granularity with DB2 authorization

In the figure above, this is the upper right option (Security at row level / DB2 access control). In this combination, DB2 grants are used for authorization at the DB2 object level (database, table, and so forth). Multilevel security is implemented only at the row level within DB2. This is discussed in 5.5.7, “Multilevel security with row granularity” on page 466.

Multilevel security at the object level with external access control

In the figure above, this is the lower left option (Security at object level / RACF access control). In this combination, external access control, such as the RACF access control module, is used for authorization at the DB2 object level. The RACF access control module has been enhanced to also use security labels to perform mandatory access checking on DB2 objects as part of multilevel security. This option is discussed in more detail in 5.5.6, “Implementing multilevel security at the object level” on page 464.

For information about the access control authorization exit, see the new DB2 V8 manual *DB2 UDB for z/OS RACF Access Control Module Guide,* SC18-7433.

Multilevel security with row-level granularity with external access control

In the figure above, this is the lower right option (Security at row level / RACF access control). This option combines both options mentioned before. It uses multilevel security to control the access to the DB2 objects, as well as multilevel security (SECLABELs) to control access at the row level within DB2.

For more information, see also the IBM Redbook *MLS / DB2 Implementation*, SG24-6480.

5.5.6 Implementing multilevel security at the object level

To implement multilevel security with DB2 at the object level, perform the following steps:

- Define security labels in RACF for all DB2 objects that require mandatory access checking by using the RDEFINE command. You have to define security labels for the RACF resource classes used by DB2 and assign them to the appropriate classes:

  – DSNR (access to DB2 subsystems)

  Also, you need to do this for all classes used by the RACF access control module:

  – DSNADM (administrative authorities)
  – MDSNBP and GSNBP (buffer pools)
  – MDSNCL and GDSNCL (collections)
  – MDSNDB and GDSNDB (database)
  – MDSNJR and GDSNJR (JAR)
  – MDSNPK and GDSNPK (package)
  – MDSNPN and GDSNPN (plans)
  – MDSNSC and GDSNSC (schema)
  – MDSNSG and GDSNSG (storage groups)
  – MDSNSM and GDSNSM (system privileges)
  – MDSNSP and GDSNSP (stored procedures)
  – MDSNSQ and GDSNSQ (sequences)
  – MDSNTB and GDSNTB (tables, views, indexes)
  – MDSNTS and GDSNTS (table spaces)
  – MDSNUF and GDSNUF (user-defined functions)
  – MDSNUT and GSNUT (Distinct Type)

  It is your responsibility to ensure the proper hierarchy of security labels. In general, the security label of an object that is higher in the object hierarchy should dominate the security labels of objects that are lower in the hierarchy. RACF and DB2 do not enforce this hierarchy; they merely enforce the dominance rules that you establish.
  When you use RACF to define security labels for the DB2 objects, take into account the following object hierarchy:

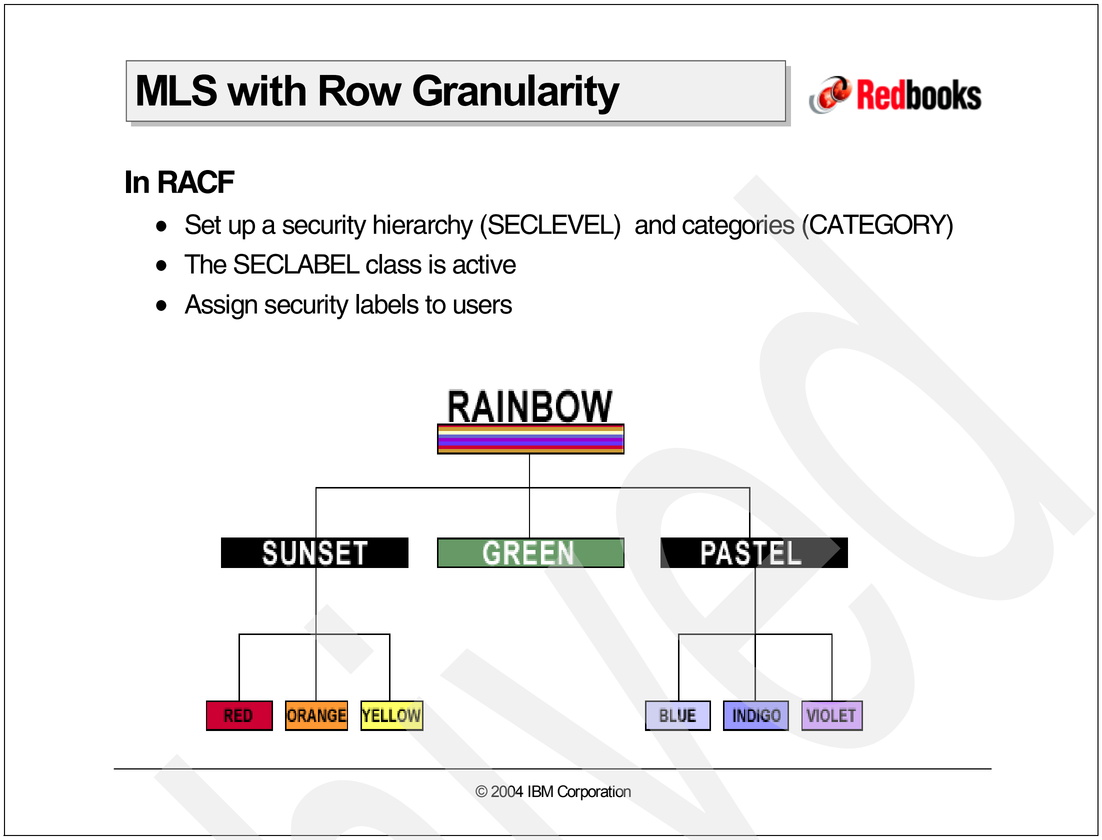
  – Subsystem or data sharing group
    - Database
      - Table space
        - Table
          - Column
          - Row
    - View
    - Storage group
    - Buffer pool
    - Plan
    - Collection
      - Package
    - Schema
      - Stored procedure or user-defined function
      - Java ARchive (JAR)
      - Distinct type
      - Sequence

The preceding example suggests dominance relationships among objects in the DB2 object hierarchy. A database should dominate a table space. That table space should dominate a table. That table should dominate a column or row.

Views are a little bit more complex. If a view is based on a single table, the table should dominate the view. However, if a view is based on multiple tables, the view should dominate the tables.

- Assign security labels to users as well. Consider giving users with SYSADM, SYSCTRL, and SYSOPR authority the security label of SYSHIGH. If you are using a TCP/IP connection, you need to define security labels in RACF for the security zones into which IP addresses are grouped. These IP addressed represent remote users.
- Activate the SECLABEL class in RACF. If you want to enforce write-down control, turn on write-down control in RACF.
- Install the external security access control authorization exit routine (DSNX@XAC), the RACF access control module.

At execution time, DB2 uses RACF to verify whether a user is allowed to access a DB2 object by comparing the user's SECLABEL against the DB2 object's SECLABEL (assigned in the DSN*, MDSN* or GDSN* profiles). The checking for dominance or equivalence is done based on the "access intent". In case of a SELECT, dominance checking is performed. In other cases, equivalence is checked.

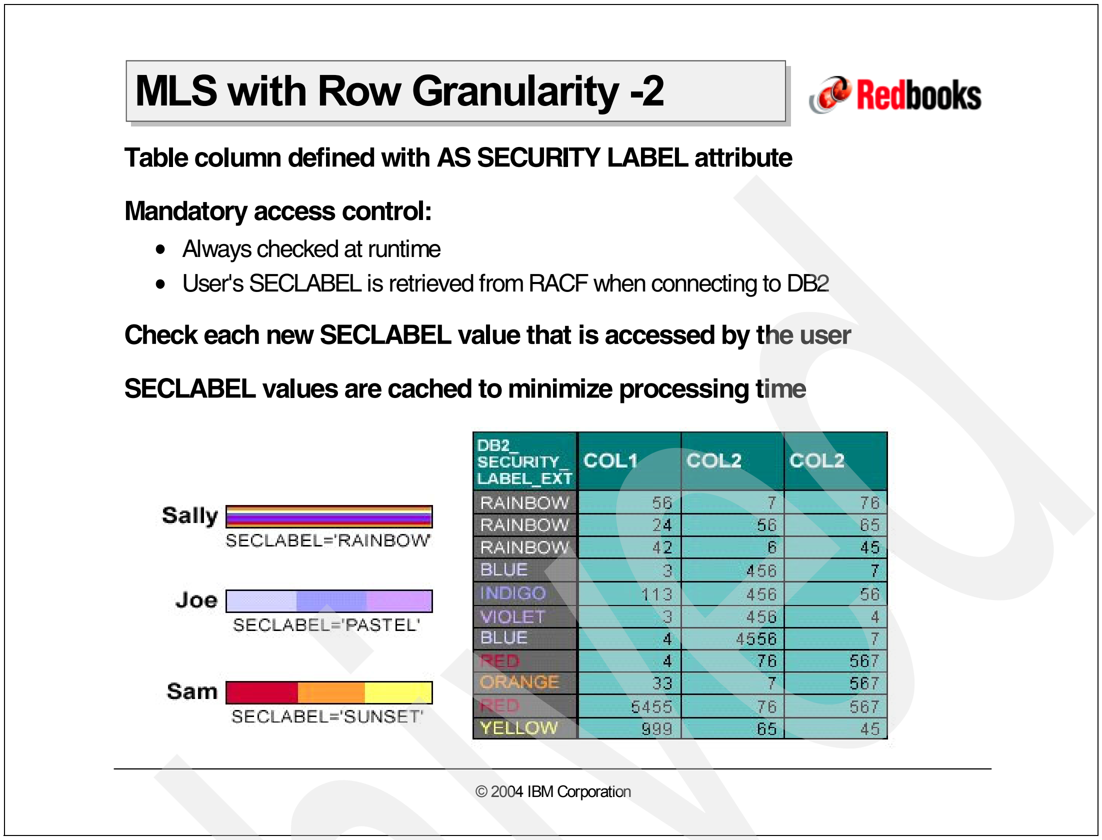


5.5.7 Multilevel security with row granularity

With the hierarchy shown on the foil established in the security server, the system understands that users with authority to access RAINBOW can access anything. Someone with authority to access PASTEL information can access any row associated with BLUE, INDIGO, VIOLET, or PASTEL. Someone with SUNSET authority can access SUNSET, RED, ORANGE, YELLOW.

This is a lot more powerful than just having an exact match on the security label, for example, where the user's label must exactly match the data's label, since it has the notion of “groups” that make security administration easier to manage.

With this additional capability, we are able to implement a hierarchical type of security scheme without requiring the application to access the data using special views or predicates.



5.5.8 Multilevel security with row granularity - DB2 setup

Customers asked for row-level security for applications that need more granular security or mandatory access control. For example, an organization may want a hierarchy in which employees can see their own payroll data, a first line manager can see his or her payroll information, and all of the employees reporting to that manager, and so on. Security schemes often include a security hierarchy and non-hierarchical categories.

To allow DB2 to verify access at the row level, using multilevel security with row granularity, you must have a column that acts as the security label (SECLABEL), with a column defined AS SECURITY LABEL. Each row value has a specific SECLABEL.

You incorporate the security label column in the table definition at CREATE TABLE time, or add the column later.

Security labels are defined and provided by RACF. When connecting to DB2, a user's SECLABEL is retrieved from RACF. When rows are accessed, DB2 checks each new SECLABEL value accessed. If access is allowed, then you can access the row. If access is not allowed, the data is not returned, and you are not even aware it exists.

Normally, when a user performs an INSERT, UPDATE operation, or LOADs a row, the user's SECLABEL is stored in the table's column that is defined with the "AS SECURITY LABEL" attribute. However, if write-down is not in effect, or write-down is in effect but the user has write-down privilege, the user can specify any SECLABEL for that row.

This is the runtime checking of the user's SECLABEL against the data's security label, in addition to DB2 GRANT and RACF PERMIT controls.

CREATE/ALTER TABLE with a seclabel

To enable row level security

- Table must have a column assigned to store the SECLABEL
- To define a column as the security label column
  - Security label column can have any column name
  - Specify "**AS SECURITY LABEL**" as the column attribute
    - This indicates that the table is defined with multilevel security with row granularity, and that the column will contain the security label values
  - Datatype must be CHAR(8)
    - Must be single byte column
    - Must be NOT NULL WITH DEFAULT
  - Cannot be a column with
    - Fieldproc, editproc, or check constraint

Table once created with SECLABEL cannot be disabled

Audit record produced if the table with security label is created, altered or dropped

5.5.9 CREATE/ALTER a table with a security label

When you CREATE a table or ALTER it, you can decide to implement row-level security by including a column that specifies the AS SECURITY LABEL attribute, or add a column with that attribute to an existing table.

The only technique to disable row level security (a table that has a column that is defined with the AS SECURITY LABEL attribute) is to drop the table, table space, or database.

You can assign any name to the security label column, but the same column name cannot be used more than once in the table. Only one security label is allowed per table.

The security label column must be data type single byte character (SBCS), CHAR(8), NOT NULL WITH DEFAULT.

This column cannot have field procedures, edit procedures, or check constraints.

When the audit trace (class 3) is active, an audit record IFCID 0142 is created. A table with a security label is treated like an audited table.

Accessing a table defined with MLS

SELECT

- User's seclabel is compared to the seclabel of the row
- If user's seclabel dominates the data seclabel -> row returned
- If user's seclabel does not dominate -> row not returned, no error

INSERT

- Value of the seclabel column for inserted row is set to the value of the user's seclabel
- If user has write-down authority, the user is allowed to set the seclabel field to any non-disjoint seclabel

5.5.10 Accessing data in a table defined with row level security

Let us now have a look at how the different operations that you normally perform against data in a table are affected by having defined multilevel security with row granularity for the table.

> **Note:** In this and the next section (MLS with row level granularity and utilities), we assume that write-down is in effect and that the user does not have write-down authority, unless otherwise mentioned.

SELECT

The security rule for select is that your current security label must dominate the security label of all the rows read. If your security label does not dominate the label of the data row, then that row is not returned.

The user's SECLABEL is compared to the data SECLABEL of the row to be selected. If user SECLABEL dominates the data SECLABEL, then the row is returned. If user SECLABEL does not dominate the data SECLABEL, then the row is not included in data returned, but no error is reported.

The user must be identified to RACF with a valid SECLABEL. If not, an authorization error and audit record (IFCID 140) are produced, provided the audit trace is active.

INSERT

The access rules for INSERT are similar, but DB2 puts the user's current SECLABEL in the row when the row is inserted, if a user does not have the write-down privilege. If the user does have the write-down privilege, or write-down checking is not in effect, then he or she can set the value of the SECLABEL column to any non-disjoint seclabel value.

Note: Since the PTF for APAR PQ94303, an SQLCODE -20264 is issued when a disjoint seclabel is specified. Before this APAR, any seclabel could be specified if the user had write-down authority.

Accessing a table defined with MLS -2

UPDATE

- User's seclabel is compared to the seclabel of the row to be updated
  - If the seclabels are equivalent
    - Row is updated
    - Value of the seclabel in the updated row is set to the value of the user seclabel
- If user has write-down authority, then down-level rows can be accessed and updated

DELETE

- User's seclabel is compared to the seclabel of the row to be deleted
  - If the seclabels are equivalent
    - Row is deleted
- If user has write-down authority, then down-level rows can be accessed and deleted

We now continue with UPDATE and DELETE access against a table defined with multilevel security with row granularity.

UPDATE

The rules for UPDATE are similar, using the SELECT rules for access to the data, and setting the SECLABEL like INSERT. Update requires equivalence for users who are not allowed to write down.

The user's SECLABEL is compared with the SECLABEL of the row to be updated. The update proceeds according to the following rules:

- If the security label of the user and the security label of the row are equivalent, the row is updated and the value of the security label is determined by whether the user has write-down privilege:
  - If the user has write-down privilege or write-down control is not enabled, the user can set the security label of the row to any valid security label.
  - If the user does not have write-down privilege and write-down control is enabled, the security label of the row is set to the value of the security label of the user.
- If the security label of the user dominates the security label of the row, the result of the UPDATE statement is determined by whether the user has write-down privilege:
  - If the user has write-down privilege or write-down control is not enabled, the row is updated and the user can set the security label of the row to any valid security label.

> **Note:** Since the PTF for APAR PQ94303, an SQLCODE -20264 is issued when a disjoint seclabel is specified. Before this APAR, any seclabel could be specified if the user had write-down authority.

  – If the user does not have write-down privilege and write-down control is enabled, the row is not updated.

- If the security label of the row dominates the security label of the user, the row is not updated.

The user must be identified to RACF with a valid SECLABEL. If not, an authorization error and an audit record are produced.

DELETE

Delete operations in a multilevel security with row granularity environment proceed according to the following rules:

- If the security label of the user and the security label of the row are equivalent, the row is deleted.
- If the security label of the user dominates the security label of the row, the user's write-down privilege determines the result of the DELETE statement:
  – If the user has write-down privilege or write-down control is not enabled, the row is deleted.
  – If the user does not have write-down privilege and write-down control is enabled, the row is not deleted.
- If the security label of the row dominates the security label of the user, it is not considered a matching row, and the row is not deleted.

Security labels and performance

The performance of accessing tables that contain a security label can be impacted if the SECLABEL column is not included in the indexes. The security label column is used whenever a table with multilevel security enabled is accessed. Therefore, it is a good idea to include the SECLABEL column in your existing indexes, especially when your queries are using index-only access today.

DB2 caches security labels (per commit scope) to avoid extra calls to RACF. While the impact is not measured yet, the impact is expected to be small. The caching would work best if there are a relatively small number of security labels to be checked compared with the number of rows accessed in a long commit scope.

DB2 performs multilevel security with row-level granularity by comparing the security label of the user to the security label of the row that is accessed. Because security labels can be equivalent without being identical, DB2 uses the RACROUTE REQUEST=DIRAUTH macro to make this comparison when the two security labels are not the same. For read operations, such as SELECT, DB2 uses ACCESS=READ. For update operations, DB2 uses ACCESS=READWRITE.

Multilevel Security and Utilities

LOAD RESUME of a table space containing tables with multilevel security (MLS) with row granularity

- User must be identified to RACF and have a valid ACEE
- Rules for LOAD RESUME are similar to the rules for INSERT
- Without write-down, seclabel set to user's current seclabel
- With write-down permission, permitted to specify a seclabel

LOAD REPLACE on a MLS table space requires write-down authority

UNLOAD and **REORG UNLOAD EXTERNAL**

- User must be identified to RACF and have a valid ACEE
- Similar to the rules for SELECT statements
- Only rows can be unloaded if the user's seclabel dominates the data seclabel
  No error returned if this is not true, only the row is not unloaded

5.5.11 DB2 utilities and multilevel security

Here we describe how DB2 utilities operate in an environment with multilevel security (MLS).

LOAD RESUME

Executing a LOAD RESUME utility against a table space containing tables with multilevel security with row granularity, requires that the user be identified to RACF and have a valid security label (SECLABEL). If the user does not have a valid SECLABEL, an authorization error message and an audit trace record (IFCID 140) is produced, provided audit trace class 1 is active. LOAD RESUME adheres to the same rules as INSERT. Without write-down authorization, the SECLABEL in the table is set to the SECLABEL associated with the user ID executing the LOAD RESUME utility. With write-down, any valid SECLABEL can be specified.

LOAD REPLACE

LOAD REPLACE deletes all rows in a table space. Therefore, write-down authority is required. If the userID associated with the job that is running the LOAD REPLACE utility does not have write-down privilege, and write-down is in effect, an error message is issued.

Note: Since the PTF for APAR PQ94303, message DSNU334I with ERROR CODE 28 is issued when the supplied value for a column defined with the AS SECURITY LABEL clause, is disjoint with respect to the user's security label, and the user has write-down authority.

UNLOAD and REORG UNLOAD EXTERNAL

Executing the UNLOAD or REORG UNLOAD EXTERNAL utility against tables with multilevel security with row granularity requires that the user associated with the job be identified to RACF, and have a valid SECLABEL. If the user does not have a valid SECLABEL, an authorization error message and an audit trace record (IFCID 140) is produced, provided audit trace class 1 is active. For each row unloaded from those tables, if the user SECLABEL dominates the data SECLABEL, then the row is unloaded. If the user's SECLABEL does not dominate the data SECLABEL, the row is not unloaded and no error is returned.

Multilevel Security and Utilities -2

Redbooks

REORG ... DISCARD of tables

- User must be identified to RACF and have a valid ACEE
- For each row unloaded from those tables, if the row qualifies to be discarded, the user seclabel is compared to the data seclabel
  - If they are the same -> row discarded
  - If they are not the same -> check for equivalence of the two seclabels
    - If equivalent -> row discarded
    - If not check if write-down privilege is in effect:
      - \+ In effect and user had write-down
        -> row discarded if user seclabel dominates the row
      - \+ In effect and user does not have write-down
        -> not discarded
      - \+ Write-down not in effect
        -> dominance is enough

REORG ... DISCARD

Executing REORG ... DISCARD on tables with multilevel security with row granularity requires that the user be identified to RACF and have a valid SECLABEL. If the user does not have a valid SECLABEL, an authorization error message and an audit trace record (IFCID 140) is produced, provided audit trace class 1 is active.

REORG with the DISCARD option adheres to the same rules as the SQL DELETE statement. For each row unloaded from those tables, if the row qualifies to be discarded, the user SECLABEL is compared to the data SECLABEL.

- If the SECLABELs are the same, the row is discarded.
- If the SECLABELs are not the same, then equivalence is checked.
  - If the SECLABELs are equivalent, the row is discarded.
  - If not, a check is done to see if write-down privilege is in effect:
    - If write-down privilege is in effect, and the user has write-down, rows that are dominated are discarded.
    - If write-down privilege is in effect, and the user does not have write-down, the row is not considered to be a match and the row is not discarded.
    - If write-down privilege checking is not in effect, then rows that are dominated are discarded.

Requirements and Restrictions

Requirements

- Requires z/OS 1.5 and Security Server (RACF) V1R5

Restrictions

- Row level security is not enforced for referential constraints
- Referential constraints cannot be defined on a seclabel column
- Sysplex parallelism is not used for queries on a table defined with a security label
- As mentioned before, the seclabel column cannot have
  - FIELDPROC, EDITPROC, check constraints
- Trigger transition tables do not have security labels
- Global temporary tables cannot have a true seclabel column
- Some additional restrictions for MQTs

5.5.12 MLS requirements and restrictions

Let us look at some of the MLS requirements and restrictions.

Requirements for MLS

As mentioned before multilevel security (with or without row granularity) requires z/OS V1.5 and Security Server (RACF) V1.5. Setting up MLS requires coordination with your security administrator, as MLS not only affects DB2, but a lot of setup work needs to be done in RACF as well.

Restrictions when using MLS with row granularity

In the following sections we describe some of the restrictions that you should be aware of when planning to use multilevel security with row granularity.

Sysplex query parallelism

Sysplex query parallelism cannot be used for queries that access tables that have a SECLABEL column defined.

Global temporary tables with multilevel security

For a declared temporary table with a column definition, no syntax exists to specify a security label on a DECLARE GLOBAL TEMPORARY TABLE statement. An attempt to specify a security label results in an error. If a DECLARE GLOBAL TEMPORARY TABLE statement uses a fullselect or a LIKE predicate or a CREATE GLOBAL TEMPORARY TABLE statement uses a LIKE predicate, the resulting temporary table can inherit the security label column from the referenced table or view.

However, the temporary table does not inherit any security attributes on that column. That means that the inherited column in the temporary table is not defined AS SECURITY LABEL. The column in the temporary table is defined as NOT NULL, with no default. Therefore, any statements that insert data in the temporary table must provide a value for the inherited column.

Materialized query tables with multilevel security

A materialized query table (MQT) is a table that contains information that is derived and summarized from other tables. As such, an MQT can contain the results of queries with expensive join and aggregation operations. The optimizer is aware of the MQT and can decide to use the MQT to speed up query processing. For more information about MQTs, see 10.2, "What is a materialized query table?" on page 700.

If one or more of the source tables for a materialized query table has multilevel security with row-level granularity enabled, some additional rules apply to working with the materialized query table and the source tables.

- When creating an MQT, if any of the source tables in the fullselect of the materialized query table definition contains a security label column:
  - If only one source table contains a security label column and the materialized query table is defined with the DEFINITION ONLY clause, the materialized query table inherits the values in the security label column from the source table. However, the inherited column is not a security label column.
  - If only one source table contains a security label column, the security label column must be included in the materialized query table definition with the AS SECURITY LABEL clause. The materialized query table will inherit the security label column from the source table. The MAINATAINED BY USER option is allowed.
  - If more than one source table contains a security label column, DB2 returns an error code and the materialized query table is not created.
- Using an ALTER TABLE statement to add a security label column to a table will fail if the table is a source table for a materialized query table.
- The REFRESH TABLE statement deletes the data currently in the MQT and then repopulates the MQT by executing the fullselect. DB2 does not check for multilevel security with row-level granularity during this operation. The row-level granularity check is enforced when using the materialized query table, either by exploiting the MQT or by using the MQT directly.

Constraints and multilevel security

- A unique constraint is allowed on a security label column.
- A referential constraint is not allowed on a security label column. Although a referential constraint is not allowed for the security label column, DB2 enforces referential constraints for other columns in the table that are not defined with a security label.
- A check constraint is not allowed on a security label column.
- Row level security checking is not enforced for referential constraints.

Field procedures, edit procedures, validation procedures, and MLS

As mentioned earlier, you cannot define a field procedure or edit procedure on a SECLABEL column (or row).

Validation procedures are allowed on a table that is defined with a security label column. When an authorized user with write-down privilege makes an INSERT or UPDATE request for a row, the validation procedure passes the new row with the security label of the user. If the authorized user does not have write-down privilege, the security label of the row remains the same.

Triggers and multilevel security

When a transition table is generated as the result of a trigger, the security label of the table or row from the original table is not inherited by the transition table. Therefore, multilevel security with row-level checking is not enforced for transition tables and transition values.

If an ALTER TABLE statement is used to add a security label column to a table with a trigger on it, the same rules apply to the new security label column that would apply to any column that is added to the table with the trigger on it.

When a BEFORE trigger is activated, the value of the NEW transition variable that corresponds to the security label column is set to the security label of the user if either of the following criteria are met:

- Write-down control is in effect and the user does not have the write-down privilege.
- The value of the security label column is not specified.

5.5.13 Multilevel security in a distributed environment

Systems that do not support MAC processing can participate in a multilevel secure network, if they are physically managed to guarantee that all information on the system has the same single security label and all users of the system are permitted to that security label. These systems are sometimes referred to as *single-level security* or *managed systems*. This management requires both physical control of the systems and careful management of the network. Managed systems must be prevented from communicating with other managed systems that do not have equivalent security labels. For more information about configuring a DB2 server to use multilevel security in a distributed environment, see *z/OS Communications Server: IP Configuration Guide*, SC31-8775.

SQL statements coming in from remote requesters can participate in a multilevel secure environment. Remote requesters in a multilevel security environment are considered managed systems.

TCP/IP support for multilevel security

A Communications Server IP stack that runs in a multilevel secure environment can be configured as either a restricted stack or an unrestricted stack. An unrestricted stack is configured with an ID that is defined with a security label of SYSMULTI. A single z/OS system can concurrently run a mix of restricted and unrestricted stacks. Unrestricted stacks allow DB2 to use any security label to open sockets. When the system is running a DB2 system that needs to communicate via TCP/IP, the recommendation is to have the DB2 use an unrestricted stack.

All users on a TCP/IP connection have the security label that is associated with the IP address that is defined on the server. If a user requires a different security label, the user must enter through an IP address that has that security label associated with it. If you require multiple IP addresses on a remote z/OS server, a workstation, or a gateway, you can configure multiple virtual IP addresses. This strategy can increase the number of security labels that are available to a client.

Network security zones

A network security zone is an administrative name for a collection of systems that require the same access control policy. IP addresses are used to map systems into security zones. A network security zone can contain a single IP address or any combination of IP addresses and subnetworks. All of the IP addresses in a security zone must have the same security label (though all IP addresses with the same security label do not have to be in the same security zone). Security zones are defined in TCP/IP via the NETACCESS statement.

SECLABELs in a TCP/IP multilevel security environment

Remote users that access DB2 by using a TCP/IP network connection use the security label that is associated with the RACF SERVAUTH class profile when the remote user is authenticated. Security labels are assigned to the database access thread when the DB2 server authenticates the remote server by using the RACROUTE REQUEST = VERIFY service.

To assign the security label in a multilevel security environment (and the MLACTIVE option is active) for incoming TCP/IP requests, you must use the TCP/IP Network Access Control capabilities of the z/OS IP Communication Server V1.5 in combination with the RACF SERVAUTH class. To use the RACF SERVAUTH class and TCP/IP Network Access Control, perform the following steps:

1. Set up and configure TCP/IP Network Access Control by specifying the NETACCESS statement in your TCP/IP profile.

   For example, suppose that you need to allow z/OS system access only to IP addresses from 9.1.37.0 to 9.1.37.255. You want to define these IP addresses as a security zone, and you want to name the security zone IBMITSO. Suppose also that you need to deny access to all IP addresses outside of the IBMITSO security zone, and that you want to define these IP addresses as a separate security zone (WORLD). To establish these security zones, use the following NETACCESS clause:

   ```
    NETACCESS INBOUND OUTBOUND
   ; IP Addr    MASK          SECZONE
     9.1.37.0   225.225.225.0 IBMITSO
     DEFAULT                  WORLD
    ENDNETACCESS
   ```

   For more information about the NETACCESS statement, see *z/OS V1.5 Communications Server: IP Configuration Reference*, SC31-8776.

2. Activate the SERVAUTH class by issuing the following TSO command:

   ```
   SETROPTS CLASSACT(SERVAUTH)
   ```

3. Activate RACLIST processing for the SERVAUTH class by issuing the following TSO command:

   ```
   SETROPTS RACLIST(SERVAUTH)
   ```

4. In the SERVAUTH class you need to define the general resource profiles for the different security zones. These profiles have to adhere to a strict naming convention, namely:

   ```
   EZB.NETACCESS.system-name.tcpip-name.security-zone
   ```

   In this statement:

   - `EZB.NETACCESS` is the RACF required qualifier for these profile names.
   - `system-name` is the MVS system name.
   - `tcpip-name` is the TCP/IP started task name (of your stack that your DB2 runs under).
   - `security-zone` is the name of the security zone. It should match the security zone names specified in the TCP/IP NETACCESS statement.

Define the IBMITSO general resource profiles in RACF to protect the IBMITSO security zone by issuing the following command:

```
RDEFINE SERVAUTH (EZB.NETACCESS.SC63.TCPIP.IBMITSO) ACC(READ) SECLABEL(CLASSFD)
```

5. For this permission to take effect, refresh the RACF database by using the following command:

```
SETROPTS CLASSACT(SERVAUTH) REFRESH RACLIST(SERVAUTH)
```

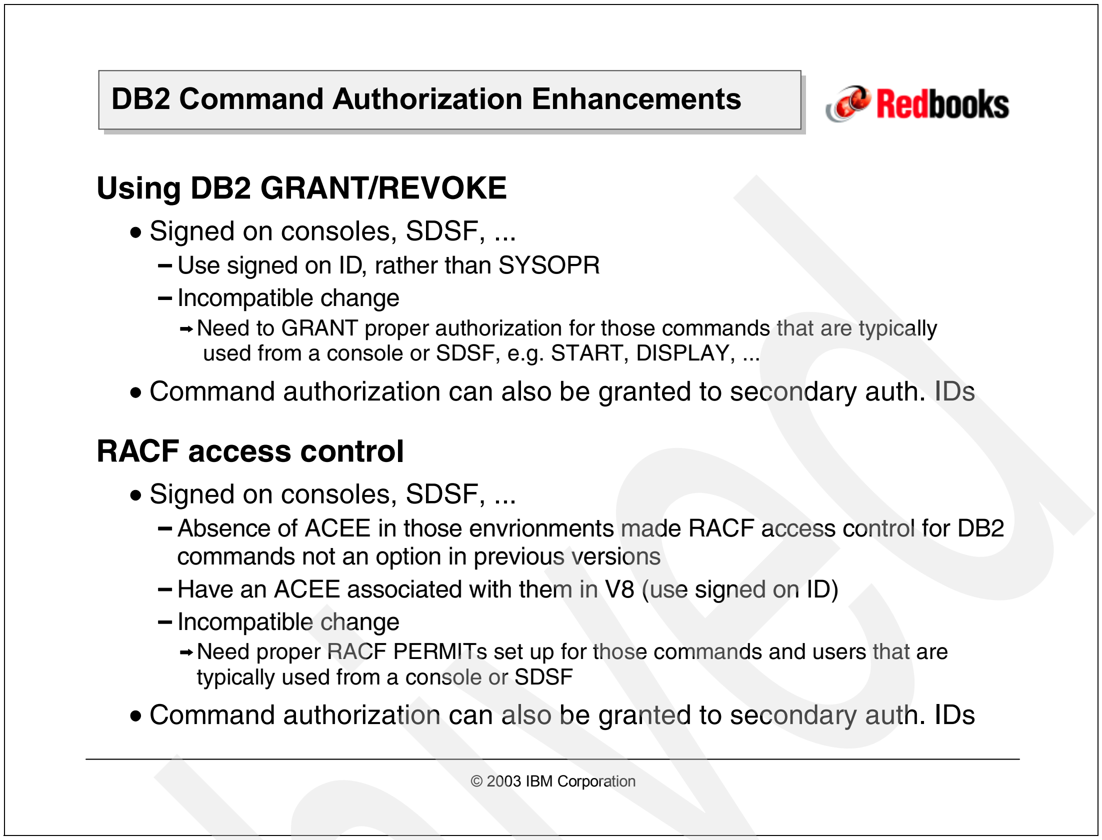
Now, suppose that USER5 has an IP address of 9.1.37.3. TCP/IP Network Access Control will determine that USER5 has an IP address that belongs to the IBMITSO security zone. USER5 will be granted access to the system using the CLASSFD security label.

Alternatively, suppose that USER6 has an IP address of 9.1.25.37. TCP/IP Network Access Control will determine that USER6 has an IP address that belongs to the WORLD security zone. As there is no profile in the SERVAUTH class, access is denied.

The SECLABEL that is assigned via the SERVAUTH class will be used to access DB2 resources.

SNA support for multilevel security

Security labels are assigned to the database access thread when the DB2 server authenticates the remote server by using the RACROUTE REQUEST = VERIFY service. The service establishes a security label to the authorization ID that is associated with the database access thread. For SNA connections, this security label is the default security label that is defined for the remote user.



5.5.14 DB2 Command authorization enhancements

This section discusses authorization enhancements related to DB2 commands. Authorizations, including DB2 commands, can be managed in DB2 via GRANT/REVOKE statements, or via RACF access control.

Managing DB2 command authorization using GRANT/REVOKE

In previous DB2 versions, when executing a command from a (logged on) console, or via SDSF, DB2 always associated user “SYSOPR” with that request. The SYSOPR user had to have the necessary authorization granted to him to execute the requested command.

In DB2 for z/OS Version 8, DB2 uses the user ID that is signed on to the console or that is running the SDSF session, with the DB2 command. This means that the signed on user ID, and no longer SYSOPR must have been granted the necessary authorization to execute the requested DB2 command.

In addition, you can grant the authorization to execute a DB2 command to a secondary authorization ID in DB2 V8. In previous DB2 versions, only the primary authorization ID was checked or, as described in the previous paragraphs, SYSOPR was used when verifying DB2 command authorization.

Therefore, in order to avoid authorization problems with the execution of DB2 commands when migrating to V8, make sure that those user IDs that issue DB2 commands in V7 via SDSF or a logged on console, have either been granted:

- The privilege to execute the command, like DISPLAY, or
- Have been granted SYSOPR authorization, or

- Have been granted another administrative authority that includes the privilege to execute the requested DB2 command.

The authorization can be granted to the primary or any of the secondary authorization IDs associated with the logged on user ID.

Managing DB2 commands using RACF access control

In previous DB2 versions, DB2 commands prefixed with a dash (-) are checked by DB2 catalog security (GRANT/REVOKE). The RACF/DB2 External Security Module specifically passes the request to DB2, because DB2 did not provide an ACEE when running this asynchronous task.

In DB2 for z/OS Version 8, an ACEE is associated with these requests. This means that RACF access control is now able to manage DB2 command authorization.

In addition, you can assign the RACF PERMIT authorization to execute a DB2 command to a secondary authorization ID.

Therefore, when you plan to use RACF access control for DB2 commands in V8, make sure that those user IDs that issue DB2 commands, have receive the PERMIT to either:

- Execute the specific DB2 command, like DISPLAY, or
- To use the SYSOPR administrative authority, or
- To use another administrative authority that includes the privilege to execute the requested DB2 command.

The PERMIT can be handed out to the primary or any of the secondary authorization IDs associated with the logged on user ID.

> **Tip:** The default DB2 DSNZPARM value for installation SYSOPR1 and SYSOPR2 is “SYSOPR”. This means that a user executing a DB2 command in Version 7 via SDSF or a logged on console, will not experience any authorization problems. Remember that for those requests, V7 uses authorization ID SYSOPR. Since user SYSOPR is installation SYSOPR1 by default, command authorization is “bypassed” in V7. Especially for those installation using SYSOPR as installation SYSOPR1 or SYSOPR2, it is important to hand out the necessary authorizations to individual user IDs or secondary authorization IDs before migrating to V8.

6

Unicode in DB2 for z/OS

List of Topics

Conversion basics

Unicode fundamentals

Unicode in DB2 for z/OS

How does Unicode affect me in V8?

Moving to Unicode data

Multiple CCSID set SQL statements

DB2 UDB for OS/390 and z/OS is increasingly being used as a part of large client-server systems. In these environments, character representations vary on clients and servers across many different platforms and across different geographies. One area where this sort of environment exists is in the data centers of multinational companies. Another example is e-commerce. In both of these examples, a geographically diverse group of users interact with a central server, storing and retrieving data.

The traditional way of encoding characters requires hundreds of different code page systems, because no single code page was adequate for all the letters, punctuations, and symbols in common use. These code pages also conflict with one another, because two code pages can use the same code points for different characters.

In order to solve these problems, the support of the Unicode encoding scheme was introduced in DB2 V7 and its use was optional. DB2 for z/OS Version 8 provides additional support for Unicode and globalized applications. Fundamental changes such as the conversion of the DB2 catalog from EBCDIC to Unicode, and Unicode parsing of SQL statements, affects everybody to some degree, even when you do not plan to store any user data in Unicode.

This chapter explains the changes in V8 and helps you to understand the impact they have.

EBCDIC Code Page CCSID 500

HEX DIGITS 1ST → 2ND ↓	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0	(SP) SP010000	& SM030000	- SP100000	ø LO610000	Ø LO620000	° SM190000	µ SM170000	¢ SC040000	{ SM110000	} SM140000	\ SM070000	0 ND100000
-1	(RSP) SP300000	é LE110000	/ SP120000	É LE120000	a LA010000	j LJ010000	~ SD190000	£ SC020000	A LA020000	J LJ020000	÷ SA060000	1 ND010000
-2	â LA150000	ê LE150000	Â LA160000	Ê LE160000	b LB010000	k LK010000	s LS010000	¥ SC050000	B LB020000	K LK020000	S LS020000	2 ND020000
-3	ä LA170000	ë LE170000	Ä LA180000	Ë LE180000	c LC010000	l LL010000	t LT010000	· SD630000	C LC020000	L LL020000	T LT020000	3 ND030000
-4	à LA130000	è LE130000	À LA140000	È LE140000	d LD010000	m LM010000	u LU010000	© SM520000	D LD020000	M LM020000	U LU020000	4 ND040000
-5	á LA110000	í LI110000	Á LA120000	Í LI120000	e LE010000	n LN010000	v LV010000	§ SM240000	E LE020000	N LN020000	V LV020000	5 ND050000
-6	ã LA190000	î LI150000	Ã LA200000	Î LI160000	f LF010000	o LO010000	w LW010000	¶ SM250000	F LF020000	O LO020000	W LW020000	6 ND060000
-7	å LA270000	ï LI170000	Å LA280000	Ï LI180000	g LG010000	p LP010000	x LX010000	¼ NF040000	G LG020000	P LP020000	X LX020000	7 ND070000
-8	ç LC410000	ì LI130000	Ç LC420000	Ì LI140000	h LH010000	q LQ010000	y LY010000	½ NF010000	H LH020000	Q LQ020000	Y LY020000	8 ND080000
-9	ñ LN190000	ß LS610000	Ñ LN200000	` SD130000	i LI010000	r LR010000	z LZ010000	¾ NF050000	I LI020000	R LR020000	Z LZ020000	9 ND090000
-A	[SM060000	] SM080000	¦ SM650000	: SP130000	« SP170000	ª SM210000	¡ SP030000	¬ SM660000	(SHY) SP320000	¹ ND011000	² ND021000	³ ND031000
-B	. SP110000	$ SC030000	, SP080000	# SM010000	» SP180000	º SM200000	¿ SP160000	| SM130000	ô LO150000	û LU150000	Ô LO160000	Û LU160000
-C	< SA030000	* SM040000	% SM020000	@ SM050000	ð LD630000	æ LA510000	Đ LD620000	¯ SM150000	ö LO170000	ü LU170000	Ö LO180000	Ü LU180000
-D	(SP060000	) SP070000	_ SP090000	' SP050000	ý LY110000	¸ SD410000	Ý LY120000	¨ SD170000	ò LO130000	ù LU130000	Ò LO140000	Ù LU140000
-E	+ SA010000	; SP140000	> SA050000	= SA040000	þ LT630000	Æ LA520000	Þ LT640000	´ SD110000	ó LO110000	ú LU110000	Ó LO120000	Ú LU120000
-F	! SP020000	^ SD150000	? SP150000	" SP040000	± SA020000	¤ SC010000	® SM530000	× SA070000	õ LO190000	ÿ LY170000	Õ LO200000	(EO)

6.1 Conversion basics

Before we start to explain what Unicode is all about, we first describe DB2's code page support in general and clarify some of the terminology.

6.1.1 Code pages, CCSIDs, and code points

A *code page* is a set of assignments of characters to code points. Within a code page, a code point can have only one specific meaning.

A *code point* is a location in a code page. In the IBM environment, the hexadecimal representation of the location is preferred. For example, code point x'C1' represents character 'A' in the EBCDIC code page 500, as shown in the visual above.

A *Coded Character Set Identifier (CCSID)* is simply a number to identify a particular code page. For example, North Americans use the US-English code page denoted by a CCSID of 37. Germans use the CCSID 273. People in Turkey use CCSID 1026. Some of these include code points for specific characters in their language. Some have the same characters but are represented by different code points in different CCSIDs.

You can refer to the following Web page for a list of the IBM code pages:

http://www.ibm.com/servers/eserver/iseries/software/globalization/codepages.html

6.1.2 Encoding scheme

An *encoding scheme* is a collection of the code pages for various languages used on a particular computing platform. For example, the EBCDIC encoding scheme is typically used on z/OS and iSeries (AS/400). The ASCII encoding scheme is used on Intel-based systems (like Windows), and UNIX-based systems.

ASCII stores the character 'A' as x'41' and the number '1' as x'31'. As mentioned above, EBCDIC stores the character 'A' as x'C1' and '1' as x'F1'. This results in a different collating sequence in EBCDIC and ASCII.

- The collating sequence in ASCII is: space, numerics, upper case characters, lower case characters.
- The collating sequence in EBCDIC is: space, lower case characters, upper case characters, numerics.

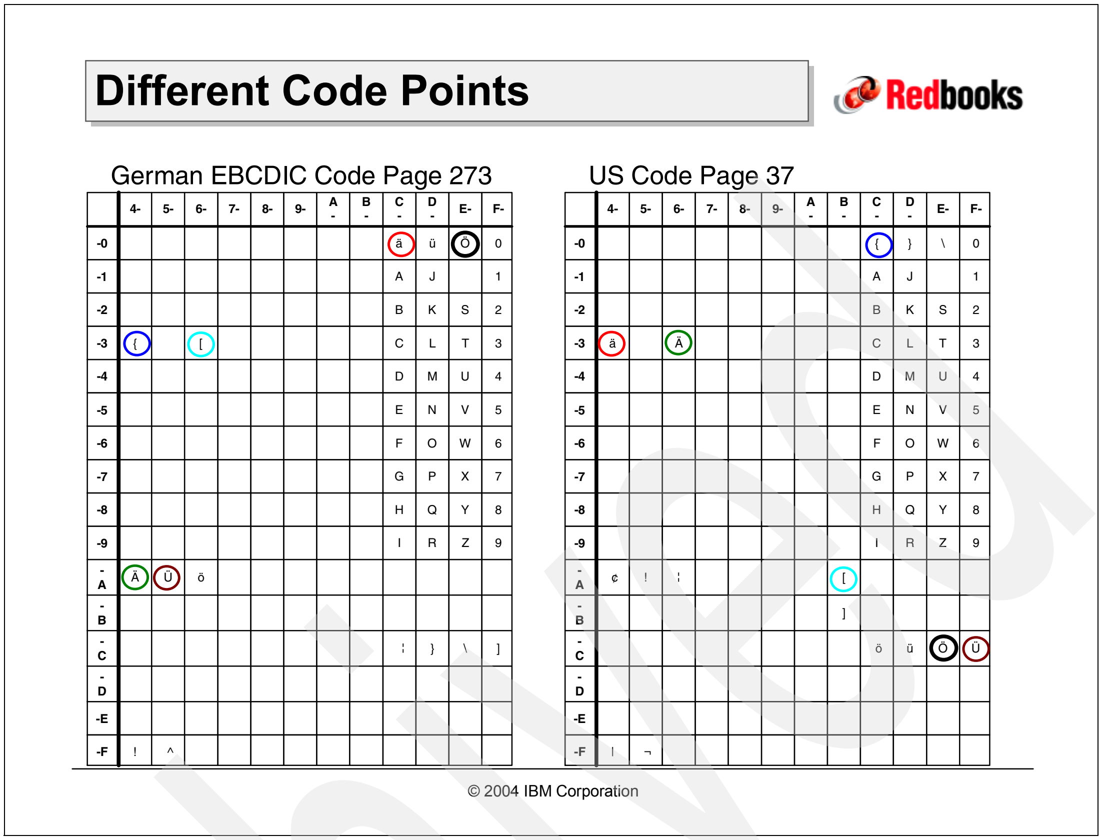
You can create table spaces with different encoding schemes within one database. Within one table space, however, you can only create tables with which use the same encoding scheme as the table space. For indexes, you cannot choose any specific encoding scheme. Indexes always inherit the table's encoding scheme.

6.1.3 CCSID set

A *CCSID set* comprises a single byte, mixed byte, and double byte CCSID which you can specify on panel DSNTIPF when you install a DB2 for z/OS subsystem.

Most languages can be encoded using a single 256 code point code page (sometimes also called a SBCS CCSID). In that case you can set up your DB2 subsystem as a MIXED=NO system. Then the mixed and double byte CCSIDs do not apply and they default to 65534, which is a reserved or dummy CCSID. For example, CCSIDs 1026, 273, and 37, mentioned above, do not have CCSIDs for mixed or double byte data. Refer to Appendix A of the *DB2 Installation Guide,* GC18-7418 for a list of available CCSID sets.

The languages that do not only use single byte CCSIDs are Chinese, Japanese, and Korean. They use double byte and mixed character sets due to the range and complexity of their symbols. Refer to "Mixed data characteristics" on page 500 for a discussion of mixed data.



6.1.4 Code page differences

Generally speaking, computers store all data as bytes. As shown on the visual above, the EBCDIC encoding for the character ’A’ is always stored as x’C1’. This is independent from the code page that has been used to bring this data into your DB2 for z/OS subsystem EBCDIC table. So for some characters, it doesn’t matter which code page you use. They are always represented by the same hex string. However, this is not true for all characters. The code pages shown above are not complete. We just picked a few characters to show you some of the differences.

Refer to Table 6-1 for a comparison of some characters, where the hexadecimal representation of the stored data is different between the German EBCDIC code page 273, and the US EBCDIC code page 37.

Table 6-1 Comparison of EBCDIC code pages

Character	Hex value in CCSID 273	Hex value in CCSID 37	Hex value in CCSID 500
{	x’43’	x’C0’	x’C0’
Ä	x’4A’	x’63’	x’63’
Ö	x’E0’	x’EC’	x’EC’
ä	x’C0’	x’43’	x’43’
[	x’63’	x’BA’	x’4A’
Ü	x’5A’	x’FC’	x’FC’

When comparing CCSID 37 with some other code pages, you may find more differences than just a few characters. For example, in CCSID 290 (or CCSIDs based on CCISD 290), the lower case characters are not in the "usual" place either.

When does Conversion Occur for Local Applications?

In general:

- No conversion

For *local* applications, conversion occurs when:

- Dealing with ASCII/Unicode tables
- Specified by application:
  - CCSID override in SQLDA (V2.3)
  - DECLARE VARIABLE (V7)
  - Application ENCODING bind option (V7)
  - CURRENT APPLICATION ENCODING SCHEME special register (V7)

6.1.5 When does conversion occur for local applications?

Before we start talking about how character conversion is performed, we want to make you aware of the situations which may or may not require conversion.

Generally speaking, we can say that conversion does not occur for local application. There are, however, some exceptions which require your data to be converted to another CCSID even if you are working locally. As also shown on the visual above, conversion occurs when:

- Dealing with ASCII or Unicode tables. See also 6.7, “Multiple CCSID sets per SQL statement” on page 546 for more information about when conversion occurs.
- Specified by application. The idea is that the application requests to have (some or all) of the data returned in a specific encoding scheme.
  - CCSID override in the SQLDA:

    The possibility for specifying a CCSID for a string column has been around since DB2 V2.3. For languages other than REXX, the assignment of any valid CCSID is done through field SQLNAME of SQLVAR. For REXX, the CCSID is in the SQLCCSID field.
  - DECLARE VARIABLE:

    If the SQL statement uses a host variable that has been declared in a DECLARE VARIABLE statement, then the DB2 precompiler automatically codes the equivalent setting in the SQLDA with a CSSID. This allows statements where a USING clause is not allowed to indicate that the data should be returned in a specific CCSID.
  - Application ENCODING bind option:

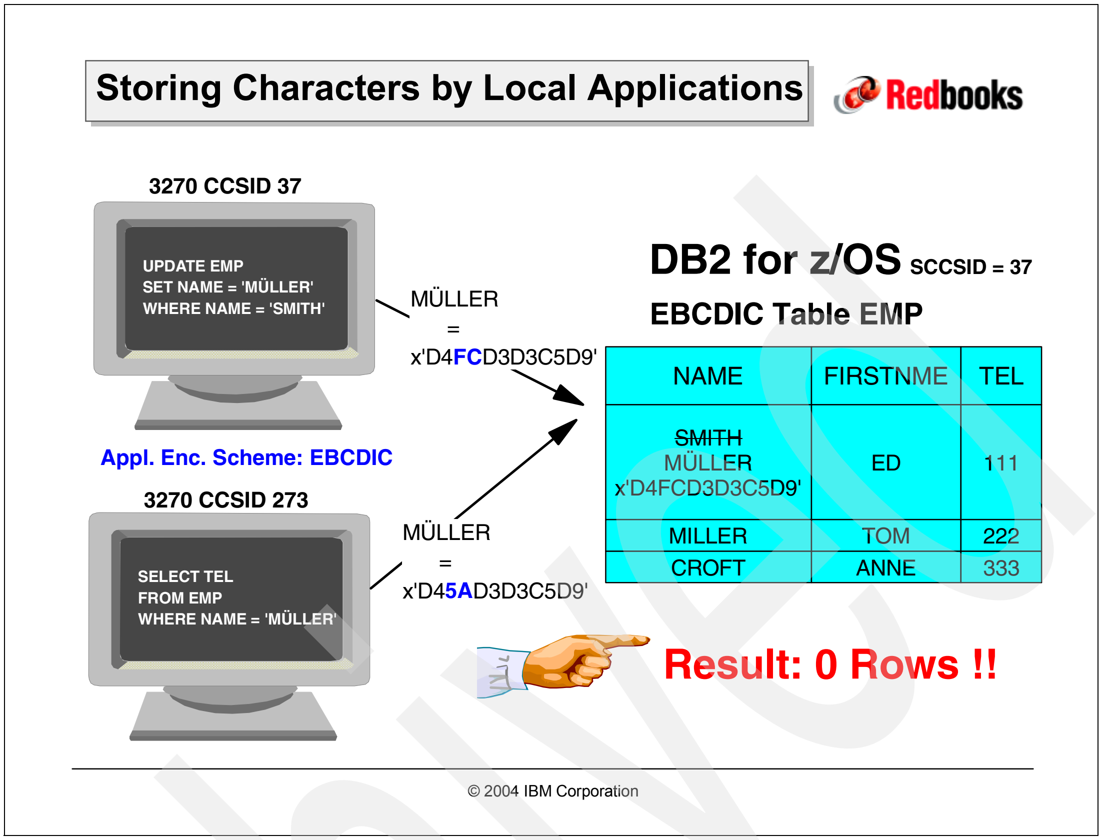
    This controls the application encoding scheme that is used for all the static statements in the plan/package.

The default is the system default APPLICATION ENCODING SCHEME specified at installation time (DSNZPARM APPENSCH). The default package application encoding option is not inherited from the plan application encoding option. For both plan and package, the system default is used when the ENCODING option is not specified during BIND.

– CURRENT APPLICATION ENCODING SCHEME special register enables an application to specify the encoding scheme that is being used by the application for dynamic statements.

  The value returned in the special register is a character representation of a CCSID. Although you can use the values ‘ASCII’, ‘EBCDIC’, or ‘UNICODE’ to SET the special register, the value set in the special register is the character representation of the numeric CCSID corresponding to the value used in the SET command. The values ‘ASCII’, ‘EBCDIC’, and ‘UNICODE’ are not stored.

DECLARE VARIABLE, the application ENCODING bind option, and the CURRENT APPLICATION ENCODING SCHEME special register have been introduced in DB2 with Version 7.



6.1.6 How characters are stored for local applications

Let us consider a case where we only work locally on a DB2 Version 7 subsystem using 3270 emulators; that is, we do not access data remotely through DB2 Connect. Even in this case, we have to be aware of how character encoding is performed, although most of the time, character conversion does not occur. For more details on when conversion occurs, see 6.1.5, “When does conversion occur for local applications?” on page 488.

The scenario on the visual above shows what happens when DB2 data is, for example, manipulated through ISPF by terminal emulators that use different CCSIDs. The 3270 emulation for the PC shown in the upper part of the visual uses CCSID 37. Because Miss Smith got married to Mr. Müller, and she wants to adopt her husband’s name, we update the name of employee ’SMITH’ in the EBCDIC table EMP to ’MÜLLER’. The hexadecimal representation of character ’Ü’ in CCSID 37 is ’FC’. That is, ’MÜLLER’ is stored as x’D4**FC**D3D3C5D9’ in the data set containing EMP.

Now somebody, using a different CCSID for her/his 3270 emulation, in our case the German CCSID 273, selects from table EMP and asks for all employees whose name is ’MÜLLER’. Since this user is using a different CCSID for their terminal emulation, ’Ü’ is now interpreted as the hexadecimal code point x’5A’, and the name MÜLLER looks like x’D4**5A**D3D3C5D9’ in hexadecimal.

DB2 compares this hexadecimal string with the data stored in EMP and does not find any match. That is, although the user of CCSID 273 is logically searching for the exact same person that the user of CCSID 37 just updated in the employee table, this entry is not displayed as a result row.

The situation described is also true for QMF and batch programs that have been coded using a different CCSID than the one that has been used to store the data.

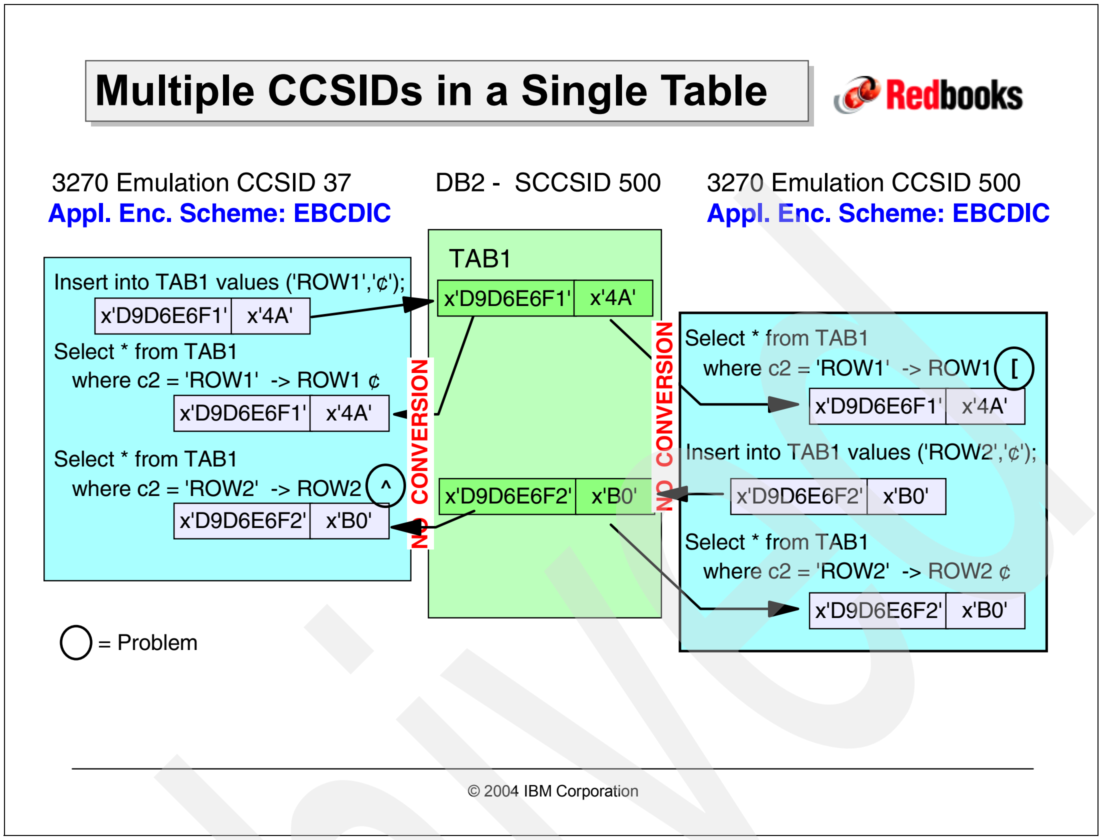
The described situation does not cause a problem for the data that is entered and retrieved by the same user. It is only a problem when different users using different CCSIDs have to retrieve each other's data.

To tackle this problem, you can force everybody to use the same CCSID on their terminal emulation program (37 in our example), but when people are spread out over different countries, using different languages and keyboards, that is not always an easy solution.

Another option is to use the ENCODING BIND option, also known as the application encoding scheme. This allows you to specify the encoding scheme used by the application. It tells DB2 how to interpret information in incoming and outgoing host variables.

If you can make sure that people using CCSID 37 in their terminal emulation program also use an application with the ENDCODING(37) BIND option, and users with 273 use ENCODING(273), you should be OK. Users using 37 in both the emulator and the application undergo no conversion (as before), but users using 273 in both their emulation and ENCODING BIND options will trigger conversion between the application encoding scheme's CCSID (273) and DB2's SCCSID 37. x'D45AD3D3C5D9' is converted to x'D4FCD3D3C5D9' before it is compared to data in the database, and the correct row will be retrieved. The use of the ENCODING BIND option is discussed in more detail in the next section.

In summary, it is extremely important that all users of a DB2 system who use applications where the data is not tagged with a CCSID should either use the same CCSID, or use techniques like the application encoding scheme, to convert the data to the correct CCSID before it is stored in the DBMS. Otherwise you end up with data corruption sooner or later.



Multiple CCSIDs in a single table - the problem

Because if is very important to specify the correct CCSID for your DB2 system, as well as the correct application encoding scheme for your applications, we now look at the potential problems and how to solve them using the application encoding scheme in more detail.

The problem is the same as described in the previous section. Because local DB2 applications, especially prior to V7, were not likely to do conversions, it is possible to end up with data from multiple CCSIDs stored in the same table, for example, because different people use a different CCSID in their terminal emulation program, and do not force proper conversion (via SQLDA CCSID override, by using the DECLARE VARIABLE construct, or by using the appropriate application encoding scheme).

This problem is not new. It has existed ever since day one of DB2. However, it was much less likely to be a problem in the early days, for a number of reasons:

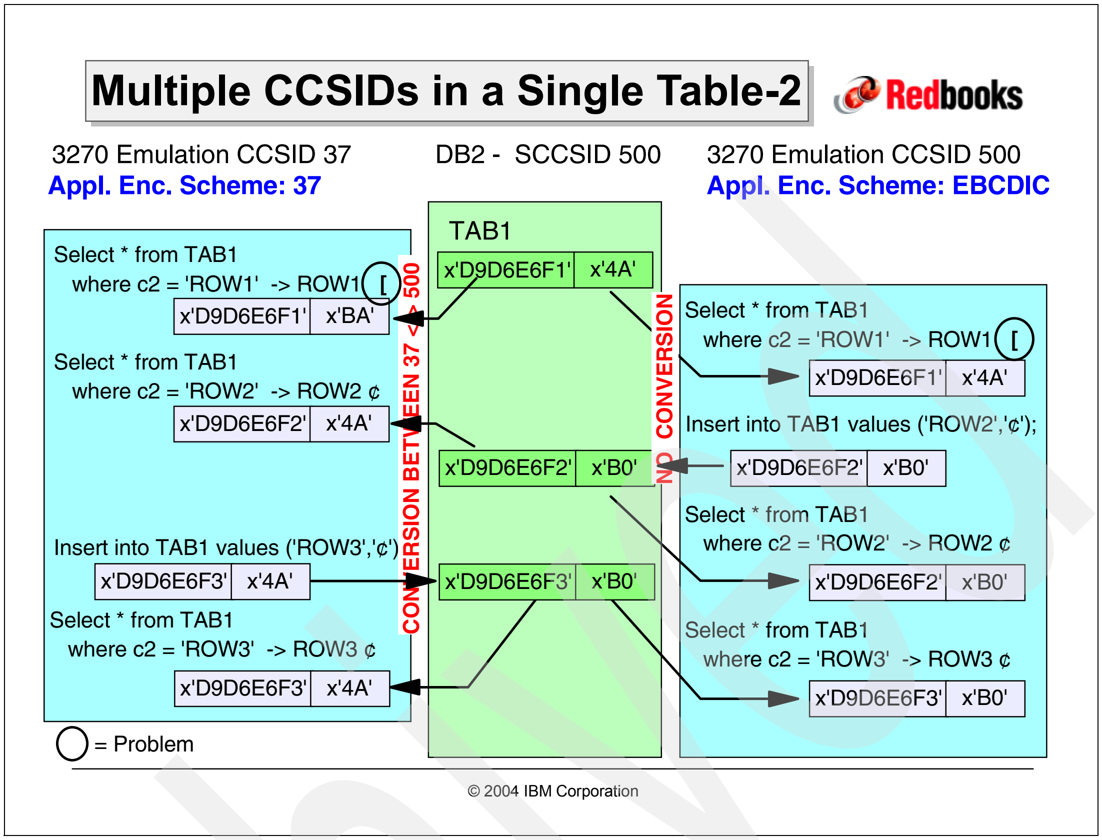
- Most applications then used local 3270 terminals that were connected via a local communication control unit, and the CCSID used by these terminals was defined inside the control unit. Therefore, it was much more difficult to change the CCSID compared to changing the CCSID in your 3270 PC terminal emulation program.
- Unlike today's configurations, where people from all over the world access a consolidated mainframe environment in a single location, in the early days, most DB2 applications were only accessed by local users, in the same building, or at most spread throughout the country.
- In addition, most people were just happy that they could store their data electronically, and were not too worried about getting all the "language specific" characters correctly represented.

Therefore, in many early DB2 installations, people did not worry too much about specifying the correct system EBCDIC CCSID (DSNHDECP SCCSID) parameter. This may have led to a situation where the data was inserted into DB2 via local applications using different CCSIDs, and without the proper override in the SQLDA (which was the only possibility to override the CCSID in pre-V7 local applications). The net result is that you can have a mix of data with different CCSIDs in a single DB2 table (or in multiple tables within the same DB2 subsystem).

This situation is illustrated in the figure above. The DB2 system uses SCCSID=500, and two 3270 emulators, one with CCSID 37 and one with CCSID 500. Both applications are bound with the ENCODING(EBCDIC) option. Both applications are inserting the same type of data, the character “¢” and the “ROWx” string.

As you can see, each application is able to correctly retrieve the row that it inserted, but sees the row that was inserted by the other application in a different (wrong) way. This is because DB2 does not do CCSID translation in this case, and stores the data as is in the table. But when user 1 (using CCSID 37) wants to retrieve data that was inserted by user 2 (using CCSID 500), translation should occur in order to be able to retrieve the data correctly.

Note: Not converting is only a problem for those characters that are different between the SCCSID code page and the code page used by the application.



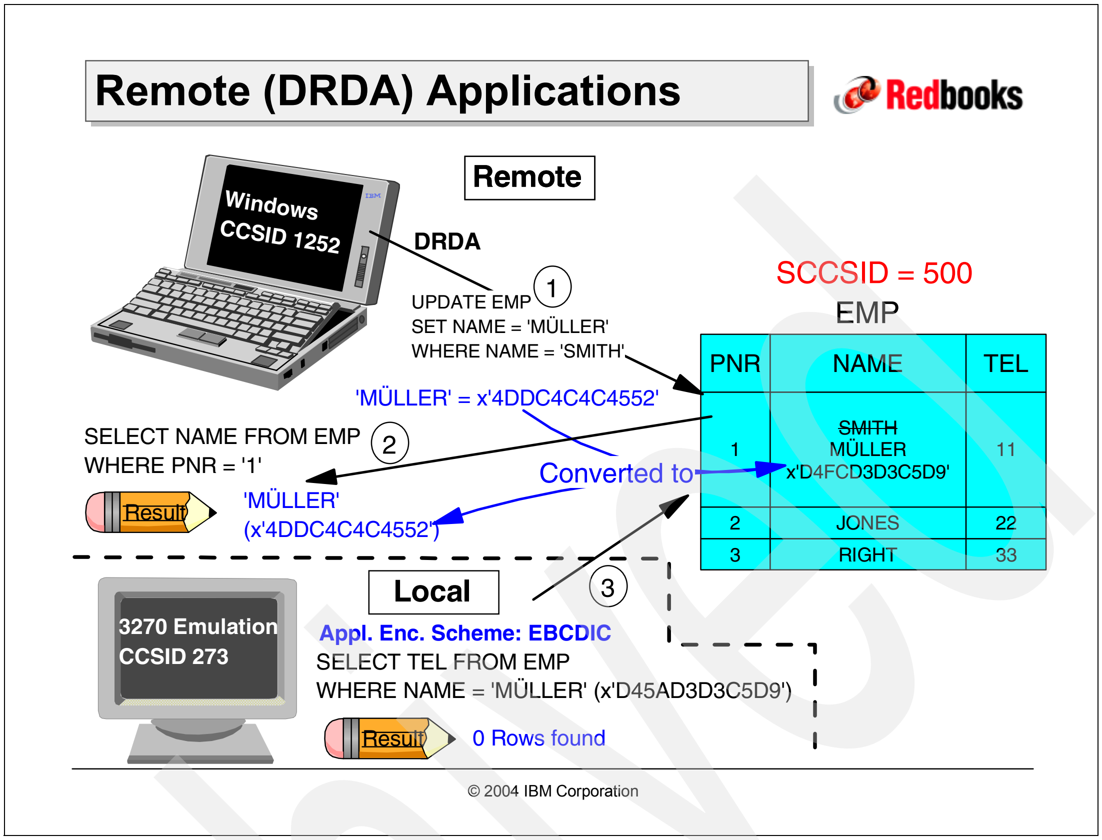
Multiple CCSIDs in a single table - a solution

To “force” translation, you can bind the application that is used by the user with CCSID 37 with the ENCODING(37) BIND option. This will trigger translation between 37 and 500, and vice versa. We did not choose to bind the application using CCSID 500 with the ENCODING BIND option, because the DB2 system SCCSID=500, and therefore there is no need to trigger conversion for that application.

The application that uses CCSID 37 (and the ENCODING(37) BIND option) can now correctly access all the data that was inserted by the other application, as well as all new data that it inserted after the application started to use the ENCODING(37) BIND option. The data inserted earlier cannot be correctly displayed, because with the ENCODING(37) BIND option, character conversion occurs between CCSID 500 (the CCSID of the table’s data) and 37 (the CCSID specified on the ENCODING BIND option). However, the data that was previously inserted into TAB1 via the CCSID 37 application was in CCSID 37 and stored as-is (because previously no conversion was done).

The application using CCSID 500 can correctly access all the data it inserted, as well as the newly inserted data from the other application (because that data is now converted at insert time from 37 to 500) and can be correctly retrieved by the CCSID 500 application without conversion. The only “problem” here is that the old data which was inserted by the application using CCSID 37 before the ENCODING BIND option was implemented (ROW1). In order to make this solution work 100%, you must track down which rows were inserted incorrectly and manually remove them, or convert them from 37 to 500.

Note that this is only a problem if you have inserted data incorrectly in the past. If you start a new application, and use the correct ENCODING BIND option where required, there is of course no need to clean up any old data, and all data will be accessible by all applications from day one.



6.1.7 When does conversion occur for remote applications?

If you are accessing your data remotely, character conversion is more likely to occur, since we are often dealing with different computer platforms using different encoding schemes. Consider these two cases, for example:

- The values of host variables sent from the requester to the current server, such as SELECT predicates or INSERT column values
- The values of result columns sent from the current server back to the requester, such as SELECTed columns

In either case, the string data can have a different representation at the sending and receiving system. The sample on the visual above assumes that the SCCSID for the DB2 for z/OS system is set to 500, which is the so-called International EBCDIC code page.

If you access your data through DRDA, character strings are always converted when different CCSIDs are involved. As shown above, (1) we update the name of employee ’SMITH’ and set her name to ’MÜLLER’. We assume that the workstation, which is used to run this update, uses the Windows CCSID 1252. That is, the code points of character string ’MÜLLER’ using CCSID 1252 are converted to the SCCSID EBCDIC 500 and stored in EBCDIC table EMP.

This conversion occurs the other way around when you select from table EMP (2) using a remote request from your workstation. Because conversion occurs for both directions, you can properly work with the data stored in table EMP.

DRDA automatically triggers character conversion when different CCSIDs are involved.

However, this is not the case for local applications. Let us assume that the SCCSID parameter for your DB2 subsystem is set to 500. On the other hand, your 3270 emulation program that is used to access your data in table EMP locally, is set to 273 (that is the German EBCDIC CCSID). This is shown in the bottom part of the figure. As discussed earlier in this chapter, data is usually not converted when the data is accessed locally (unless one of the conditions described earlier applies).

Therefore, as shown in (3), if you run a SELECT statement asking for all rows of table EMP, whose values in column NAME equal to the string MÜLLER (spelled in CCSID 273), you do not receive any result row.

The reason is that the string 'MÜLLER' is represented by x'D4**5A**D3D3C5D9' in EBCDIC CCSID 273. This hexadecimal string is used to compare the data that is stored in table EMP. Since CCSID 500 was used to store the data (when the DRDA user updated it), 'MÜLLER' has been stored as x'D4**FC**D3D3C5D9'. These two strings are not the same and therefore, the SELECT statement does not return any result row.

> **Important:** Again, this example clearly demonstrates the importance of specifying a CCSID in DB2 that matches the CCSID used by your terminal emulators and applications, or use the application encoding scheme bind option (or special register). Failure to do so can lead to losing data and data corruption.

To deal with this problem, you can use the application encoding scheme (ENCODING) BIND option. In this case, you can bind your local application with ENCDODING(273). This will trigger conversion between 273 and 500 (the DB2 SCCSID), and vice versa.

Two Conversion Methods

- Native DB2 - SYSIBM.SYSSTRINGS
- z/OS support for Unicode

```
COMMAND INPUT ===> /d uni,all

RESPONSE=SC63
 CUN3000I 02.37.19 UNI DISPLAY 752
  ENVIRONMENT: CREATED       04/17/2004 AT 10.08.19
               MODIFIED      04/17/2004 AT 10.08.20
               IMAGE CREATED 12/08/2003 AT 17.02.33
      SERVICE: CHARACTER      CASE           NORMALIZATION
COLLATION
      STORAGE: ACTIVE      424 PAGES
               LIMIT     51200 PAGES
     CASECONV: NORMAL
    NORMALIZE: DISABLED
      COLLATE: DISABLED
   CONVERSION: 00850-01047-ER              01047-00850-ER
               00037-01200(13488)-ER       01200(13488)-00037-ER
               00037-01208-ER              01208-00037-ER
               00437-01208-ER              01208-00437-ER
               00037-00367-ER              01252-00037-ER
               00037-01252-ER              00367-00037-ER
               00500-01200(13488)-ER       01200(13488)-00500-ER
               01047-01200(13488)-ER       01200(13488)-01047-ER
               01047-01208-ER              01208-01047-ER
               01208-01200-ER              01200-01208-ER
               01383-01200-ER              01200-01383-ER
               00932-01200-ER              01200-00932-ER
.....
```


6.1.8 Two ways of doing character conversion

There are two methods for character conversion support in DB2 for z/OS V8. The methods are used in the following order:

- DB2 catalog table SYSIBM.SYSSTRINGS (this has been available since DB2 V2.3):
  - If DB2 does not provide a conversion table for a certain combination of source and target CCSIDs, you will get an error message, unless a conversion can be found by the other conversion method.
  - If the conversion is incorrect, you might get an error message or unexpected output. To correct the problem, you need to understand the rules for assigning source and target CCSIDs in SQL operations.
- z/OS conversion services:

  z/OS conversion services are used since V7. They are certainly used for conversion to and from Unicode, but also support other conversions. You must customize z/OS Unicode support in order to have Unicode data in DB2. See Appendix B., “Unicode setup” on page 1039.

Note: APAR OA04069 provides automatic configuration of a default conversion image. When no UNI=xx is specified in the system parameters at IPL-time, Unicode Services will automatically load a pre-built image that contains all the EBCDIC and ASCII conversion tables. Detailed information of the conversion tables can be found in the APAR text. The pre-built image must be page fixed in storage and will require 39 Megabytes (9862 pages) of real storage. (A quick way to validate the successful loading and page fixing of the pre-built image, is by issuing the console command DISPLAY UNI,STORAGE. The result should show 9862 pages of storage utilized by the active image.)

Although the prebuilt image contains all possible conversions, 39 MB of page fixed real storage is a lot. Therefore, make sure that you have enough real storage available to accommodate the conversion image.

In order to save storage, it is recommended that you build you own conversion image, based on the conversions that are required in your installation. So far, from feedback from North American and European customers, they seldom have conversion images bigger than 2 MB, so they are much smaller than the default 39 MB.

DB2 Version 7 sometimes uses conversion services provide by the Language Environment product. LE conversions are no longer used in DB2 Version 8.

To ensure that conversions between EBCDIC and UTF-8 are as fast as possible, in some cases DB2 V8 performs so-called "inline conversion" instead of calling the z/OS Conversion Services. As a general rule, inline conversion can be done when a string consists of single-byte characters in UTF-8. This conversion enhancement is not available in V7, nor is it available for conversions between EBCDIC and UTF-16 and vice versa.

In addition, z/OS V1.4 has improved the performance of EBCDIC to UTF-8 (and vice versa) conversions by streamlining the conversion process, and V1.4 dramatically outperforms z/OS 1.3 conversions.

On top of that, zSeries machines have hardware instructions that assist CCSID conversion. These instructions were first implemented on the z900, and have been enhanced on the z990 machines.

Tip: The way to get conversions done as quickly as possible is to use DB2 V8, and use z/OS V1.4 or above, on a z990 machine.

Conversion Types RT- ES

Redbooks

Round-Trip (RT) conversions (e.g. ASCII or EBCDIC to Unicode)

- Data integrity (of character data) from the source CCSID to the target CCSID, and back to the source
- Potential incorrect representation of the characters in the target CCSID. However, when the characters are converted back to the source CCSID, they regain their original hexadecimal values and representation.
- RT can be a problem for application that see the data in the target CCSID

Enforced Subset (ES) conversions (e.g. Uni to ASCII/EBCDIC)

- Characters that exist in both the source and target CCSID are OK
- Characters in the source CCSID but not in the target CCSID are replaced by a substitution character.
- This substitution is permanent when converting back to the source CCSID because it is not possible to retrieve the original hexadecimal values.

6.1.9 Round-trip versus enforced subset conversion

The following methods are used for conversion:

- Round-trip conversion:

  The integrity of all character data is maintained from the source coded character set identifier (CCSID) to the target CCSID and back to the source.

  When performing a round-trip conversion, you may see incorrect representation of the characters displayed in the target CCSID. The integrity is preserved, however. When the characters are converted back to the source CCSID, they regain their original hexadecimal values and representation.

- Enforced subset conversion (substitution):

  Characters that exist in both the source and target CCSID have their integrity maintained. Characters in the source CCSID but not in the target CCSID are replaced. Replaced values are also referred to as substitution characters. All source characters that do not have a corresponding character in the target CCSID map to the same substitution character. For EBCDIC encoding, these appear on most display stations as a solid block. For ASCII encoding, these substitution characters appear differently.

  This substitution is permanent when converting back to the source CCSID because it is not possible to retrieve the original hexadecimal values.

DB2 uses a combination of RT and ES conversions. The trend, however goes toward ES conversions. When your SQL statement requires a conversion from/to Unicode, you can use:

- ASCII/EBCDIC to Unicode: Performed as round-trip conversions
- Unicode to ASCII/EBCDIC: Generally performed as enforced subset conversion

MIXED DATA

EBCDIC double-byte coded character set identifiers (CCSIDs)

National Language	MCCSID	SCCSID	GCCSID	User-defined Characters
Japanese (Extended Katakana)	930	290	300	4370
Japanese (Katakana-Kanji)	1390	8482	16684	4370
Japanese (Extended Katakana)	5026	290	4396	1880
Japanese (Extended English)	939	1027	300	4370
Japanese (Latin-Kanji)	1399	5123	16684	4370
Japanese (Extended English)	5035	1027	4396	1880
Korean	933	833	834	1880
Korean	1364	13121	4930	1880
Simplified Chinese	935	836	837	1880
Simplified Chinese	1388	13124	4933	1880
Traditional Chinese	937	28709	835	6204

6.1.10 Mixed data characteristics

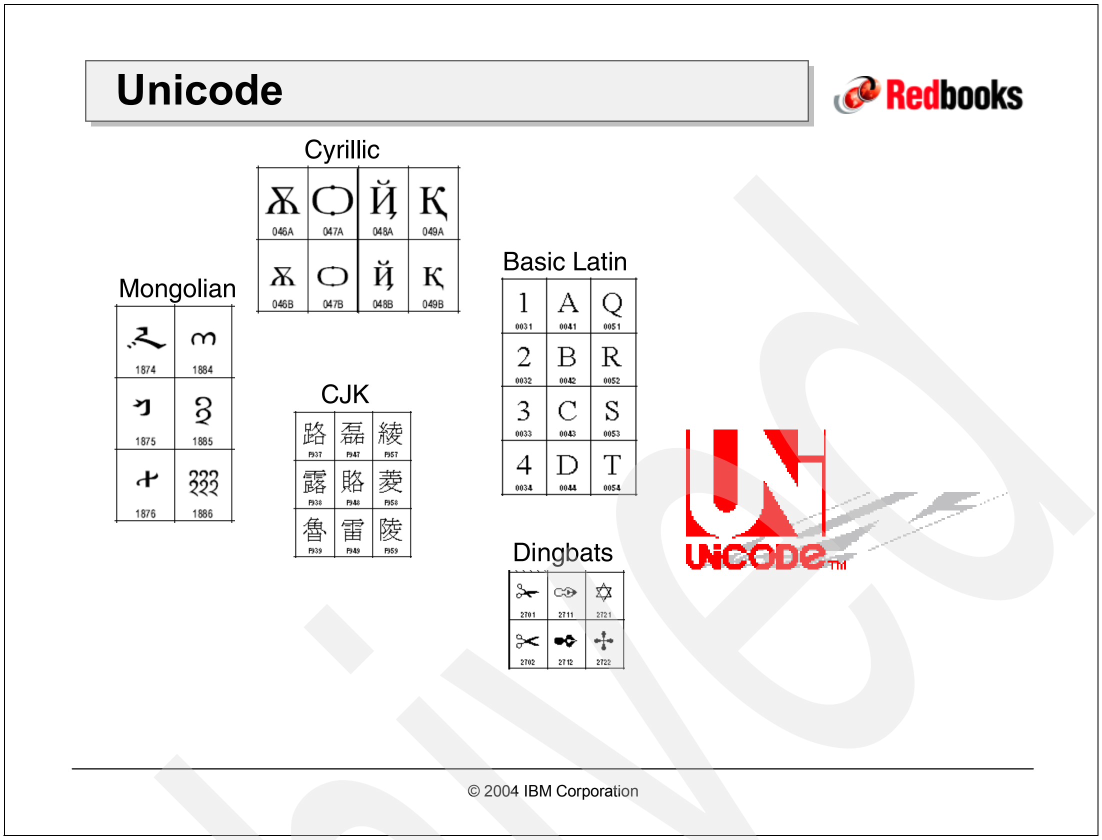
Within DB2, you have the option use just SBCS (Single Byte Character Set) CCSIDs, or use mixed and DBCS CCSIDs for the storage or your data. You can do this by either specifying MIXED DATA = YES or NO on installation panel DSNTIPF. If you decide to allow mixed EBCDIC and ASCII data within your DB2 subsystem, you must specify CCSIDs for mixed and DBCS.

Mixed and DBCS (Double Byte Character Set) CCSIDs are only available for a certain number of SBCS CCSIDs, namely, CCSIDs for Japanese, Korean, and Chinese. That is, for CCSIDS such as 273 for Germany, the mixed and DBCS CCSIDs do not exist.

The mixed data option has no effect on the Unicode CCSID field. Regardless of the setting for mixed data, Unicode UTF-8 data is considered to be mixed data and is processed according to the rules for mixed data.

Mixed data is capable of representing SBCS and MBCS (Multi-Byte Character Set). SBCS data can be compared to mixed without conversion to mixed, because it is a subset of the mixed repertoire. This is true for ASCII, EBCDIC, and Unicode.

- If MIXED=YES is specified for EBCDIC data, code points x’0E’ and x’0F’ have a special meaning. They are known as shift-in and shift-out controls for character strings that include double-byte characters.
- ASCII uses first byte code points. If the first byte is within a certain range, say x’81’ - x’9F’, then it is the first byte of a DBCS character. For example, x’8155’ is a DBCS character.



6.2 Unicode

The description of the basics of character conversion and CCSID usage in the previous sections shows that dealing with EBCDIC and ASCII code pages in a multinational environment might cause some difficulties. These difficulties exist because two encodings can use the same hexadecimal number for two different characters, or use different numbers for the same character. Servers, especially nowadays, need to be able to support many different encoding schemes. This means that whenever data is passed between different encodings or platforms, that data always runs the risk of data loss if not converted properly.

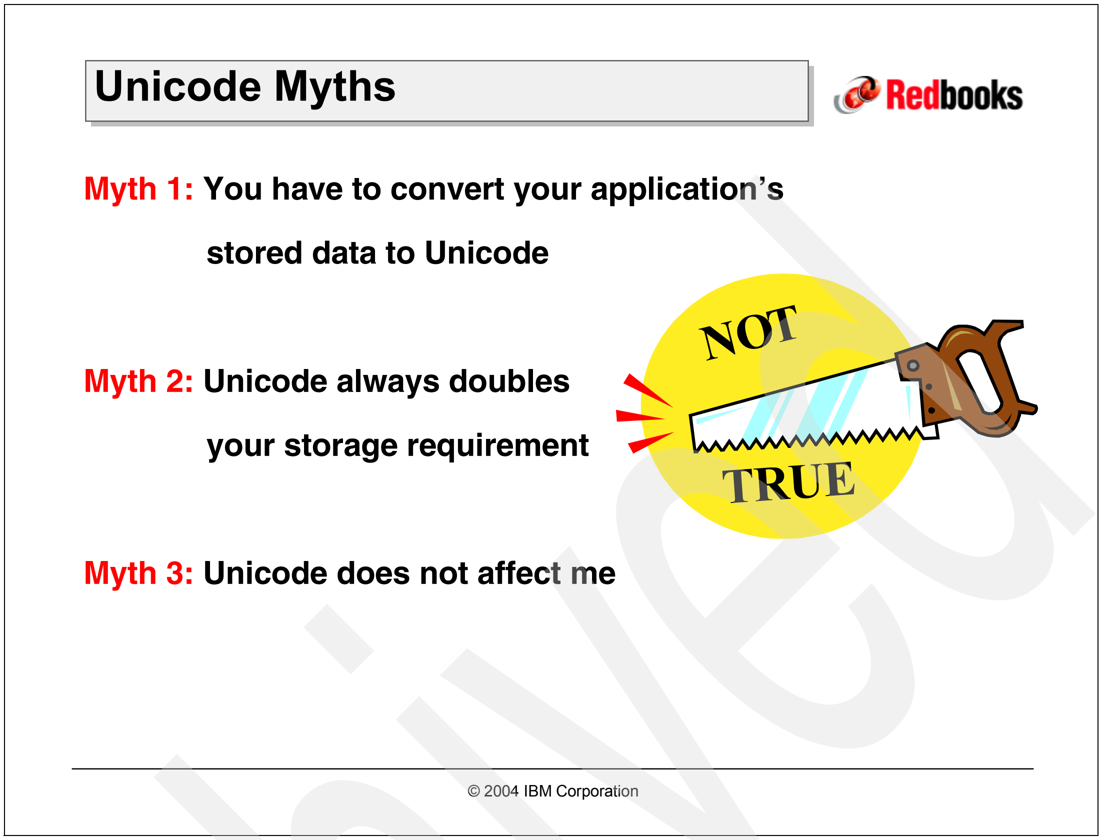
Unicode can make a big difference here!

Unicode provides a unique code point for every character, no matter what the platform, no matter what the program, no matter what the language is. The Unicode standard has been adopted by such industry leaders as Apple, HP, IBM, Microsoft, Oracle, SAP, Sun, and many others. Unicode is required by modern standards such as XML, Java, LDAP, CORBA 3.0, etc., and is the official way to implement ISO/IEC 10646. It is supported in many operating systems, all modern browsers, and many other products. The Unicode standard, and the availability of tools supporting it, are among the most significant recent global software technology trends.

Incorporating Unicode into client-server or multi-tiered applications and Web sites offers significant cost savings over the use of legacy character sets. Unicode enables a single software product or a single Web site to be targeted across multiple platforms, languages, and countries without re-engineering. It allows data to be transported through many different systems without corruption.

Since Unicode is still pretty new for DB2 for z/OS users, and quite different from what most DB2 people are used to, we discuss and describe the different Unicode standards in a bit more detail in the next few sections. For more information about Unicode, see also:

http://www.unicode.org



6.2.1 Let us destroy some myths

Since the enhanced Unicode support is one of the main changes for DB2 for z/OS Version 8, a lot has been written and talked about. However, there seem to be some myths surrounding Unicode that are hard to dispel.

Myth 1: You have to convert your application's stored data to Unicode

Version 8 enhances the infrastructure to support storing and retrieving data in Unicode format. This includes the conversion of the DB2 catalog to Unicode. However, the intention is not for all user data to be stored as Unicode, but to provide a choice. There is no need for you to convert all your data from EBCDIC to Unicode. With the support for multiple CCSIDs in a single SQL statement, there is even less need to convert all your data to Unicode than there is in Version 7. (However, you should be aware that there may be performance implications when using multiple CCSID set SQL statements.)

Myth 2: Unicode always doubles your storage requirement

There is a common misunderstanding, that the conversion of your data to Unicode always doubles its size. The DB2 Family uses Unicode Transformation Format UTF-8 and UTF-16. A detailed description of this format follows within the next few pages. It is important to know that in UTF-8, the first 127 code points are the same as that of 7-bit ASCII, with one byte being used for characters such as A-Z, a-z, and 0-9. Other characters are stored as one to four bytes, with accented characters often taking two bytes, and Far Eastern characters taking three to four bytes. Therefore, the actual increase for your system depends on the nature of your data.

Myth 3: Unicode does not affect me

Everyone is impacted by the changes in DB2 Version 8 regardless of whether user data is to be stored as Unicode. Most of the remainder of this chapter describes in detail why and where you are affected.

Importance of Unicode Knowledge

The DB2 catalog is stored in Unicode

The parsing of SQL statements is performed in Unicode

The DB2 precompiler generates DBRMs, which are stored in Unicode

For SQL statements with multiple CCSIDs, comparison of string values is often done in Unicode

The results of SQL statements can be returned to you in Unicode

The Unicode collating sequence is different from the one you are used to for EBCDIC data

Unicode data may take more space

6.2.2 Importance of Unicode knowledge

The enhanced Unicode support is one of the biggest changes within DB2 for z/OS V8. With V8, Unicode is around everywhere. Refer to the following list for a quick impression about which areas are impacted by the enhanced Unicode support:

- The DB2 catalog is stored in Unicode
- The parsing of your SQL statements is performed in Unicode
- The DB2 precompiler generates DBRMs, which are stored in Unicode
- If you use SQL statements containing multiple CCSID sets, the comparison of character values is very often done in Unicode
- The results of SQL statements are often returned to you in Unicode
- The Unicode collating sequence is different from the one you are used to for EBCDIC data.
- Unicode data may take more space. This is important for the storage of your data within DB2 and also for your application programs.

More detail regarding the topics listed above can be found in 6.5, “How does Unicode affect me?” on page 524.

Unicode Fundamentals

Current Unicode Standard, 4.0

- Contains character codes for 96248 characters
- Unique number assigned to every character
- Assignment between this number and the character the same for all encodings but are stored in different formats (UTF-8, UTF-16, ..)

Three forms of Unicode encoding

- UTF-8
  - Unicode Transformation Format in 8 bits
- UTF-16
  - Unicode Transformation Format in 16 bits
- UTF-32
  - Unicode Transformation Format in 32 bits
  - Replaces UCS-4

6.3 Unicode fundamentals

Historically, two projects started independent attempts to create a single unified character set. Those two projects were:

- ISO 10646 of the International Organization for Standardization (ISO)
- Unicode project

In the early nineties, both projects fortunately realized that two different unified character sets did not make sense. They joined their efforts and have agreed to keep the code tables of the Unicode and ISO 10646 standards compatible. Therefore, Unicode 3.0 corresponds to ISO 10646-1:2000, Unicode 3.2 to ISO 10646-2:2001, and the latest Unicode standard 4.0 to ISO ISO 10646-2:2003.

As stated previously, Unicode is a standard for coding every available character world-wide. The latest version of this standard, Version 4.0, contains 96248 graphic and format characters. The number of characters that are defined by the Unicode standard continues to grow and will probably never be complete. This is especially true if you consider that apart from living languages, some very old and dead languages are also part of the Unicode standard.

All Unicode versions since 2.0 are compatible. No existing characters will be removed, and only new characters will be added.

A special number is assigned to every single character. You can refer to `http://www.unicode.org` to find out more about Unicode. You can find the code charts, which show you which number is assigned to which character within the Unicode standard on Web site `http://www.unicode.org/charts/`. If you prefer to have a list of all available characters on your workstation so that you do not have to be online every time you want to find a specific character in the unicode code page, you can download a 'Unibook' from the Web site `http://www.unicode.org/unibook/`.

The international standard ISO 10646 defines the Universal Character Set (UCS). UCS and Unicode are just code tables that assign integer numbers to characters. Several alternatives exist for how a sequence of such characters or their respective integer values can be represented as a sequence of bytes.

Currently, the following three forms of Unicode encoding are probably the most used:

- **UTF-8:**

  Unicode Transformation Format in 8 bits.
- **UTF-16:**

  Unicode Transformation Format in 16 bits.
- **UTF-32:**

  Unicode Transformation Format in 32 bits. This encoding replaces UCS-4.

Refer to the following pages to find out more about how these encodings differ.

There are some additional Unicode encoding forms that we will only mention here for completeness. They are not used by DB2 to store data.

- **UCS-2:**

  Universal Character Set coded in 2 octets. UCS-2 was published as a part of the original UNICODE standard. UCS-2 is a fixed-width 16-bit encoding standard, with a range of 2*16 code points. The UNICODE standard originally hoped this many code points would be more than enough and hoped to stay with this range. UCS-2 is a subset of UTF-16.
- **UCS-4:**

  Universal Character Set coded in 4 octets. UCS-4 was also published as a part of the original UNICODE standard. UCS-4 is a fixed-width 32-bit encoding standard, with a range of 2*31 code points. The 2*31 code points are grouped into 2*15 planes, each consisting of 2*16 code points. The planes are numbered from 0. Plane 0, the Basic Multilingual Plane (BMP), corresponds to UCS-2 above.

Note: The assignment between the Unicode number and a character is unique for all forms of Unicode encodings such as UTF-8, UTF-16, and so on.

UTF-8 Encoding

UTF - Unicode Transformation Format

Unicode	UTF-8
U+00000000 - U+0000007F	0xxxxxxx
U+00000080 - U+000007FF	110xxxxx 10xxxxxx
U+00000800 - U+0000FFFF	1110xxxx 10xxxxxx 10xxxxxx
U+00010000 - U+001FFFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

Examples:

Character	Unicode	Bit Pattern (binary)	UTF-8(binary) Half bytes in hexadecimal
'A'	U-41	1000001	01000001 4 1 => x'41'
'§'	U-A7	10100111	11000010 10100111 C 2 A 7 => x'C2A7'

6.3.1 UTF-8

UTF-8 is one of the available ways to encode Unicode characters. As you can see from the upper table in the figure above, all Unicode characters from U+00000000 to U+0000007F can be represented in UTF-8 using one byte. In this byte, the high order bit is always set to 0. For this range, the UTF-8 encoding is identical to the one of ASCII.

For all Unicode characters from U+00000080 to U+000007FF, the UTF-8 representation needs two bytes. For characters from U+00000800 to U+0000FFFF, three bytes are used, and for U+00010000 to U+001FFFFF UTF-8 takes up 4 bytes.

To become a little bit more familiar with the way the encoding is done, let us look at some encoding examples:

Example 1:

The Unicode code point for character ’A’ is U+00000041. Since this number is within the first range in the upper table above, you can easily tell that the UTF-8 encoding for ‘A’ is also x’41’.

You can also perform the following steps to come to the same conclusion:

1. Determine the bit representation of x’41’. The resulting bit pattern is 100 0001 as shown in the lower table on the visual above. Since the high order bit of a UTF-8 encoded character that takes up one byte must always be 0, only bit patterns with a maximum of 111 1111 can be represented. As you probably can tell, binary 111 1111 is equal to x’7F’.
2. Join the 7 digit bit stream 100 0001 with the leading 0. The resulting 8 bit binary value is 0100 0001.

3. To easily derive the hexadecimal value, you must now split the 8 bits (1 byte) into to half bytes. The first four bits, 0100, are equal to hexadecimal 4, and the second four bits are equal to hexadecimal 1.
4. As a result, you can now tell that the UTF-8 hexadecimal representation for character ’A’ is x’41’.

Example 2:

Now let us look at the second row on the example shown on the visual above.

1. As mentioned before, you can find the code point for this character on Web site `http://www.unicode.org/charts/`. The code point for the paragraph sign (’§’) is U+000000A7 and larger than U+0000007F. Therefore the representation of this character in UTF-8 needs two bytes. The binary representation of x’A7’ is 1010 0111, which are eight bits.
2. Refer to the upper table on the visual above. You can see that if two bytes are used, the first three bits of the first byte is predefined to 110. Therefore it is always in the range of x’C0’ to x’FD’ and it indicates how many bytes follow for this character (two bits means a two-byte character). For the second byte, the first two bits are predefined to 10. Therefore it is always in a range of x’80’ and x’BF’.
3. Now take the bit sequence 1010 0111 and append this to the “available” bits from the back. For our example, the second byte ends up in binary 10***100111***, that is, the first two bits are the predefined ones and the remaining six bits are taken from the bit sequence 1010 0111 (x’A7’).

   The first byte ends up in 110000***10***, that is, the first three bits are the predefined ones, and the remaining two bits are placed on bit positions 7 and 8.
4. Now we must take byte one and two, and cut them into half bytes; that is, 1100 00***10*** 10***10*** ***0111*** has the following four half bytes:
   - 1100, which is 12 decimal, that is, x’C’
   - 0010, which is 2 decimal, that is, x’2’
   - 1010, which is 10 decimal, that is, x’A’
   - 0111, which is 7 decimal, that is, x’7’

The combination of these half bytes results in x’C2A7’, which is the UTF-8 encoding for the paragraph sign.

UTF-8 Encoding - 2

Examples:

Character	Unicode	Bit Pattern (binary)	UTF-8(binary) Half bytes in hexadecimal
ᅖ	U+1156	1000101010110	11100001 10000101 10010110 E 1 8 5 9 6 => x'E18596'

Encoding Ranges:

Unicode	UTF-8
U+00000000 - U+0000007F	x'00' - x'7F'
U+00000080 - U+000007FF	x'C080' - x'DFBF'
U+00000800 - U+0000FFFF	x'E08080' - x'EFBFBF'
U+00010000 - U+001FFFFF	x'F0908080' - 'F7BFBFBF'

Encoding UTF-8

The upper part of the visual above shows a third example of how to encode a character in UTF-8.

The Unicode code point of the character used above is U+00001156. The x’1156’ is equal to the binary number 1 0001 0101 0110. As shown on the previous visual, this number must now be adjusted to the UTF-8 bit pattern. The result is a 3-byte string. The final hexadecimal number, which is used to store the character above in the UTF-8 column, is x’E18596’.

The table in the lower part of the visual shows a different way to look at the encoding in UTF-8. As you can see, Unicode code points U+00000000 - U+0000007F are always encoded using one byte. The hexadecimal values for this byte are in the range between x’00’ and x’7F’.

The hexadecimal values for those characters whose Unicode code points are between U+00000080 and U+000007FF always lay in the range between x’C080’ and x’DFBF’, and so on. This table helps you to check if hexadecimal values you find for your character data are in a valid range, and it also helps you if you try to decode hexadecimal values for UTF-8 columns back to the Unicode code points. Refer to the next page to learn more about how to decode UTF-8 data.

Decoding UTF-8

The following data is stored as UTF-8 in a table:
x'50C39CC39F'

How to decode these values?

- x'50' is < x'7F' => represents 1 character, **U+50, 'P'**
- x'C3' is > x'7F' => must be at least two bytes
  - x'C39C' is between x'C280' and x'DFBF' => two byte char
  - x'C39C' = 1100001110011100
    - 1101 1100 (binary) = x'DC'
    - **U+DC is character 'Ü'**
- x'C39F' is between x'C280' and x'DFBF' => two byte char
  - x'C39F' = 1100001110011111
  - 000 1101 1111 (binary) = x'DF'
  - **U+DF is character 'ß'**

Decoding UTF-8

On the previous two pages, we explained how the encoding from a Unicode code point to its UTF-8 representation format is done. Encoding and decoding is usually done by DB2. What is important for you to know is how to work with Unicode hexadecimal constants in the future.

To analyze your data on a hexadecimal basis, the first thing you must be aware of is which type of Unicode encoding was used to store the data (UTF-8 or UTF16 are the only two formats used by DB2). This is important, because you must first find out if the characters you are currently looking at have been encoded using one, two, three, or even four bytes. Once you know the encoding format, you can recalculate the Unicode code points from the hexadecimal values.

Refer to the example shown on the visual above. Let us assume that your DSN1PRNT output shows string x'50C39C339F' and you know that the table storing this data uses Unicode, UTF-8.

Start looking at this string byte-wise.

The first byte, x'50', lies within the range of x'00' and x'7F'. This means that we have a one-byte representation of a character, where the hexadecimal value is the same as the Unicode code point. In our example, you can find the first character on position U+00000050, which is reserved for a 'P'.

The next byte, x'C3' is greater than x'7F'. Therefore, it cannot be a one-byte representation of a character. If we include the next byte, there is a two-byte string of x'C39C', which is within the range between x'C280' and x'DFBF'. This is a valid Unicode character.

X'C39C' equals binary 110***0 0011*** 10***01 1100***. If we remove the bits that are a fixed part of the two-byte UTF-8 string, 000 1101 1100 remains. 1101 1100 binary is equal to x'DC'. If you look at the Unicode code page (for example at http://www.unicode.org/charts/), you can see that U+000000DC represents the character 'Ü'.

The next remaining two bytes also start with x'C3', indicating another two-byte character. The x'C39F' is equal 110***0 0011*** 10***01 1111***. Again we must remove the fixed part of the bit pattern and end up with 000 1101 1111, which is equal to x'DF'. Code point U+000000DF in Unicode represents the character 'ß'.

UTF-16 Encoding

Each character is represented as either two or four bytes

- **For U+00000000 - U+0000FFFF:**
  - Hexadecimal two byte string
- **For U+00010000 - U+0010FFFF:**
  - Hexadecimal four byte string
    - U' = U value - U+00010000
    - Determine binary value of U' (yyyyyyyyyyxxxxxxxxxx)
    - W1 = 110110yyyyyyyyyy
    - W2 = 110111xxxxxxxxxx
    - Assign 10 high-order bits of U' to W1
    - Assign 10 low-order bits of U' to W2
    - Cut into half bytes
    - Calculate hexadecimal values for half bytes
    - Combine hexadecimal values

6.3.2 UTF-16

As described previously, UTF-8 encoding uses one to four bytes to represent a Unicode character, that is, a Unicode code point. The first 127 characters, whose representation in UTF-8 only use one byte, are the main characters used for a lot of languages. Only a few characters which are part of the EBCDIC CCSIDs need two or even more bytes for the encoding.

For languages such as Japanese, Korean, and Chinese, the character encoding in EBCDIC was DBCS, that is, all characters need two bytes for the representation of every single character. With UTF-8, most of those characters are represented using three bytes.

In UTF-16 encoding, characters are represented using either two or four bytes. The first 65536, that is, x’10000’, can be represented with two bytes.

This means that characters in languages such as Japanese, Korean, and Chinese can be represented in two bytes (as their code points are before U+10000). This is a big advantage for these languages, because otherwise, if they use UTF-8, their storage needs would increase by 50 percent.

(There is an extension mechanism in UTF-16 (called surrogates - or planes) which allows for a million or so characters to be coded as two successive code points. (2 times 2 bytes)).

The visual above gives you a formal description of how Unicode code points are represented in UTF-16. For all characters, whose Unicode code points are less then U+00010000, the encoding is just the two-byte hexadecimal value.

For characters whose Unicode code points are greater than or equal to U+00010000 (and up to U+00010FFFF), the encoding is a bit more complicated, as described on the visual. (Characters beyond U+00010FFFF cannot be represented in UTF-16.) You can calculate the hexadecimal representation in UTF-16 as follows:

1. Calculate a value U' where U' = U - U+00010000.
2. Take the hexadecimal value of U' and determine the binary value for it. Consider U' to be something like yyyyyyyyyyxxxxxxxxxx.
3. Assume two binary strings:
   - W1= 110110yyyyyyyyyy
   - W2 = 110111xxxxxxxxxx
4. Fill up W1 and W2 with the y and x values taken from the binary string of U', that is, assign the 10 high-order bits of the 20 bit U' to the 10 low-order bits of W1, and the 10 low-order bits of U' to the 10 low-order bits of W2.
5. W1 contains 16 bits, that is, two bytes or four half bytes. The same is true for W2. Convert W1 and W2 to hexadecimal.
6. Combine the eight hexadecimal values to one value, which is now the UTF-16 encoding of the corresponding character.

This description is very formal and may not be so easy to understand. Refer to the next visual for an example of how to encode a Unicode code point in UTF-16.

UTF-16 Encoding Examples

- Character: 'A' Unicode code point: U+00000041
  - UTF-16: x'0041'
- Character: ň ' Unicode code point: U+00000148
  - UTF-16: x'0148'
- Character: ' Unicode code point: U+000200D0
  - Calculate: x'200D0' - x'10000' = x'100D0'
  - x'100D0' = 0001 0000 0000 1101 0000 (binary)
  - W1 = 1101100001000000
    - 1101 = x'D', 1000 = x'8', 0100 = x'4', 0000 = x'0'
  - W2 = 1101110011010000
    - 1101 = x'D', 1100 = x'C', 1101 = x'D', 0000 = x'0'
  - UTF-16 : x'D840DCD0'

UTF-16 encoding examples

As shown in the visual above, characters such as 'A' or any other character, which is represented by a Unicode code point less than U+0010000 the Unicode number, equals the hexadecimal value in UTF-16, which is used to store the data.

For the character that is represented by U+000200D0, the encoding works as follows:

- Calculate x'200D0' - x'10000'. The result is x'100D0'.

> **Tip:** The easiest way to do all these hexadecimal and binary calculations is to use a scientific calculator that comes with your Windows operating system, for example. You can simply select between the different bases which you want to use for your calculations, and you can also use Edit -> Copy to copy long binary strings.

- The binary value for x'100D0' is 1 0000 0000 1101 0000 (17 bits). Add enough leading zeros so that the string length is 20 bits, that is, the string we start looking at is 0001 0000 0000 1101 0000.
- Now you must calculate the first two bytes of the hexadecimal value, which is used to encode the character shown above. To do that, first take the bit pattern W1 as shown on the previous visual and replace all with the first 10 digits of the 20-bit string that we have just calculated. You end up with a 16-bit string, which can be divided into four half bytes:
  - 1102 = x'D'
  - 1000 = x'8'
  - 0100 = x'4'
  - 0000 0 x'0'

These hexadecimal values can be combined to x'D840' and represent the first two bytes of the four-byte encoding.

You can now do the same with W2. You end up with x'DCD0', which are the last two bytes of the four-byte representation.

- As a last step, you must combine the two two-byte strings, that you have just calculated. The value which is stored in your DB2 table is x'D840DCD0'.

UTF-16 Decoding

Check the value of the first two bytes. Let this value be W1.

If W1 < x'D800' or W1 > x'DFFF'

- The character value U is the value of W1.

If W1 is between x'D800' and x'DFFF'

- Check if there are two more bytes. Let these two bytes be W2.
  - If there are no more two bytes, this is not a valid character
  - Check if W2 is between x'DC00' and x'DFFF'. If not, the 4 bytes do not represent a valid character
- Construct a 20-bit Unicode character
  - Take the 10 low-order bits of W1 as the 10 high-order bits
  - Take the 10 low-order bits of W2 as the 10 low-order bits
- Add x'10000' to the number you have just created and obtain the character from the Unicode tables

UTF-16 decoding

As for UTF-8, you do not simply have a need to know how UTF-16 is encoded. Probably more often, you must decode whatever you find in your DB2 data sets.

If you know that your data is encoded in UTF-16, you must start looking at your data somewhat differently than for UTF-8, because UTF-16 always uses either two or four bytes. Instead of just checking the value of the first byte, you must check whether the first two bytes are less than x’D800’ or greater than x’DFFF’. If this is the case, the decoding is very simple, because the hexadecimal value of these two bytes equal the Unicode code point.

If the first two bytes are between x’D800’ and x’DFFF’, you must check whether there is another pair of bytes available. If this is not the case, the encoding of the first two bytes is not a valid UTF-16 encoded character.

Otherwise (and this should be the case), you must continue to check the next two bytes (W2). Their value should be between x’DC00’ and x’DFFF’. If the second group of two bytes have any other value, the four bytes you are currently looking at are not a valid UTF-16 representation.

If the first group of two bytes, W1, is between x’D800’ and x’DFFF’, and the second group of two bytes, W2, is in the range between x’DC00’ and x’DFFF’, you must construct a 20-bit long bit stream. Take the 10 low-order bits of W1 as the 10 high-order bits of the Unicode code point, and take the 10 low-order bits of W2 as the 10 low-order bits of the Unicode code point. Convert this binary stream to hexadecimal and add x’10000’ to this value. The hexadecimal value you receive this way is the Unicode code point. You can now use any listing of Unicode characters to search for the corresponding character.

Let us check out the next visual for a sample of the decoding procedure described above.

UTF-16 Decoding Example 1

Hexadecimal string found in a UTF-16 table:

x'D800 DF38 1820 D840 DCD0'

Which characters are represented here?

1. x'D800' is not less than x'D800' and is not greater than x'DFFF'
2. x'D800' is between x'D800' and x'DFFF'
3. There are at least two more bytes, which are x'DF38'
4. x'DF38' is between x'DC00' and x'DFFF'
5. x'D800' = 1101100000000000 (binary)
6. x'DF38' = 1101111100111000 (binary)
7. 00000000001100111000 (binary) = x'338'
8. Calculate x'338' + x'10000' = x'10338'
9. U+10338 represents GOTHIC LETTER THIUTH, which is: 𐌸

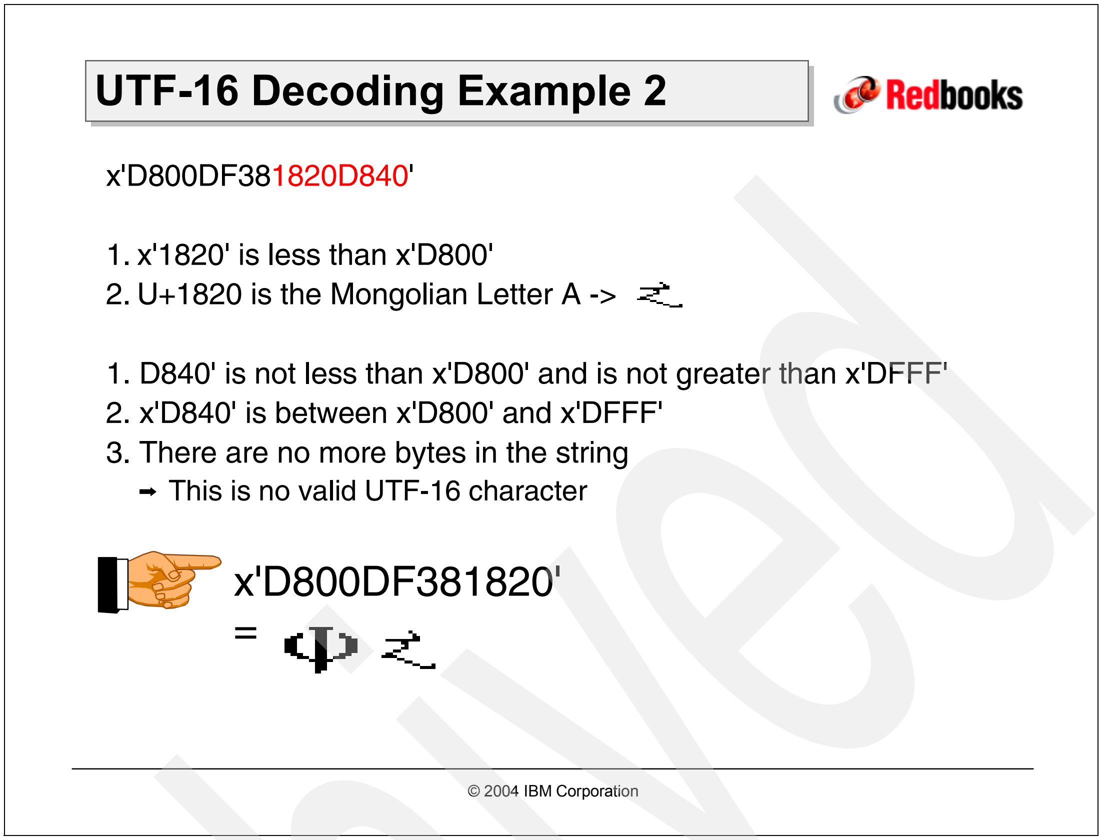
UTF-16 decoding examples

After looking at the formal way to decode UTF-16 encoding, we provide an example to illustrate the process.

Assume that you look at some data, for example, from a DSN1PRNT output, and find the following hexadecimal string: x'D800DF381820D84DCD0'. You know that the table space you are looking at only contains UTF-16 data, and you want to know, first of all, whether the data that is stored is valid UTF-16 encoding, and second, which characters are stored here. To find the answers to your questions, you must perform the steps described on the visual.

- Since UTF-16 encoding always uses two or four bytes to represent a character, you must start looking at the first two bytes. The first two bytes are x'D800'. If they are less than x'D800' or greater than x'DFFF', their hexadecimal value equals the Unicode number of the represented character. This is not the case in our example.
- Next, check whether x'D800' is between x'D800' and x'DFFF'. This is true for our example, because "between" includes the upper and lower ranges. The fact that the value is in the specified range indicates that this might be a UTF-16 character, which uses four bytes for the character representation.
- If, in UTF-16 a character is represented using four bytes, the second two bytes must always be in a range between x'DC00' and x'DFFF'. If this is not the case, the four bytes you are currently looking at are invalid and do not represent a UTF-16 encoded character.
- In our example, the second two bytes are x'DF38', which is definitely within the specified range.
- Now we need three more steps to identify the Unicode code point. Convert x'D800', which are the first two bytes, to a binary representation.

- Do the same for bytes three and four.
- Take the 10 low-order bits of byte one and two as the 10 high order bits of a new binary string, and continue with the 10 low-order bits of byte three and four as shown under step 7 on the visual above.
- Now add x'10000' to the hexadecimal value of the binary string you have just constructed. (Remember that when we encoded a UTF-16 string, we subtracted x'10000', so for decoding we need to add it back.)
- The result of the prior addition now equals the Unicode number, which you can just look up in appropriate lists of Unicode characters. In our case, the Gothic letter THIUTH is stored here.



UTF-16 decoding examples continued

Now that we have successfully identified the first character, we can continue with the next two bytes. The next two bytes contain x'1820'.

This is a very easy case, because x'1820' is less than x'D800', which means that the Unicode number equals this one. If you look up U+1820, you can see that it represents the Mongolian Letter A.

Now let us check the remaining part of the hexadecimal string, which is x'D840'.

- x'D840' is between x'D800' and x'DFFF'. That is, it seems to be the first two bytes of a four-byte encoded character.
- Unfortunately there are no more hexadecimal values available, which means that x'D840' is not valid.

Character Encoding Comparison

Redbooks

Character	ASCII	UTF-8	UTF-16 (Big Endian format)
A	'41'x	'41'x	'0041'x
a	'61'x	'61'x	'0061'x
9	'39'x	'39'x	'0039'x
Å (The character A with Ring accent)	'C5'x	'C385'x Note: 'C5'x becomes double byte in UTF-8	'00C5'x
顠 U+9860	'CDDB'x (CCSID 939)	'E9A1A0'x	'9860'x
𠃐 U+200D0	N/A	'F0A08390'x	'D840DCD0'x

Note: UCS-2/UTF-16 and UCS-4/UTF-32 are using a technique called Zero Extension

6.3.3 Character encoding comparison

Refer to the figure above for a summary example of the different encoding schemes and formats. As you can see again, UTF-8 data varies between one and four bytes. UTF-16 always uses at least two bytes.

Today, countries that use CCSIDs like ASCII 939 (which is for DBCS), now need two bytes for the encoding of their characters. This is also true if they use UTF-16 for the encoding of their Unicode data, whereas if they used UTF-8, almost all characters would need three bytes for the encoding.

For UTF-16, an additional note indicates that the BIG ENDIAN FORMAT is used here.

Endianess only affects UTF-16 and UTF-32, which is not covered in this chapter. There is a distinction between these types:

- Big Endian:

  Used by pSeries, zSeries, iSeries, Sun, HP.

  For a four byte word, the byte order is 0,1,2,3. For a 2-byte word it is 0,1.

  The encoding of character 'A' on the visual above is a 2-byte word, which is stored as x'0041' in Big Endian format.

- Little Endian:

  Used by Intel based machines, including xSeries.

  For a four byte word, the byte order is 3,2,1,0. For a 2-byte word it is 1,0.

  The encoding of character 'A' in the Little Endian Format would be x'4100'.

Since this book deals with DB2 for z/OS, within DB2, only the Big Endian format is used.

Within a byte, the Endianess does not matter. A byte is always ordered from leftmost significant bit to rightmost least significant bit. Bit order within a byte is always 7,6,5,4,3,2,1,0.

Unicode CCSIDs within DB2

Redbooks

DSNHDECP - USCCSID/UMCCSID/UGCCSID

CREATE DATABASE/TABLESPACE/TABLE/stored procedure/UDF - CCSID UNICODE option

CCSIDs used for Unicode encoding in tables depends on the column type and subtype (FOR xxx) option in your DDL:

- CHAR / VARCHAR
  - FOR SBCS DATA
    - 7-bit ASCII CCSID 367
    - Is a subset of UTF-8
  - FOR MIXED DATA
    - UTF-8 CCSID 1208
    - This CCSID is the **default for Unicode tables**
    - Allowed, even with MIXED=NO specifies on DSNTIPF
- FOR GRAPHIC / VARGRAPHIC
  - UTF-16 CCSID 1200

6.4 Unicode data within DB2

Until now, we have only described Unicode in general and talked about the two encoding formats, UTF-8 and UTF-16, which are used by DB2. Let us now look in more detail at how Unicode can be used within your DB2 for z/OS subsystem.

6.4.1 Specifying the Unicode CCSID in DSNHDECP

With DB2 V8, as well as in DB2 V7, you must specify valid Unicode CCSIDs on installation panel DSNTIPF. Currently, the only valid CCSID here is 1208.

The 1208 equals UTF-8 as encoding format for your Unicode mixed data. By specifying 1208, which is the default setting for UMCCSID, the CCSIDs for USCCSID is set to 367, which equals the 7-bit ASCII encoding. In addition to that, UGCCSID is set to 1200, indicating that UTF-16 is used for GRAPHIC Unicode data.

As you can see, the default setting of 1208 in field 'Unicode CCSID' on panel DSNTIPF is the CCSID for mixed data. This is different from what you are used to, regarding EBCDIC data. For EBCDIC, you usually only specify the CCSID for your SBCS data (unless you are in a mixed environment). The reason for this is that the default setting for Unicode data is also the mixed CCSID. This is even true if your setting for the parameter MIXED is set to NO.

You will probably not use the Unicode SBCS 367 very often, because you can just encode 127 characters, that is, the seven bit ASCII characters using this CCSID. The advantage of SBCS data in a Unicode environment is that it will perform faster for certain types of processing. LIKE predicates for example, will perform faster.

6.4.2 Specifying Unicode as the encoding scheme when storing DB2 data

You can specify that you want to use Unicode within your DB2 subsystem in many different places. Refer to the list below for some examples:

- CREATE DATABASE mydb CCSID UNICODE

  The default encoding scheme for objects (that do not explicitly specify the CCSID clause) created within this database is Unicode. You can, however, create table spaces using other encoding schemes within this database.

- CREATE TABLESPACE myts IN mydb CCSID UNICODE

  This is the way to explicitly specify to use Unicode for a table space, which resides in a database. As mentioned above, it is possible to have table spaces with different encoding schemes within one database.

- CREATE TABLE t1 (c1 char(10)) CCSID UNICODE

  You can specify an encoding scheme for every table that you create. You cannot, however, have tables with different encoding schemes within one table space. Also, you cannot create a table within a table space specifying a different encoding scheme than the table space has.

  If you do not specify a CCSID within your CREATE TABLE statement, the created table inherits its encoding scheme from the database or table space in which it resides. If you create a table which implicitly creates a table space, and you do not specify a CCSID in this CREATE statement, the table space and the table inherit the database's CCSID.

- CREATE PROCEDURE mysp (in in_parm1 char(10) CCSID UNICODE) ...

  Within the parameter list of your CREATE PROCEDURE statement, you can specify the CCSID that is used to interpret your parameters. If you do not specify anything, the SCCSID for the default encoding scheme is used.

If you do not specify the CCSID option, the system's "DEF ENCODING SCHEME" (DSNHDECP ENSCHEME) parameter determines the default encoding scheme. Acceptable values are EBCDIC, ASCII, or UNICODE. The default is EBCDIC. ENSCHEME specifies the default format in which to store data in DB2. DDL uses the default encoding scheme in the following cases:

- CREATE DATABASE
- CREATE DISTINCT TYPE
- CREATE FUNCTION
- CREATE GLOBAL TEMPORARY TABLE
- DECLARE GLOBAL TEMPORARY TABLE
- CREATE TABLESPACE (in DSNDB04 database)

Note: The "encoding scheme" parameter is not to be confused with the "application encoding scheme" parameter. "Encoding scheme" applies to how objects (that you specify in DDL) will store their data by default. "Application encoding scheme" applies to how applications need to see their host variables interpreted by DB2 and returned from DB2.

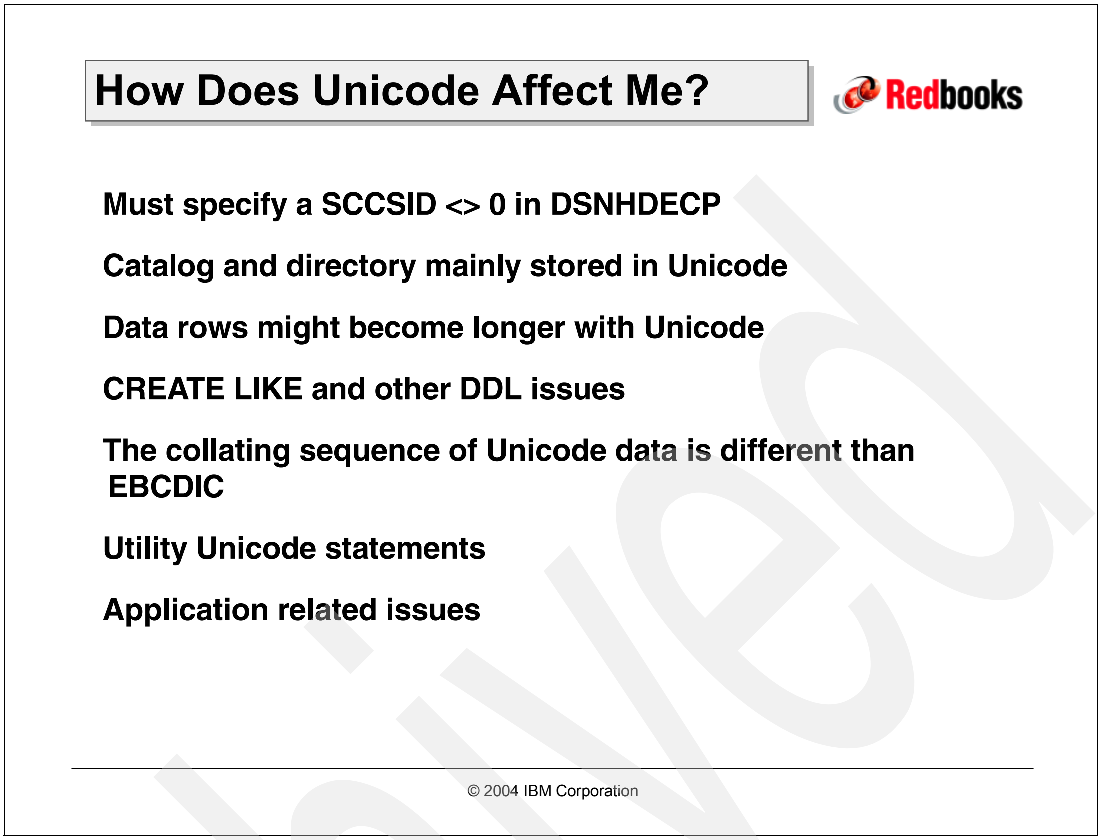
6.4.3 Unicode encoding form depends on the DB2 column type

As shown in the figure above, the actual format of Unicode encoding depends on the DB2 column type and the column subtype. Unicode, of course, only applies to character data types, CHAR, VARCHAR, GRAPHIC and VARGRAPHIC in your Unicode table.

- When you use CHAR or VARCHAR, you have the option to specify a column subtype by using the FOR clause. You can specify FOR SBCS DATA, FOR MIXED DATA, and FOR BIT DATA.
  - When you specify FOR SBCS DATA, the data is stored using CCSID 367 (also known as 7-bit ASCII). CCSID 367 is a subset of UTF-8, and only allows you to data that can be represented by a single UTF-8 character.
  - When you specify FOR MIXED DATA, the data is stored using CCSID 1208 (also known as UTF-8).

  Note: FOR MIXED DATA is the default when you create a CHAR or VARCHAR column in a Unicode table. Your system does NOT have to be a MIXED=YES system to allow the specification of the FOR MIXED DATA clause on CHAR/VARCHAR Unicode column (as is the case for EBCDIC and ASCII).

- When you use GRAPHIC or VARGRAPHIC columns in a Unicode table, their data is encoded in UTF-16 (CCSID 1200).



6.5 How does Unicode affect me?

With DB2 for z/OS Version 8, Unicode is ubiquitous. Everybody is affected by the changes regarding Unicode that come with Version 8.

In this section we describe the following areas that are affected by Unicode:

- You must now specify a valid SCCSID in your DSNHDECP.
- Most parts (character columns) of the catalog table spaces and one of the directory are stored in Unicode.
- Data rows may become longer when stored in Unicode. This can impact your application programs as well as your database design.
- The CREATE LIKE SQL statement is one of several impacts regarding DDL.
- The collating sequence of Unicode data is different from EBCDIC. This can affect ORDER BY results, as well as range predicates.
- Literals are affected by Unicode.
- You need to keep an eye your buffer pool, EDM pool, and so on, since data stored as Unicode may require more space to store, and you may have to increase the size of these pools.
- Utility Unicode statements also need consideration.

We now look at these areas in more detail.

6.5.1 SCCSID greater than 0 required

DB2 Version 8 forces you to specify a SCCSID which is other than 0, because DB2 V8 is evan more dependent on having correct CCSIDs being used than previous versions.

A first step in this direction came with the PTFs for APAR PQ56697, which became available in April 2003 for DB2 V6 and V7. These PTFs add provisions to alert you if you are using an undefined CCSID, or the DB2-supplied DSNHDECP in SDSNLOAD, and that these practices are unsupported in DB2 Versions 6 and 7, and will be disallowed in the future.

After applying this PTF:

- Assembling a DSNHDECP with an invalid (0) CCSID for ASCII or EBCDIC produces warning messages. The assembly ends with CC=4 and a new MNOTE warning. (The undefined single-byte EBCDIC CCSID (SCCSID=0) will be disallowed in the near future. Contact IBM for assistance with choosing a valid single byte EBCDIC CCSID. Do not choose a different CCSID without guidance from IBM. See the explanation text for message DSNT526I.
- A warning message is shown at DB2 startup time when undefined CCSIDs are used in DSNHDECP (DSNT526I).
- A warning message (DSNT527I) is displayed at DB2 startup time if DB2 does not find a DSNHDECP, other than the one that is shipped in SDSNLOAD. The DSNHDECP that is shipped in SDSNLOAD is not intended to be used by customers. That is why the warning message is issued.
- The DB2 precompiler produces warning messages when it finds that it is running with a DSNHDECP that has undefined CCSIDs in it (DSNH526I).
- The DB2 precompiler produces warning messages when it finds that the DSNHDECP that is shipped in SDSNLOAD is used (DSNH527I).

After applying the PTF for APAR PQ71079 (still open at the time of writing of this publication):

- DB2 will no longer allow you to create a DSNHDECP with SCCSID=0 (EBCDIC).

In DB2 for z/OS V8:

- DB2 will not start with SCCSID=0 (EBCDIC) or ASCCSID=0 (ASCII).
- DB2 will not allow you to create a DSNHDECP with ASCCSID=0.
- The precompiler will not run when it finds a DSNHDECP with SCCSID=0, ASCCSID=0, or both.
- DB2 will make sure that you do not change the CCSIDs in the system by accident, as changes in the subsystem’s CCSIDs are not supported. In V8, the CCSID information is also recorded in the BSDS (and can be displayed by the display log map utility), and at startup time DB2 checks to make sure they match the values in the DSNHDECP. If the values do not match, message DSNT108I will be issued, and DB2 startup processing will terminate (DB2 does not start successfully).

Accessing the Unicode Catalog

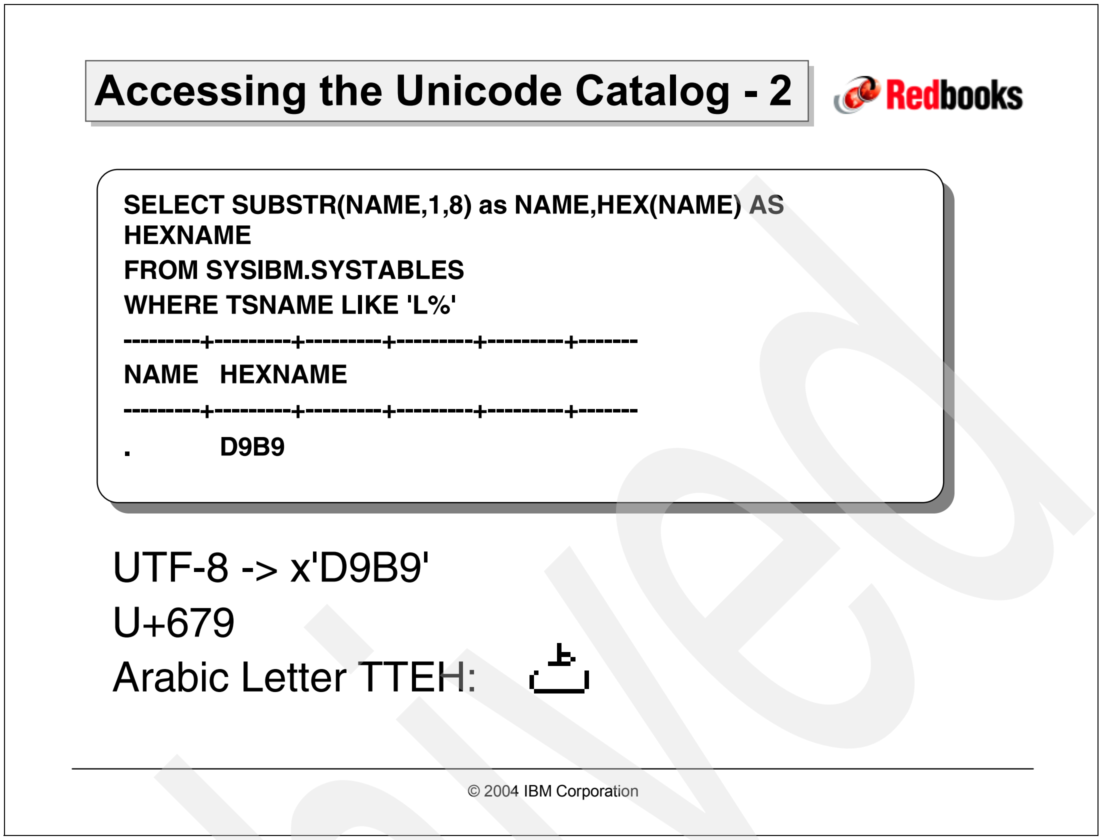
```
SELECT SUBSTR(NAME,1,8) AS NAME,
       HEX(NAME) AS HEXNAME
  FROM SYSIBM.SYSSTOGROUP

---------+---------+---------+---------+-----
NAME       HEXNAME
---------+---------+---------+---------+-----
DSN8G810   44534E3847383130
SABI       53414249
Z!L\       5A214C5C
|          7C
```


6.5.2 Catalog and directory mainly stored in Unicode

To enable Unicode names and literals in DDL, most parts of the DB2 catalog are converted to UTF-8 in DB2 Version 8. The only two exceptions are tables SYSIBM.SYSDUMMY1 in the newly created table spaces DSNDB06.SYSEBCDC and SYSIBM.SYSCOPY in DSNDB06.SYSCOPY. This conversion to Unicode is being done at the same time as the enabling of the long names support during enabling-new-function mode (ENFM) processing.

Although the way the catalog data is stored changes “under the covers”, you can still access the catalog tables using SPUFI, for example. As long as the character that is stored in the catalog tables can be represented in the CCSID used by the application encoding scheme, and can also be represented in the CCSID used by your 3270 emulation, the data is displayed as for V7, that is, in EBCDIC (SPUFI’s encoding scheme) — CCSID 37 in our case.

You can see this in the upper SELECT statement on the visual above. The hexadecimal representation (HEXNAME column) shows that the column data is really stored in Unicode, but the application encoding scheme is used to convert all characters to a valid character in CCSID 37, which is also used by our terminal emulation.



Accessing the Unicode catalog

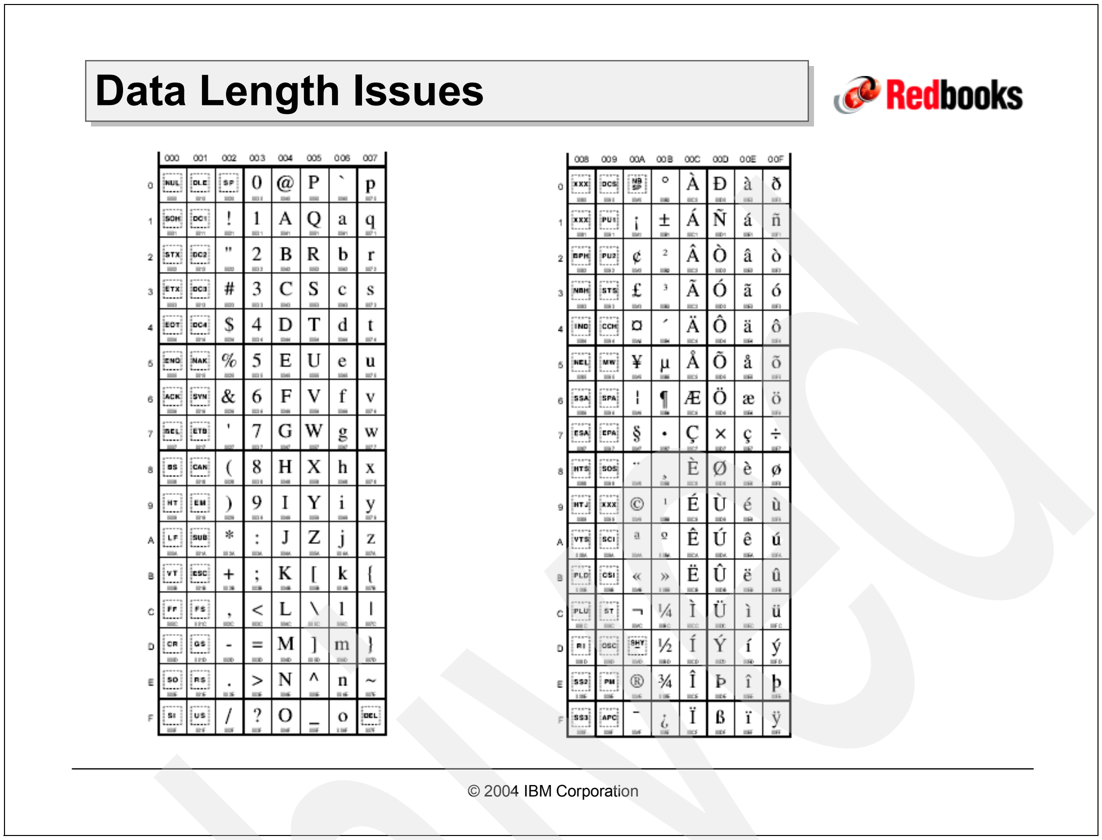
If, however, you start using more "fancy" characters for your DB2 objects, like table names, index names, and so on, you may not be able to represent those characters from all clients.

This is what we did in the example above. We used the DB2 Control Center to create a table with the keyboard set to URDU and entered the Arabic letter TTEH as table name enclosed in apostrophes (delimited table name). The creation of the table was successful, and since we did not use a specific table space name, DB2 chose L. If we now select from SYSTABLES where the table space name equals 'L', the resulting row looks like the one shown on the visual above. Since the arabic letter TTEH cannot be displayed using EBCDIC CCSID 37 (our terminal emulation's CCSID), you just see a '.' instead. From the hexadecimal value (HEXNAME column) you can see that it really is the Arabic letter TTEH which is stored as table name. The hexadecimal value is x'D9B9'. Follow the procedure described in "Decoding UTF-8" on page 510, if you want to check again how to decode the hexadecimal value x'D9B8' into the Unicode code point U+0679, which represents TTEH.

Tip: Although possible, we recommend that you do *not* use characters for your DB2 object names which are not part of a common subset that is representable on all clients.

For some objects, whose names must be passed to z/OS, the characters used must be convertible to EBCDIC. Here are a few examples:

- Database names are part of the data set qualifier.
- Table space names are part of the data set qualifier.
- DBRM names are PDS member names.
- UDF and stored procedures (when no explicit external name is specified), exits, and FIELDPROCs are PDS member names.



6.5.3 Data length issues

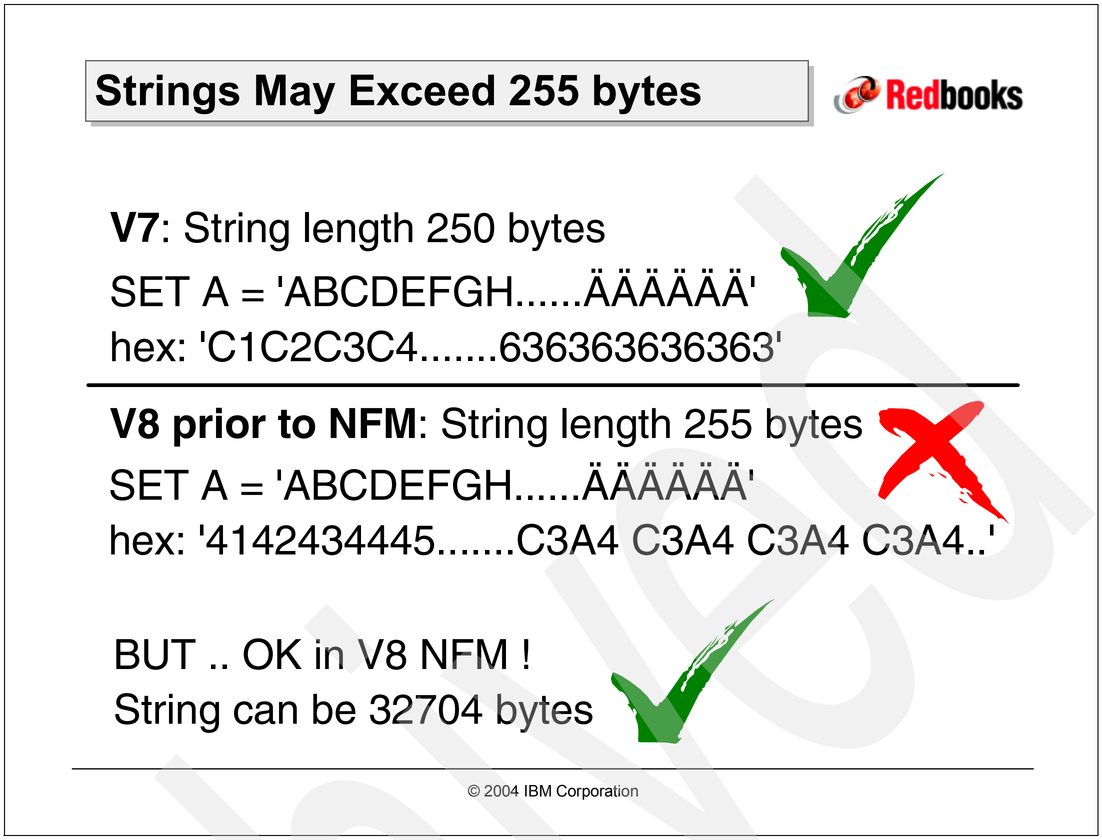
Conversions can cause the length of a string to change. This can be true in both directions!

Let us first have a look at expanding conversions, those in which the conversion from one CCSID to another causes the data to become longer. As stated previously, all characters that can be represented by the first 127 ASCII characters can be stored using just one byte in UTF-8. If you refer to the visual above, you can see that this covers the major part of letters used in many countries today. If, however, you are also using characters that require code points greater than U+7F, the representation of these characters requires at least two bytes.

As an example, you may want to use the character é, which is very common in French. Once you start using this character for your DB2 objects or in your application data with a Unicode encoding scheme, you must take into account that the storage of these characters requires more space than it used to previously in EBCDIC or ASCII.

Attention: Always remember that you must allocate the width of your data columns according to the storage length and not according to their display length. For example the storage of string ‘René’ requires five bytes instead of the four bytes which one could assume just looking at it in UTF-8. Therefore using varying length strings may be appropriate.

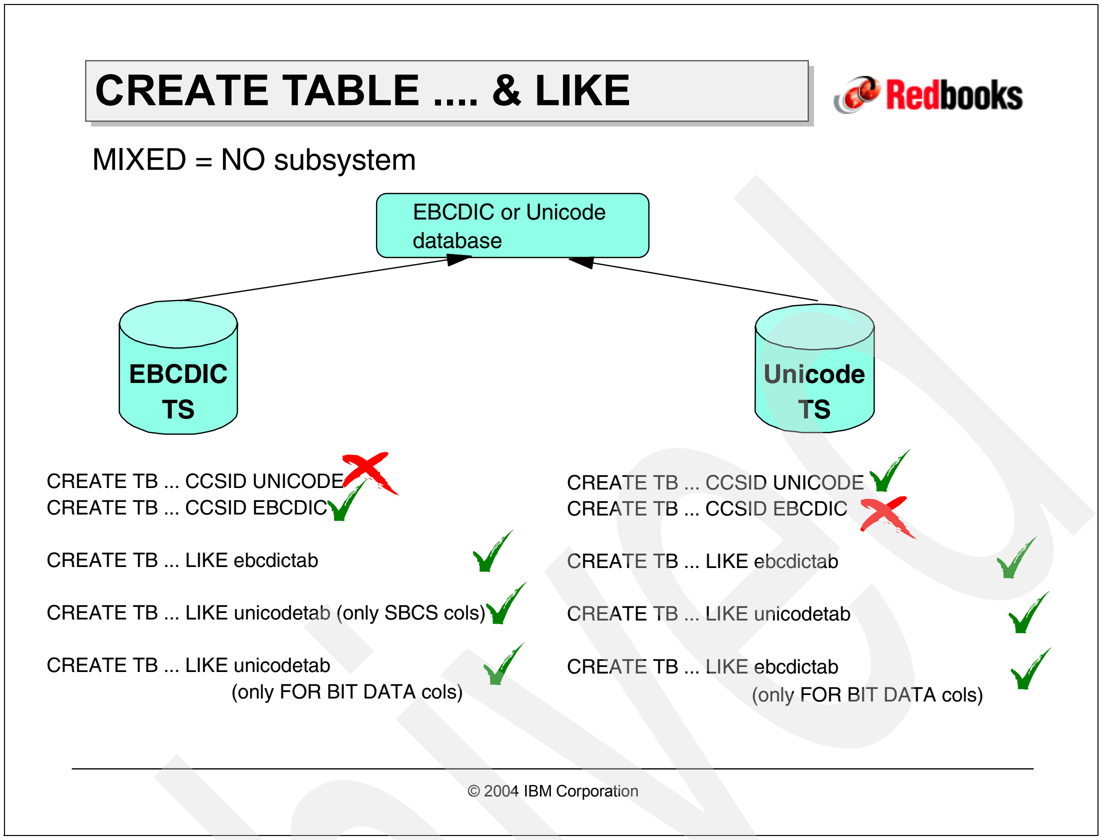
This, however, does not affect your application programs if you do not start using application encoding scheme Unicode. As long as you stick to EBCDIC, for example, DB2 converts the data to the program’s CCSID based on the application encoding scheme and the precompiler CCSID. Refer to 6.5.7, “Application programming issues” on page 535 for more details.



Are you using strings close to the limit of 255 bytes?

If you are using strings in your SQL statements, it is possible that those strings that were valid in V7 are invalid in V8 compatibility mode. This is due to the potential increase in number of bytes when they must be converted to Unicode. These statements might be flagged as too long in Version 8 (exceeding 225 bytes), and you will receive message DSNH102I (during a precompilation) or SQLCODE -102 (otherwise).

This is, however, is only a problem while you are running your DB2 V8 subsystem in compatibility and enabling-new-function mode. Once you enter the new-function mode, the 255 byte limit is raised to 32704 bytes.



6.5.4 CREATE LIKE and other SQL issues

In this section, we look at Unicode and EBCDIC objects. We do not explicitly mention ASCII, because it is similar to what we discuss for EBCDIC.

If you have a database defined with CCSID Unicode, you can create both EBCDIC and Unicode table spaces in this database. This is also true for a database defined with CCSID EBCDIC. The CCSID that you have specified in your CREATE DATABASE statement becomes the default encoding scheme for this database. This means that if you subsequently create table spaces in this database, they inherit the database's default encoding scheme. You can, however, override this default by specifying another CCSID in the CREATE TABLESPACE statement. If you do not specify a CCSID clause in your CREATE DATABASE statement, the option defaults to the value of field DEF ENCODING SCHEME on installation panel DSNTIPF (ENSCHEME DSNHDECP).

Table 6-2 shows information you can see in the DB2 catalog table SYSIBM.SYSDATABASE, depending on which CCSID is associated to the database either explicitly or implicitly through the system default encoding scheme.

Table 6-2 CCSID related values in SYSIBM.SYSDATABASE

	Encoding_ Scheme	SBCS_CCSID	DBCS_CCSID	MIXED_CCSID
EBCDIC	E	37	0	0
Unicode	U	367	1200	1208
DSNDB04	blank	0	0	0

If you create an EBCDIC database and your DB2 subsystem has mixed data disabled (MIXED NO), the system's default encoding scheme is stored in column SBCS_CCSID and no CCSID is stored in DBCS and mixed columns. For Unicode databases, all three CCSID columns are always filled with the exact same values as shown in Table 6-2.

Database DSNDB04 is neither EBCDIC nor UNICODE and no CCSIDs are associated with the default database. We will refer to this special behavior a little bit later in this section.

When you create a table space, unless you explicitly specify the CCSID clause on the CREATE TABLESPACE statement, its CCSID is the same as the CCSID of the database.

As you can see in the figure above, for both Unicode and EBCDIC table spaces, there are some restrictions regarding tables which can be defined in them. First we describe the behavior of EBCDIC table spaces.

If you do not use the CCSID parameter in your CREATE TABLE statement, the table space's encoding scheme is the default. Therefore, it is not very surprising that you can create EBCDIC tables in EBCDIC table spaces. In contrast to that, you cannot create a table using CCSID UNICODE in an EBCDIC table space. All objects in a table space must have the same encoding scheme.

If your DB2 subsystem is defined with MIXED=NO, all characters that are stored in EBCDIC tables take up one byte of storage and they are all encoded in the specific encoding scheme, which is associated with the table space. In addition to that, only the SBCS_CCSID can be used to code your data.

CREATE TABLE ... LIKE

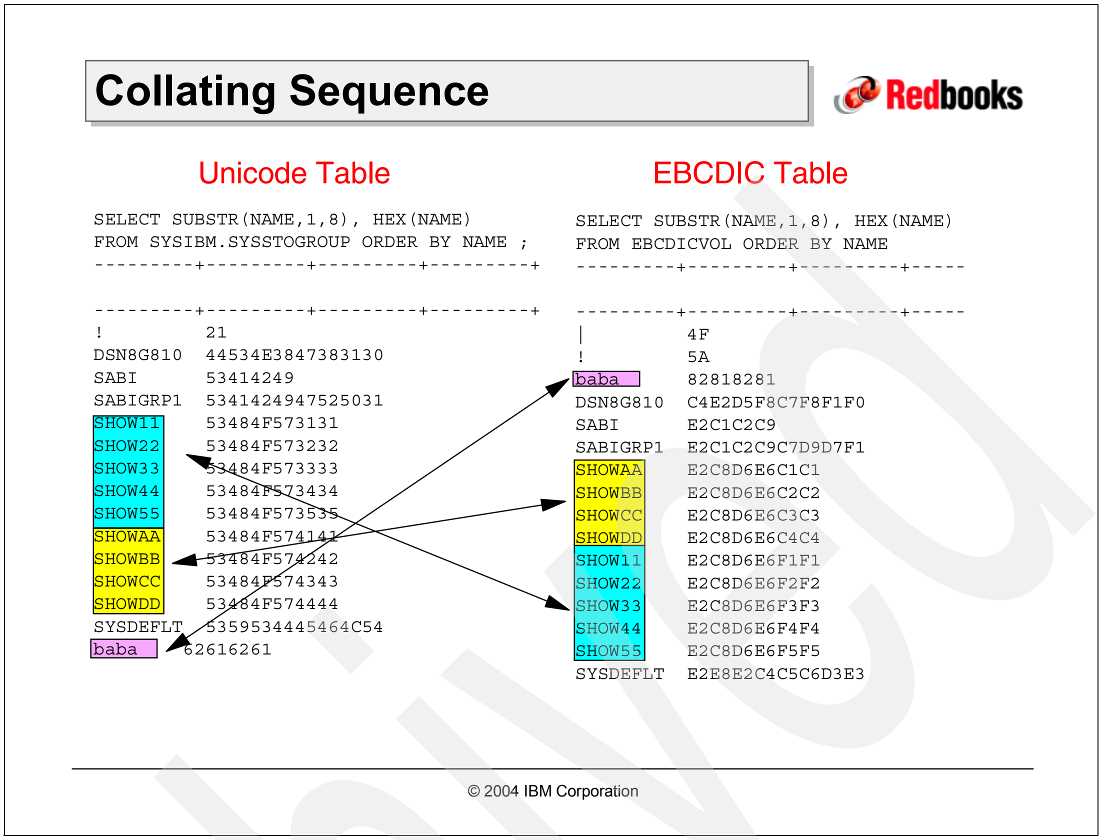
Let us now have a look at the CREATE TABLE ... LIKE .. statement. If you use this statement to create a table in your EBCDIC table space, which is LIKE another EBCDIC table, the statement works fine.

If the table specified in the LIKE clause is a Unicode table, where columns are FOR MIXED DATA (the default for Unicode tables), the CREATE TABLE statement currently fails. This happens because on a MIXED=NO system, we cannot have "mixed string" EBCDIC data. (This behavior may change in the future.)

As shown in Table 6-2, an EBCDIC database (or table space) created in a MIXED=NO subsystem does not have mixed_CCSIDs associated and therefore cannot be used.

> **Tip:** CREATE TABLE TEST1 LIKE SYSIBM.SYSDATABASE works fine. The implicitly created table space is now being created in the default database DSNDB04. As shown in Table 6-2, DSNDB04 does not have any encoding scheme associated with it. In this case, the implicitly created table space can inherit the LIKE table's encoding scheme, which is Unicode in case of SYSIBM.SYSDATABASE. Since mixed is the default for Unicode table spaces, the definition of the table and the storage of the data would match.

Table SYSIBM.SYSDUMMY1 used to be part of table space DSNDB06.SYSSTR. The SYSSTR table space is encoded in Unicode starting with DB2 V8. In order to provide a consistent behavior when working with this "dummy "table, it has been moved to a separate EBCDIC table space called DSNDB06.SYSEBCDC.



6.5.5 Collating sequence

As you can see from the visual above, whenever you access Unicode data, the collating sequence as result of an ORDER BY statement is different than the one in EBCDIC. The Unicode collating sequence is equivalent to the ASCII behavior, as the first 127 code points are identical to Unicode.

This means that whenever you run a query on your V8 NFM catalog, the results may return in a different sequence. However, this is only an issue if the columns being ordered contain a mix of numeric, upper case and lower case characters and special characters.

See Table 6-3 for an overview of how the characters are ordered in Unicode versus EBCDIC.

Table 6-3 Collating Sequence in Unicode vs. EBCDIC

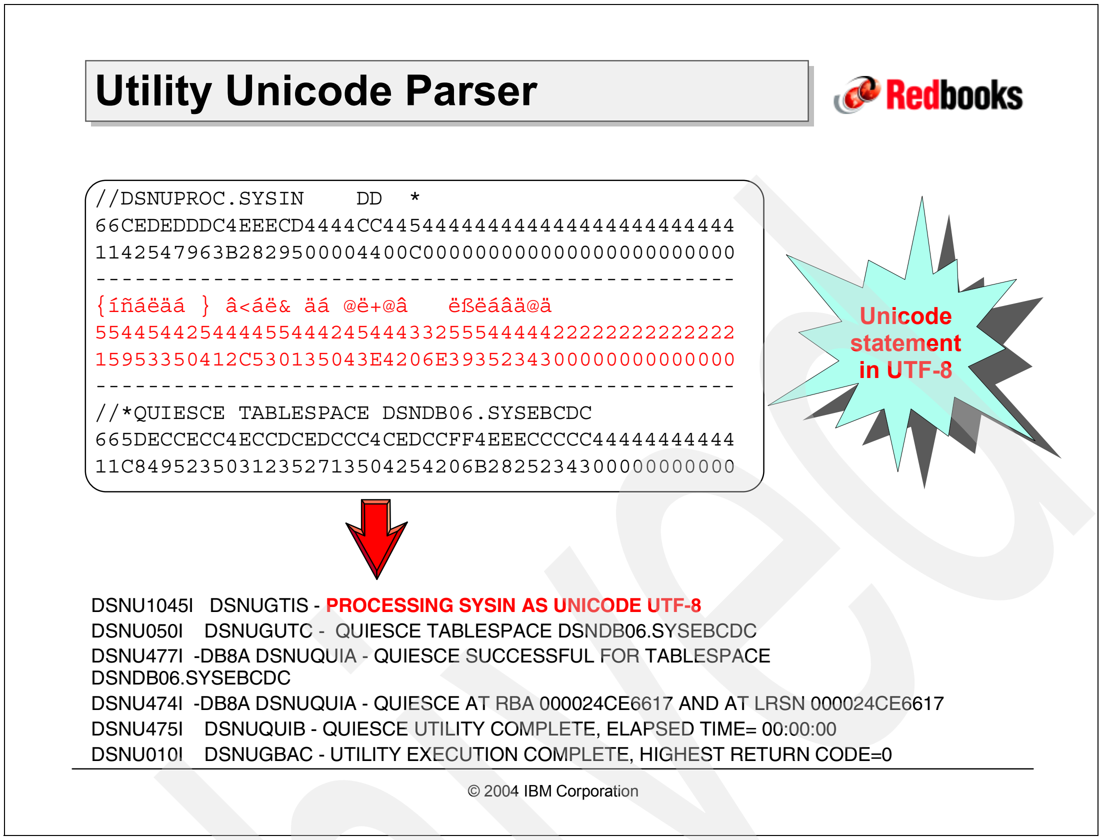
EBCDIC	Hex value	UNICODE/ASCII	Hex value
space	x'40'	space	x'20'
lower case	x'81 - 89' x'91 - 99' x'A1 - A9'	numerals	x'30 - 39'
upper case	x'C1 - C9' x'D1 - D9' x'E1 - E9'	upper case	x'40 - 4F' x'50 - 5A'
numerals	x'F0 - F9'	lower case	x'61 - 6F' x'70 - 7A'

Range predicates and MIN/MAX functions may also be impacted where they span sets of characters. Refer to “ORDER BY sort order” on page 564 for a more detailed description.

Tip: A possible workaround for the ORDER BY issue is the usage of the CAST function to convert the data to EBCDIC format, as this will result in the original sort order. The following statement returns the data in EBCDIC sequence:

```
SELECT CAST(NAME AS CHAR(20) CCSID EBCDIC) AS E_NAME
  FROM SYSIBM.SYSTABLES
  WHERE NAME  LIKE 'T%'
 ORDER BY E_NAME
```

Note, however, that DB2 cannot use an index to avoid the ORDER BY sort in this case. The use of this construct may impact performance, especially on large result sets.



6.5.6 Utility Unicode statements

As you can see in the figure above, starting with V8, you can specify your utility control statement in EBCDIC or UTF-8. By utility control statement, we mean those input data sets provided to the DSNUTILB program with DD names SYSIN, SYSLISTD or SYSTEMPL or the contents of the UTSTMT field passed to the DSNUTILU stored procedure.

Refer to the visual above. The output on the SYSPRINT data set will continue to be in EBCDIC. This is also true for messages displayed on the console.

For a more detailed description of the new utility stored procedure interface DSNUTILU for Unicode, also refer to 9.14.4, “Utility Unicode statements” on page 693 and subsequent pages.

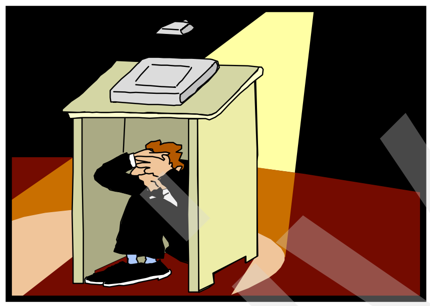
Application Programming Issues

Unicode SQL parser

Program Preparation

Functions, Routines, Stored Procedures

DBRMs & SYSIBM.SYSSTMT & SYSIBM.SYSPACKSTMT

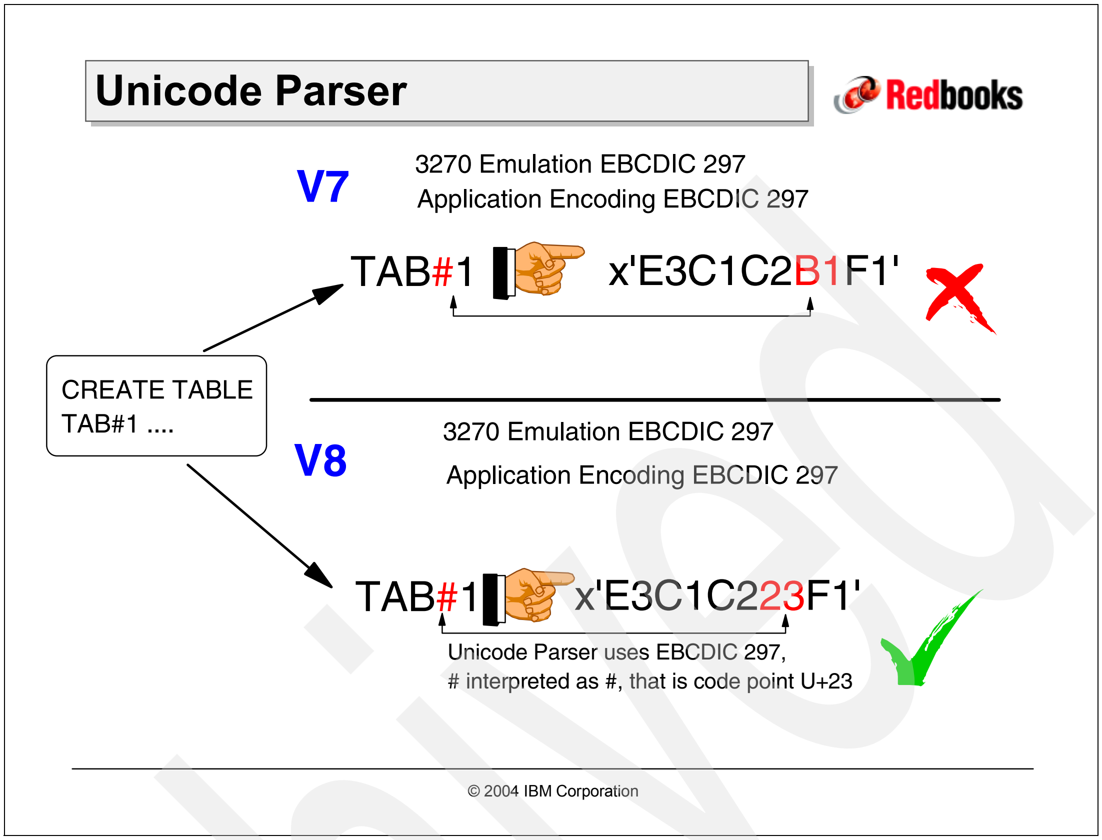


Be prepared!

6.5.7 Application programming issues

Starting with DB2 V8, Unicode is everywhere. This also requires some changes to the way programs can access DB2 data. These changes are described in detail on the next few pages. They include:

- An explanation of why the SQL parser needs to be able to read Unicode data
- A list of changes to the program preparation steps
- A discussion about the concept of application encoding schemes
- A description of how Unicode changes the behavior of functions, routines, and stored procedures
- A reminder regarding the way you must start working with your DBRMs and catalog tables SYSIBM.SYSSTMT, and SYSIBM.SYSPACKSTMT



Unicode parser

The use of an EBCDIC parser in a DB2 subsystem that is used for global e-commerce creates a few interesting problems:

- One problem is that you cannot include string constants from multiple character sets in a single source SQL statement. To solve this problem, you must use one of the following techniques:
  – Host variables (or parameter markers for dynamic SQL) and DECLARE VARIABLE (or a descriptor) to specify the different CCSIDs that you need to use
  – Hexadecimal string constants

  Both techniques are obviously not very convenient.
- A second problem is that various EBCDIC code pages are inconsistent regarding code points of various special characters. Those characters include “$@#|¬, as well as Katakana lower case characters. This makes it very difficult for an EBCDIC parser to correctly process SQL statements that use these characters. Even though the DB2 V7 parser tries to recognize these characters in different locations in different code pages, it does not always work out.

  For example, some EBCDIC CCSIDs represent the ‘#’ character as different hex code points. This can cause parsing errors. The figure above shows that in case you are using the French EBCDIC CCSID 297, you cannot use character ‘#’ for the creation of a table in V7, and you get an SQLCODE -104.

  To avoid some of these problems, you can sometimes use a different way to code your SQL statements. For example:

```
COLA ¬= 5    can be coded as    COLA <> 5
```

This avoids the use of the ¬ (NOT) sign that tends to jump around in different EBCDIC code pages. A similar problem exists with the | (CONCAT) operator. For example:

```
COLA || COLB¬= 5    can be coded as    COLA CONCAT COLB
```

DB2 V8 introduces the Unicode SQL parser. This adds some of the key functionality which leads DB2 from a basic Unicode implementation (data only), to enhanced Unicode exploitation. Unicode parsing in V8 transforms the traditional EBCDIC parser into a parser which accepts the syntax regardless of the EBCDIC CCSID. The Unicode parser converts all SQL statements that are not currently encoded as Unicode UTF-8, to that format before parsing.

This technique solves both problems discussed before.

Because you can code your SQL statement in Unicode, and the statement can also be processed (parsed and interpreted) in Unicode (instead of having to convert to EBCDIC, before DB2 could work on it), you can code the literals in Unicode as well. This way, you can code any existing character as part of a literal in your Unicode SQL statement.

The second problem we discussed is illustrated in the figure above. In this example we assume that the user's 3270 emulation is set to the French EBCDIC code page 297, and we want to create a table named 'TAB#1'. This is a valid name.

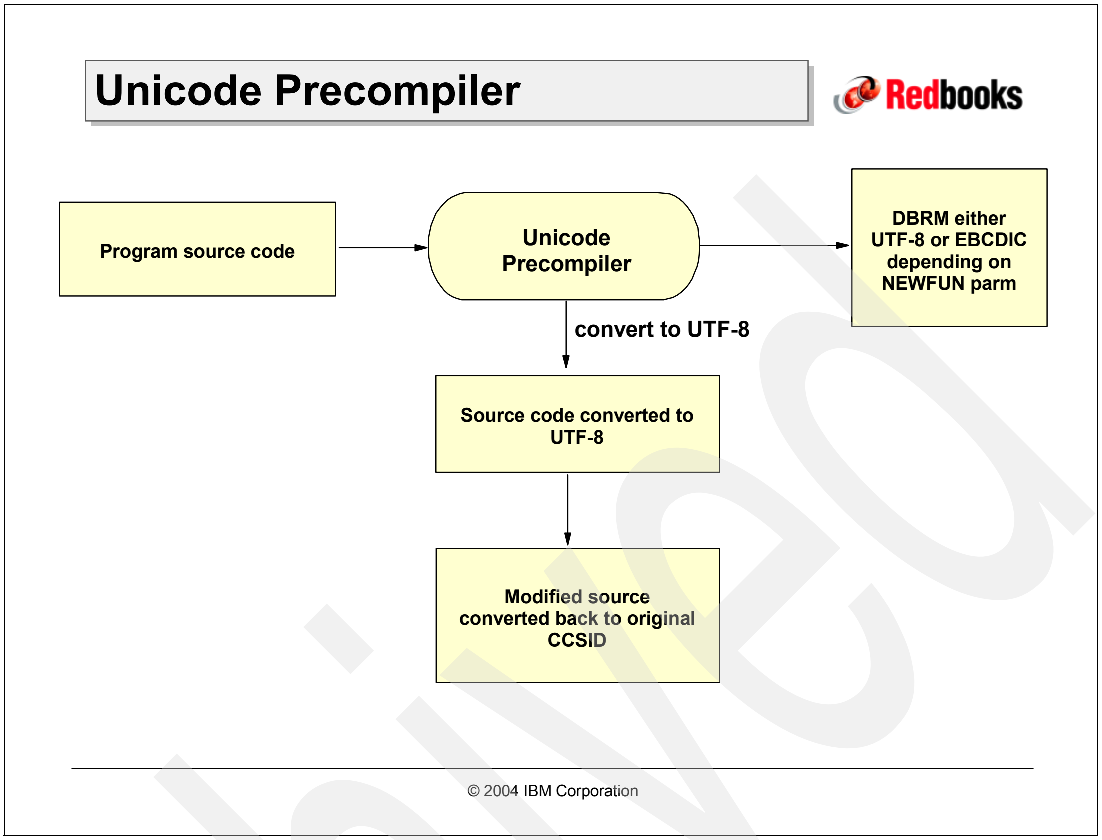
In V7, the creation fails, because DB2 is not always able to interpret the '#' (x'B1' in CCSID 297) as a '#' because of its different locations in certain EBCDIC code pages. The CREATE TABLE statement fails with the following message:

```
CREATE TABLE TAB#1 (COLA CHAR(5))
DSNT408I SQLCODE = -104, ERROR:  ILLEGAL SYMBOL " ". SOME SYMBOLS THAT MIGHT BE LEGAL
ARE: ( LIKE
```

In V8, when using application encoding scheme (297), which is the same as the terminal emulator's CCSID (as it should be), the statement works fine. When you type a '#' in your 3270 emulator with EBCDIC 297, it is coded as x'B1'. The value x'B1' is now converted to Unicode using the application encoding scheme, which is set to EBCDIC 297 in our example. x'B1' is converted to Unicode code point U+23, which is represented as x'23' in UTF-8. This is also illustrated in Example 6-1.

Example 6-1   Creating TAB#1

```
CREATE TABLE TAB#1 (COLA CHAR(5));
RESULT OF SQL STATEMENT:
DSNT400I SQLCODE = 000,  SUCCESSFUL EXECUTION

SELECT SUBSTR(NAME,1,20) AS NAME,
          HEX(NAME) AS HEX_NAME
     FROM SYSIBM.SYSTABLES
     WHERE DATE(CREATEDTS) = CURRENT DATE;

NAME                 HEX_NAME
-----------------------------------------
TAB#1                5441422331
```

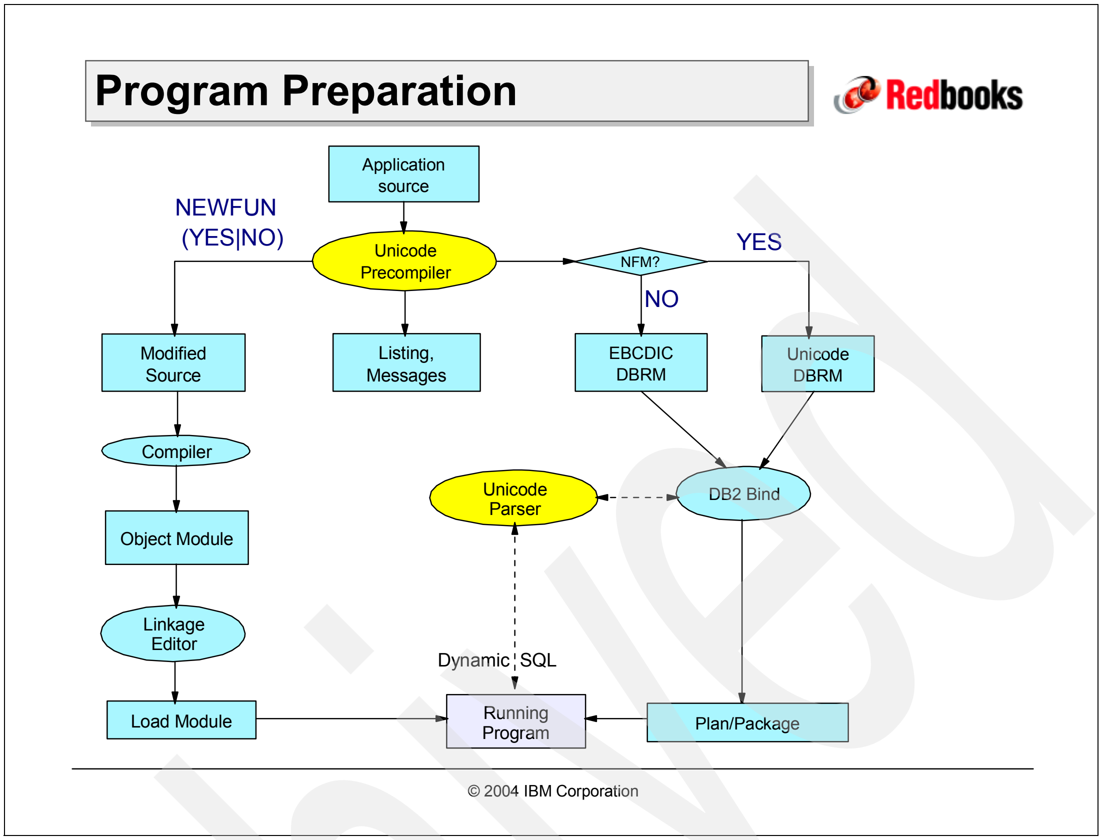


Unicode precompiler

When migrating your DB2 UDB for OS/390 and z/OS from V7 to V8, you must go through three different modes. Irrespective of the mode in which you are currently running your DB2 subsystem, DB2 V8 always uses a Unicode precompiler (or precompiler services).

As you can see from the visual above, the Unicode precompiler converts the program source code to Unicode UTF-8, performs the precompilation, and then converts all statements, including the generated and modified statements, back to the system CCSID as specified on the panel DSNTIPF during installation (unless you specify the CCSID precompiler option, in which case the source is interpreted as being in this CCSID, and is converted back to that CCSID after the precompilation is done).

Apart from the modified source, the Unicode precompiler also generates the corresponding DBRM. If you do not specify any additional precompiler options, the DBRM is generated in EBCDIC as long as your subsystem is not running in new-function mode (as explained in more detail in the next section).



NEWFUN precompiler/coprocessor option

Only NFM allows the use of SQL functions that are new in DB2 Version 8. Since the precompiler executes outside DB2, it cannot ascertain the current mode of DB2. In addition to that, you may want to be able to run a partial test (precompilation and compilation without binding or executing) of an application that uses new syntax. Therefore, a new precompiler option (*NEWFUN*) has been added which tells the precompiler whether or not to allow new syntax, as well as to tell whether or not to produce a DBRM in EBCDIC or Unicode.

If you specify a value of *NEWFUN(NO)*, the precompiler rejects any source SQL statements that contain new V8 syntax. A successful precompilation produces an EBCDIC DBRM, which is compatible with DB2 V7 and earlier releases. The DBRM can be bound on DB2 V7 or V8.

If you specify a value of *NEWFUN(YES)*, the precompiler accepts source SQL statements that contain new V8 SQL syntax. A successful precompilation produces a DBRM that is marked as V8-dependent and is therefore not compatible with V7. This happens regardless of whether the program contains new V8 syntax or not. As a consequence, it cannot be bound on a V7 nor on a V8 subsystem that is not yet running in NFM. The DBRM that is produced as a result of the precompilation is in Unicode.

The default value for the NEWFUN precompiler parameter is set to *NO* during compatibility and enabling-new-function modes. For a new V8 subsystem or for a subsystem which has successfully been converted to new-function-mode, the default changes to *YES*. The advantage of changing the default is that there is no need to change all the precompile jobs once you get to NFM.

This behavior is also illustrated on the visual above. As you can see, regardless of whether NEWFUN is set to YES or NO, DB2 invokes the V8 SQL parser. The V8 SQL parser uses Unicode UTF-8 for parsing. If the source program's SQL statements are not in UTF-8, the precompiler converts them to UTF-8 for parsing (as explained in the previous section).

You can also refer to Table 6-4. This is a slightly different way to explain the dependencies between the different modes and the value used for the NEWFUN option.

Table 6-4 Dependencies and binding for DB2 V8

Value of NEWFUN keyword	**NO**	**YES**	**YES**
Are new V8 functions used?	**No**	**No**	**Yes**
Is DBRM V8 new object and V8-dependent?	No	Yes	Yes
Can V7 or earlier bind?	Yes	No	No
Can V8 bind before new-function mode	Yes	Yes	No
Can V8 bind in new-function mode	Yes	Yes	Yes

For more information, see 12.8.22, “Application programming” on page 970.

Functions and Routines

Functions

- LENGTH, SUBSTR, POSSTR, LOCATE
  - For SBCS and MIXED (UTF-8) they are byte-oriented
  - For DBCS (UTF-16) they are double-byte-oriented
- New character based functions (and enhanced existing functions)
  - New CHARACTER_LENGTH, SUBSTRING, POSITION
  - New parm indicating how to count (CODEUNITS32, CODEUNITS16, OCTETS)
- Cast functions
  - Unicode generally accepted where CHAR is accepted
  - For CHAR functions, UTF-8 is result data type

Routines

- UDFs, UDTs, and stored procedures allow Unicode parameters
- Parameters converted as necessary between CHAR(UTF-8) and GRAPHIC(UTF-16)
- Date, time, timestamp passed as UTF-8 (ISO format)

Functions and routines

Functions and routines have been enhanced to be able to deal with Unicode data.

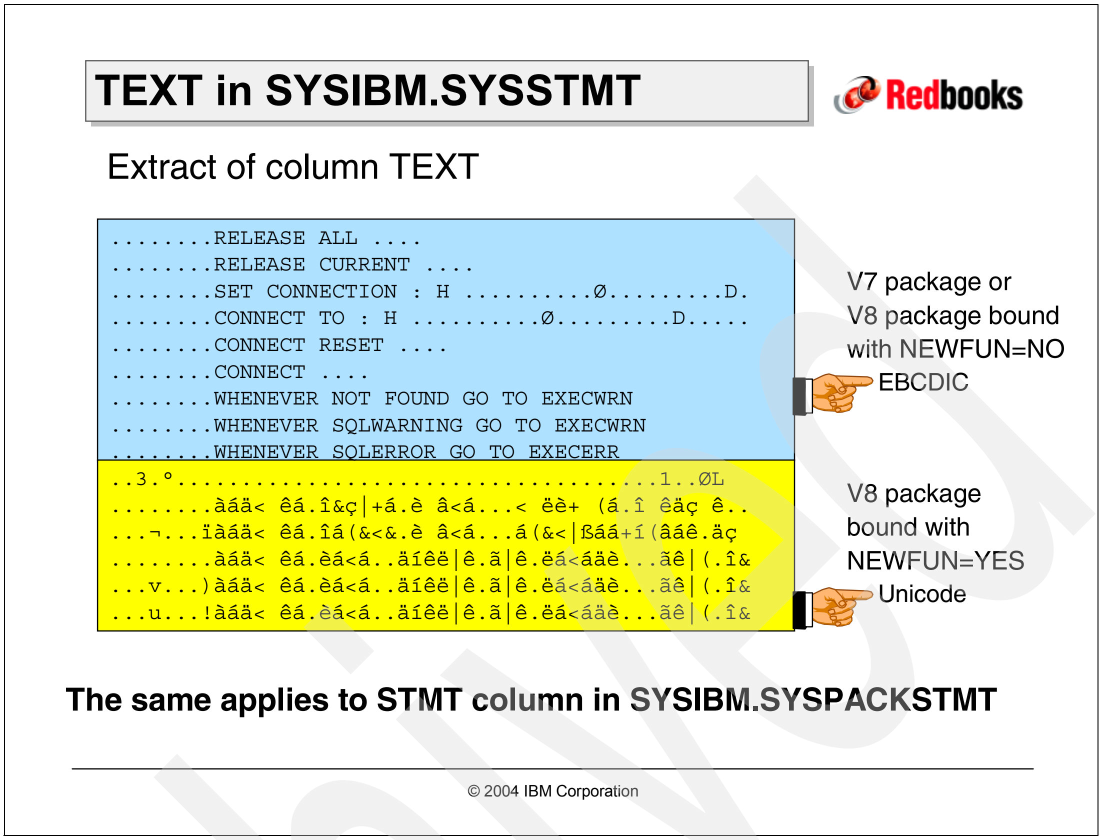
If you use the LENGTH function, DB2 returns the actual storage length; in other words, it counts the number of bytes.

To handle the problem of varying length UTF-8 characters, DB2 has enhanced a number of existing functions and introduces three new functions (CHARACTER_LENGTH, POSITION and SUBSTRING). You can now indicate to these functions how you want DB2 to count, byte-wise or character-wise. You can specify the following new keywords:

- CODEUNITS32
- CODEUNITS16
- OCTETS

For more information on these new functions an how to the new options, see 4.22.4, “Character-based string functions” on page 377.

Routines, UDFs, and stored procedures allow for the specification of Unicode parameters. Parameters are converted when required between CHAR (UTF-8) and GRAPHIC (UTF-16). The character representation of date, time, and timestamps are passed as UTF-8.



6.5.8 SYSIBM.SYSSTMT and SYSIBM.SYSPACKSTMT

Once you are in NFM, and the parameter NEWFUN is set to YES, the text of your DBRMs, which is stored in column TEXT of catalog table SYSIBM.SYSSTMT, or the STMT column of SYSIBM.SYSPACKSTMT, is encoded in Unicode. These columns are marked with the FOR BIT DATA attribute. This means that if you look at the column, using SPUFI for example, you are no longer able to easily read the contents, because columns with the FOR BIT DATA attribute are never converted, and these columns are now in Unicode.

If you see readable information in this column as shown on the visual above in the blue (upper) frame, this package has either never been bound since DB2 has been migrated to V8, or the NEWFUN option was set to NO. In this case, the information in column TEXT is still stored in EBCDIC.

You can also use the V8 Visual Explain (VE) tool to look at those statements. VE takes care of the conversion for you, or you can use a tool like DB2 Administration Tool.

A “quick and dirty” approach is to download the SPUFI output to your PC in binary and look at it using the NOTEPAD editing tool. Because the first 127 characters in Unicode are the same as their ASCII counterparts, these characters can easily be displayed on an ASCII-based workstation.

If you know that your SQL statements for a certain package or DBRM are in Unicode, you can use the CAST function to convert the TEXT column to EBCDIC, as shown in Example 6-2.

Example 6-2 Retrieving statement text from SYSSTMT

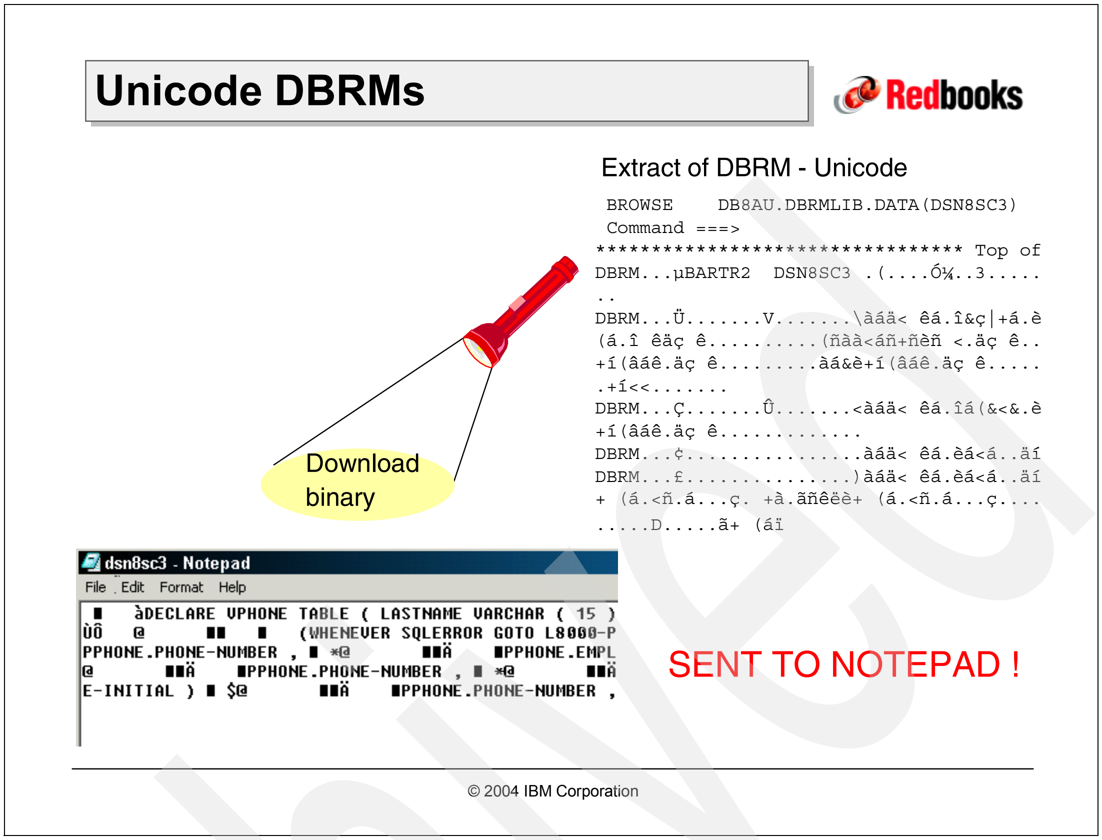
```
SELECT NAME, STMTNO, STMTNOI,
       CAST(CAST(TEXT AS VARCHAR(3500) CCSID 1208) AS VARCHAR(3500) CCSID EBCDIC)
  FROM SYSIBM.SYSSTMT
  WHERE NAME = 'unicode-dbrm'
        AND NOT (STMTNO=0 AND SEQNO=0 AND SECTNO=0)
  ORDER BY PLNAME, STMTNO, STMTNOI;
```

A similar statement (see Example 6-3) can be used to obtain the SQL statement text from SYSPACKSTMT.

Example 6-3 Retrieving statement text from SYSPACKSTMT

```
SELECT NAME, STMTNO, STMTNOI,
       CAST(CAST(STMT AS VARCHAR(3500) CCSID 1208) AS VARCHAR(3500) CCSID EBCDIC)
  FROM SYSIBM.SYSPACKSTMT
  WHERE NAME = 'unicode-package'
        AND NOT (STMTNO=0 AND SEQNO=0 AND SECTNO=0)
  ORDER BY STMTNO, STMTNOI;
```

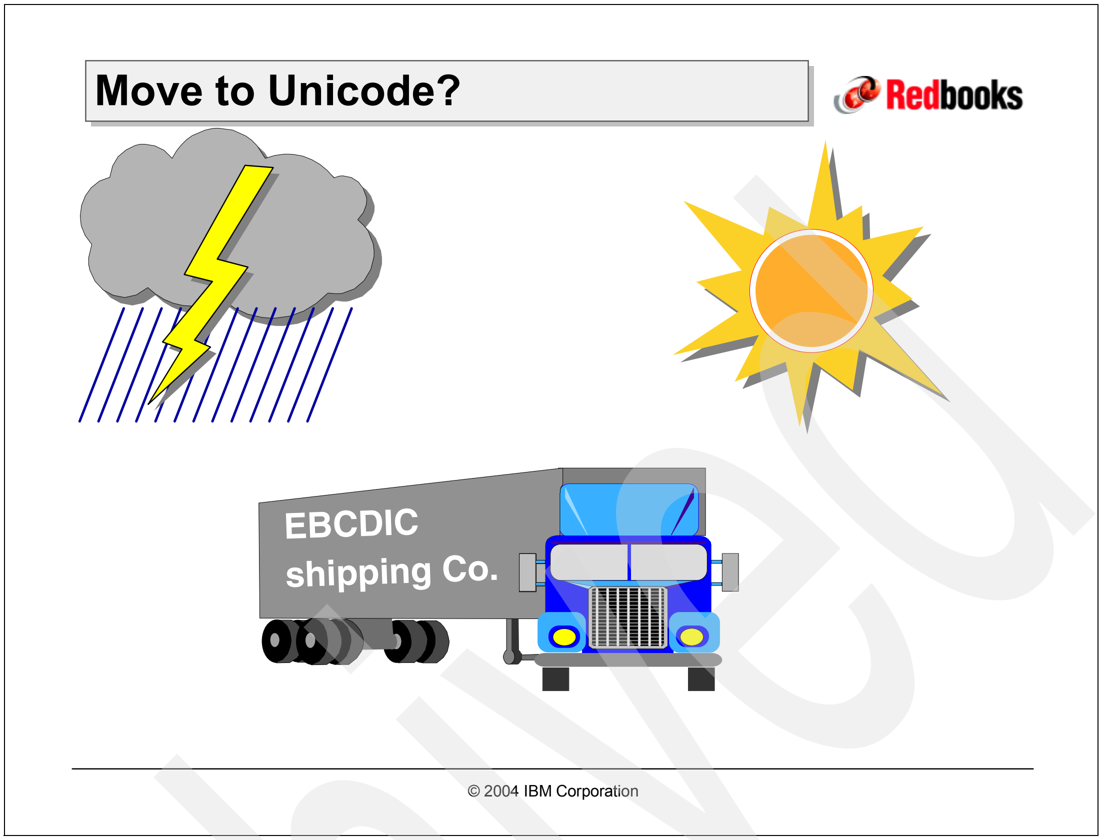
Please note that due to the nature of the data (some of the data is not textual data) in the TEXT column, warnings and errors are possible.



DBRMs in Unicode - How to read them?

The same technique can be used for your DBRMs. Once you start precompiling and your programs with the parameter NEWFUN set to YES, the DBRMs are stored in Unicode and you cannot read them easily any more. The workaround described in the previous section, that is, downloading and looking at it using NOTEPAD, also applies here.

Tip: the PTF for APAR OA07685 (still open at the time of writing of this publication) will impement functionality that will allow you to browse Unicode data under ISPF, making the use of the Notepad technique no longer required.



6.6 Move to Unicode data?

The decision whether to store data as Unicode in the database should be considered carefully on a case-by-case basis. Obviously the main benefit to be realized is the ability to store multilingual data in a single DB2 system, database, or even table. If you are using Unicode-based technologies such as Java and XML, you may also wish to store data as Unicode to avoid conversion issues.

If you plan to start storing your data as Unicode, you must remember that storage size does not equal the displayed size. This means that you may have to increase column length to cater to maximum storage lengths. (This is also true in mixed EBCDIC systems because of the use of shift-in and shift-out characters.)

In addition to potential column length increase, the data sets may also grow in size, which requires additional DASD. The amount of increase that you can expect depends upon the nature of the data as discussed previously.

> **Note:** You do not have to move your data to Unicode if you do not have any special needs for it. For the time being, you can easily continue to work with EBCDIC.
> However, with the V8 enhancements that allow you to have multiple CCSID sets in a single SQL statement (discussed next), you can now gradually migrate your EBCDIC data to Unicode data. In DB2 V7, you were almost always forced to move all the data that was touched by an application to Unicode in one conversion operation. If you shared tables between applications, which is very often the case, you may have to convert multiple applications at the same time, or potentially the entire DB2 subsystem. In V8 you can take it one step at a time.
>
> **Important:** However, bear in mind that there are cases where multiple CCSID SQL statements do not perform as well as single CCSID statements.

Multiple CCSID Sets per SQL

A statement is a multiple CCSID set SQL statement if it references one or more of the following:

- Table objects with different CCSID sets
- Any of the following new functions:
  - GX
  - UX
  - XML2CLOB
  - CAST with CCSID clause
- The SQL statement is one of the following:
  - SET assignment
  - VALUES INTO
  - SET special register
  - VALUE
  - CALL
- A table UDF

6.7 Multiple CCSID sets per SQL statement

Even though you can store data in ASCII, since DB2 for MVS V5, and in Unicode, since DB2 for z/OS and OS/390 V7, DB2 V7 does not allow you to reference multiple table objects defined with different encoding schemes in the same SQL statement.

With DB2 V8, this restriction is removed. Once your DB2 subsystem is running in enabling-new-function mode (ENFM), it allows you to access multiple CCSID sets per SQL statement. (This functionality is not available in compatibility mode.) With character columns in the DB2 V8 catalog being in Unicode, it became an absolute necessity to implement this feature. Otherwise, your applications, or vendor products, that join DB2 catalog tables with your own EBCDIC tables, or vendor EBCDIC tables, would run into errors.

Most of the other new features of DB2 V8 are only available once you successfully migrate to new-function mode. This feature is an exception. It is also available in enabling-new-function mode, because already during enabling-new-function mode, the catalog tables are migrated step by step to Unicode.

The visual above lists the cases that make an SQL statement become a multiple CCSID set statement.

Note: When an SQL statement has a single CCSID set, the “old” rules apply; for example, all constants and special registers referenced in the statement use the CCSID set of the statement, where the CCSID set of the statement is the CCSID of the table (or view) you are running your query against. If no table (or view) is present in the statement, the default EBCDIC encoding scheme is used for the string constant. The CCSID is the appropriate character string CCSID for the encoding scheme.

If a statement is considered to be a multiple CCSID set SQL statement, the CCSID set associated with a string constant or special register always comes from the application encoding scheme.

Graphic string constants in V8

DB2 for z/OS Version 8 supports two (new) types of hexadecimal graphic string constants. When used, they turn the statement into a multiple CCSID set SQL statement, as shown in the figure:

- **UX’xxxx’** represents a string of graphic Unicode UTF-16 characters, where x is a hexadecimal digit. The number of digits must be a multiple of 4 and must not exceed 32704. Each group of 4 digits represents a single UTF-16 graphic character. For example, the UX constant for ’ABC’ is UX’004100420043’. When a UX constant is referenced in a statement containing a single CCSID set, it is converted to the DBCS CCSID of the statement’s CCSID set.
- **GX’xxxx’** represents a string of graphic characters, where x is a hexadecimal digit. The number of digits must be a multiple of 4. Each group of 4 digits represents a single double-byte graphic character. The hexadecimal shift-in and shift-out (’OE’X and ’OF’X), which apply to EBCDIC only, are not included in the string. The CCSID assigned to a GX constant is the DBCS CCSID associated with the application encoding scheme. For dynamic statements, this is the CURRENT APPLICATION ENCODING SCHEME special register. For static statements, this is the ENCODING bind option.

  If the MIXED DATA install option is set to NO, a GX constant cannot be used. Instead, a UX constant should be used. A GX constant cannot be used when the encoding scheme is Unicode.

Statements using the CAST function with the CCSID clause

In V8, with the ability to combine objects with multiple CCSIDs in a single SQL statement, it is often important that you can CAST between different encoding schemes and/or CCSIDs. To allow you to do so, the CAST function has been enhanced to allow you to specify an encoding scheme or specific CCSID.

A simplified syntax diagram is shown in Figure 6-1. We only show the character-related data types in the data type specification, as the CCSID clause can only be specified on those. Numeric data types or the BLOB data type can of course be specified in the CAST function, but cannot use the CCSID clause.

When you use the CCSID clause in a CAST function, this makes the statement a multiple CCSID set statement, and the multiple CCSID string comparison rules apply. These are explained in more detail in 6.7.4, “String comparisons with multiple CCSID sets” on page 556.

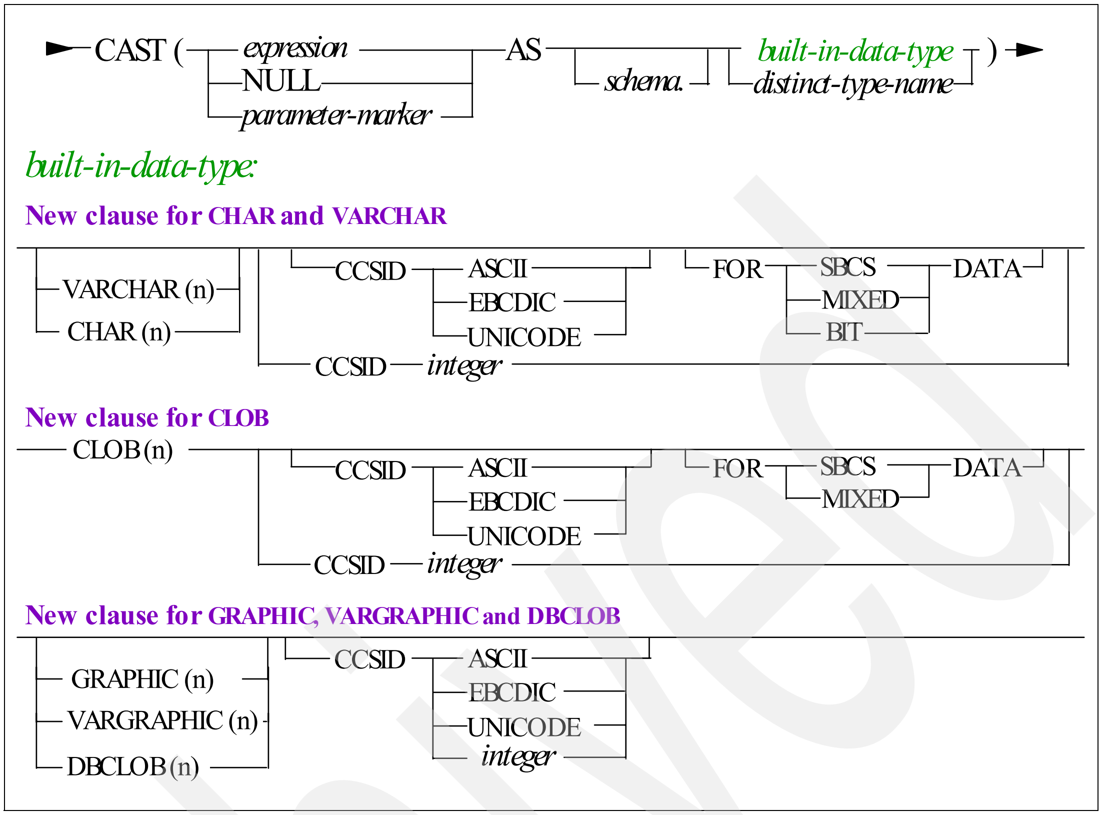


Figure 6-1 CAST function with CCSID clause

To illustrate the use of the CCSID clause in the CAST function, let us look at the following example. Assume we have defined the following tables defined in the catalog: TA, TB, T1, T2. Example 6-4 shows the retrieval of these table names, and order the result by the table name.

Example 6-4 SELECT from catalog with ORDER BY

```
SELECT NAME
  FROM SYSIBM.SYSTABLES
  WHERE NAME LIKE 'T%'
 ORDER BY NAME
```

When you are in V8 compatibility mode (or in V7), the catalog is in EBCDIC, and the result of the query is:

```
TA
TB
T1
T2
```

Once the SYSTABLES catalog table is converted to Unicode during enabling-new-function mode processing, the result looks like this:

```
T1
T2
TA
TB
```

To return the result of this query in EBCDIC collating sequence, which most people are used to after all these years, you can rewrite the query as shown in Example 6-5.

Example 6-5 SELECT using CAST with CCSID

```
SELECT CAST (NAME AS VARCHAR(128) CCSID EBCDIC) AS E_NAME
  FROM SYSIBM.SYSTABLES
  WHERE NAME LIKE 'T%'
 ORDER BY E_NAME
```

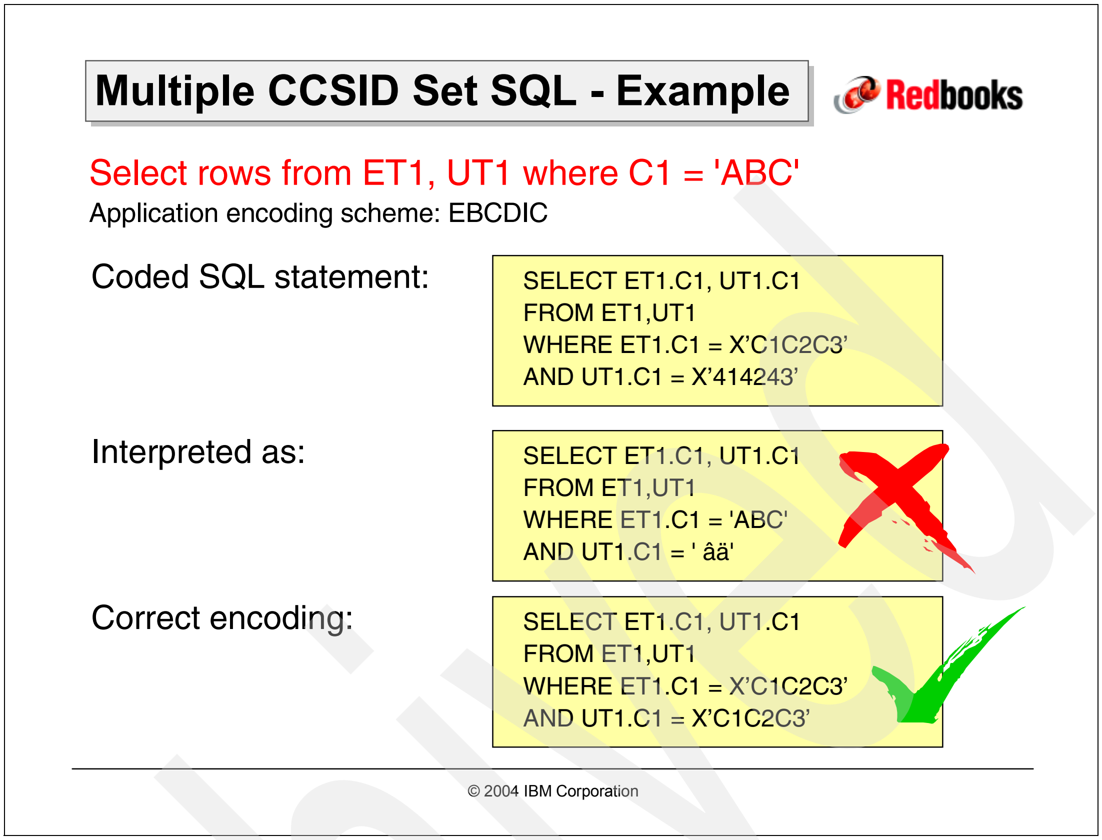
The CAST function will convert the NAME column that is in Unicode (because SYSTABLES is in Unicode) to EBCDIC, and use the converted column (now in EBCDIC) to sort on. Since the ORDER BY is executed after processing the SELECT list, the result of the ORDER BY on the E_NAME column, that is now in EBCDIC, is that rows are returned in EBCDIC order.

Note, however, that DB2 cannot use an index on NAME, in case you created a user-defined index on that column of SYSTABLES; there is no system-defined index on NAME. The index would be in Unicode order, and we have transformed the column to EBCDIC before sorting, so a Unicode index cannot be used.

Table UDF

The decision for a table UDF which DB2 uses follows the method of finding the best fit as described in the *DB2 SQL Reference,* SC18-7426. In order to be able to find a UDF, DB2 must resolve the parameters. Once DB2 has decided which UDF to use, it is processed using the UDFs application encoding scheme.

Depending on which encoding scheme is used to find the best fitting UDF, the length of the character strings may vary and therefore the best fit could be a different one depending on which encoding scheme is used. Using the default application encoding scheme to evaluate the character strings gives you more flexibility than V7, where DB2 always used EBCDIC to resolve the character strings for a UDF.



6.7.1 Multiple CCSID sets per SQL statement

When a statement references table objects with multiple CCSID sets, there is a need to determine which CCSID set to use in the various semantic rules. This need further requires every string expression, such as a string constant, a special register, etc., in the statement to have a CCSID associated with it.

All of the following examples use the naming conventions shown in Table 6-5.

Table 6-5 Naming conventions for examples in this section

Name	Meaning
ET1	EBCDIC Table
UT1	Unicode Table
AT1	ASCII Table
x'.....'	Hexadecimal string constant
UX'.....'	Unicode string constant
GX'.....'	Graphic string constant

Example 6-6 shows the importance of knowing the CCSID associated with the string constants in the predicates.

Example 6-6 Multiple CCSID SQL statement

```
SELECT ET1.C1, UT1.C1
  FROM ET1,UT1
  WHERE ET1.C1 = X’C1C2C3’
    AND UT1.C1 = X’414243’
```

The statement joins two tables, an EBCDIC table ET1 and a Unicode table UT1. The EBCDIC column ET1.C1 is compared with a hexadecimal constant X’C1C2C3’. The Unicode column UT1.C1 is compared with a hexadecimal constant X’414243’. (Note that x’C1C2C3’ in EBCDIC represents the same letters as x’414243’ in Unicode.)

Should the hexadecimal constants use EBCDIC or Unicode? Or should DB2 use EBCDIC for the comparison of ET1.C1=X’C1C2C3’ and Unicode for UT1.C1=X’414243’ ? This decision influences the result set and must be consistent for all those decisions which need to be made by DB2.

In fact, coding the statement as shown in Example 6-6 is probably not going to give you the result that you want. Let us look at the rules which govern multiple CCSID set SQL statements.

Assume that the application encoding scheme for the package which issues this statement is EBCDIC. In this case, DB2 uses the EBCDIC SBCS CCSID set to “interpret” both hexadecimal constants. If the hexadecimal constants are as shown in Example 6-6, this converts to an SQL statement shown in Example 6-7.

Example 6-7 Conversion of hexadecimal constants

```
SELECT ET1.C1, UT1.C1
  FROM ET1,UT1
  WHERE ET1.C1 = ’ABC’
    AND UT1.C1 = ’ âä’
```

This is not what we intended with this query. The purpose was to compare both C1 columns with value ’ABC’. In EBCDIC x’C1C2C3’ is ’ABC’, but x’414243’ equals ’ âä’. In order to have the Unicode and the EBCDIC column evaluated against ’ABC’, your query must look like Example 6-8.

Example 6-8 Correct encoding of hexadecimal values

```
SELECT ET1.C1, UT1.C1
  FROM ET1,UT1
  WHERE ET1.C1 = X’C1C2C3’
    AND UT1.C1 = X’C1C2C3’
```

Rule: When a statement contains table objects with different CCSID sets, the CCSID set associated with a string constant or a special register is determined by the application encoding scheme.

Why You Need to Know More ....

Redbooks

Character conversion can influence:

- The length of your result rows

```
SELECT HEX(U.WORKDEPT||A.PROJECT),U.LASTNAME
FROM AT1 A,UT1 U
WHERE A.LASTNAME=U.LASTNAME  AND A.LASTNAME = 'HAAS';
---------+---------+---------+---------+---------+---------
                                LASTNAME
---------+---------+---------+---------+---------+---------
413030C384C384C384C384          HAAS
```

- The format of your result rows

```
SELECT HEX(FIRSTNME)AS FIRSTNAME
FROM ASCII_TB1
WHERE LASTNAME ='GEYER';
---------+---------+----
  FIRSTNAME
---------+---------+----
8E8E8E8E8E8E8E8E8E8E8E
```


6.7.2 What else do you need to know?

Before we describe how DB2 does data comparison and concatenation in more detail, it is important to know what happens to your data when you start using multiple CCSIDs in your SQL statements.

Let us assume that you are working in the following environment:

- UT1 - Unicode table with columns WORKDEPT and LASTNAME. The only row in this table has the following values:
  - WORKDEPT: A00, defined as CHAR(3), SBCS
  - LASTNAME: HAAS
- AT1 - ASCII table with one column, PROJECT, which just contains one row with value ÄÄÄÄ, defined as CHAR(4), SBCS.

Length of result rows - changes

Example 6-9 shows a SELECT statement producing a result row with concatenated columns from a Unicode table UT1 and an ASCII table AT1. If you look at the result row, it looks exactly as you would have expected it to look.

Example 6-9 Select with concatenating two columns

```
SELECT     U.WORKDEPT||A.PROJECT ,U.LASTNAME
FROM AT1 A,UT1 U
WHERE A.LASTNAME=U.LASTNAME  AND A.LASTNAME = 'HAAS';
---------+---------+---------+---------+---------+----
                 LASTNAME
---------+---------+---------+---------+---------+----
A00ÄÄÄÄ          HAAS
```

Apparently, neither the length nor the format has changed, but if you look at the bytes as actually stored, you get the result shown in Example 6-10.

Example 6-10 Select hex value of concatenated value

```
SELECT HEX(U.WORKDEPT||A.PROJECT),U.LASTNAME
FROM AT1 A,UT1 U
WHERE A.LASTNAME=U.LASTNAME  AND A.LASTNAME = 'HAAS';
---------+---------+---------+---------+---------+---------
                                LASTNAME
---------+---------+---------+---------+---------+---------
413030C384C384C384C384          HAAS
The value for WORKDEPT is ‘A00’ and PROJECT is ‘ÄÄÄÄ’
```

As you can see, the representation of ’A00ÄÄÄÄ’ takes up 11 bytes instead of the original seven bytes now. This is, because DB2 has to convert the ASCII data to Unicode. Depending on which Unicode CCSID DB2 uses in a particular conversion, the number of bytes used to store a character may increase from one to four bytes. In Example 6-10, the representation of character ’Ä’ in Unicode (UTF-8) takes up two bytes x’C384’. Refer to 6.3.1, “UTF-8” on page 507 to learn more about Unicode UTF-8 encoding.

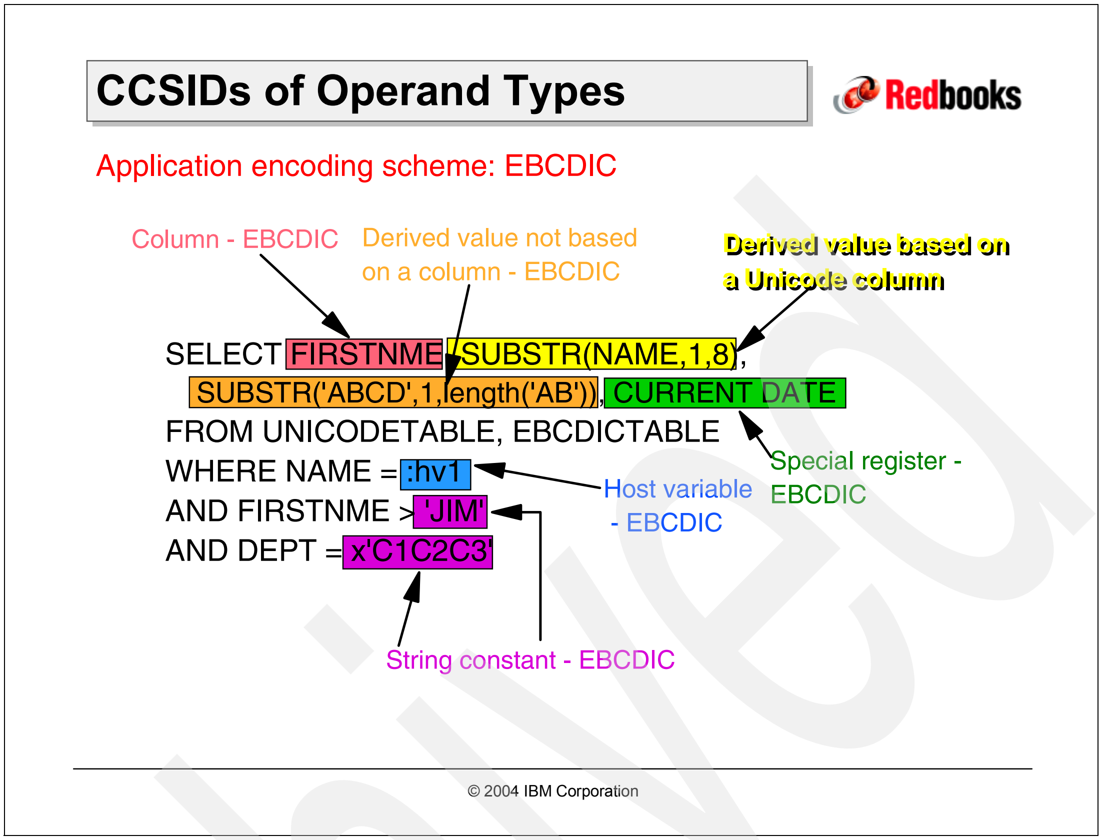
Encoding scheme of your result rows - changes

Apart from the changes in length, the encoding scheme of your result data can also change if you reference more than one CCSID in your SQL statement. Example 6-11 below shows how the hexadecimal representation of ’ÄÄÄÄÄÄÄÄÄÄ’ in ASCII CCSID 437. The x’8E’ is used to represent the character ’Ä’, whereas in the result row shown above in Example 6-10, x’C384’ is used.

Example 6-11 Single CCSID SELECT statement - hex representation

```
SELECT HEX(FIRSTNME)AS FIRSTNAME
FROM ASCII_TB1
WHERE LASTNAME ='GEYER';
---------+---------+----
  FIRSTNAME
---------+---------+----
8E8E8E8E8E8E8E8E8E8E
```

If, for some reason, you must look at your result data using the hexadecimal representation, you must know which encoding scheme is used for the table in order to be able to interpret the byte sequence correctly. According to the examples shown above, and after reading the Unicode chapter, you might be able to convert x’C384’ back from UTF-8 to ASCII CCSID 437, but without knowing that what you see is UTF-8, you cannot tell which character is represented here. In this case, you could have also read x’C384’ as characters “Cd” (where x’C3’ stands for “C”, and x’84’ stands for “d” in EBCDIC code page 37).



6.7.3 Operand types

Before we can go into more detail regarding how DB2 deals with multiple CCSID sets in SQL statements, we must first look at the terminology.

For the purpose of CCSID determination, string expressions in SQL statements are divided into six types, called operand types. Refer to Table 6-6 for a more detailed description of each operand type and its associated CCSID.

Table 6-6 Operand types for CCSID determination

Operand type	CCSID of the operand type
Columns	CCSID from the containing table
String constants	CCSID associated with the application encoding scheme: ► For dynamic statements, this is the CURRENT APPLICATION ENCODING SCHEME special register. ► For static statements, this is the ENCODING bind option. UX'...' hexadecimal constant is always Unicode DBCS. GX'...' The CCSID is the graphic string CCSID for the application encoding scheme, which must be ASCII or EBCDIC.
Special registers	CCSID associated with the application encoding scheme: ► For dynamic statements, this is the CURRENT APPLICATION ENCODING SCHEME special register. ► For static statements, this is the ENCODING bind option.
Host variables	CCSID specified in the DECLARE VARIABLE statement, associated with the application encoding scheme, or specified in SQLDAID of SQLDA.
Derived value based on a column	CCSID of the source of the derived value: A derived value based on a column is an expression, other than a column, a constant, a special register, or a host variable, whose source is directly or indirectly based on a column. The CCSID of such an expression is the CCSID of its source. `Example: SUBSTR(column_1,5,length(column_2))` The CCSID is the one of column_1. Note that column_2 has no influence on the CCSID of SUBSTR. Another example of this operand type is a cast specification that is based on a specific column.
Derived value *not* based on a column	CCSID of the source of the derived value: A derived value not based on a column is an expression, other than a column, a constant, a special register or a host variable, whose source is not directly or indirectly based on any column. The CCSID of such an expression is the CCSID of its source. `Example: SUBSTR('ABCD',1,length('AB'))` The CCSID which is used here is the CCSID of the string constant ABCD. Follow the rules for string constants to find out which CCSID ABCD has. Two other examples for this operand type are: ► UDFs ► CAST specification if not column based

String Comparison

Which operand supplies the CCSID for character conversion?

First operand	Column value or derived value based on a column	Derived value not based on a column	String constant	Special register	Host variable
Column value or derived value based on a column	Note 1	first	first	first	first
Derived value not based on a column	second	Note 1	Note 1	Note 1	Note 1
String constant	second	Note 1	Note 1	Note 1	Note 1
Special register	second	Note 1	Note 1	Note 1	Note 1
Host variable	second	Note 1	Note 1	Note 1	Note 1

Note 1: If the CCSIDs are different , convert (both) operands to Unicode

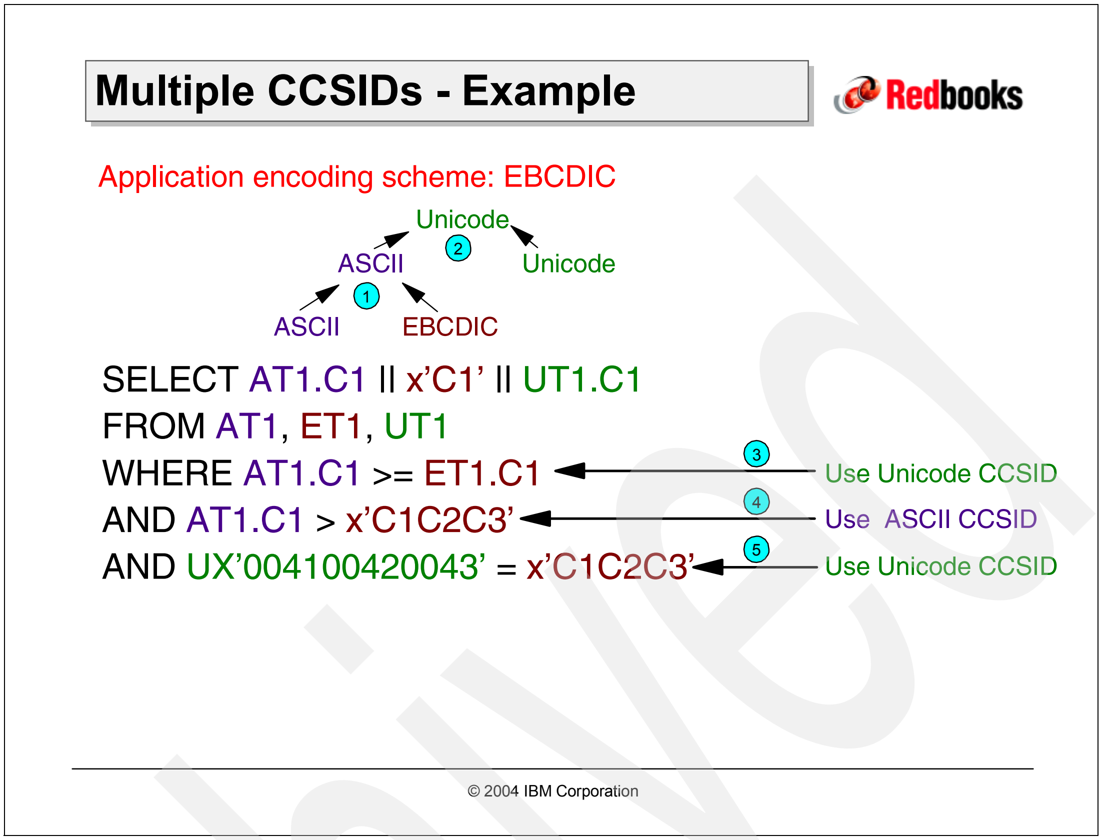
6.7.4 String comparisons with multiple CCSID sets

You can use the table shown on the visual to find out which operand supplies the CCSID for the character conversion, if character conversion is required.

Typically the evaluation of semantic rules involves two operands, operand 1 and operand 2. CCSID conversion is required if all of the following are true:

- The CCSID of operand 1 is different from the CCSID of operand 2.
- Neither CCSID is x'FFFF', that is, neither operand is defined as FOR BIT DATA, nor BLOB.
- The operand selected for conversion is neither NULL, nor empty.

> **Note 1:** If the CCSID sets are different, both operands are converted, if necessary, to Unicode.



6.7.5 Multiple CCSID set SQL statements - example

Let us now look at an example to become more familiar with the "string comparison" table provided earlier.

The visual shows a sample SELECT statement with multiple CCSIDs involved. The statement joins three tables.

> **Important:** The evaluation for CCSID conversion in (most) comparisons and concatenations is a pair-wise evaluation.

The following explanation demonstrates how comparisons are done, and how to use the table from the previous section to find out which operand supplies the CCSID for the comparison or concatenation.

1. AT1.C1 || x'C1' is evaluated first. AT1.C1, which is the first operand, is a column. X'C1', which is the second operand, is a string constant. If you refer to the table on the previous visual, you see that the first operand supplies the CCSID for this concatenation. The result of this first comparison is a column value based on a column, whose associated CCSID is ASCII. Therefore the resulting CCSID of the expression AT.C1 || X'C1' is ASCII.
2. The expression AT.C1 || X'C1' (ASCII) becomes the first operand for the second operation. The second operand (UT.C1) is a column with Unicode CCSID, because the table's encoding scheme is Unicode. The concatenation of these two operands ends up in column one, row one on the string comparison table ("column value or derived value based on a column" for both operands), which means that since the encoding schemes of both operands are different, they are both converted to Unicode.

3. AT1.C1 is an ASCII column. ET1.C1 is an EBCDIC column. Since two columns with different encoding schemes are compared here, they are both converted to Unicode.
4. AT1.C1 is an ASCII column. X’C1C2C3’ is a hexadecimal constant, which is interpreted in the application encoding scheme, which is EBCDIC. Since the column value is always favored, the comparison table shows ’first’ here, which means that the hexadecimal value x’C1C2C3’ is converted to ASCII.
5. UX’414243’ is an Unicode constant. The hexadecimal values are interpreted as Unicode values, that is, it converts to ’ABC’. x’C1C2C3’ is an hexadecimal constant and is therefore interpreted according to the application encoding scheme, which is EBCDIC in our example. The string comparison table shows a ’1’, which means that in case of different encoding schemes, both operands are converted to Unicode.

String Comparison - 2

Redbooks

Which operand supplies codepage within encoding scheme for conversion:

		second operand		
		SBCS Data	Mixed Data	DBCS Data
first operand	SBCS Data	N/A	second(1)	second
	Mixed Data	first(1)	N/A	second
	DBCS Data	first	first	N/A

(1) - For ASCII and EBCDIC data:

- If MIXED DATA = YES, SBCS operand converted to MIXED
- If MIXED DATA = NO, MIXED operand converted to SBCS

6.7.6 Which code page within the encoding scheme is used?

In the previous example we looked at the general encoding scheme of the data which is a candidate for conversion. In addition to that, conversion can also force the data to be changed, for example, from SBCS to mixed, or DBCS data.

The table above shows which operand is selected for conversion when comparing operands with different SBCS, mixed, or DBCS attributes.

As shown in the very beginning of this section, the length of your result data may change if a statement contains multiple CCSID sets. The string then becomes a varying-length string. That is, the data type of the string becomes VARCHAR, CLOB, VARGRAPHIC, or DBCLOB.

Refer to Table 6-7 to determine the result length of CCSID conversion. The “x” in Table 6-7 represents LENGTH (string in bytes).

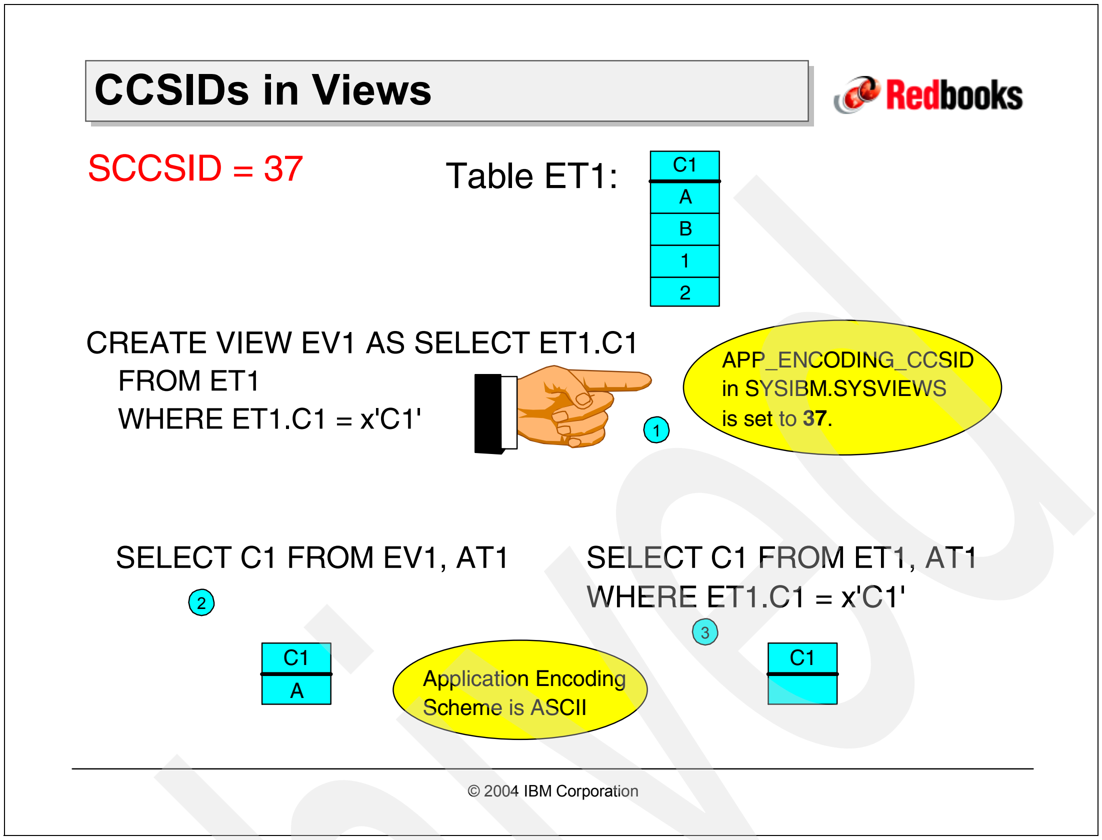
Table 6-7 Result length of CCSID conversion

To CCSID		***EBCDIC***			***ASCI***			***Unicode***		
From CCSID		**SBCS**	**Mixed**	**DBCS**	**SBCS**	**Mixed**	**DBCS**	**SBCS**	**Mixed**	**DBCS**
EBCDIC	**SBCS**	X	X	X*2[1]	X	X	X*2[1]	X[1]	X*3	X*2
	Mixed	X	X	X*2[1]	X	X	X*2[1]	X[1]	X*3	X*2
	DBCS	X*0.5[1]	X+2	X	X*0.5[1]	X	X	X*0.5	X*1.5	X
ASCII	**SBCS**	X	X	X*2[1]	X	X	X*2[1]	X[1]	X*3	X*2
	Mixed	X	X*1.8	X*2[1]	X	X	X*2[1]	X[1]	X*3	X*2
	DBCS	X*0.5[1]	X+2	X	X*0.5[1]	X	X	X*0.5	X*1.5	X
Unicode	**SBCS**	X	X	X*2	X	X	X*2	X	X	X*2
	Mixed	X	X*1.25	X	X	X	X	X	X	X*2
	DBCS	X*0.5	X+2	X	X*0.5	X	X	X*0.5	X*1.5	X

1. These conversions are not allowed because of the high probability of data loss and getting SQLCODE -332 returned.

Tip: The catalog table SYSIBM.SYSCOLUMNS contains the necessary information for you to find out which encoding scheme applies to which column and whether it is SBCS, mixed, or DBCS. You can refer to the CCSID column to find out the encoding scheme. The FOREIGNKEY column shows which subtype is associated with a specific column.

Note: Column CCSID is new in V8.



6.7.7 CCSID sets of a view

In this section we discuss how to deal with CCSIDs in views. First, we discuss the case where the view only contains a single CCSID. Next, we look at views with multiple CCSIDs.

Single CCSID views

If a view only contains one CCSID, the "old" conversion rules apply. That is, if, for example, you create a view and the underlying table is an EBCDIC table, and your DB2's system CCSID is set to 37, the APP_ENCODING_CCSID column in SYSIBM.SYSVIEWS is set to 37 as well. This situation is shown as "1" in the visual above. This means that at creation time of the view, the CCSID value associated with each column and each expression in the fullselect is determined and locked in.

As a consequence, even if you reference this view from applications that use different application encoding schemes, using multiple CCSID SELECT statements, DB2 always interprets x'C1' as "A", as shown in "2".

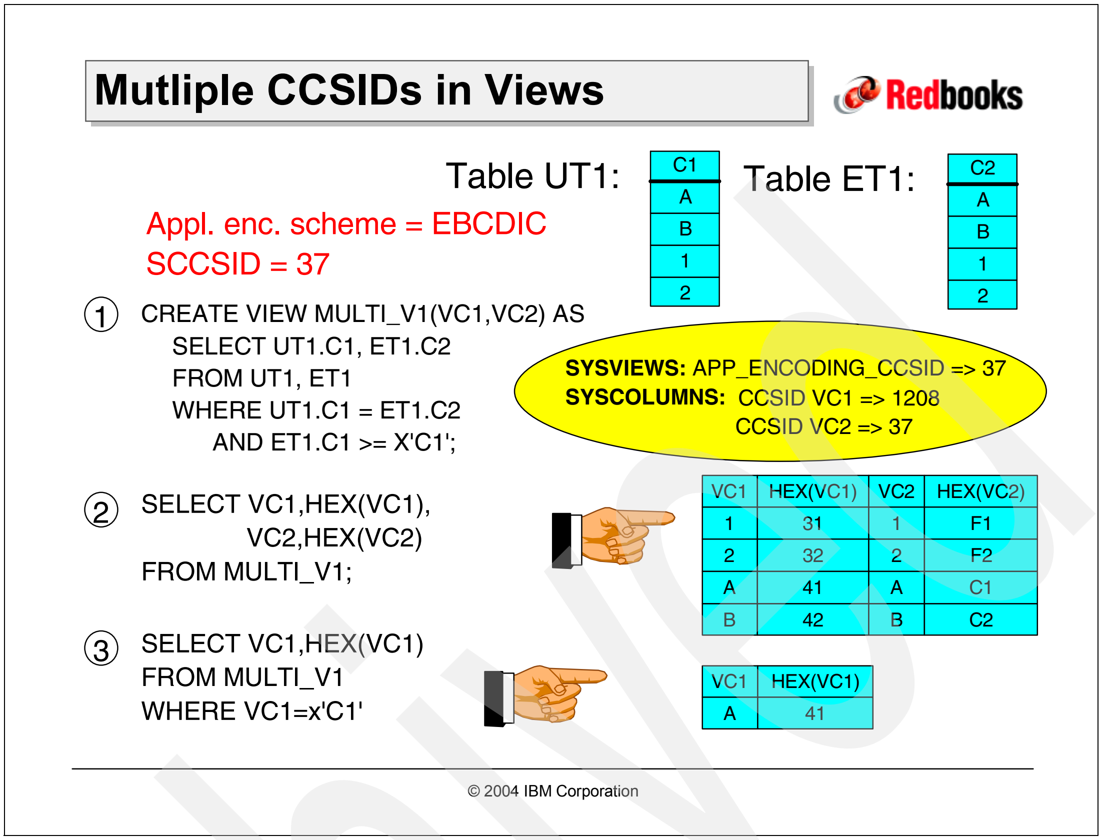
If DB2 would not have this information available in SYSIBM.SYSVIEWS, it would interpret x'C1' as "Á" (A with accent), which is the ASCII character for code point x'C1' and therefore not return any rows.

Note: If you access your tables directly, instead of using the view, as shown in “3”, and you are using an application encoding scheme that is different from the one in column APP_ENCODING_CCSID in catalog table SYSIBM.SYSVIEWS, you may receive different results than when you use the view to select from.

Note also that this can create some interesting challenges, when a view needs to be dropped and recreated as part of a schema change of any of the underlying tables.

Important: The described situation can lead to differences in results between working with views, or working with tables directly. If you use views to select your data, you always receive the exact same result independent of the application encoding scheme you are using. If you are accessing your tables directly, and use operands for comparison, which are dependent on the application encoding scheme, such as hexadecimal constants, it is likely that you receive different results.

Tip: APP_ENCODING_CCSID is a new column in catalog table SYSIBM.SYSVIEWS. This column is used to lock in all predicate CCSIDs at view creation time.



Multiple CCSID views

If you create a view which refers to tables with multiple CCSIDs, every column of the view has a specific CCSID associated with it.

Case "1"

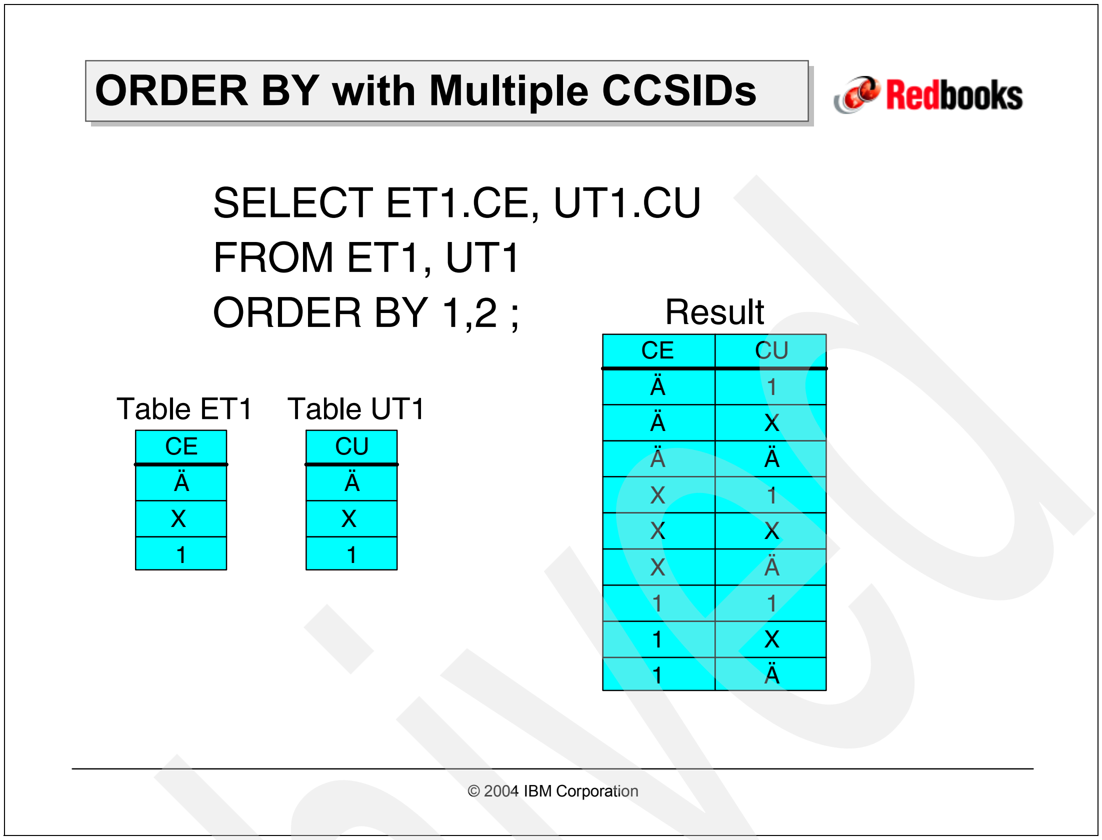
View MULTI_V1, in the figure above, refers to a Unicode and an EBCDIC table. In SYSIBM.SYSCOLUMNS, DB2 stores CCSID 1208 for column VC1 and CCSID 37 for column VC2.

Case "2"

If you select VC1 and VC2 in conjunction with the hex values of the returned characters, you can see that the characters keep their encoding schema. The values returned for column VC1 are represented as Unicode values, whereas the characters in column VC2 are shows as EBCDIC.

Case "3"

The fact that DB2 remembers that both columns are using different encoding schemes, can also be verified if you look at case "3" in the figure above. On the one hand, DB2 needs to know that the contents of column VC1 are stored in Unicode. On the other hand, if DB2 would not treat the comparison as it does in a multiple CCSID statement — that is, interpreting the hexadecimal constant x'C1' as EBCDIC, which is the application encoding scheme, and converting it to Unicode later for the actual comparison — then the SELECT statement would not have returned a single row. This is true, because none of the hexadecimal values stored in column VC1 equals x'C1'. As you can see from the visual above, DB2 returns one result row, because x'C1' has been interpreted as EBCDIC character 'A'. Prior to the comparison of the values in column VC1, this character is converted to Unicode, which is x'41'. As you can see from the resulting table in part "2", x'41' is present in VC1.



6.7.8 Special considerations regarding multiple CCSID sets

When talking about SQL statements which contain multiple CCSID sets, there are some additional things that you must take into account.

ORDER BY sort order

The statement shown on the visual above joins two tables, an EBCDIC table ET1 and a Unicode table UT1. Each column in the select list has its own CCSID. The ORDER BY clause produces the order based on the code points of the EBCDIC column ET1.C1 followed by the code points of the Unicode column UT1.C1.

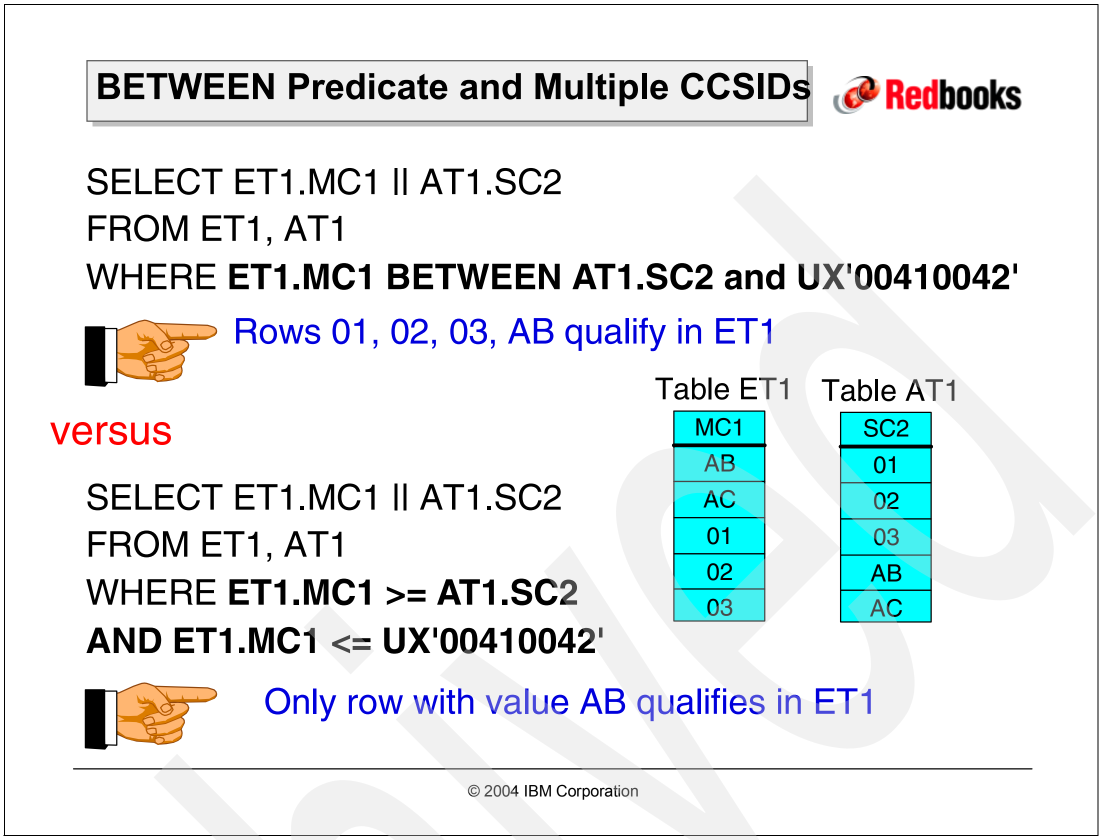
In EBCDIC code page 37, the characters shown on the visual have the following code points:

- Ä -> x'63'
- X -> x'E7'
- 1 -> x'F1'

In Unicode, the characters shown on the visual have the following code points:

- Ä -> x'C384', U+C4
- X -> x'58', U+58
- 1 -> x'31', U+31

As shown in the result, this is not different from any other ORDER BY; all values are ordered based on their hex representation. However, when using multiple CCSID set SQL statements, you can have multiple collating sequences in a single result set.



BETWEEN predicate

Generally, the BETWEEN predicate can be decomposed into basic predicates connected by the AND logical operator, that is, the two statements shown in Example 6-12 are equivalent in Version 7.

Example 6-12 BETWEEN - equivalent statements

```
    SELECT COL1, COL2
    FROM TAB1
    WHERE COL1 BETWEEN 'A' and 'Z'

 is equivalent to:

    SELECT COL1, COL2
    FROM TAB1
    WHERE COL1 >= 'A'
    AND COL1 <= 'Z'
```

This is also true for V8, as long as you do not have multiple CCSID sets in your SELECT statements. The logical equivalence is only true when the CCSIDs used in the comparison are the same between the original predicate and the decomposed one.

In the example on the visual above, the statement joins two tables, an EBCDIC table ET1, and an ASCII table AT. In order to avoid CCSID conversion conflict on the left-hand side (LHS) operand, BETWEEN predicates are processed in the following two steps:

1. Evaluate the high and low bound values.
2. Evaluate the result of step 1 with the LHS operator.

In the case of the example shown in the figure, this means that the following evaluation steps are performed:

1. Evaluate: AT1.SC1 AND UX’00410042’

   The CCSID of UX’00410042’ is always Unicode 1200, which is a DBCS.

   If you refer to the string comparison tables shown in 6.7.4, “String comparisons with multiple CCSID sets” on page 556 and 6.7.6, “Which code page within the encoding scheme is used?” on page 559, the value UX’00410042’ is converted to ASCII DBCS.
2. Now the LHS operand, ET1.MC1, which is EBCDIC mixed, is evaluated. This evaluation ends up in a conversion to the Unicode DBCS CCSID.

As a consequence, for example, the collating sequence that is used for the comparison is the Unicode collating sequence, which is different from the EBCDIC collating sequence. To be more concrete, UX’00410042’ has code points x’41’ and x’42’.

Let us now look at the values that are checked if you decompose the BETWEEN predicate. Refer to the second SQL statement in the figure above. The following values are evaluated here:

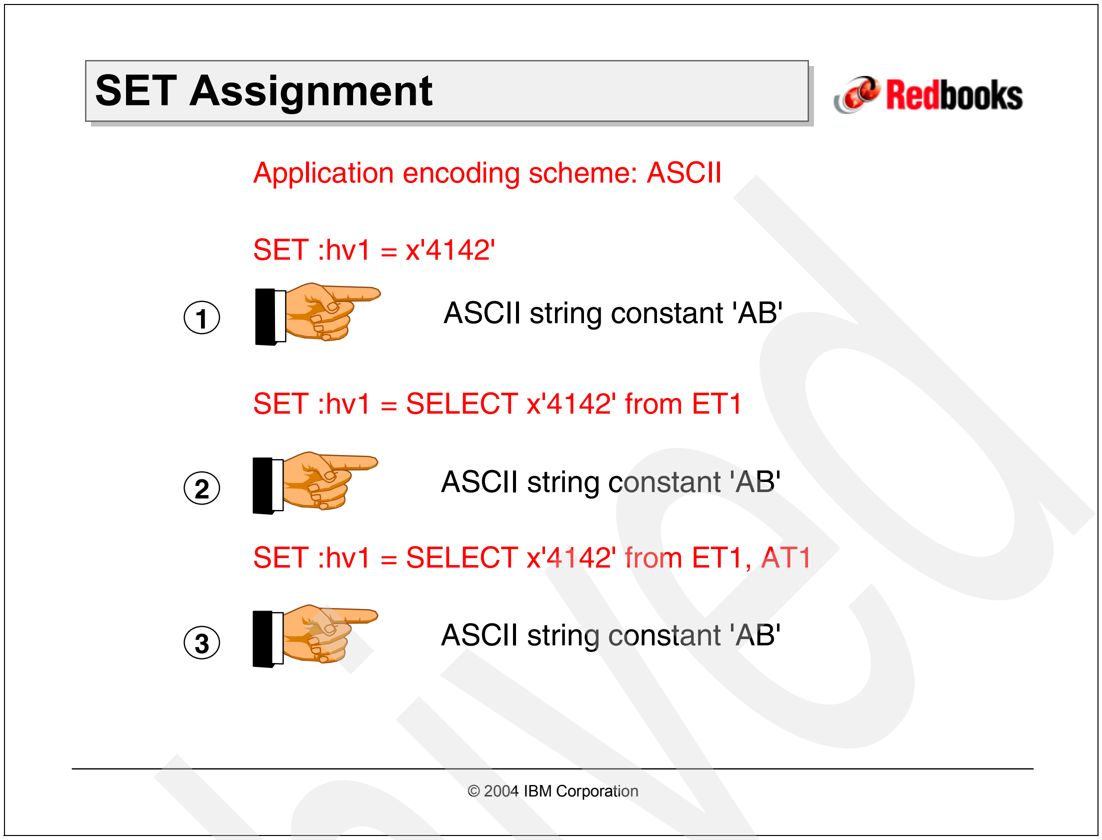
1. WHERE ET1.MC1 >= AT1.SC1

   Here we have two columns with different CCSIDs. Following the conversion rules, those are converted to Unicode mixed CCSIDs.
2. AND ET1.MC1 <= UX’00410042’

   UX’00410042’ is converted to EBCDIC DBCS.

Again, as the collating sequence for EBCDIC is different than the one in Unicode, in case 1, all rows qualify the request, whereas the decomposed statement results in just one row.

The same CCSID consideration applies to the transitive closure rule. If the generated predicate is a range predicate, that is, the operator is >, >=, <, or <=, and would use a different CCSID in CCSID conversion, the transitive closure predicate is not generated. This might lead to a performance degradation.

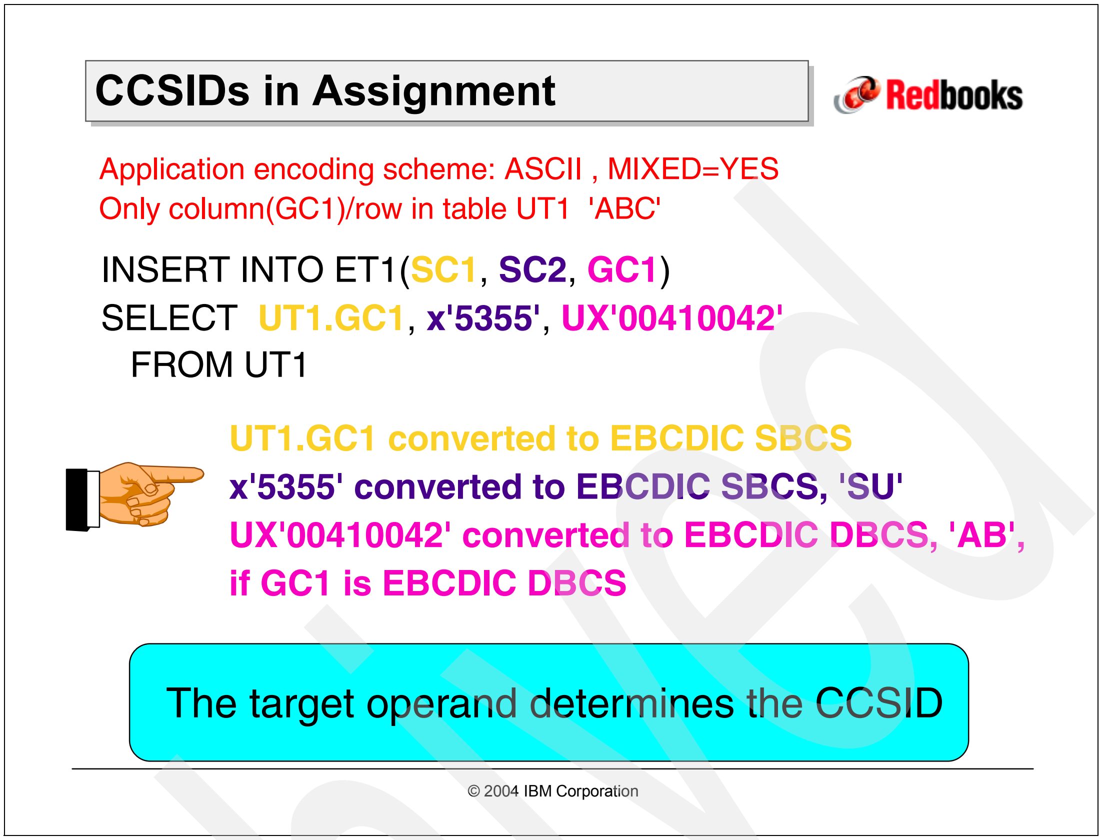


6.7.9 CCSID for SQL typically not associated with table objects

The SET assignment, VALUES INTO, SET special register, VALUE, and CALL statements are statements that, prior to Version 8, do not have a table context. Beginning with DB2 V8 ENFM, they are treated as multiple CCSID set statements, which means that the new CCSID conversion rules apply.

Refer to the visual above:

- In case number 1, DB2 behaves exactly the same as it does in V7, that is, the application encoding scheme is used to interpret the hexadecimal string.
- In case number 2, following the regular conversion rules, the string constant would inherit the encoding scheme from the EBCDIC table. That is, x'4142' would convert to the appropriate character for the EBCDIC CCSID but not to x'AB'. Since, as mentioned above, the SET assignment statement is considered to be a multiple CCSID statement, the string constant is converted according to the application encoding scheme, which is ASCII in our example. Therefore, x'4142' is nicely converted to 'AB'.
- In case number 3, apart from the SET assignment statement, the subselect itself is a multi-CCSID statement. Therefore, the hexadecimal constant is interpreted using the application encoding scheme.



6.7.10 CCSID conversion for assignments and CAST functions

In the evaluation of assignment and cast rules, there is a concept of a *source* operand and a *target* operand. The CCSID set of the *target* operand determines the CCSID set in CCSID conversion.

- For assignment, the CCSID value of the *target* operand is used in the actual conversion.
- For cast, when DBCS CCSID is cast to SBCS or mixed CCSID, it is always converted to the mixed CCSID.

Refer to the example shown in the visual. Although the INSERT statement contains multiple CCSIDs, the values which are inserted into the EBCDIC table ET1 are all converted to EBCDIC, and then inserted. That is, if you want to insert "SU" into SC2, you must code SU in ASCII (the application encoding scheme), that is, exactly as it has been done in the example. Then "SU" is going to be converted to EBCDIC and stored as x'E2E4'.

This is a little bit different for Unicode constants, because those are always interpreted as Unicode and then, as shown in our example, converted to whatever column GC1 is. If it is EBCDIC DBCS, then it is converted to that type. If it is mixed, then it is converted to mixed.

Multiple CCSID Sets Restrictions

The following *view-name* must be single CCSID set view

- CREATE TABLE LIKE *view-name*
- CREATE GLOBAL TEMPORARY TABLE LIKE *view-name*
- DECLARE GLOBAL TEMPORARY TABLE LIKE *view-name*

The result columns of the fullselects below must be in a single CCSID set

- DECLARE GLOBAL TEMPORARY TABLE AS (fullselect) WITH NO DATA / DEFINITION ONLY
- CREATE TABLE AS (fullselect) WITH NO DATA / DEFINITION ONLY
- ALTER TABLE ADD (fullselect) WITH NO DATA / DEFINITION ONLY
- CREATE TABLE AS (fullselect) refreshable-table-options
- ALTER TABLE ADD (fullselect) refreshable-table-options

Fullselect SELECT list colums in MQT must be a single CCSID and must be the same CCSID set as the table space specified in the IN clause

6.7.11 Restrictions on multiple CCSID sets

There are some restrictions when using DDL statements with multiple CCSID sets.

Views

You cannot use the following DDL statements, if the columns of the underlying views are referencing multiple CCSID sets:

- CREATE TABLE LIKE view-name
- CREATE GLOBAL TEMPORARY TABLE LIKE view-name
- DECLARE GLOBAL TEMPORARY TABLE LIKE view-name

This restriction applies, because it is not allowed to have different encoding schemes within one table. That is, all columns within one table have to be either EBCDIC, ASCII, or Unicode. The statements fail with an SQLCODE = -873 “ERROR: The statement referenced data encoded with different encoding schemes or ccsids in an invalid context”.

Declared temporary tables

When you create a declared temporary table as WITH NO DATA (or the equivalent DEFINITION ONLY keyword), the outermost SELECT list of the *fullselect* in the statement below must not reference data that is encoded with different CCSID sets.

- DECLARE GLOBAL TEMPORARY TABLE AS (*fullselect*) WITH NO DATA

Materialized query tables

For MQTs the same restriction applies. The outermost SELECT list of the *fullselect* in the statements below must not reference data that is encoded with different CCSID sets:

- CREATE TABLE AS (*fullselect*) WITH NO DATA
- CREATE TABLE AS (*fullselect*) refreshable-table-options
- ALTER TABLE ADD (*fullselect*) refreshable-table-options

The following statement fails with SQLCODE -20058, "ERROR: The fullselect specified for materialized query table xxxx is not valid", because the resulting columns for the table are EBCDIC (COL1) and Unicode (COL2).

```
CREATE TABLE MQTTEST1 AS
       (SELECT COL1, UX'0048' AS COL2
          FROM EBCDICTABLE)
       DATA INITIALLY DEFERRED REFRESH DEFERRED
```

In contrast to that, the following statement ends successfully:

```
CREATE TABLE MQTTEST2 AS
       (SELECT COL1
          FROM UNICODETAB, EBCDICTAB)
       DATA INITIALLY DEFERRED REFRESH DEFERRED
```

Although the SELECT statement is a multiple CCSID set statement, COL1 is the only column that is returned in the SELECT list and it is (of course) one CCSID set, which is Unicode.

The single CCSID set that the columns returned in the SELECT list are in must be the same CCSID set as the table space specified in the IN clause or specified in the CCSID clause.

7

Network computing

List of Topics

Redbooks

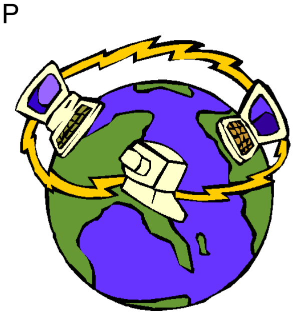
CDB enhancements
- Requester database ALIAS
- Location server ALIAS

Member routing in TCP/IP
- Using SYSIBM.IPLIST
- Data sharing member subsetting via DRDA TCP/IP

DDF and RRS accounting rollup

DRDA data stream encryption

Other enhancements



In this chapter we examine the following enhancements:

- Location alias names:

  A location alias name is an alternative location name you can use to access a DB2 subsystem or database through the network. DB2 V8 brings in a number of changes related to the use of aliases.

  Requester location alias names allows you to access a given database on many DB2 for LUW systems, even if thousands of them exist with the same database name. Server location alias names allow you to connect to a DB2 for z/OS subsystem by using more than one location name.

- Member routing in TCP/IP:

  Member routing in TCP/IP consists of two enhancements. The first one allows subsetting of the members that you can access in a DB2 data sharing group as a DB2 for z/OS application requester. The second one builds on the server location alias name support and allows you to connect to a subset of data sharing members from any DRDA application requester.

- Rollup accounting data for DDF and RRSAF threads:

  In Version 8, DB2 accounting data collection is enhanced to optionally accumulate accounting data for DDF and RRSAF threads. This is controlled by a new DSNZPARM parameter, ACCUMACC, and can be dynamically turned on and off.

- DRDA data stream encryption:

  To achieve more effective security in a distributed computing environment, DB2 Version 8 provides the ability to authenticate via encrypted userid, or encrypted userid password, and provide support for encrypting security sensitive data.

- Other network computing enhancements:

  The following items are some of the other network enhancements in DB2 V8:

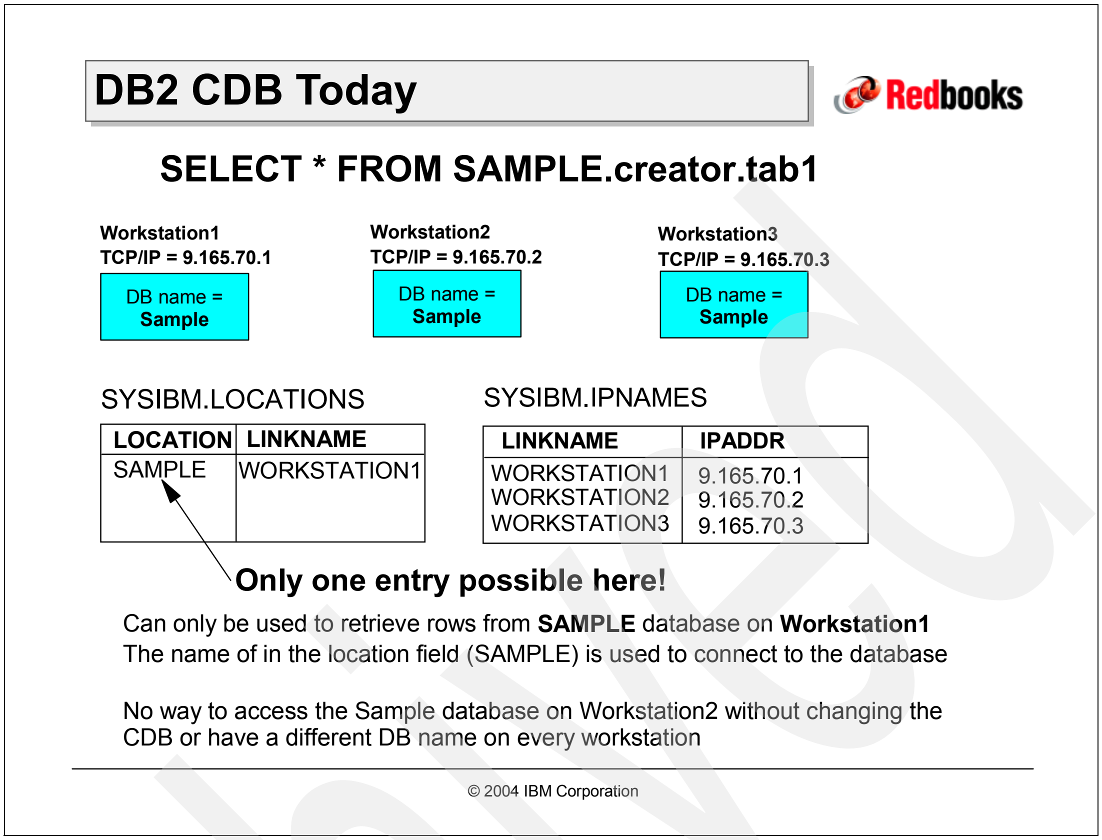
  – There is a change in terminology for “Type 1 Inactive Threads” and “Type 2 Inactive Threads”

  – VTAM® conversation allocation requests can now time-out with RC00D31033.

  – Query blocks can now be larger than 32K for DB2 as a server.

  – Limited SQL functionality in private protocol is provided.

  – Changes have been made to the -DISPLAY LOCATION() command.

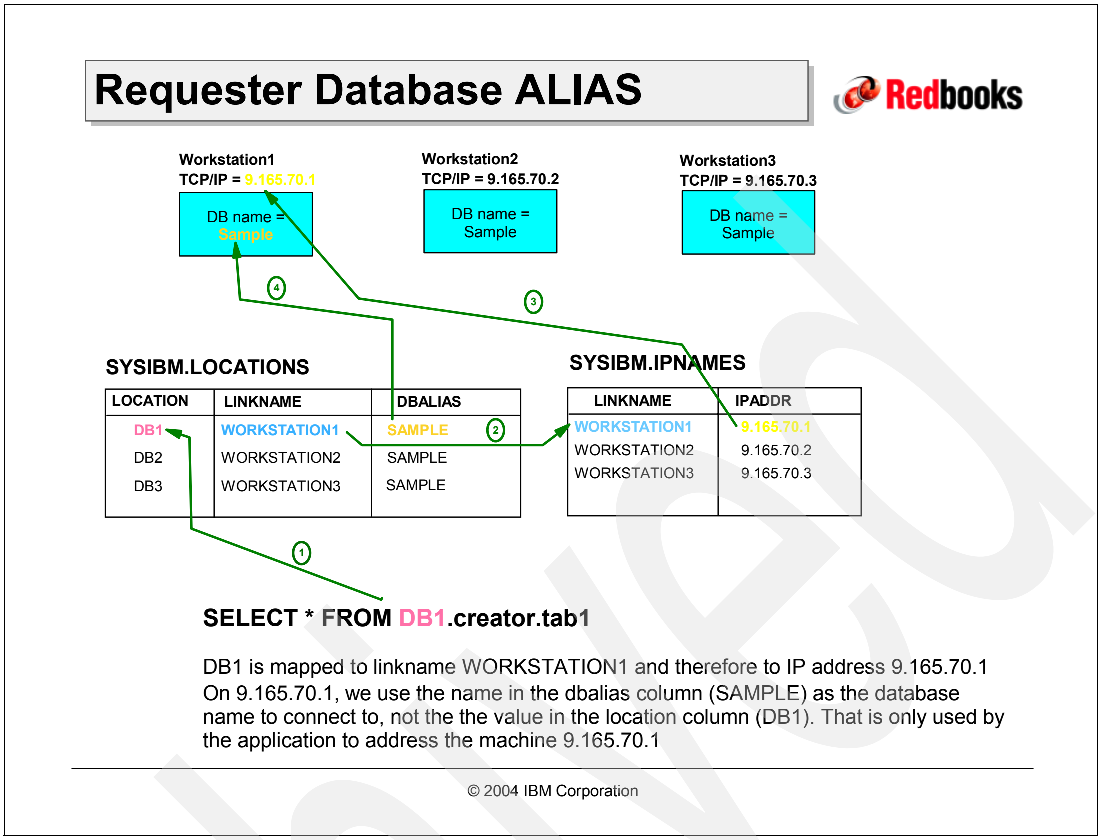


7.1 DB2 CDB enhancements

Today the CDB, or Communications Database, does not really exist anymore. Its tables have been integrated into the DB2 catalog's DSNDB06.SYSDDF table space, but as this term is still commonly used to address those catalog tables that contain information about communicating with other DB2 systems, we continue to use the term CDB throughout this publication.

When connecting to a DB2 UDB for z/OS and OS/390 system through DRDA, you address the entire DB2 subsystem by using its *location name*. A DB2 UDB for LINUX/UNIX/Windows database is known in the network by its *database name* at a particular instance of the database server. If the requester is a DB2 UDB for z/OS system, you must specify the database name of the DB2 UDB for LUW system you want to connect to, in the LOCATION column of the SYSIBM.LOCATION catalog table.

Up to DB2 Version 7, there is always a one-to-one mapping between location name and database name, since the value in the LOCATION column is used as the database name when connecting to a DB2 for LUW system. Prior to DB2 Version 8, there is no way to access multiple DB2 UDB for LUW databases that have the same database name (even when they reside on different machines), unless you catalog an alternate *dbalias* on the server for the *database name* to be accessed. Then you specify this alternate dbalias value in the LOCATION column. Cataloguing of an extra dbalias on each server for each access database name can become an administrative challenge.



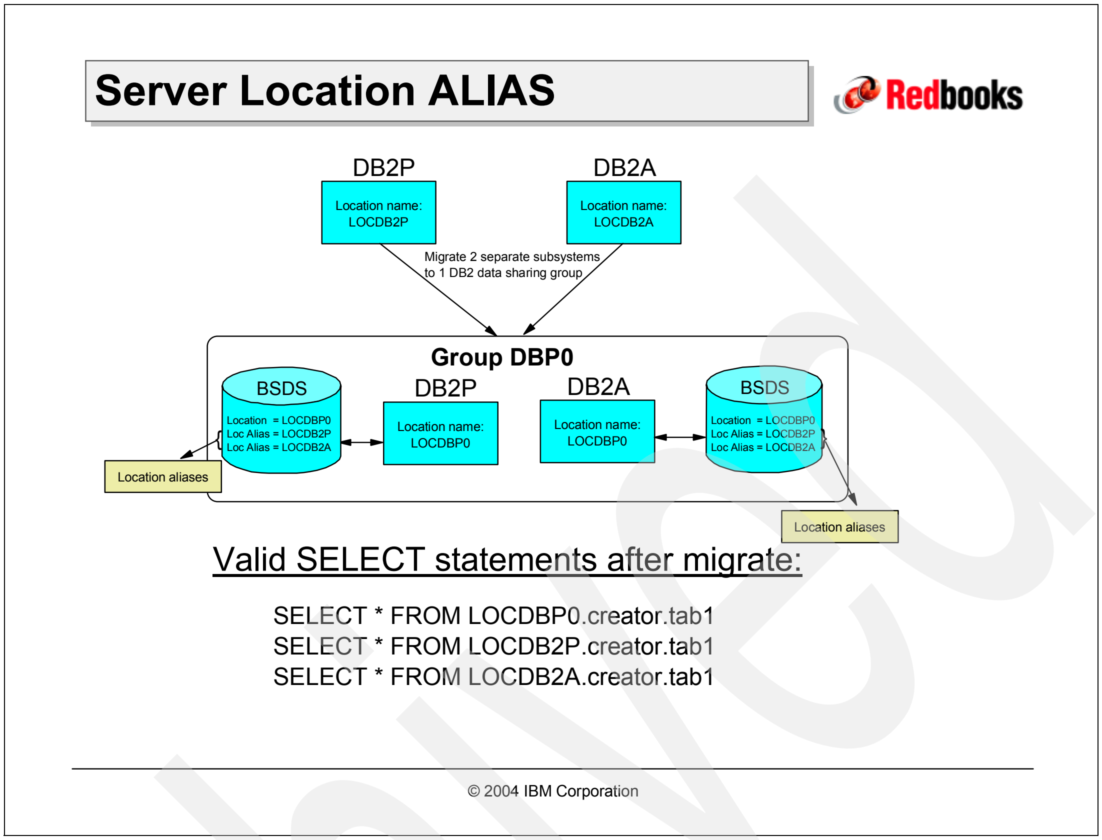
7.1.1 Requester database ALIAS

This restriction regarding names is removed in DB2 Version 8.

A new column DBALIAS is added to the SYSIBM.LOCATIONS table. Here is how it works:

1. You continue to specify the value of the LOCATION name field as the first qualifier of your three-part table name in your SELECT statement.
2. The mapped LINKNAME links you to the corresponding entry in SYSIBM.IPNAMES, which provides the correct TCP/IP address for the workstation you want to access.
3. The entry in column DBALIAS of SYSIBM.LOCATIONS points your SELECT statement to the real database name on the DB2 UDB for Linux, UNIX, and Windows that you want to connect to.

You can now access the SAMPLE database on every LINUX/UNIX/Windows system, even if thousands of them exist with the same database name.



7.1.2 Server location ALIAS

As mentioned before, a DB2 UDB for z/OS server is known in the network by its location name. This name is used by applications to identify a DB2 subsystem or a DB2 data sharing group. When two or more DB2 subsystems are consolidated into a single DB2 data sharing group, multiple locations must be consolidated into a single location (because the entire data sharing group uses the same location name). This means that all applications that use the old location name (used when the DB2 was still a stand-alone subsystem) need to be changed to access the location name of the data sharing group.

Another situation is where you need to move an application from one DB2 system to another. If this application is accessed remotely, you must change the location name in all CONNECT statements, and the first qualifier of the three-part object names of all remote applications using that application, to reflect the new location name of the DB2 subsystem.

To ease this type of migration, DB2 Version 8 allows you to define up to eight alias names in addition to the location name for a DB2 subsystem or data sharing group. A *location alias* is an alternative name that a requester can use to access a DB2 subsystem. DB2 accepts requests from applications that identify the DB2 server with its location name or any of its location alias names.

You do not have to change the location names in your applications programs to be able to access your data after migrating to a new data sharing group. The only thing you must do is to add the additional location alias names to your BSDS data sets on each member of the data sharing group, or DB2 subsystem.

The DB2 Change Log Inventory utility allows you to define up to eight alias names in the BSDS, in addition to the location name. The Print Log Map utility prints any location alias names defined for the DB2 subsystem.

DDF Communication Record

```
**** DISTRIBUTED DATA FACILITY ****
                 COMMUNICATION RECORD
                14:26:17 SEPTEMBER 30, 2003
LOCATION=DB8O
ALIAS=DB8OGRP
LUNAME=LU1 PASSWORD=(NULL) GENERICLU=(NULL)
   PORT=33744 RPORT=33745
```

Output from Print Log Map utility (DSNJU004)

Input statement for Change Log Inventory utility (DSNJ003)

DDF ALIAS=DB8OGRP

7.1.3 DDF communication record

The extra location alias names are defined in the BSDS with the Change Log Inventory utility (DSNJU003). The syntax for the Change Log Inventory utility is:

```
DDF ALIAS = aliasname,aliasname2
```

The ALIAS keyword identifies an alias name for the local location specified by the LOCATION keyword. Up to eight alias names can be specified with the ALIAS keyword, separated by commas. The alias names can be added or replaced by running another DSNJU003 Utility with the ALIAS keyword, specifying a new list of names (more or less) which will replace the old list of names. You always need to specify the entire list of aliases that you want to be known to the system. If you want to add an alias DB8ZGRP to the existing alias DB8OGRP in the visual above, you must specify:

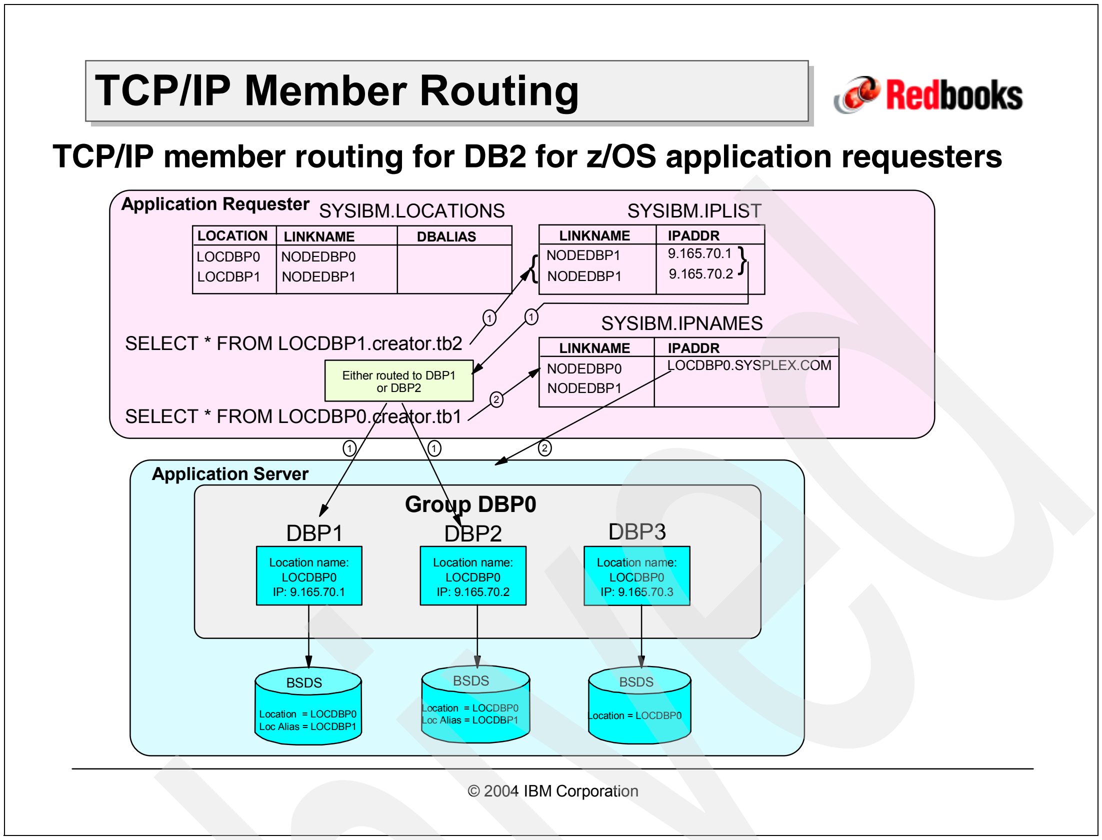
```
DDF ALIAS = DB8OGRP,DB8ZGRP
```

In order to conform to the Change Log Inventory standards, a new keyword NOALIAS is introduced to remove all the location alias definitions from the BSDS, specified by the ALIAS keyword.

```
DDF NOALIAS
```

Note: Alias names are never displayed in messages.

The distributed data facility communication record in the BSDS data sets has been changed to show the location alias names you have specified for your subsystem. This visual shows the output of the Print Log Map utility (DSNJU004). You can see the location alias name DB8OGRP we added for our subsystem.



7.2 Member routing in TCP/IP

Two enhancements are provided in the area of TCP/IP member routing.

- The first one applies only to DB2 for z/OS application requesters.
- The second one applies to any DRDA TCP/IP AR, including DB2 for z/OS V8 application requesters, that want to connect to a DB2 data sharing group.

7.2.1 Member routing in TCP/IP for DB2 for z/OS AR using SYSIBM.IPLIST

Currently in a data sharing environment, remote TCP/IP connections are normally set up to automatically balance connections across *all* members of a data sharing group. This is not a good solution in all cases. Sometimes you want to be able to route requests from DB2 UDB for z/OS DRDA application requesters to specific members of your data sharing group, similar to the support for SNA connections that use the SYSIBM.LULIST table.

To achieve this, we combine the server location alias feature (at the DRDA application server), described earlier, with the use of a new table in the catalog, namely SYSIBM.IPLIST (at the DB2 for z/OS application requester).

The visual describes a sample implementation.

The location name LOCDBP0 represents all three DB2 subsystems in the data sharing group: DBP1, DBP2, and DBP3. Previously, location LOCDBP0 is set up to route requests using the group domain name to all available members.

Now an additional location, LOCDBP1, is to be used to route requests only to members DBP1 and DBP2:

- At the DRDA AS, a location alias LOCDBP1, has been defined in the BSDS for data sharing members DBP1 and DBP2. Note that this alias does not exist in the BSDS for DBP3.
- At the DB2 for z/OS AR, we need to update the catalog tables in the following way:
  - In SYSIBM.LOCATIONS, we have two entries. One for the “normal” location name of the data sharing group (LOCDBP0) to address all members of the data sharing group, and one for the location alias (LOCDBP1).
  - In SYSIBM.IPLIST we have two entries for the link name that corresponds with the link name in SYSIBM.LOCATIONS (NODEDBP1); one pointing to IP address 9.165.70.1, the other one to 9.165.70.2.
  - In SYSIBM.IPNAMES, we also have an entry with the same link name. This is mandatory. When using SYSIBM.IPLIST, you must also have an entry in SYSIBM.IPNAMES with the same link name. Note, however, that you should not specify an IP address in the IPADDR column in SYSIBM.IPNAMES for the NODEDBP1 entry. Otherwise you receive an error (00D31203).

When resolving the remote system you want to access, DB2 first checks the IPLIST table, and then the IPNAMES table. So when executing the SELECT * FROM LOCDBP1.creator.tb2 statement:

1. DB2 first looks in SYSIBM.LOCATIONS for a matching location name (LOCDBP1), and picks up the linkname (NODEDBP1).
2. DB2 then searches the new catalog table SYSIBM.IPLIST for matching linkname entries (NODEDBP1) and picks up their IP addresses.(9.165.70.1 and 9.165.70.2).
3. DB2 then checks SYSIBM.IPNAMES for a matching linkname (NODEDBP1) and makes sure it is there, and no IPADDR is supplied.
4. At the DRDA AS side, LOCDBP1 is checked in the BSDS, and is found to be an alias for DBP1 and DBP2, so the request can only go to those members of the data sharing group.

If a request comes in for location name LOCDBP0, for example SELECT * FROM LOCDBP0.creator.tb1, the entry in SYSIBM.SYSIPNAMES is used (as there is no matching linkname in SYSIBM.IPLIST) and the request is routed to all available members in group DBP0.

Defining multiple location names is not confined to a TCP/IP network, but can also be used to control which members are used to process requests for an application in a SNA data sharing group.

When using dynamic VIPA to perform workload balancing, the IPLIST must contain the member specific dynamic VIPA for each DB2 subsystem that is to be routed to.

The following section shows the catalog definitions at the DB2 for z/OS AR (Table 7-1 to Table 7-3).

To route to all members, rows are inserted for location LOCDBP0 in SYSIBM.LOCATIONS and SYSIBM.IPNAMES. The LOCATION column in SYSIBM.LOCATIONS contains the group location name, LOCDBP0.

To route requests to only DBP1 and DBP2, rows are inserted for location LOCDBP1 in SYSIBM.LOCATIONS, SYSIBM.IPNAMES, and SYSIBM.IPLIST. LOCDBP1 is the server location alias name defined in the BSDS of the members of the data sharing group that you want to address (DBP1 and DBP2).

Table 7-1 Example of TCP/IP member routing using SYSIBM.LOCATIONS

LOCATION	LINKNAME	PORT	TPN	DBALIAS	IBMREQD
LOCDBP0	NODEDBP0				
LOCDBP1	NODEDBP1				

A row is inserted into SYSIBM.IPNAMES where the LINKNAME contains the same LINKNAME used for LOCDBP0 and the IPADDR column contains the group domain name. Another a row is inserted into SYSIBM.IPNAMES where the LINKNAME contains the same LINKNAME used for LOCDBP1 and the IPADDR column contains blanks.

Table 7-2 SYSIBM.IPNAMES

LINKNAME	SECURITY_OUT	USERNAMES	IPADDR	IBMREQD
NODEDBP0			LOCDPB0.SYSPLEX.COM	
NODEDBP1				

Lastly, rows are inserted into SYSIBM.IPLIST for DBP1 and DBP2 where the LINKNAME contains the same LINKNAME for LOCDBP1 and the IPADDR column contains the member-specific names (member specific DVIPA in our case).

Table 7-3 Example of TCP/IP Member Routing using SYSIBM.IPLIST

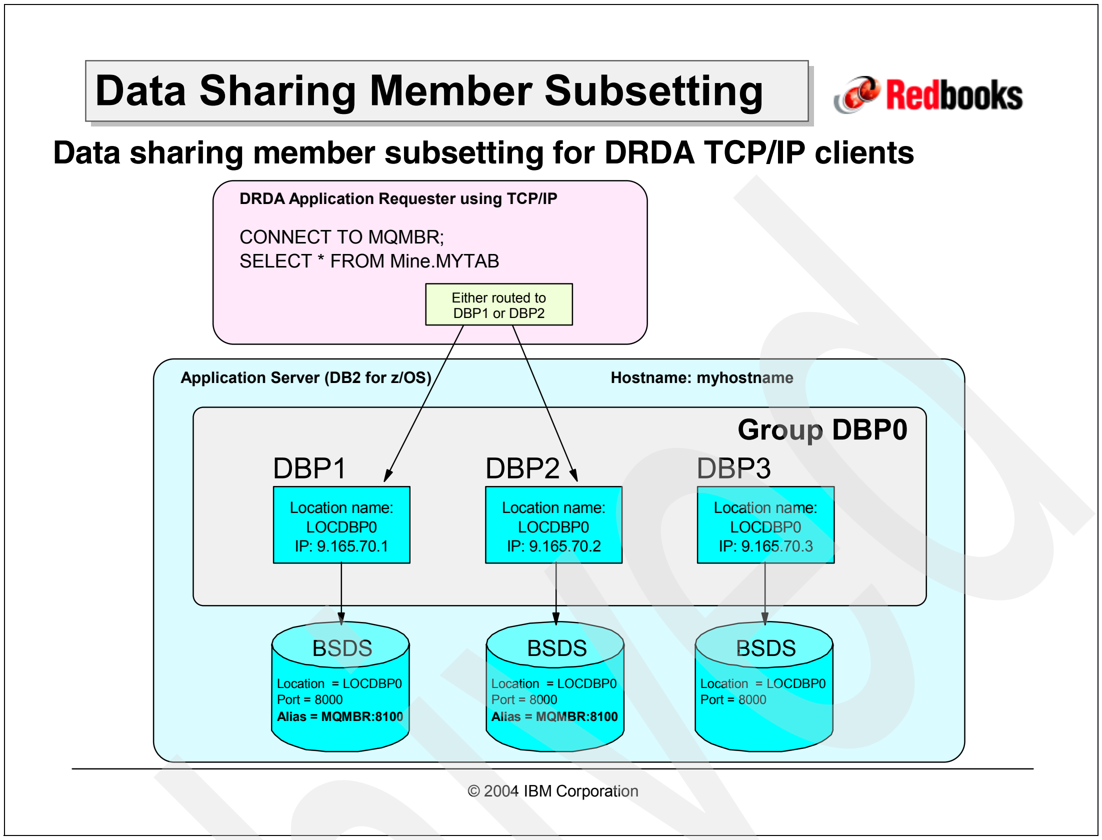
LINKNAME	IPADDR	IBMREQD
NODEDBP1	9.165.70.1	
NODEDBP1	9.165.70.2	

When an application wants to route requests to the least loaded member of the data sharing group, it connects using location LOCDBP0. If an application wants to route requests to member DBP1 or DBP2, it connects to location LOCDBP1.

Each member of the DBP0 DB2 data sharing group, has LOCDBP0 defined as the location name.LOCDBP1 is defined as a location alias name using the Change Log Inventory (DSNJU003) utility in the BSDS of DBP1 and DBP2 only.

In a sysplex environment, by using this new SYSIBM.IPLIST table on the DB2 for z/OS AR, you can define a specific member or a subset of members in a data sharing group to route requests to. At the DRDA AS server (the data sharing group that you want to connect to) server location alias names need to be defined allowing applications to access the DB2 server with the alternate location names.

Note: Because the setup of the IPLIST determines the members of the data sharing group you can connect to, this enhancement only applies to configurations where DB2 for z/OS is also the application requester. Similar support for non-DB2 for z/OS AR is provided by the enhancement described next.



7.2.2 Data sharing member subsetting for DRDA TCP/IP clients

In our description of the previous enhancement, we discussed the use of the new table in the catalog (SYSIBM.IPLIST) to enable DB2 for z/OS application requesters to subset the members of the data sharing group they can connect to. That enhancement does not apply to non-DB2 for z/OS application requesters like DB2 Connect.

By using V7 functionality, you can use the following techniques to subset requests to only a few members of the data sharing group via:

- Enabling some, but not all of the members with the dynamic VIPA/sysplex distributor support
- Disabling DDF server support on some of the members that you want to exclude.

It is obvious that neither of these options are optimal solutions.

In V8, you can also route DRDA requests to a subset of members of a data sharing group, from any DRDA TCP/IP application requestor, like DB2 Connect, based on DB2 location alias names. Subsetting a data sharing group extends the ALIAS support at a DB2 for z/OS V8 DRDA application server. When you have ALIAS names defined with valid port numbers, they become *subset location names*.

To use location subsetting, at least one ALIAS name must be defined with an ALIAS TCP/IP port number. To do this, the syntax of the DDF statement of DSNJU003 (Change Log Inventory) has been enhanced to accept a numeric TCP/IP port number (value must be greater than 0 and less than 65535) preceded by a colon (":") after any entered ALIAS name. If a value preceded by a colon (":"is not entered after an ALIAS name), then this ALIAS name will be used by DB2 as an alternate LOCATION name for this DB2 subsystem.

Each ALIAS port number must be unique and must be different than that already specified for, or being entered for, the PORT and RESPORT parameters. Like the current processing of the PORT and RESPORT values, only numeric port values will be supported. Service name values are not supported. Here is a sample ALIAS statement:

```
ALIAS=DB2A,MQDB2Z:8002,DB2X,MYALIAS1,SIEBEL1:10001
```

In this example, MQDB2Z and SIEBEL1 will be the only ALIAS names to support subsets of the data sharing group. DB2A,DB2X, and MYALIAS1 are also ALIAS names, that can only be used as alternate LOCATION names for this DB2 subsystem.

Example 7-1 shows how the subset location names show up in the output of the print log map utility. In this example, location DSNT2 has two subset location names, DSHT2PROD, listening on port 50251,and DT25 listening on port 50265.

Example 7-1 Print log map output for subset location names

```
                   **** DISTRIBUTED DATA FACILITY ****
                           COMMUNICATION RECORD
                         00:25:55 MARCH 30, 2004
LOCATION=DSNT2
ALIAS=DSNT2PROD:50251,DT25:50265
LUNAME=STBDT25 PASSWORD=(NULL) GENERICLU=(NULL) PORT=50200 RPORT=50205
DSNJ401I  DSNUPBHR BACKUP SYSTEM UTILITY HISTORY RECORD NOT FOUND
```

Currently with both DB2 UDB for z/OS V7 and V8 (without alias port numbers), when DDF is started for a subsystem member of a data sharing group, only a single WLM resource group representing all of the “active” members of a DB2 data sharing group is created.

At DDF start up time, with subsetting support installed in V8, for each ALIAS name defined to be a subset location name that also has an ALIAS port value, the subsetting support will perform the following actions (only if DDF is being started in a subsystem member of a data group):

- Issue calls to WLM sysplex routing services to register ALIAS names in addition to the normal call to register the DB2 data sharing group LOCATION name.
- Add the ALIAS port numbers to the TCP/IP socket SELECT call for the SQL port listener.

To have the list of members participating in the ALIAS subset automatically managed by the operating system, WLM sysplex routing services are used to maintain a “weighted” server list of the DB2 data sharing group for each ALIAS name being used as a subset. (An ALIAS name is used for subsetting if an ALIAS port value exists in the BSDS DDF record.)

If the WLM request to register is unsuccessful for any ALIAS, then (existing) error message DSNL044I is issued to inform you of the error. DDF startup processing continues, but since the ALIAS name was not successfully registered, this DB2 subsystem will not have its information added to the list of members participating in the subset.

As mentioned earlier, the ALIAS port numbers are added to the TCP/IP SELECT socket call for the SQL port listener. This will permit the use of the z/OS Sysplex Distributor (if set up) to send requests destined for an ALIAS port to only the members participating in the ALIAS subset. It will also permit the member to respond to requests made against the ALIAS port.

Prior to issuing the TCP/IP SELECT socket call, DDF attempts to issue a TCP/IP bind against each ALIAS port number. If the bind fails, then the (existing) message DSNL515I will be issued with the ALIAS port value as part of the message. DDF continues its startup, and continues issuing the TCP/IP binds for other ALIAS ports. DDF will then issue the TCP/IP SELECT socket call for the SQL request listener only for its PORT and those ALIAS ports whose TCP/IP bind call was successful. If the TCP/IP SELECT socket call is unsuccessful, then the (existing) error message DSNL512I is issued.

Once DDF startup is complete, with subsetting support installed and active in a member of a data sharing group, the SQL listener will now accept TCP connection requests against its primary SQL PORT and any of the ALIAS ports that did not cause a TCP/IP bind error.

Once DDF in all the members of a data sharing group has completed its startup, multiple WLM sysplex routing services resource groups will exist; one for the LOCATION name of the data sharing group which should have all members registered in it, and one each for the ALIAS names that had ALIAS ports defined in any of the members.

This visual shows how we can use the location alias support to subset two members (DBP1 and DBP2) of the data sharing group (DBP0) for the MQ related workload (because these are the only two members that have the DB2- MQ support installed). As shown below, you need to define DB2 location alias names in the BSDS, as well as an additional port number.

Thus, to define the subset for the above requirement, each of the members would have the following definitions:

- DBP1: LOCATION=LOCDBP0,PORT=8000,RESPORT=8001,ALIAS=MQMBR:8100
- DBP2: LOCATION=LOCDBP0,PORT=8000,RESPORT=8001,ALIAS=MQMBR:8100
- DBP3: LOCATION=LOCDBP0,PORT=8000,RESPORT=8001

For DB2 Connect to access any of the subsets, one would define its node, db, and dcs db profiles as follows:

- db2 catalog tcpip node locdbp0 remote myhostname server 8000
- db2 catalog tcpip node mqmbr remote myhostname server 8100
- db2 catalog dcs db locdbp0 as locdbp0 parms ',,,,,sysplex'
- db2 catalog dcs db mqmbr as mqmbr parms ',,,,,sysplex'
- db2 catalog db locdbp0 as locdbp0 at node locdbp0 authentication server
- db2 catalog db mqmbr as mqmbr at node mqmbrs authentication server

Assuming that the subsetting support is installed and setup/activated, a DRDA application requester connection to an alias name comes into DDF as before. DDF analyzes the request. If the ALIAS name being referenced is being used to form a subset, and both the WLM registering and TCP/IP bind were successful, then DDF will ask WLM for the current "weighted" server list for the provided ALIAS name, and return it to the DRDA AR.

Since DDF can be stopped dynamically via the STOP DDF command or when a new system parameters module is loaded which specifies that MAXDBAT=0, then if the subsystem was previously successfully registered in a WLM sysplex routing services resource group, DDF will appropriately de-register the subsystem from all ALIAS resource groups.

Restriction: This enhancement is only provided for DRDA connections using TCP/IP. Support for VTAM/APPC and DB2 private protocol is not provided. In addition, to activate this subsetting feature, you must define a location alias with a TCP/IP alias port. The system that you are "subsetting" must be a DB2 data sharing group, of course.

Rollup Accounting Data Redbooks

Optionally accumulate accounting data for DDF and RRSAF threads

- New DSNZPARMs - ACCUMACC and ACCUMUID
- Can be deactivated (and activated) dynamically
  - To allow for more detailed monitoring when required

Accounting data accumulated by either, or combination of

- User ID
- Transaction name
- Workstation name

Reduces the need for CMSTAT=ACTIVE

- Reduces number of SMF records written

7.3 Rollup of DB2 accounting data for DDF and RRSAF threads

With WebSphere, WebLogic, and other e-business application servers or distributed applications that connect to DB2 for OS/390 via DDF, to optimize the use of DB2’s resources, it is recommended to use CMTSTAT=INACTIVE (DSNZPARM). This normally allows DB2 to separate the DBAT from the connection (at commit time) and reuse the DBAT to process somebody else’s work while your connection is inactive. This is called DB2 inactive connection support (sometimes wrongly called 'type 2 inactive' threads).

A side effect of using CMTSTAT=INACTIVE is that DB2 cuts an accounting record when the connection becomes inactive (that is normally on every COMMIT or ROLLBACK). If you are running high volume OLTP in this environment, the large volume of accounting records can be a problem, as you can end up flooding SMF, compromising your ability to do charge-back or do performance monitoring and tuning.

When using the RRS attach, you can run into a similar problem. WebSphere on z/OS drives the RRS signon interface on each new transaction, and DB2 cuts an accounting record when this signon happens. Some of these transactions are very short (for example, just one SELECT statement followed by a COMMIT), but still result in a DB2 accounting record being produced.

In Version 8, DB2 accounting data collection is enhanced to optionally accumulate accounting data for DDF and RRSAF threads.

Two new installation options are added to activate this behavior. The DB2 installation panel DSNTIPN - Tracing Panel, contains two new fields to accommodate for this enhancement:

- "DDF/RRSAF Accum". This new field specifies whether DB2 accounting data should be accumulated by end user for DDF and RRSAF threads. The related DSNZPARM parameter is ACCUMACC.
  - If NO is specified (the default), then DB2 continues to write (as in V7) an accounting record when a DDF thread is made inactive, or when signon occurs for an RRSAF thread.
  - If 2-65535 is specified, then DB2 writes an accounting record every 'n' occurrences of the "end user" on the thread, where 'n' is the number specified for this parameter. (The meaning of "end user" is discussed next).
- "Aggregation Fields". This new field specifies which aggregation criteria is to be used for DDF and RRS accounting record rollup. The corresponding DSNZPARM is ACCUMUID. This field can take a value from 0 to 6. Aggregation is based on the following three fields that have to be provided by the application (for example, WebSphere):
  - ID of the end user (QWHCEUID, VARCHAR 128). Note that the end user ID does not necessarily have to be the authorization ID that is used to connect to DB2, but it will be set to it by default (especially from DB2 Connect clients).
  - End user transaction name (QWHCEUTX, 32 bytes)
  - End user workstation name (QWHCEUWN, 18 bytes)

  These values can be set by DDF threads via "Server Connect" and "Set Client" (SQLESETI) calls, RRSAF threads via the RRSAF SIGN, AUTH SIGNON, and CONTEXT SIGNON functions, and as properties in Java programs when using the new Java Universal Driver.

  Aggregation is done based on any of the following fields of combinations thereof:
  - 0 : End user ID, AND end user transaction name, AND end user workstation name (null values accepted)
  - 1 : End user ID
  - 2 : End user transaction name
  - 3 : End user workstation name
  - 4 : End user ID AND end user transaction name (null values accepted)
  - 5 : End user ID AND end user workstation name (null values accepted)
  - 6 : End user transaction name AND end user workstation name (null values accepted)
  - 7 : End user ID, AND end user transaction name, and end user workstation name (all non-null)
  - 8 : End user ID AND end user transaction name (all non-null)
  - 9 : End user ID AND end user workstation name (all non-null)
  - 10 : End user transaction name AND end user workstation name (all non-null)

  The default value is 0 (zero). The ACCUMUID value is ignored if ACCUMACC=NO (no DDF/RRS rollup).

  Options 7-10 were introduced via the maintenance stream by APAR/PTF PQ90547/UQ95032. For example, given a thread with workstation name = "MYWORKSTATION" and end user ID = "null", the thread would qualify for rollup if ACCUMUID is set to 5, but would not qualify if ACCUMUID is set to 9. Threads that do not qualify for rollup write individual accounting records. In addition, any thread with *all* key values set to "null" will *not* qualify for rollup, and an individual accounting records is written. It is important to note that even for ACCUMUID values where null values are accepted, at least 1 of the key values must be non-null.

  For DDF/RRSAF rollup accounting, a "null" value is defined as a value that is exclusively a sequence of x'00' bytes or exclusively a sequence of x'40' bytes. If x'00' and x'40' bytes are intermixed for a given key value, this will be treated as a non-null value.

DB2 always creates separate rollup buckets for DDF and RRS threads so that rollup records will not intermix the two types of accounting records into a single rolled up accounting record, even when the aggregation fields are the same. The type of thread rolled into a given bucket can be identified by QWHCATYP.

If data accumulation is activated, then, when a DDF unit of recovery (UR) ends (end-commit or end-abort), or a RRSAF signon occurs, instead of immediately writing an accounting record at that time, DB2 adds the accounting values to the current values for this end user's use of the thread ("end user" is defined by the six criteria above). If the thread does not already have accumulated accounting values for this end user, then a new entry is created.

DB2 externalizes the end user's accumulated accounting data when the number of occurrences for this "end user" value reaches the threshold value specified in ACCUMACC.

Even when you specify a value between 2 and 65535 for ACCUMACC, DB2 may choose to write an accounting record prior to the nth occurrence of the "end user" in the following cases:

- When a storage threshold is reached for the accounting rollup blocks.
- When no updates have been made to the rollup block for 10 minutes, that is, the "end user" has not performed any activity for over 10 minutes that can be detected in accounting.

There are certain cases where detailed accounting data for the DDF and RRSAF threads is desired, such as detailed performance monitoring. The ACCUMMACC DSNZPARM can be dynamically altered to activate or deactivate accounting data aggregation on the fly.

Today, some installations avoid the extra overhead of cutting an accounting record every time a thread becomes inactive in an environment with very high transaction rates, by disabling thread pooling, and using CMTSTAT=ACTIVE. (If so, you should ensure that some kind of client-side pooling is active (such as WebSphere connection pooling), so that threads are kept open for the application connections, and the applications performs regular commits, but do not disconnect.) This enhancement to DB2 accounting will provide some relief in this area as well. Instead of writing an accounting record every time a thread gets pooled, an accounting record is only written after 'n' occurrences of ACCUMUID, where 'n' is the value of the new ACCUMACC DSNZPARM.

The DB2 statistics record IFCID 1 has been enhanced and contains the following additional fields to track rollup accounting information:

QWSDARTH Number of rollup accounting records written due to rollup threshold exceeded.

QWSDARSG Number of rollup accounting records written due to rollup accounting storage threshold exceeded.

QWSDARST Number of rollup accounting records written due to staleness threshold exceeded.

QWSDARIR Number of records failing to qualify for accounting rollup due to all rollup key fields are "null" or "null" values not being permitted.

DRDA Data Stream Encryption

Redbooks

DB2 V8 as a requester supports these authentication mechanisms

- Encrypted User ID and encrypted password
- Encrypted User ID and encrypted security-sensitive data
- Encrypted User ID, encrypted password, and encrypted security-sensitive data

DB2 V8 as a server supports these authentication mechanisms

- Encrypted User ID and password (already available in V7)
- Encrypted User ID and encrypted security-sensitive data
- Encrypted User ID, encrypted password, and encrypted security-sensitive data

Only if z/OS ICSF is installed, configured, and active

7.4 DRDA data stream encryption

As a server, DB2 z/OS and OS/390 Version 7 accepts encrypted userIDs and passwords (The userIDs and passwords are encrypted using 56-bit DES and a shared private key generated using the Diffie-Hellman distribution algorithm). However, DB2 z/OS V7 requesters always send the userIDs and passwords in clear text. Also, security-sensitive user data is always sent in the clear.

To achieve more effective security in a distributed computing environment, DB2 for z/OS Version 8 provides the ability to authenticate via encrypted userID, or encrypted userID and password, and provides support for encrypting security-sensitive data, based on the security option specified.

Prior to Version 8, DB2 for z/OS servers provided software support for the DES decryption by loading a required BSAFE service (licensed by IBM) into the distributed address space. DDF directly invoked the BSAFE service for userid-password decryption and Diffie-Hellman services.

In Version 8, the encrypted security mechanisms will use the z/OS Integrated Cryptographic Service Facility (ICSF) for encryption, decryption, and Diffie-Hellman services. If ICSF is not installed and configured properly, then DB2 will continue to use the existing BSAFE services, but only for the security mechanisms supported by DB2 servers in prior releases.

Two new DRDA security options are added to the SECURITY_OUT column of SYSIBM.IPNAMES table:

- The option 'D' implies that the userID and security-sensitive data are encrypted.
- The option 'E' implies that the userID, password, and security-sensitive data are encrypted.

The SECURITY_OUT column option 'P' (Password) is modified to encrypt userID and password, if the server supports encryption. If the server does not support encryption, then the userID and password will flow in the clear as before.

For data stream encryption, for performance reasons, we do not encrypt the entire network stream. Only security-sensitive user data items is encrypted. The following DRDA objects are encrypted:

- SQL statements that are being prepared, executed, or bound into an RDB package.
- SQL Program Variable Data consisting of input data to SQL statement during open or execute. This also includes a description of the data.
- SQL Data Reply Data consisting of output data from RDB processing of an SQL statement. It also includes a description of the data.
- Query Answer Set Data consisting of the answer set data resulting from a query.
- Input or output LOB data.
- Description of the data returned from the server during DESCRIBE.

The RDB name, package name, section, consistency token, etc. are not encrypted. The SQLCA is not encrypted.

DB2 Version 8, as a DRDA application requester, supports the following DRDA authentication mechanisms, if TCP/IP protocols are used:

- Encrypted User ID and encrypted password
- Encrypted User ID and encrypted security-sensitive data
- Encrypted User ID, encrypted password, and encrypted security-sensitive data

DB2 Version 8, as a DRDA application server, supports the following DRDA authentication mechanisms if TCP/IP protocols are used:

- Encrypted User ID and encrypted password
- Encrypted User ID and encrypted security-sensitive data
- Encrypted User ID, encrypted password, and encrypted security-sensitive data.

Encryption mechanism

During connect processing, requester and server connection keys are exchanged and a shared private key is generated. The connection keys and the shared private key are generated using the standard Diffie Hellman distribution algorithm. The 56-bit encryption key is generated from the shared private key.

Integrated Cryptographic Services Facility

The Integrated Cryptographic Services Facility is a software element of z/OS that works with a required hardware cryptographic feature and the Security Server (RACF) to provide secure, high-speed cryptographic services in the z/OS environment. ICSF supports cryptography by IBM's Common Cryptographic Architecture (CCA) which is based on the DES algorithm. See the *z/OS ICSF Administrator's Guide*, SA22-7521, for more information.

Restrictions:

- DRDA data stream encryption is only supported when using TCP/IP connections. The function will not be supported over SNA connections.
- A DB2 for z/OS requester can support the DRDA data stream encryption security mechanisms only if z/OS ICSF is installed, configured, and active. If ICSF is not installed and configured properly, then the DB2 for z/OS server will use the existing BSAFE services only for the encryption security mechanisms supported in prior releases, and the DB2 z/OS server will not support data stream encryption.
- In addition, this function is only available in new-function mode.

Other Network Enhancements

Redbooks

Terminology change

- "Type 1 Inactive Thread" is now "Inactive DBAT"
- "Type 2 Inactive Thread" is now "Inactive Connection"
- "Pooled DBAT" is a DBAT that is not associated with a connection

VTAM conversation allocation requests can now timeout (like TCP/IP)

DRDA query blocks larger than 32K

- Does not apply to DB2 Private Protocol

Add DDF accounting string to RRSAF

New-function mode SQL cannot be used in DB2 Private Protocol

Write accounting record if KEEPDYNAMIC(YES)

DB2 Universal Driver for SQLJ and JDBC

-DISPLAY LOCATION command change

7.5 Other network computing enhancements

In this section we list a number of other V8 enhancements that are related to network computing.

7.5.1 Terminology changes for inactive thread support

DB2 Version 8 uses the term *"Inactive DBAT"* instead of "Type 1 Inactive Thread", and uses the term *"Inactive Connection"* instead of "Type 2 Inactive Thread". The DBAT that is decoupled from the connection, when the connection becomes inactive, is called a *"pooled DBAT"*, as it returns to a pool of DBATs where it can be reused by other connections. These terms are much more descriptive of the actual status of the threads and connection, and brings the terminology more in line with DB2 UDB on other platforms.

The DB2 Version 8 installation clist panels and DB2 documentation reflect this new terminology.

7.5.2 Time-out for allocate conversation requests

In DB2 for z/OS Version 8, DDF will search for threads waiting for a VTAM allocate conversation request to complete every three minutes. When an allocate request has waited for a session for more than three minutes, DDF issues a deallocate abend conversation to VTAM to force VTAM to abnormally terminate the request. The remote SQL statement fails with a SQLCODE of -904 with the reason code of 00D31033.

This is an indicator that there may be a network problem and the network administrator should be notified of the communication failure.

7.5.3 Larger query blocks

A query block is a group of rows that fit in to a (query) block and is sent as a single network message. The default query block size used by DB2 in Version 7 is 32K. The number of rows that DB2 puts into a query block depends on the row length, as well as the OPTIMIZE FOR n ROWS clause. Blocking is used by both DB2 Private Protocol and DRDA. When blocking is not used, DB2 will send a single row across the network.

To support the larger network bandwidths and customer requirements to control the amount of data returned on each network request, the DRDA architecture has been enhanced to allow query blocks up to 2M. This way, the requester can ask for a query block size of up to 2M. This allows a requester to better manage and control the size of blocks returned from DB2. DB2 as a requester continues to always request a block size of 32K, but as a server it can support any block size. DB2 Connect Version 8, by default, also continues to use its standard RQRIOBLK size of 32767bytes.

7.5.4 Limitations on SQL statements allowed in DB2 Private Protocol

It is the DB2 direction to sunset the DB2 distributed Private Protocol and encourage customers to use the Distributed Relational Database Architecture™ (DRDA) protocol. The reasons are many, the most obvious ones are that DB2 Private Protocol is single platform only, the available functions are limited, not easily portable, and the performance is not on par with the applications that use DRDA protocol.

In Version 8, DB2 for z/OS limits the kind of SQL statements allowed in an application that uses DB2 Private Protocol, to SQL statements supported in V8 compatibility mode. This is equivalent to limiting the SQL functionality to DB2 Version 7. Using SQL statements that are only supported in new function mode, return an SQLCODE -142, when the application is using DB2 Private Protocol.

7.5.5 Write accounting records at the server when using KEEPDYNAMIC(YES)

In V7 there are a number of reasons that indicate to DB2 that it is time to produce a DB2 accounting record, for example, at thread deallocation time, or when a connection becomes inactive (DDF). When using inactive connections (CMTSTAT=INACTIVE), a connection cannot become inactive, and its associated DBAT pooled at commit time, when it touches a package that is bound with KEEPDYNAMIC(YES). There is also no accounting record produced, and the WLM enclave, that governs the priority of the work that thread is doing, remains active.

This is not an ideal situation for two reasons:

- An accounting record is only produced when the thread is deallocated and potentially contains a large number of executions of the program. This makes monitoring and charge back accounting difficult. Users need an accounting record written at every transaction to give them better granularity for monitoring and charge back.
- The WLM enclave also remains active for a long time, which means that period goals are not effective. Executions that happen after the thread has been running for some long time will run in the lowest period, and do not get the service they need.

The intent of this enhancement is not to make the connection become inactive (Type 2 inactive thread). Since there are KEEPDYNAMIC sections that must be maintained and remain associated with the client application's connection, the connection cannot become inactive.

However, with this enhancement, when a UOW completes (after commit or rollback) in a DRDA application that has touched a package that is bound with KEEPDYNAMIC(YES), DB2 will now produce an accounting record and delete the WLM enclave, as if the connection had become inactive, **IF** there is nothing else that would otherwise prevent the connection from becoming inactive (like an open held cursor, or a declared temporary table that has not been dropped). Private Protocol connections will not be affected.

When another request arrives from the client system, a new WLM enclave is created and a new accounting interval is started.

With this enhancement, DB2 “pretends” the connection went inactive, even if there are KEEPDYNAMIC sections present, but there are no open held cursors or active declared temporary tables.

Users will notice that more accounting records are produced and more enclaves are being established, where they were not before. When classifying DDF work in WLM, the use of period goals for this type of applications can now be considered.

7.5.6 Add DDF accounting string to RRSAF

A new accounting string parameter is provided on the RRS SIGNON, AUTH SIGNON, CONTEXT SIGNON and the SET_CLIENT_ID function. This new parameter can be specified with the existing client user ID, application program name, workstation name, and accounting token parameters. This RRSAF function sets the character strings that are passed to DB2 when the next SQL request is made.

The effect of this new parameter is that accounting strings can be set using an inexpensive RRSAF function, and are available on the next SQL call.

7.5.7 DB2 UDB Universal Driver for SQLJ and JDBC

DB2 UDB for Multiplatforms Version 8 was the first member of the DB2 Family to introduce the new JDBC driver, called the IBM DB2 Universal Driver for SQLJ and JDBC. This new Java driver architecture is the future basis of all DB2-related Java efforts. It supports a so-called JDBC type 2 and type 4 driver.

These drivers are currently supported on DB2 UDB for Linux, Unix, and Windows and DB2 for z/OS Version 8. They are also available for DB2 for z/OS and OS/390 Version 7, through the maintenance stream via UQ85607.

Please refer to 5.1, “DB2 Universal Driver for SQLJ and JDBC” on page 389 for more information.

-DISPLAY LOCATION

Redbooks

Prior to Version 8

- Displays details for ALL Locations
  - -DISPLAY LOCATION
  - -DISPLAY LOCATION()
  - -DISPLAY LOCATION(*)

DB2 Version 8

- Now behaves the same as -DISPLAY DATABASE command
- -DISPLAY LOCATION()
  - Fails with message DSN9010I

7.5.8 -DISPLAY LOCATION command

In DB2, prior to Version 3, the -DISPLAY LOCATION command allowed no parameters (-DISPLAY LOCATION), and displayed all locations. In Version 3, the -DISPLAY LOCATION command was enhanced to accept parameters and the DETAIL keyword was added.

When this enhancement was introduced, the idea was that providing a blank parameter, -DISPLAY LOCATION(), should provide the same output as before the parameter was introduced. That is “-DISPLAY LOCATION()” should behave as “-DISPLAY LOCATION”. Hence, -DISPLAY LOCATION, -DISPLAY LOCATION(), and -DISPLAY LOCATION(*) display all locations, whereas adding a specific parameter (-DISPLAY LOCATION (WTSCPOK) displays only matching locations.

However, this behavior of -DISPLAY LOCATION with an empty parameter is different from the behavior of -DISPLAY DATABASE with an empty parameter. If you do a -DISPLAY DB() SPACENAME(), the command fails with message DSN9010I.

In an effort to make all commands behave in a more predicable and similar manner, DB2 Version 8 changes the semantics of the -DISPLAY LOCATION command with an empty parameter. -DISPLAY LOCATION() will behave the same way as -DISPLAY DATABASE(). Both commands will fail with message DSN9010I.

8

Application enablement

List of Topics

Redbooks

General stored procedure and UDF enhancements

- MAXFAILURES
- Better WLM resource management

Deprecation of some stored procedure features

- No more COMPJAVA stored procedures
- No new DB2 managed stored procedures

SQL stored procedure enhancements

DB2 V8 Development Center integration

Implicit RRSAF connections

New CURRENT PACKAGE PATH and SCHEMA special register

ODBC for USS enhancements

In this chapter we discuss the DB2 for z/OS Version 8 enhancements that are related to applications enablement. There are numerous enhancements in this area, and they usually fit into a number of different categories. We address some specific topics in this chapter and provide pointers to other chapters that discuss related topics as well.

Stored Procedure and UDF Enhancements

Redbooks

Specify maximum number of failures at stored routine level

- More granular control over stored procedure and user-defined function failure management
- User can specify for each stored procedure or user-defined function how many times a routine can experience failures before it is stopped
  - Failure is a program exception or unusual termination
- Specified at routine level, instead of subsystem level (MAX ABEND COUNT - DSNZPARM STORMXAB)
- Specified on CREATE/ALTER FUNCTION/PROCEDURE
  - **STOP AFTER SYSTEM DEFAULT FAILURES**
  - **STOP AFTER n FAILURES**
  - **CONTINUE AFTER FAILURE**
- Does not apply to sourced and SQL scalar UDFs

8.1 Stored procedure and UDF enhancements

In this section we use the term *stored routines*. This signifies both stored procedures and user-defined functions (UDFs).

8.1.1 Specify maximum failures at stored routine level

In V7, you have a DB2 DSNZPARM called STORMXAB that allows you to specify the number of times a stored procedure or an invocation of a user-defined function is allowed to terminate abnormally, after which SQL CALL statements for the stored procedure or user-defined function are rejected. As it is a DSNZPARM, the value applies to all stored routines.

In V8, you can specify that a stored routine is to be put in a stopped state after some number of failures, at the stored routine level. Next we list the SQL statement options you can use to do this.

STOP AFTER nn FAILURES

Specifies that this routine should be placed in a stopped state after nn failures. The value nn should be an integer from 1 to 32767.

STOP AFTER SYSTEM DEFAULT FAILURES

Specifies that this routine should be placed in a stopped state after the number of failures indicated by the system parameter MAX ABEND COUNT (DSNZPARM STORMXAB). This is the default.

CONTINUE AFTER FAILURE
Specifies that this routine should not be placed in a stopped state after any failure.

These options must not be specified for SQL functions, or sourced functions (SQLSTATE 42849, SQLCODE -20102).

To support this enhancement, a new column MAX_FAILURE is added to the SYSIBM.SYSROUTINES catalog table, where:

nn Specifies the allowable failures for this routine before it is stopped. The value *nn* can be an integer from 1 to 32767.

-1 Indicates that the DB2 DSNZPARM STORMXAB is used.

0 (zero) If zero is specified, the routine will never be stopped.

8.1.2 Adding the number of failures to DISPLAY command output

The output of the DISPLAY FUNCTION SPECIFIC and DISPLAY PROCEDURE commands is changed to display the number of times the execution of a stored routine failed. This makes it easier to monitor the well-being of your stored routines environment. A sample command result is shown in Example 8-1.

Example 8-1 -DIS PROC output with new FAIL column

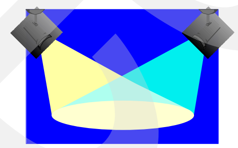
```
DSNX940I  -DB8A DSNX9DIS DISPLAY PROCEDURE REPORT FOLLOWS -

 ------- SCHEMA=SYSPROC
 PROCEDURE          STATUS ACTIVE QUED MAXQ TIMEOUT FAIL WLM_ENV
 DSNWZP
                    STARTED     0    0    1       0    0 DB8AWLM1
 DSNACCMG
                    STARTED     0    0    1       0    0 DB8AWLM1
 DSNUTILS
                    STARTED     0    0    1       0    0 WLMENV1
 DSNX9DIS DISPLAY PROCEDURE REPORT COMPLETE
 DSN9022I  -DB8A DSNX9COM '-DISPLAY PROC' NORMAL COMPLETION
```

The FAIL column indicates the number of times a stored routine has failed. DB2 resets this value to 0 each time you execute the START FUNCTION or START PROCEDURE command.

Stored Procedure and UDF Enhancements -2

Better WLM resource management for stored routines

- Better resource management by exploiting capabilities of z/OS workload manager
- WLM determines appropriate resource utilization and recommends changes in number of tasks operating in stored procedure address space
  - TCBs are added/removed based on WLM recommendation
  - Attaching a new TCB is much cheaper that starting a new address space
  - MAXTCB parm is used as a maximum
- Recommendation: Specify a fairly large number for MAXTCB and let WLM do the rest



8.1.3 Better WLM resource management for stored routines

DB2 V8 is enhanced to exploit Workload Manager (WLM) functions that will allow System Resource Manager and Workload Manager to determine appropriate resource utilization and recommend changes in the number of tasks operating in a WLM managed stored procedure address space. DB2 will add or delete tasks based on WLM's recommendations.

The NUMTCB parameter is provided to Workload Manager as a maximum task limit. We recommend that the customer specify a reasonable number in NUMTCB, except when 1 is required. For a full discussion of NUMTCB, see *DB2 for z/OS Stored Procedures: Through the CALL and Beyond*, SG24-7083.

Note that V8 introduces an additional DSNZPARM related to stored procedures, called MAX_ST_PROC. It specifies the maximum number of "active" stored procedures a thread is allowed to have. The default is 2000. If you have more than allowed by this DSNZPARM, an SQLCODE -904 is returned.

Other Stored Procedure Related Changes

Redbooks

LANGUAGE COMPJAVA stored procedures

- COMPJAVA uses HPJ (High Performance Java compiler)
- VA Java no longer support compiled Java link library files
- No longer supported in V8
- Use LANGUAGE JAVA instead
  - With better performance as well

Deprecation of DB2-established stored procedures

- Cannot create new DB2-established stored procedure in V8
  - Remove the 'NO WLM ENVIRONMENT' option SQLCODE -199
- Existing DB2-established stored procs continue to run
- After ALTER PROC to WLM-managed, you cannot ALTER it back
- Recommendation: Migrate all your stored procs to be WLM managed

8.2 Other stored procedure related changes

In this section we discuss two changes that may affect your existing stored procedures. In V8:

- COMPJAVA stored procedures are no longer supported
- You cannot create new DB2-established stored procedures, or alter stored procedures to become DB2-established stored procedures.

8.2.1 LANGUAGE COMPJAVA stored procedures no longer supported

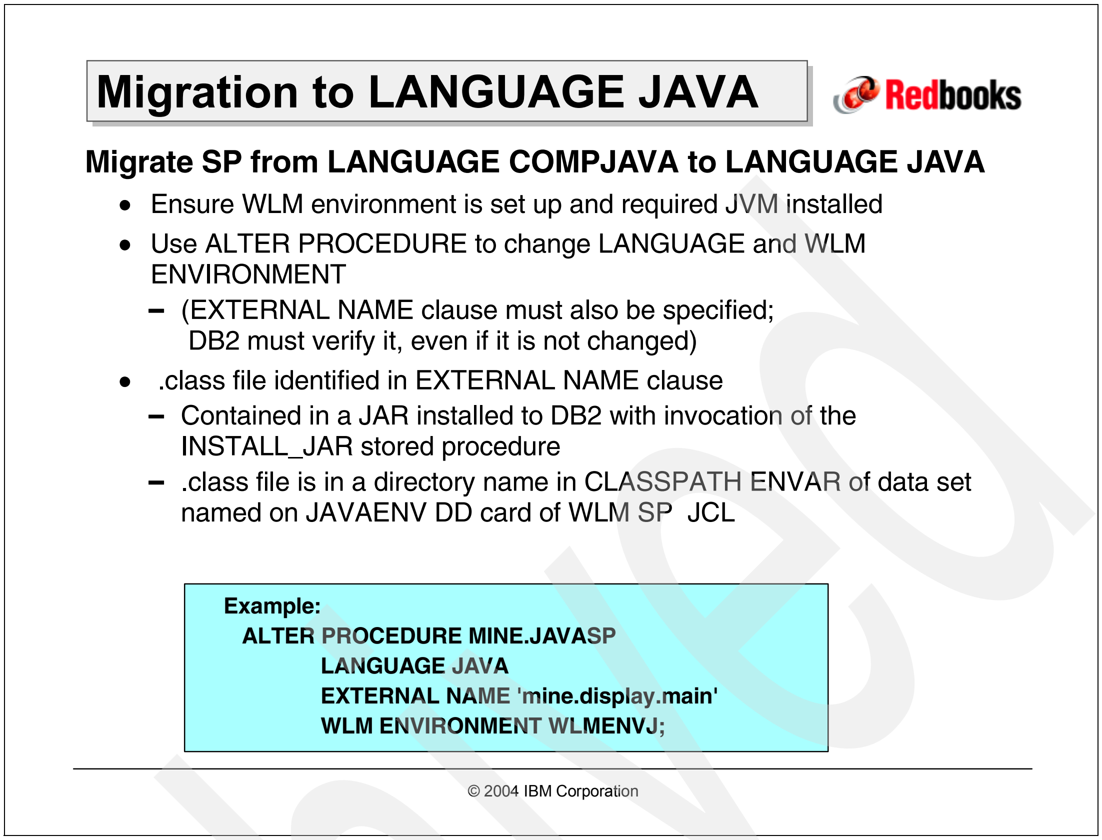
As the HPJ (High Performance Java) compiler is no longer supported, DB2 can no longer support LANGUAGE COMPJAVA stored procedures, and V8 does not allow you to run or create LANGUAGE COMPJAVA stored procedures.

8.2.2 Deprecation of DB2-established stored procedures

DB2 Version 8 removed the "NO WLM ENVIRONMENT" option on the CREATE PROCEDURE statement. This means that you can no longer create any new DB2-established (sometimes also called DB2-managed) stored procedures in V8.

However, if you have existing DB2-established stored procedures, they continue to run in V8. You just cannot create any new ones (or alter WLM-managed stored procedures back to DB2-managed).

Therefore the recommendation is to convert to WLM-managed stored procedures as soon as possible.



8.2.3 Migrating to LANGUAGE JAVA stored procedures

As mentioned earlier, after migrating to Version 8, you can no longer define or run COMPJAVA stored procedures. The only type of Java stored procedures that are supported are LANGUAGE JAVA (interpreted Java) stored procedures. You can convert LANGUAGE COMPJAVA stored procedures to LANGUAGE JAVA by following these steps:

1. Ensure that the WLM environment is configured and that the required JVM is installed.
2. Use ALTER PROCEDURE to change the LANGUAGE and the WLM ENVIRONMENT. The EXTERNAL NAME clause must also be specified. For example:

```
ALTER PROCEDURE MINE.JAVASP LANGUAGE JAVA EXTERNAL NAME 'mine.display.main' WLM
ENVIRONMENT WLMENVJ;
```

3. Ensure that the `.class` file that is identified in the EXTERNAL NAME clause of the ALTER PROCEDURE is present in one of the following places:
   - In a JAR that was installed to DB2 by an invocation of the INSTALL_JAR stored procedure
   - In a directory in the CLASSPATH ENVAR of the data set that is named on the JAVAENV DD statement of the WLM stored procedures address space JCL

Migrating to WLM-managed SP

Define JCL procedures for the WLM-managed SP address spaces

Define application environments in WLM for these SPs

Stop currently running DB2-managed SP (-STO PROC(...))

ALTER PROCEDURE ... WLM ENVIRONMENT ...

Relink your stored procedures with the RRS attach DSNRLI

(Manually start WLM-managed SP address space in case you are still running in WLM compatibility mode)

- Remember that z/OS V1.3 requires WLM goal mode

Restart the procedure -STA PROC(...)

8.2.4 Migrating to WLM-managed stored procedures

If you have existing stored procedures that use DB2-established address spaces, you need to move as many as possible to a WLM environment. To move stored procedures from a DB2-established environment to a WLM-established environment, follow these steps:

1. Define JCL procedures for the WLM stored procedures address spaces. Member DSNTIJMV of data set DSNxxx.SDSNSAMP contains sample JCL procedures for starting WLM-established address spaces.
2. Define WLM application environments for groups of stored procedures and associate a JCL startup procedure with each application environment.
3. Enter the DB2 command STOP PROCEDURE(`proc-name`) to stop the DB2-established stored procedure that you are about to change.
4. For each stored procedure, execute ALTER PROCEDURE with the WLM ENVIRONMENT parameter to specify the name of the application environment.
5. WLM managed stored procedures use Resource Recovery Services attachment facility (RRSAF). Therefore you must relink all of your existing DB2-established stored procedures with DSNRLI, the RRSAF language interface module. See the linkage editor control statements below for an example:

```
//SYSLIN DD *
  ENTRY MYSTPROC
  REPLACE DSNALI(DSNRLI)
  INCLUDE SYSLMOD(MYSTPROC)
  NAME MYSTPROC(R)
```

6. If WLM is operating in compatibility mode, start the new WLM-established stored procedures address spaces by using this z/OS command:

```
START address-space-name
```

Note: If you make these changes under DB2 V7, and z/OS is at V1.2 or below and is running in WLM compatibility mode, you must manually start the new address spaces. If you do this conversion when you are already running in V8, you will be in goal mode, because DB2 V8 requires z/OS V1.3 (or above) and z/OS V1.3 only runs in goal mode.

7. If WLM is operating in goal mode, the address spaces start automatically.
8. Restart the stored procedure in DB2, using the -START PROCEDURE(`proc-name`) command.

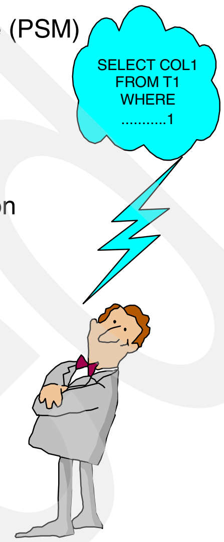
SQL Stored Procedure Enhancements

Benefits .....

- Enhances usability and power of SQL procedure language (PSM)
- DB2 Family compatibility
- Conforms to SQL standards

V8 Enhancements

- New SQL procedure statements to return status information from SQL procedure to the calling application
  - RETURN statement
  - SIGNAL/RESIGNAL support
  - GET DIAGNOSTICS
  - ITERATE
  - Enhanced LOB and variable support
- With V8 SQL statement limit extended to 2MB
  - SQL Procedure must be completely stated in a single SQL statement
- Integrated debugger



8.3 SQL stored procedure enhancements

DB2 Version 8 introduces many enhancements to SQL stored procedures, sometimes called PSM (Persistent Stored Modules). Numerous enhancements were introduced to enhance DB2 Family compatibility and conformance to the SQL standards. This will help people to port existing SQL stored procedures to the zSeries platform.

These enhancements are discussed in more detail in the following sections:

- RETURN statement
- SIGNAL/RESIGNAL support
- GET DIAGNOSTICS support
- ITERATE statement support
- Enhanced LOB and variable support
- Long SQL statements (up to 2 MB)
- Support for using the integrated debugger with SQL stored procedures

RETURN Statement

Redbooks

Before Version 8, two methods available for returning status information from an SQL procedure

- Define extra parameter for status -- CREATE PROCEDURE statement includes an additional parameter (OUT or INOUT) for status information as part of parameter list. Invoking application must provide this additional parameter on CALL statement.
- Leave conditions (errors or warnings) unhandled -- any conditions not handled in procedure are returned to caller in SQLCA

V8 provides an additional method to return status information to caller

- Return an integer value to the invoking application via new RETURN statement that you can code in the SQL procedure
- Compatibility with other platforms
- GET DIAGNOSTICS statement (extended) to return status information from a RETURN statement of an SQL procedure

8.3.1 RETURN statement

Before V8 there were two methods for returning error information from an SQL stored procedure.

The first method was to use an additional parameter (OUT or INOUT) that gets passed back from the stored procedure to the caller. Invoking applications must be aware of the existence of this extra parameter.

The second approach was to leave error or warning conditions unhandled. Any conditions not handled by the procedure are returned to the caller via the SQLCA, and the calling program must deal with the problem. (This support was introduced via PQ56323 for V6 and V7. Before this APAR, unhandled errors on SQL statements issued inside SQL procedures were not returned to the caller.)

RETURN Statement -2

For SQL procedure, it returns an integer status value to the invoking application

```
Example:  RETURN  value of 0 if successful, and -2 if not
     BEGIN
       .........
     GOTO FAIL;

     SUCCESS:  RETURN;
     FAIL:        RETURN -2;
     END
```

The caller may access the value using

- RETURN_STATUS of GET DIAGNOSTICS
- Directly from SQLCA in SQLERRD[0]
- Return value parameter marker in the escape clause CALL syntax in CLI or ODBC applications

Additional GET DIAGNOSTICS information

- MESSAGE_TEXT clause provides the current message text
- MESSAGE_LENGTH clause used to obtain length of the current message text

For SQL scalar functions, it returns the result of the function (V7)

Using the RETURN statement for the SQL procedure status

DB2 Version 8 introduces a third method: the use of the RETURN statement. The statement is also available on other platforms.

You can use the RETURN statement in an SQL procedure to return an integer status value. If you include a RETURN statement, DB2 sets the SQLCODE in the SQLCA to 0 and the caller must retrieve the return status of the procedure in either of the following ways:

- By using the RETURN_STATUS item of GET DIAGNOSTICS statement to retrieve the return value of the RETURN statement.
- By retrieving SQLERRD[0] of the SQLCA, which contains the return value of the RETURN statement. (SQLERRD[0] is used in C, whereas SQLERRD(1) is used in Assembler, COBOL and PL/I to indicate the same field in the SQLCA.)

If you do not include a RETURN statement in an SQL procedure, by default, DB2 sets the return status to 0 for an SQLCODE that is 0 or positive and sets it to -1 for a negative SQLCODE.

The MESSAGE_TEXT clause of the GET DIAGNOSTICS can be used to obtain additional information about the current message text (or the SQLERRDC in the SQLCA).

Note that the RETURN statement is also used in SQL scalar functions. In that case, it is not used to pass back a return code. It is used to pass back the result of the SQL scalar function.

SIGNAL/RESIGNAL Statement

Redbooks

Allow notification of an application when an exception occurs in SQL procedure so that program can take action (i.e., terminate)

- Other DB2 platforms support SIGNAL/RESIGNAL statements within SQL procedures
- Part of the SQL standard

Before Version 8

- SQL stored procedure cannot signal to the calling program that an exception occurred

With Version 8

- SIGNAL/RESIGNAL statements allow SQL procedure to specify a specific SQLSTATE and message text to raise a condition within the SQL procedure
- Existing support for SIGNAL statement within trigger body unchanged

8.3.2 SIGNAL/RESIGNAL support in SQL procedures

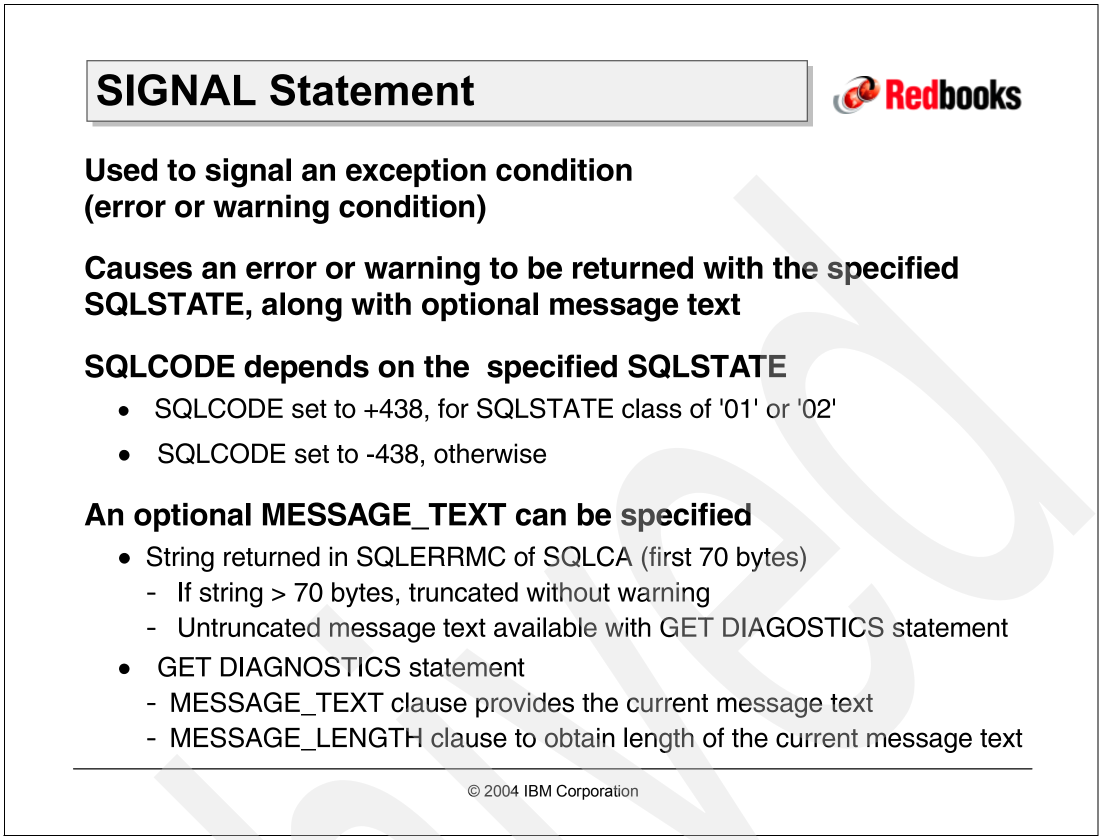
Before V8, SQL procedures did not have an easy way to signal the calling program with a specific message and SQLSTATE that an error occurred (unless you use a separate parameter, of course).

DB2 V8 allows the calling application to be notified with a specific SQLSTATE and provide a message text, when an error occurs in the SQL procedure. To enable this feature, two new statements are introduced for the SQL procedure language:

- SIGNAL
- RESIGNAL

Both statements are explained in more detail in the following sections.

> **Note:** The existing support for the SIGNAL statement inside a trigger body is unchanged.



8.3.3 Using the SIGNAL statement to raise a condition

In DB2 for z/OS Version 8, you can use the SIGNAL statement anywhere in a SQL procedure to set a specific SQLSTATE along with an optional message text.

In the following code fragment, DB2 generates an SQLSTATE 23503 (SQLCODE -530) when you attempt to insert an order for a missing customer. You can intercept the error in a handler, and pass a more meaningful message back to the caller as the code fragment in Example 8-2 shows.

Example 8-2 Using SIGNAL

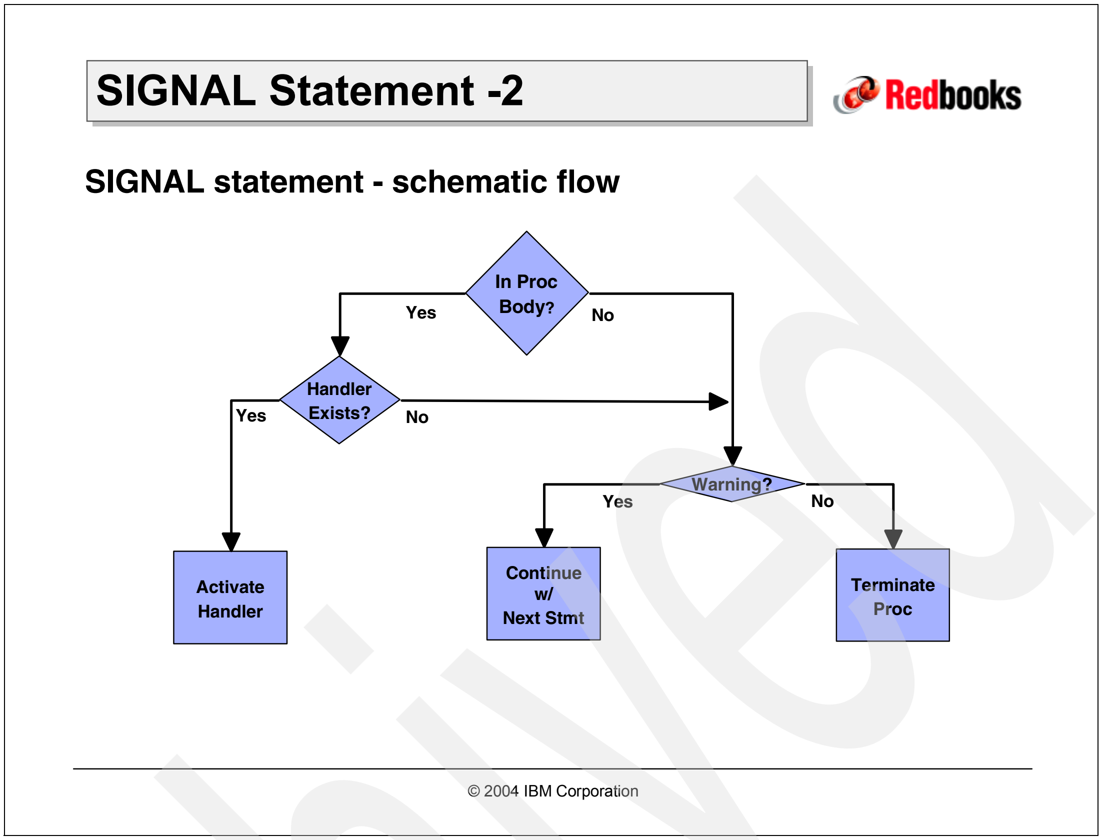
```
DECLARE EXIT HANDLER FOR SQLSTATE VALUE ‘23503’
   SIGNAL SQLSTATE ‘75001’
      SET MESSAGE_TEXT = ‘Customer is unknown’;
INSERT INTO ORDERS (....)
      VALUES (....);
```

As shown in the example above, instead of returning a general SQLSTATE 23503, the procedure returns a specific SQLSTATE 75001, and a more meaningful message indicating that the customer is not found. The provided message text is returned in the SQLCA SQLERRMC field (up to 70 bytes), or the full message can be obtained by using the MESSAGE_TEXT clause on the GET DIAGNOSTICS statement.

The capability to set a specific SQLSTATE in case of an error is useful for packaged applications such as DB2 extenders, which have their own SQLSTATEs that they want to return to the invoking application.

Note that when you use SIGNAL (or RESIGNAL) to set the SQLSTATE, the value of SQLCODE returned to the invoking application is a constant (you cannot set it yourself). It is based on the class code (first 2 bytes) of the SQLSTATE:

- Class code 00 is not allowed
- Class code 01 or 02 causes SQLCODE +438
- All other class codes cause SQLCODE -438



Schematic flow when using the SIGNAL statement

The figure above shows the schematic flow of what happens depending on where the SIGNAL statement appears in the SQL procedure and whether it signals an error or a warning:

- If the SIGNAL is in the procedure body, but not part of a handler, and a handler exists, the handler is activated.
- If the SIGNAL is in the procedure body and there is no handler defined for this condition, then the procedure:
  - Continues if a warning is signaled
  - Exits if an exception is signaled
- If SIGNAL is part of a handler, then the procedure:
  - Continues if a warning is signaled
  - Exits if exception is signaled

SIGNAL Statement -3

Example:

```
CREATE PROCEDURE SUBMIT_ORDER
      (IN ONUM INTEGER, IN CNUM INTEGER, IN PNUM INTEGER,
       IN QNUM INTEGER)
      LANGUAGE SQL
      MODIFIES SQL DATA

    BEGIN
       DECLARE EXIT HANDLER FOR SQLSTATE VALUE '23503'
         SIGNAL SQLSTATE '75002'
            SET MESSAGE_TEXT = 'Customer number is not known';
       INSERT INTO ORDERS (ORDERNO, CUSTNO, PARTNO, QUANTITY)
              VALUES (ONUM, CNUM, PNUM, QNUM);
    END
```


Sample usage of the SIGNAL statement

In the figure above, we provide a small example of the usage of the SIGNAL statement. Assume that a PK-FK relationship exists between ORDERS and CUSTOMERS. SQLSTATE 23503 indicates that there is no entry in the PK table (customers) for the FK table (ORDERS) row that we are inserting. Instead of returning SQLSTATE 23503, we return an SQLSTATE 75002, and a helpful message, “Customer number is not known”.

> **Note:** The SQLSTATE ‘value’ can be both an sqlstate-string-constant, as in the visual above, or an SQL variable-name, declared within the compound-statement. The value of the SQL variable has to be a valid 5-byte SQLSTATE.

RESIGNAL Statement

Redbooks

Used within a handler to resignal an exception condition

Causes an error or warning to be returned with the specified SQLSTATE, along with optional message text

Set SQLSTATE to a specific value

- SQLCODE set to +438 if SQLSTATE class is '01' or '02'
- SQLCODE set to -438 otherwise

An optional MESSAGE_TEXT can be specified

- String returned in SQLERRMC of SQLCA (first 70 bytes)
  - If string > 70 bytes, truncated without warning
  - Untruncated message text available with GET DIAGOSTICS statement
- GET DIAGNOSTICS statement
  - MESSAGE_TEXT clause provides the current message text
  - MESSAGE_LENGTH clause to obtain length of the current message text

8.3.4 The RESIGNAL statement

The RESIGNAL statement is (only) used inside a handler. It can be used to change a previously encountered SQLSTATE into a new one (that is more meaningful in the context, or is specific to a certain product).You can use the RESIGNAL command within the body of a handler as shown in Example 8-3.

Example 8-3 Using RESIGNAL

```
DECLARE OVERFLOW CONDITION FOR SQLSTATE VALUE ‘22003’;
DECLARE EXIT HANDLER FOR OVERFLOW
   RESIGNAL SQLSTATE ‘22375’
   SET MESSAGE_TEXT ‘Attempt to divide by zero’;
```

Note that when you use RESIGNAL (or SIGNAL) to set the SQLSTATE, the value of SQLCODE returned to the invoking application is a constant (you cannot set it) based on the class code (first 2 bytes) of the SQLSTATE:

- Class code 00 is not allowed
- Class code 00 or 01 causes SQLCODE +438
- All other class codes cause SQLCODE -438

As with the SIGNAL statement, you can also provide a message text. The provided message text is returned in the SQLCA SQLERRMC field (up to 70 bytes), or the full message can be obtained by using the MESSAGE_TEXT clause on the GET DIAGNOSTICS statement.

RESIGNAL Statement -2

Redbooks

Example:

```
CREATE PROCEDURE divide (  IN numerator INTEGER,
                           IN denominator INTEGER,
                           OUT divide_result INTEGER)
     LANGUAGE SQL CONTAINS SQL
BEGIN
  DECLARE overflow CONDITION FOR SQLSTATE '22003';
  DECLARE EXIT HANDLER FOR overflow
    RESIGNAL SQLSTATE '22375';
  IF denominator = 0 THEN
    SIGNAL overflow;
  ELSE
    SET divide_result = numerator / denominator;
  END IF;
END
```


8.3.5 Example using the RESIGNAL statement

In the visual above, we use the RESIGNAL statement in an EXIT handler. We can also use it in a CONTINUE handler.

Note: The SQLSTATE 'value' can be both an sqlstate-string-constant, as in the visual above, or an SQL variable-name, declared within the compound-statement. The value of the SQL variable has to be a valid 5-byte SQLSTATE.

The full power of the RESIGNAL statement comes to its rights when using multiple nested compound statements in your SQL procedure. Error codes at a deeper nesting level can be analyzed and changed (RESIGNAL) at a higher level. For more elaborate examples, see *DB2 SQL Procedural Language for Linux, UNIX, and Windows*, ISBN 0-13-100772-6.

ITERATE Statement

Redbooks

ITERATE statement now also supported in DB2 for z/OS (V8)

- Already supported in DB2 for iSeries and DB2 for LUW

ITERATE statement causes the program to return to the beginning of a labeled loop

label **you specify must reference a FOR, LOOP, REPEAT or WHILE statement**

ITERATE is now a reserved word in SQL statements

Example:

```
ins_loop:
LOOP
   FETCH hv_dept...
   IF hv_dept ^='D11' THEN
     ITERATE ins_loop;
   ELSEIF ...
```


8.3.6 ITERATE statement support in SQL procedures

Example 8-4 shows the use of the ITERATE statement. An ITERATE statement causes the flow of control to be passed back to the top of the LOOP statement, so you do not have to use a GOTO statement instead.

Example 8-4 Using the ITERATE statement

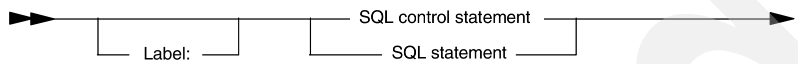
```
CREATE PROCEDURE ITERATOR ()
   LANGUAGE SQL MODIFIES SQL DATA
BEGIN
   DECLARE v_dept CHAR(3);
   DECLARE v_deptname VARCHAR(29);
   DECLARE v_admdept CHAR(3);
   DECLARE at_end INTEGER DEFAULT 0;
   DECLARE not_found CONDITION FOR SQLSTATE 02000 ;
   DECLARE c1 CURSOR FOR
           SELECT deptno,deptname,admrdept FROM department ORDER BY deptno;
   DECLARE CONTINUE HANDLER FOR not_found SET at_end = 1;
   OPEN c1;
   ins_loop:
   LOOP
      FETCH c1 INTO v_dept, v_deptname, v_admdept;
      IF at_end = 1 THEN LEAVE ins_loop;
       ELSEIF v_dept = 'D11' THEN ITERATE ins_loop;
      END IF;
      INSERT INTO department (deptno,deptname,admrdept)
                       VALUES('NEW', v_deptname, v_admdept);
   END LOOP;
   CLOSE c1;
END
```

Enhanced label, LOB and variable Support

Redbooks

In DB2 for z/OS V8, you can have a statement label at the beginning of any statement within an SQL procedure

- Enhances functionality of SQL procedures
- Enhances DB2 Family compatibility (with LUW and iSeries)



LOB SQL variables are supported in DB2 for z/OS V8

- LOB parameters already supported in V7
- Now also as a variable, for example, **DECLARE hv_book CLOB (2M)**;

Relaxing restrictions on variable names

- In V8, SQL variables and parameters can have the same name
  – If SQL var and parm have same name, an unqualified reference to the name is assumed to be the variable (since declared more locally than parm)
- In V8, SQL variables can be reserved words
  – When ambiguous, use delimited version, such as SET "PATH" = 'ABC';

8.3.7 Enhanced label, LOB, and variable support

In the following sections we discuss support for enhanced labels, LOBs, and variables.

Enhanced label support

When SQL stored procedures were introduced in DB2 for OS/390, they included limited support for statement labels within SQL procedures. More specifically, V6 and V 7 only support labels on the assignment statement, the compound statement, LOOP, REPEAT, and WHILE statements within an SQL procedure.

The label provides a destination for the GOTO statement, for example:

```
label1: SET x = y;
...
IF y = 1 GOTO label1;
```

With this enhancement, DB2 for z/OS introduces support for a statement label at the beginning of any statement within an SQL procedure.

This enhances compatibility with both DB2 LUW and DB2 for iSeries.

LOB SQL variables

The initial support for SQL procedures in DB2 for OS/390 did not include support for either LOB parameters or SQL variables. Version 7 added support for LOB parameters, however, there was still no support for defining a LOB SQL variable within an SQL procedure. V8 adds support for LOB SQL variables. You can now declare variables in an SQL procedure as CLOB, BLOB, or DBCLOB, as shown in Example 8-5.

Example 8-5 Declaring LOB SQL variables

```
CREATE PROCEDURE PROCESS_BOOK (IN bookNumber INTEGER, OUT bookPages INTEGER)
LANGUAGE SQL
BEGIN
   DECLARE v_numRecords INTEGER DEFAULT 1;
   DECLARE v_counter INTEGER DEFAULT 0;
   DECLARE deptNumber INTEGER DEFAULT 0;
   DECLARE v_book CLOB(2M);
   DECLARE v_binary BLOB(2M);
   DECLARE v_dbcs DBCLOB(2M);
   .....
END
```

DB2 LUW and DB2 for iSeries also support LOB data types in the definition of SQL variables within SQL procedures.

Relaxed restrictions on variable names

In the following sections we describe some restrictions that have been relaxed in V8.

SQL variable and parameter can have the same name

An additional change is to remove a restriction that an SQL variable cannot have the same name as a parameter for that procedure. This restriction was previously only enforced by DB2 for OS/390; it was not enforced by the other DB2 platforms.

If an SQL variable has the same name as an SQL parameter, then an unqualified reference to the name is assumed to be a variable. This makes sense because the SQL variable is declared more locally than the parameter.

In the following procedure (Example 8-6), the SQL variable P1 takes precedence over the SQL parameter of the same name because it is declared more locally than the parameter.

Example 8-6 Using the same name for a variable and a parameter

```
CREATE PROCEDURE CLAIRE2 (OUT P1 INTEGER, OUT P2 INTEGER) LANGUAGE SQL
RICK1: BEGIN
   DECLARE P1 INT;
   SET CLAIRE2.P1=5;
   SET RICK1.P1=7;
   SET P1=10;
   SET P2=P1;
+ END
```

The values returned from this procedure are:

- P1: 5
- P2: 10

The results indicate that the SQL parameter P1 was only set by the assignment statement which referred to P1 by the qualified name RICK1.P1. The assignment statement which referred to P1 without a qualifier resulted in the value 10 (‘implicitly qualified by RICK1) indicating that the SQL variable overwrote the qualified assignment of the value 7 to the SQL variable P1.

SQL procedure variable names can be reserved words

In DB2 V7, the names of variables and parameters within SQL procedures cannot be reserved words.

This enhancement increases DB2 Family compatibility with DB2 for LUW and DB2 for iSeries.

In DB2 for z/OS V8, if an SQL variable with a name of a reserved word is used in a context where its use would be ambiguous, then you specify the name of the variable as a delimited identifier. The following coding (Example 8-7) is now allowed.

Example 8-7 Using reserved words in SQL procedures

```
CREATE PROCEDURE P1 LANGUAGE SQL
 BEGIN
   DECLARE PATH CHAR(8);
   SET PATH = 'ABC';   <-- This refers to the special register
   SET "PATH" = 'ABC'; <-- This refers to the SQL variable
 END
```

Potential SET CURRENT PATH and CONNECT behavior change

Before V8, SET CURRENT PATH and CONNECT statements were not behaving as documented in the manuals in some cases.

In V8, DB2 behaves as described in the SQL Reference regarding an unresolved name, and whether it resolves to an identifier, SQL parameter, or SQL variable.

- In the SET PATH statement, the name is checked as an SQL parameter name or SQL variable name. If not found as an SQL variable or SQL parameter name, it will then be used as an identifier.
- In the CONNECT statement, the name is used as an identifier.

This way DB2 for z/OS behaves as documented, and the same way as DB2 for LUW and DB2 for iSeries.

DB2 UDB V8 Development Center

Functionality

- Evolution of Stored Procedure Builder
- Support for entire family of DB2 servers
- From mainframe point of view, it can be used to create stored procedures
- Enhanced z/OS support including specialized SQL IDs (package owner, build owner, secondary ID and advanced build options)
- Support for developing SQL and JAVA stored procedures on zSeries
- Support for viewing live database tables, views, triggers, stored procedures and user-defined functions
- and more....

8.4 DB2 UDB V8 Development Center

The DB2 Development Center (DC), included in the V8.1 UDB Application Development Client (ADC) component, is the follow-on product to the DB2 Stored Procedure Builder (SPB) in the DB2 V7.2 UDB Application Development Client.

Development Center supports the entire family of DB2 servers using the DRDA architecture. It communicates with the DB2 UDB V8.1 distributed servers (Linux, UNIX, and Windows), DB2 UDB for OS/390 V6 and V7 and DB2 UDB for z/OS V8, as well as currently supported DB2 UDB releases on iSeries.

From a mainframe perspective, the tool can be used to build stored procedures. Development Center supports creating SQL stored procedures on all supported versions of DB2 for OS/390 and z/OS (currently V6, V7, and V8). Java stored procedures can be created with Development Center on DB2 V7 and V8.

The Development Center Online Help is an excellent source for additional information on the Development Center.

Development Center -2

Testing and debugging of SP on zSeries

- Test (invoke) stored procedure and user-defined function written in any language
- Saved object test settings, including parameter values and pre-execution and post-execution SQL scripts
- Via a wizard, you can create and generate code for new SQL and Java stored procedures
- Enhanced debugging of SQL stored procedures with variable value change support using an integrated SQL debugger
- New feature of wizard
  - Ability to insert code fragments into the generated code

8.4.1 Testing and debugging stored procedures with DC

Using the Development Center (DC) you invoke and test stored procedures written in any language. To make it easier to test, you can save object test settings, including parameter values. You can also use pre- and post-execution SQL scripts.

Via a wizard, you can create and generate code for new SQL and Java stored procedures. You can also import existing SQL and Java stored procedures.

You can remotely (from within Development Center) debug SQL stored procedures that execute on DB2 UDB for z/OS servers using the SQL Debugger. The SQL Debugger is integrated into various client development platforms including DB2 Development Center. With the SQL Debugger, you can observe the execution of SQL procedure code, set break points for lines, and view or modify variable values.

With the new wizard you have the ability to insert code fragments into the generated code.

Development Center -3

Including SQL code fragments

- Code fragments
  - User-defined sections of source code or comments inserted at predefined locations in the generated source code
- Text files that you create, which you and your team members can reuse
- Especially useful when standard set of error handling functions, headers, variable declarations, etc. included in all stored procedures for a given project

Debugging

- Tightly integrated into Development Center
- Implementation of the SQL debugger includes:
  - Greater stability
  - Support for variable value change while debugging
  - Viewing sections of large variables, such as large objects (LOBs)

Including SQL code fragments

The new DC allows you to include your own code fragments into the generated code. You can insert user-defined sections of source code (or comments) at pre-defined locations in the generated code. These are text files that you create, and can be shared among a work group, such as all developers. A good example of this would be a standard set of error handling functions, headers, or variable declarations. This way, they can be coded once and included by everybody.

Debugging

As mentioned before, you can use the integrated debugger to debug SQL stored procedures on z/OS. The prerequisites and setup for debugging SQL stored procedures on z/OS are:

- Workstation:
  - DB2 Connect
  - DB2 Development Center UDB V8.1.4 (FixPak 4)
- z/OS:
  - DB2 UDB for z/OS Version 8
  - Run installation job <hlq>.SDSNSAMP(DSNTPSMP)
  - C Compiler
  - Run installation job <hlq>.SDSNSAMP(DSNTIJSD).

    This job provides six new DB2-supplied stored procedures. It is recommended to process these stored procedure in the same WLM AE with the DB2-supplied stored procedures used by the SQL Assist set up in <hlq>.SDSNSAMP(DSNTIJMS).

This job includes support for the debugger providing:

- New DDL definitions
- New BINDs
- New authorizations

– A WLM procedure must be defined for executing the SQL stored procedure. This WLM procedure optionally includes a //PSMDEBUG DD statement used to collect information when debugging problems with the SQL Debugger. The //PSMDEBUG statement defines a physical sequential data set with RECFM=VBA, LRECL=4096. This data set should only be included in the WLM procedure when requested by IBM Level 2 as the //PSMDEBUG statement presence causes records to be written to it for SQL debugger problems, which will impact performance.

Here are some reference Web documents:

- DB2 Development Center — The Next-Generation AD Tooling for DB2

  http://www7b.software.ibm.com/dmdd/library/techarticle/0207alazzawe/0207alazzawe.html

- DB2 Integrated SQL Debugger IBM DB2 Stored Procedure Builder V7.2

  http://www7b.software.ibm.com/dmdd/library/techarticle/alazzawe/0108alazzawe.html

Implicit RRSAF database connections

Allow RRSAF applications to exploit *implicit connections* to DB2 for z/OS

- Similar to implicit connection support in CAF

Today only explicit RRSAF connection

- Issue IDENTIFY, CREATE THREAD, SIGNON to establish a database connection
- (TERMINATE THREAD - TERMINATE IDENTIFY to end)

V8 implicit RRSAF connection

- Just start issuing SQL statements and IFI calls
- RRSAF will set up the connection for you
  - Subsystem name used depends on SSID in DSNHDECP
  - Plan name used depends on DBRM issuing first SQL call
  - Authorization ID used is that of the address space or ACEE (if exists)

8.5 Implicit RRSAF connections to DB2

With this enhancement, DB2 applications using RRSAF (RRS attach facility) can make implicit connections to DB2, simply by including SQL statements or IFI calls. RRSAF will make the required connection to DB2.

Comparing with CAF

A *CAF (call attach facility) application* can use two ways to connect to DB2, using an explicit, or an implicit connection.

- When using an *explicit connection*, you issue a CONNECT and OPEN to connect to the DB2. To disconnect, you issue a CLOSE and DISCONNECT.
- When using an *implicit connection*, a CAF application just issues SQL statements or IFI calls. CAF establishes the implicit connection to DB2 using default values for subsystem name (from DSNHDECP) and plan name (DBRM name associated with the first SQL call).

In V7, a RRSAF application can only use explicit connections. It issues an IDENTIFY, optionally a SIGNON, and a CREATE THREAD. To end a connection explicitly, you use TERMINATE THREAD, TERMINATE IDENTIFY.

This enhancement changes RRSAF to allow the application to implicitly connect, just by issuing SQL statements or IFI calls (similar to CAF). When an implicit connect is requested, RRSAF will issue an IDENTIFY and CREATE THREAD using default values for the subsystem name and the plan name.

- The default value for the subsystem name is the name specified by the SSID parameter in DSNHDECP. RRSAF uses the installation default DSNHDECP, unless your own DSNHDECP is in a library provided in a STEPLIB or JOBLIB concatenation, or in the link list. In a data sharing group, the default subsystem name is the group attachment name.
- The default value for the plan name will be the name of the database request module (DBRM) associated with the module making the (first) SQL call. If your program can make its first SQL call from different modules with different DBRMs, then you cannot use a default plan name. You must use an explicit call using the CREATE THREAD function.
- The authorization ID is set from the 7-byte user ID associated with the address space, unless an authorized function has built an ACEE for the address space. If an authorized function has built an ACEE, DB2 passes the 8-byte user ID from the ACEE.

If your application includes both SQL and IFI calls, you must issue at least one SQL call before you issue any IFI calls. This ensures that your application uses the correct plan, as described above for the plan name.

As before, you must make sure the RRSAF language interface load module, DSNRLI is available.

Tip: As with CAF, using an explicit connection using RRSAF gives you more control over the behavior of the database connection. If this is important for your application, you should not use implicit connections.

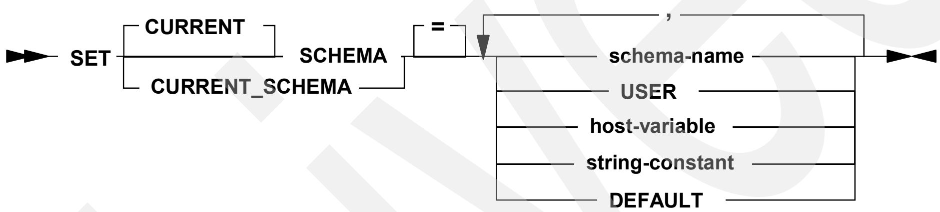
SET [CURRENT] SCHEMA

Problem: Today use CURRENT SQLID as implicit qualifier for dynamic SQL

- However, using CURRENT SQLID also affects authorization:
  - CURRENT SQLID must be authorized to do dynamic CREATE, ALTER, GRANT REVOKE
  - CURRENT SQLID must be primary or secondary auth ID (if not SYSADM or SYSCTRL)
- Static SQL uses the BIND option QUALIFIER for this purpose
  - Only affects unqualified SQL

Solution : SET SCHEMA / CURRENT SCHEMA special register for dyn SQL

- CURRENT SCHEMA is a special register of VARCHAR(128) with same value as CURRENT SQLID at initialization time
- Use SET SCHEMA to change
- Only used as qualifier
- Can be any value (not limited to primary or secondary auth ID like CURRENT SQLID



8.6 SET [CURRENT] SCHEMA

Customers often run applications with different qualifiers in effect for unqualified names, without having to have separate copies of the application code itself. They then deploy the application in different environments (sometimes on the same DB2 subsystem) without have to change any of the application code.

With DB2 for z/OS and OS/390 V7, an application can be coded without qualifiers for object names (also known as unqualified SQL), and an implicit qualifier will be used for unqualified names.

For static SQL statements, the DB2 QUALIFIER option for BIND can be used, to specify the implicit qualifier for unqualified object names (except for those contexts which use the SQL PATH to resolve the name of an object). The QUALIFIER bind option affects the implicit qualification of unqualified object names. If the QUALIFIER bind option is not specified, the OWNER of the plan or package is used as the implicit object qualifier for static SQL statements.

For dynamic SQL statements, the CURRENT SQLID special register can be used to specify the implicit qualifier for unqualified object names (except for those contexts which use the SQL PATH to resolve the name of an object). However, unlike the QUALIFIER bind option, the use of the CURRENT SQLID special register has other effects, as it is also used for authorization checking on dynamic CREATE, ALTER, GRANT, and REVOKE statements, and the value is also used as the owner (or definer) of objects created with dynamic CREATE statements.

The current (V7) support for specifying an implicit qualifier for dynamic SQL statements (CURRENT SQLID) is not quite the same as the support for static SQL statements (the QUALIFIER bind option).

The bundling of qualification of object name with the authorization checking, and ownership of objects is not an ideal solution for dynamic SQL statements.

Again DB2 for z/OS Version 8 comes to the rescue. V8 introduces a new SET (CURRENT) SCHEMA SQL statement, and a new special register CURRENT SCHEMA. It is only used to qualify object names in unqualified dynamic SQL statements (when DYNAMICRULES(RUN) are in effect, the default).

The CURRENT SCHEMA special register is initialized with the value of the CURRENT SQLID at initialization time. You can then change it in your application using the SET SCHEMA SQL statement. You can set the CURRENT SCHEMA special register to *any* valid string (as long as the string you provide is a VARCHAR(128)). This is different from changing the CURRENT SQLID. There you have to specify you your primary authorization ID or one of your secondary authorization IDs (unless you are SYSADM or SYSCTRL). The CURRENT SCHEMA special register is only used for qualifying unqualified SQL in your program, nothing else.

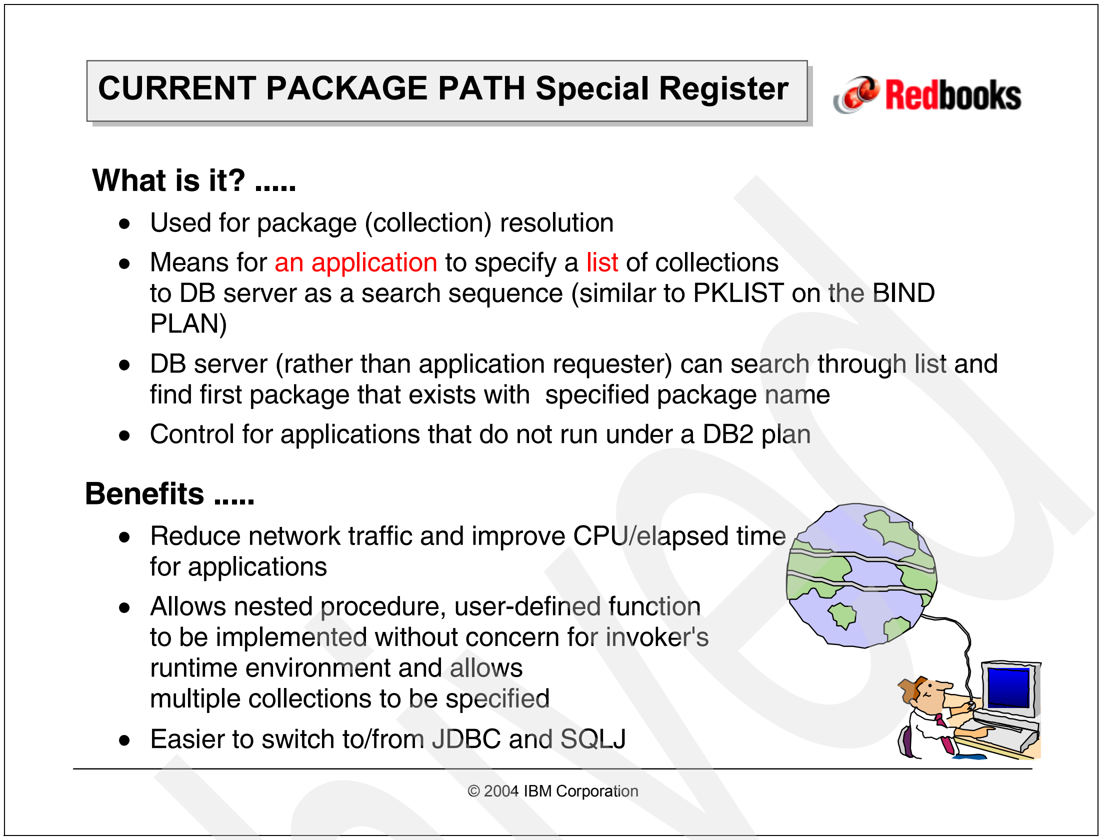
> **Note:** The QUALIFIER BIND option is used to qualify unqualified static SQL statement, or dynamic statement when DYNAMICRULES(BIND) is in effect.
>
> Currently the SET SCHEMA statement is not supported in the SQL procedure language.

Setting the CURRENT SCHEMA special register does not affect any other special register. Therefore, the CURRENT SCHEMA is not to be included in the SQL path that is used to resolve the schema name for unqualified references to function, procedures, and user-defined types in dynamic SQL statements. To include the current schema value in the SQL path, whenever the SET SCHEMA statement is issued, also issue the SET PATH statement including the schema name from the SET SCHEMA statement.

When the name of the object to be created is specified as an unqualified name, in case of a dynamic CREATE statement, the value of CURRENT SCHEMA must be the same as the CURRENT SQLID special register. Otherwise an SQLCODE -20283 is issued, as shown in Example 8-8.

Example 8-8 SQLCODE -20283

```
---------+---------+---------+---------+---------+---------+---------+---------+-
SET SCHEMA = 'TEST' ;
---------+---------+---------+---------+---------+---------+---------+---------+-
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 0
---------+---------+---------+---------+---------+---------+---------+---------+-
CREATE TABLE TESTTB (COLA  CHAR(10));
---------+---------+---------+---------+---------+---------+---------+---------+-
DSNT408I SQLCODE = -20283, ERROR:  A DYNAMIC CREATE STATEMENT CANNOT BE
         PROCESSED WHEN THE VALUE OF CURRENT SCHEMA DIFFERS FROM CURRENT SQLID
DSNT418I SQLSTATE   = 429BN SQLSTATE RETURN CODE
DSNT415I SQLERRP    = DSNXODDL SQL PROCEDURE DETECTING ERROR
DSNT416I SQLERRD    = 2  0  0  -1  0  0 SQL DIAGNOSTIC INFORMATION
DSNT416I SQLERRD    = X'00000002'  X'00000000'  X'00000000'  X'FFFFFFFF'
         X'00000000'  X'00000000' SQL DIAGNOSTIC INFORMATION
---------+---------+---------+---------+---------+---------+---------+---------+-
```



8.7 CURRENT PACKAGE PATH special register

CURRENT PACKAGE PATH is a new special register in DB2 V8. It it used when DB2 is looking for a matching package (same name, version number as in the invoking program) to load. Today, plans have the possibility of specifying a search sequence of collections for DB2 to look in to find a matching package, via the PKLIST BIND option. Today in V7, there is no PKLIST option for applications that do not run under a DB2 plan.

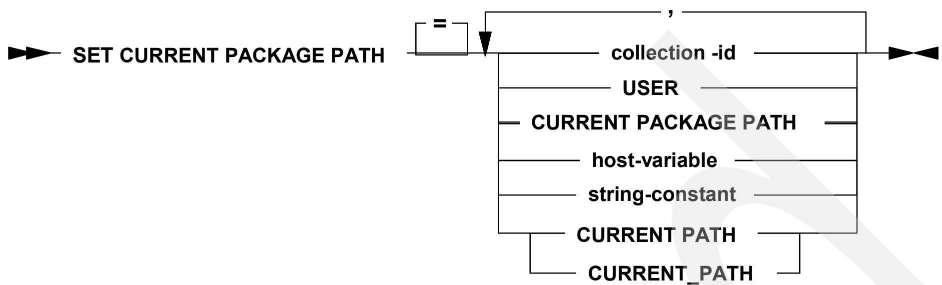
With the CURRENT PACKAGE PATH special register, DB2 V8 introduces a way *for the application* to specify a list of collections for DB2 to look for a certain package. This enhancement is especially important for SQLJ applications to provide a list of collections to look for a matching package.

The advantage of the CURRENT PACKAGE PATH is that it contains a *list* of collections. This is in contrast to the CURRENT PACKAGESET special register that can only contain the name of a single collection. In the case where the package is not found in that collection, the application has to change the CURRENT PACKAGESET special register and try again.

By using CURRENT PACKAGE PATH, you can specify a list of collections for DB2 to look in to find the package. Only after all collections have been searched unsuccessfully, is an SQLCODE -805 returned. This can be especially important in a distributed environment where every time you need to change the CURRENT PACKAGESET special register, it requires a trip across the wire. Because the CURRENT PACKAGE PATH can contain a list of collections, only a single trip is required.

SET CURRENT PACKAGE PATH

Redbooks



- USER, CURRENT PACKAGE PATH, and CURRENT PATH can only be specified once
- Note that you can specify the existing CURRENT PACKAGE PATH and concatenate additional collections, ahead or after

SET :oldCPP = CURRENT PACKAGE PATH;
SET CURRENT PACKAGE PATH = CURRENT PACKAGE PATH, prodcoll ;
CALL PRODSP (:hv1, :hv2);
SET CURRENT PACKAGE PATH = :oldCPP;

8.7.1 SET CURRENT PACKAGE PATH statement

The SET CURRENT PACKAGE PATH statement assigns a value to the CURRENT PACKAGE PATH special register. The statements must be embedded in an application. It is an executable statement that cannot be dynamically prepared. The CURRENT PACKAGE PATH is a VARCHAR(4096) value.

No validation that the collections exist is done at the time that the CURRENT PACKAGE PATH special register is set. For example, a collection ID that is misspelled is not detected, and this could affect the way subsequent SQL operates. At package execution time, authorization to the specific package is checked, and if this authorization check fails, the next collection is checked.

The SET CURRENT PACKAGE PATH statement is executed by the database server to which the application is currently connected, and is therefore classified as a non-local SET statement in DRDA. The SET CURRENT PACKAGE PATH statement requires a new level of DRDA support:

- If the application is connected to the local server when the SET CURRENT PACKAGE PATH statement is issued, the CURRENT PACKAGE PATH special register at the local server is set.
- Otherwise, when the application is connected to a remote server when the SET CURRENT PACKAGE PATH is issued, the CURRENT PACKAGE PATH special register at the remote server is set.

Combining CURRENT PACKAGE PATH and CURRENT PACKAGESET

These are the rules for combination:

- If you set the special register CURRENT PACKAGE PATH *or* CURRENT PACKAGESET, DB2 skips the check for programs that are part of a plan and uses the values in these registers for package resolution.

  When CURRENT PACKAGE PATH is set, the server that receives the request ignores the collection that is specified by the request and instead uses the value of CURRENT PACKAGE PATH at the server to resolve the package. Specifying a collection list with the CURRENT PACKAGE PATH special register can avoid the need to issue multiple SET CURRENT PACKAGESET statements to switch collections for the package search, as you would have to in V7.

- If you set CURRENT PACKAGE PATH, DB2 uses the value of CURRENT PACKAGE PATH as the collection name list for package resolution. For example, if CURRENT PACKAGE PATH contains the list COLL1, COLL2, COLL3, COLL4, then DB2 searches for the first package that exists in the following order:

  ```
  COLL1.PROG1.timestamp
  COLL2.PROG1.timestamp
  COLL3.PROG1.timestamp
  COLL4.PROG1.timestamp
  ```

- If you set CURRENT PACKAGESET *and not* CURRENT PACKAGE PATH, DB2 uses the value of CURRENT PACKAGESET as the collection for package resolution. For example, if CURRENT PACKAGESET contains COLL5, then DB2 uses COLL5.PROG1.timestamp for the package search.

Table 8-1 shows examples of the relationship between the CURRENT PACKAGE PATH special register and the CURRENT PACKAGESET special register.

Table 8-1 Scope of SET CURRENT PACKAGE PATH statement

SQL Statements	What happens
SET CURRENT PACKAGESET SELECT ...FROM T1 ...	The collections in PACKAGESET determine which package is invoked.
SET CURRENT PACKAGE PATH SELECT ...FROM T1 ...	The collections in PACKAGE PATH determine which package is invoked.
SET CURRENT PACKAGESET SET CURRENT PACKAGE PATH SELECT ...FROM T1 ...	The collections in PACKAGE PATH determine which package is invoked.
SET CURRENT PACKAGE PATH CONNECT TO S2 ... SELECT ...FROM T1 ...	The local server sends one collection at a time from PACKAGE PATH at the local server to remote server S2 that is to be used for package resolution.[1]
SET CURRENT PACKAGE PATH = 'A,B ' CONNECT TO S2 ... SET CURRENT PACKAGE PATH ='X,Y ' SELECT ...FROM T1 ...	The collections in PACKAGE PATH that are set at server S2 determine which package is invoked.
SET CURRENT PACKAGE PATH SELECT ...FROM S2.QUAL.T1 ...	Three-part table name. On implicit connection to server S2, PACKAGE PATH at server S2 is inherited from the local server. The collections in PACKAGE PATH at server S2 determine which package is invoked.

SQL Statements	What happens
Notes: [1]. When CURRENT PACKAGE PATH is set at the requester (and not at the remote server), DB2 passes one collection at a time from the list of collections to the remote server until a package is found or until the end of the list. Each time a package is not found at the server, DB2 returns an error to the requester. The requester then sends the next collection in the list to the remote server.	

CURRENT PACKAGE PATH and stored procedures and UDFs

When a stored procedure calls another program, DB2 determines to which collection the called program's package belongs, in one of the following ways:

- If the stored procedure executes SET CURRENT PACKAGE PATH, the called program's package comes from the list of collections in the CURRENT PACKAGE PATH special register. For example, if CURRENT PACKAGE PATH contains the list COLL1, COLL2, COLL3, COLL4, then DB2 searches for the first package (in the order of the list) that exists in these collections.
- If the stored procedure does not execute SET CURRENT PACKAGE PATH and instead executes SET CURRENT PACKAGESET, the called program's package comes from the collection that is specified in the CURRENT PACKAGESET special register.
- If the stored procedure does not execute SET CURRENT PACKAGE PATH or SET CURRENT PACKAGESET:
  - If the stored procedure definition contains NO COLLID, DB2 uses the collection ID of the package that contains the SQL statement CALL.
  - If the stored procedure definition contains COLLID collection-id, DB2 uses collection-id.

ODBC for USS Enhancements

We discuss enhancements to ODBC for programs running in UNIX System Services on z/OS

- Userid and password authentication (validation, not just syntax checking) on SQLConnect and SQLDriverConnect
- ODBC support for long names
- ODBC support for SQL statements up to 2 MB
- SQLCancel() support
- ODBC Unicode support
  - Update, insert, delete and fetch Unicode data through ODBC application variables
  - Unicode strings within the ODBC application programming interface (which allow you to use Unicode SQL statements in your ODBC application)

8.8 ODBC for UNIX System Services enhancements

In this section we describes the ODBC/CLI enhancements in DB2 V8:

- ODBC SQLConnect user and password support
- ODBC support for long names
- ODBC support for SQL statements up to 2 MB
- SQLCancel support
- ODBC Unicode support

8.8.1 ODBC SQLConnect user and password support

When you run an ODBC application on z/OS, you use RRS or CAF to connect to the database. (In case of a DB2 UDB for z/OS, the database to which you connect represents an entire DB2 subsystem.)

Actually, the DB2 thread is created when allocating the connection handle. After you have created the connection handle, the ODBC application can start the DRDA communication with the DB2 subsystem. To do this, you must use the SQLConnect or SQLDriverConnect API.

You have always been able to specify a user ID and password argument on the SQLConnect and SQLDriverConnect API calls. However, they were not passed to the DB2 UDB for z/OS system. These keywords were only checked to validate if they were syntactically correct, which basically means that they were not allowed to exceed the length restrictions for user ID and password and that they did not contain blank values.

The user ID that was used to establish the thread and checked for authorization in DB2 was the user ID that was used to logon to the system. At that time, the user ID and password were verified with the system's security software, for example RACF.

In terms of compatibility with other DB2 platforms, this behavior has been changed in DB2 V8. Now the values for the user ID and password arguments on the input to SQLConnect and SQLDriverConnect APIs are propagated to the target DB2 system. For compatibility with existing application programs, the user authentication is only performed when both a user ID and password are provided on the API call.

> **Attention:** Applications which try to connect to a local DB2 system with an invalid user ID or password fail with SQLCODE -922. That means that if you used values in your existing applications that did not represent real user IDs, because they have not been checked until now, you must make sure that those values are either valid or set to blank or NULL. When connecting to a remote DB2, and you specify a user ID but no password, you receive an SQLCODE -1403, or SQLCODE -30082 when the user ID or password is wrong.

This function has been made available in DB2 V7 through the maintenance stream, via APAR PQ58787 (PTF UQ67626).

8.8.2 ODBC long name support

ODBC for z/OS has been enhanced to be able to support the long names in V8 new function mode. This means changes to the following ODBC functions.

Support for longer names in the INI file variables

The keyword strings for the following initialization keywords will be extended to support longer names:

- CLISCHEMA
- COLLECTIONID
- CURRENTFUNCTIONPATH
- CURRENTSQLID
- SCHEMALIST
- SYSSCHEMA

ODBC catalog functions

As a result of the name changes in the catalog, the ODBC catalog API needs to change the lengths of the host variables and data types in the SQLDA when fetching result sets from catalog tables. However, most of these changes are transparent to the external API. Only the data type of the REMARKS column returned by SQLColumns() and SQLTables() is changed to VARCHAR(762).

8.8.3 Support for statements up to 2MB

ODBC also supports long SQL statements, up to 2 MB. This enhancement requires APAR PQ88582/PTF UQ91257.

ODBC for USS Unicode Support

The following DB2 ODBC elements support this new functionality:

- A new initialization keyword CURRENTAPPENSCH (in the .INI file) to specify the current encoding scheme (EBCDIC, ASCII, or Unicode). When you set this keyword to Unicode, generic ODBC APIs support UTF-8 data.
- New APIs with the suffix W, called wide APIs, are introduced to support UCS-2 data. (V6 PTF UQ60475 and in V7 with PTF UQ60476 and V7 as well)
  Wide APIs accept Unicode UCS-2 string arguments only, and does not require that the CURRENTAPPENSCH keyword is set to Unicode.
  - For example: The equivalent wide API for the SQLConnect () function call is SQLConnectW().
- The non-wide functions, for example SQLColumnPivileges, have been changed to accept UTF-8 string arguments and return all character string data in the result set in UTF-8 encoding scheme.
- New SQL_C_WCHAR data type to support UCS-2 data
- Additional SQLGetInfo() attributes to query the CCSID settings of the DB2 subsystem in each encoding scheme, for example SQL_ASCII_SCCSID

8.8.4 ODBC Unicode support

As we mentioned before, one of the major enhancements of DB2 V8 is the exploitation of Unicode in many different areas. ODBC is one of the important interfaces to DB2.

Up to DB2 V7, only the EBCDIC encoding scheme was fully supported for ODBC. There was no support for Unicode and only partial support for ASCII encoding scheme. DB2 V8 now provides you with the ability to:

- Update, insert, delete, and fetch Unicode data through ODBC application variables.
- Use Unicode strings within the ODBC application programming interface (which allow you to use Unicode SQL statements in your ODBC application)

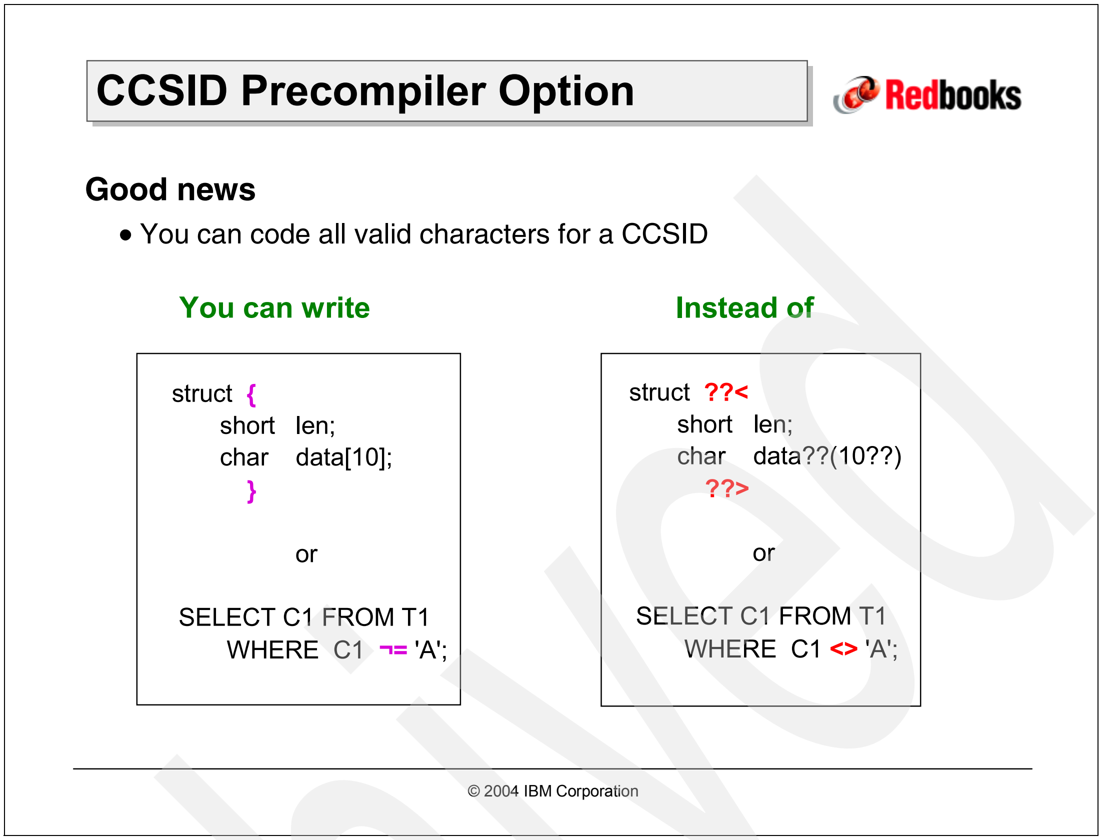
The following DB2 ODBC elements support this new functionality:

- DB2 V8 introduces a new initialization keyword CURRENTAPPENSCH (in the .INI file) to specify the current encoding scheme (EBCDIC, ASCII, or Unicode). When you set this keyword to Unicode, generic ODBC APIs support UTF-8 data.
- In addition to the normal ODBC API calls, a set of new APIs with the suffix W, called wide APIs, are introduced to support UCS-2 data.

  Wide APIs accept Unicode UCS-2 string arguments only. The equivalent wide API for the SQLConnect () function call is SQLConnectW(). Wide APIs were enabled in V6 with PTF UQ60475 and in V7 with PTF UQ60476, and are available in the V8 base code.

  (After the installation of the PTF for APAR PQ86094, the use of the wide API does not require that the CURRENTAPPENSCH keyword is set to Unicode. Actually, when using the wide API or SQL_C_WCHAR as the symbolic C data type, CURRENTAPPENSCH is not checked. When using these, UCS-2 is always assumed.)

- The non-wide functions, for example SQLColumnPivileges, have been changed to accept UTF-8 string arguments and return all character string data in the result set in UTF-8 encoding scheme.
- A new SQL_C_WCHAR data type to support UCS-2 data is now available.
- Additional SQLGetInfo() attributes to query the CCSID settings of the DB2 subsystem in each encoding scheme, for example, SQL_ASCII_SCCSID, are provided.



8.9 CCSID Precompiler option

In DB2 Version 8, the precompiler works in Unicode, irrespective of the mode DB2 is running in. Therefore, if the SQL statements of your source program are not in Unicode UTF-8 (which is most likely the case in most traditional programming languages), the DB2 Version 8 precompiler converts them to UTF-8 for parsing.

A new precompiler keyword `CCSID(n)` tells the precompiler the CCSID the source program is written in, so the precompiler can convert from that CCSID to CCSID 1208 (UTF-8). (If you want to prepare a source program that is written in a CCSID that cannot be directly converted to or from CCSID 1208, you must create an indirect conversion. For information about indirect conversions, see *z/OS Support for Unicode: Using Conversion Services*, SA22-7649.)

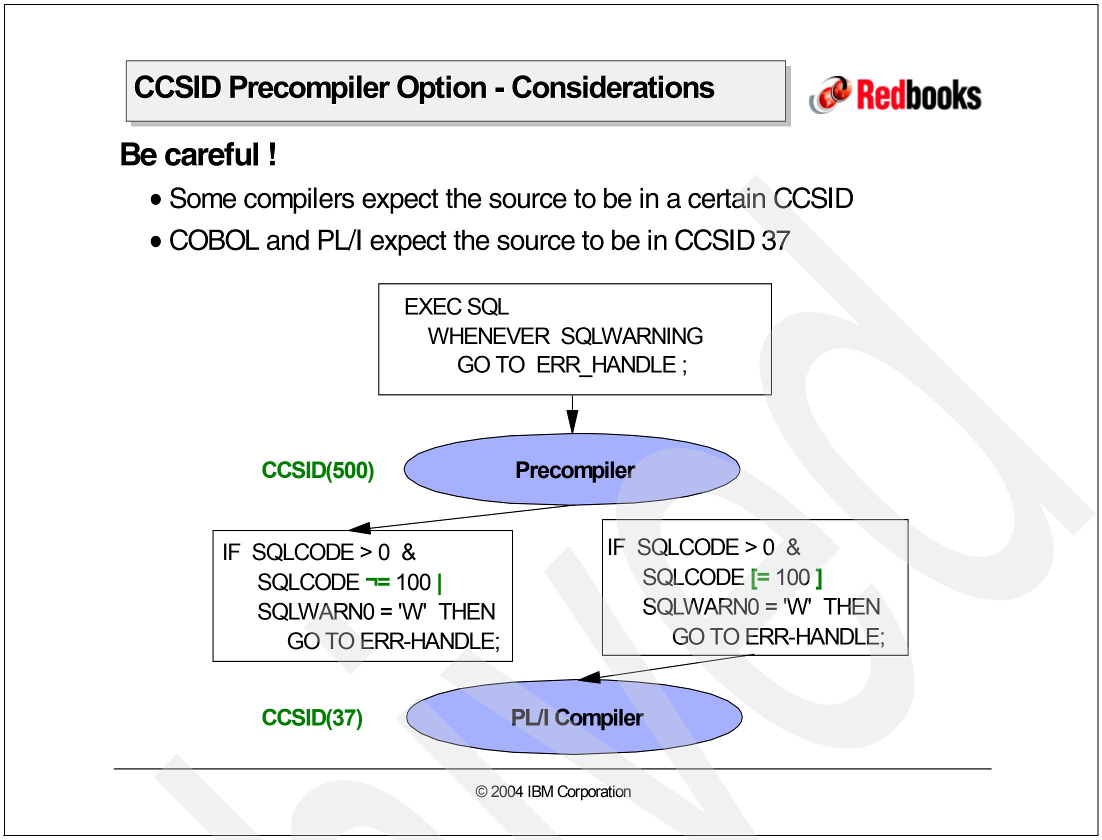
8.9.1 Coding characters in your “own” CCSID

Using the new CCSID(n) option during precompilation allows you to specify the numeric value “n” of the CCSID in which the source program is written. The number “n” must be either 65535, or in the range 1 through 65533.
The default setting is the EBCDIC system CCSID (SCCSID DSNDECP value) as specified on the panel DSNTIPF during installation.

Your source program is converted by the precompiler from the CCSID value that you specify, to CCSID 1208 (UFT-8) that is used by the precompiler in V8. After the precompilation, the program is converted back to the original CCSID. In general, the precompiler produces the following output:

- An output listing (SYSPRINT data set) in the CCSID of the source program.
- A modified source program (written to SYSCIN, and input to the compiler or assembler), written in the CCSID of the source program.
- A DBRM, where the SQL statements and the list of host variable names use the following character encoding schemes:
  - EBCDIC, for the result of a DB2 Version 8 precompilation with NEWFUN NO or a precompilation in an earlier release of DB2.
  - Unicode UTF-8, for the result of a DB2 Version 8 precompilation with NEWFUN YES.

The advantage of this option is that you can code your source program in the CCSID of your terminal emulator. As shown in the figure above, you no longer have to use “??<” to indicate curly braces in a C-program, but you can use the “{“ directly.



8.9.2 CCSID Precompiler option - considerations

However, you have to be careful when using the CCSID precompiler option.
In the figure above, we code our program in CCSID 500. When invoking the precompiler, we specify the CCSID(500) option. The precompiler does its job and produces a modified source that looks like the box on the left. Note that the precompiler substituted the “WHENEVER SQLWARNING” statement with a number of PL/I statements. In there, we see the “¬ =” (not equal) and “ | “ (or) symbols, all produced in CCSID(500).

The PL/I compiler assumes CCSID(37) when compiling a program.The special characters mentioned above are at different code points in CCSID(37), and the compiler produces an error. The statements are seen by the PL/I compiler as the box on the right, and the characters “ [“ and “] “ are not valid in PL/I.

For example, when using CCSID(500) to compile sample program DSNTEP2, you get the following error (Example 8-9).

Example 8-9 PL/I compilation error

```
Compiler Messages
 Message     Line.File  Message Description
 IBM1357I E  1687.1     Character with decimal value 186 does not belong to
                        the PL/I character set. It is assumed to be a NOT
                        symbol.
Failing statement
1687.1        IF SQLCODE>0 & SQLCODE =100   SQLWARNO='W'


Failing statement in hex
1687.1        IF SQLCODE>0 & SQLCODE =100   SQLWARNO='W'
FFFF4F44444444CC4EDDCDCC6F454EDDCDCCB7FFF4B4EDDECDDF77E7
1687B1000000009602833645E00002833645AE1000B028361950ED6D
```

Note that in the hex representation of the statement, we find x'BA' and x'BB'. They are the "¬" (not) sign and " | " (or sign) in CCSID(500), but not in CCSID(37) that is used by the PL/I compiler to interpret these statements, and hence an error is produced.

When you are using precompiler services, the compiler directly calls the DB2 precompiler, then the COBOL compiler passes the CCSID to DB2.

9

Utility enhancements

List of Topics

- New utilities BACKUP SYSTEM and RESTORE SYSTEM
- Delimited data support for LOAD and UNLOAD
- RUNSTATS enhancements
- Defaults for better performance
- REORG TABLESPACE enhancements
- REBUILD INDEX enhancements
- COPY enhancements
- REPAIR enhancements
- Changes to utilities to support online schema evolution
- Offline utility (DSN1*) enhancements
- Unicode utility statements

This chapter describes the enhancements to DB2 utilities. It consists of the following topics:

- New utilities:
  - BACKUP SYSTEM
  - RESTORE SYSTEM
- Delimited data support for:
  - LOAD
  - UNLOAD
- RUNSTATS enhancements
- Defaults for better performance
- REORG TABLESPACE enhancements
- REBUILD INDEX enhancements
- COPY enhancements
- REPAIR enhancements
- Changes to utilities to support:
  - Online schema evolution
  - Multi-level security

System Level Point-in-Time Recovery

Easier, more flexible, less disruptive, faster backup and recovery

Handle large numbers of table spaces and indexes

Two new utilities are introduced

- BACKUP SYSTEM for fast volume-level backups
  - DB2 databases and logs
  - Data sharing group scope
  - z/OS V1R5 required for new COPYPOOL function and fast replication
- RESTORE SYSTEM
  - To an arbitrary point-in-time
  - Handles CREATEs, DROPs, LOG NO events
  - Data sharing group scope

9.1 System level point-in-time recovery

Enhancements to system level point-in-time recovery for DB2 provide improved usability, more flexibility, and faster backup and recovery. You can recover your data to any point-in-time, regardless of whether you have uncommitted units of work. Data recovery time improves significantly for large DB2 systems that contain many thousands of objects. Two new utilities are used for system level point-in-time recovery:

- The *BACKUP SYSTEM* utility provides fast volume-level copies of DB2 databases and logs for an entire DB2 subsystem or DB2 data sharing group. It relies on new DFSMShsm services in z/OS Version 1 Release 5 that allow for fast volume level backups. BACKUP SYSTEM is less disruptive than using the SET LOG SUSPEND command for copy procedures. An advantage for data sharing environments is that BACKUP SYSTEM has a group scope (compared to SET LOG SUSPEND, which has a member scope).
- The *RESTORE SYSTEM* utility recovers a complete DB2 system or a data sharing group to an arbitrary point-in-time. RESTORE SYSTEM automatically handles any creates, drops, and LOG NO events that may have occurred between the point the backup is taken and the point-in-time that you recover to.

More details on this feature can be found in 3.16, "System level point-in-time recovery" on page 191.

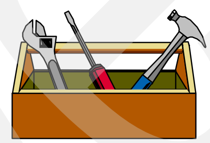
LOAD / UNLOAD Delimited Input / Output

Redbooks

LOAD / UNLOAD utilities will accept / produce delimited files

Benefits of these enhancements include:

- Eases the import / export of large amounts of data from DB2 for z/OS to other operating system platforms and vice versa
- Eliminates the requirement to write a program to convert non-z/OS platform data into the positional format for the DB2 for z/OS LOAD utility, or to use INSERT processing
- Unloads data from DB2 for z/OS in delimited file format and loads / imports it into another RDBMS



9.2 LOAD / UNLOAD delimited input / output

Most relational database management systems, including DB2 on Linux, UNIX, and Windows (LUW) platforms, are capable of unloading data in delimited format, where each record is a row, and columns are separated by commas, and optionally delimited with double quote (“) marks, for example. On the other hand, most other systems cannot unload data into the positional format required by the DB2 for z/OS and OS/390 Version 7 LOAD utility. If you want to move data to DB2 for z/OS and OS/390, you must therefore, either write a program to convert the data into the positional format that the LOAD utility understands, or use insert processing, thereby not exploiting the performance advantages of the LOAD utility.

The DB2 for z/OS Version 8 LOAD utility is enhanced to accept data from a delimited file. The UNLOAD utility is also enhanced to produce a delimited file when unloading the data. These enhancements help to simplify the process of moving/migrating data into and out of DB2 for z/OS.

For example, you can save the data from your spreadsheet as a comma separated value (CSV) file, and load the saved data into a DB2 for z/OS table using the FORMAT DELIMITED option.

This function is totally compatible with other members of the DB2 family.

Delimited Files - Reminder

A delimited file is a sequential file with row and column delimiters

Is a string of characters consisting of cell values ordered by row, then by column

Row (Record) delimiters not needed since the end of record is inherent in the file structure

Columns are separated by column delimiters

Character strings are delimited by character delimiters

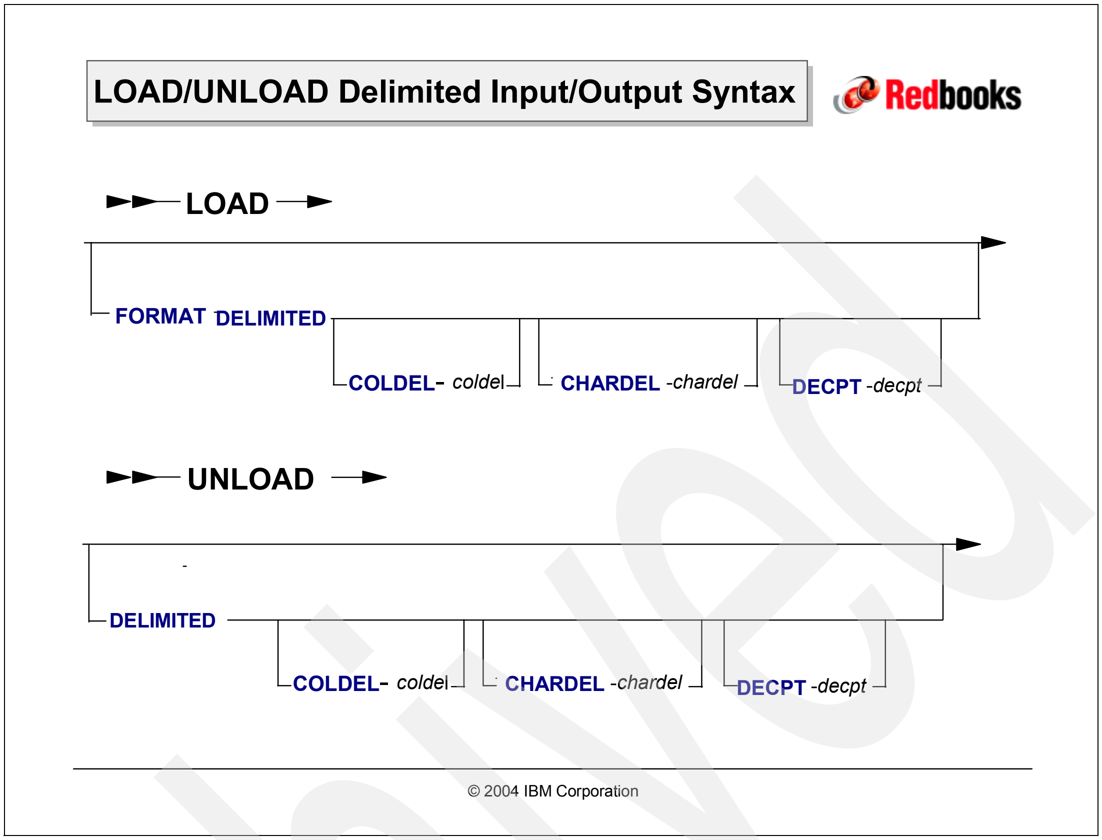
In z/OS, a row is a single BSAM record

Examples: "Bart", "Steegmans", "ITSO"

"Ravi", "Kumar", "ITES"

9.2.1 Delimited files - reminder

A delimited file, in general, is a sequential file with row and column delimiters. Each delimited file is a string of characters consisting of cell values ordered by row, and then by column. Columns within each row are separated by column delimiters. Rows are separated by row delimiters. The beginning and ending of each individual cell value may be indicated by character delimiters. In z/OS a row is a BSAM record.



9.2.2 LOAD/UNLOAD delimited input/output syntax

This visual shows the changes to the syntax of LOAD and UNLOAD utilities to support this enhancement.

> **Note:** It is interesting to note that LOAD uses FORMAT DELIMITED, whereas UNLOAD uses only the DELIMITED keyword.

Delimited Files - LOAD / UNLOAD

Redbooks

LOAD / UNLOAD allow specification of new keywords

- [FORMAT] DELIMITED
  - COLDEL - Column delimiter (default is comma)
  - CHARDEL - Character delimiter (default is quotation marks)
  - DECPT - Decimal point (default is period)

  These are characters to be found in the input file, or to be produced in output file

Double character delimiter recognition is supported (see example)

- Applies to CHAR, CLOB, and VARCHAR only

```
"what a " "nice" " day"
        LOADs as  ->      what a "nice" day

I am 6" tall.
        UNLOADs as ->   "I am 6"" tall."
```


9.2.3 Delimited files - LOAD / UNLOAD

A delimited file on z/OS is a sequential file consisting of one or more fixed or variable length records. Since the end of the record is inherent in the file structure, record delimiters, such as CRLF (carriage return line feed), are not used. The LOAD utility syntax has been changed and an additional option DELIMITED is added for the keyword FORMAT, or just the DELIMITED keyword during UNLOAD.

The DELIMITED option specifies that the input file is a delimited file. This is a BSAM file with column and character data string delimiters. In this format, all fields in the input data set are character strings or numeric data in external format. Each column value is separated from the next by a column delimiter character (the default is a comma).

When you specify DELIMITED, you can optionally specify COLDEL, CHARDEL and DECPT to indicate the delimiter characters that are different from the defaults.

- COLDEL specifies a single column delimiter character (the default is comma) that is used in the input file for LOAD, or in the output file for UNLOAD when DELIMITED is specified.
- CHARDEL specifies a single character string delimiter (the default is double quote) that is used in the input file for LOAD or in the output file for UNLOAD when DELIMITED is specified. The character string delimiter is permitted within character string input fields. Two successive character delimiters within the enclosing character delimiters are interpreted as a single character that is part of the character string.

  For example, when using the default double quote as a character delimiter:

  – The LOAD utility loads “what a ““nice”” day” as, what a “nice” day
  – The UNLOAD utility unloads I am 6” tall as, “I am 6”” tall”

- DECPT specifies a single decimal point character (the default is period) that is used in the input file for LOAD or in the output file for UNLOAD when DELIMITED is specified.

UNLOAD / LOAD (FORMAT) DELIMITED only supports unloading data from, and loading data into a single table. This is compatible with the rest of the DB2 Family. Thus, (FORMAT) DELIMITED specified with multiple FROM TABLE specifications, or not specifying FROM TABLE with the table space specification when the table space contains multiple tables, results in a syntax error.

For example, consider the following situations:

- A segmented table space DELIMITD.DELIMITS has two tables, EMP1 and EMP2, and you want to unload data from table EMP1.

  You can unload data from table EMP1 in one of the following two ways:

  ```
  UNLOAD DATA FROM TABLE EMP1 DELIMITED ... or
  UNLOAD TABLESPACE DELIMITD.DELIMITS FROM TABLE EMP1 DELIMITED ...
  ```

The TABLESPACE specification results in syntax error if FROM TABLE EMP1 ... is not included in the utility control statement.

- A segmented table space DELIMITD.DELIMITS has two tables, EMP1 and EMP2, and you want to unload data from both tables EMP1 and EMP2.

You can unload data from tables EMP1 and EMP2 in one of the following two ways:

```
UNLOAD DATA FROM TABLE EMP1 DELIMITED ...
UNLOAD DATA FROM TABLE EMP2 DELIMITED ... or

UNLOAD TABLESPACE DELIMITD.DELIMITS FROM TABLE EMP1 DELIMITED ...
UNLOAD TABLESPACE DELIMITD.DELIMITS FROM TABLE EMP2 DELIMITED ...
```

The FROM TABLE and TABLESPACE specifications result in syntax error if within a single UNLOAD statement, there is more than one DELIMITED or FROM TABLE specification.

Considerations when using DELIMITED LOAD/UNLOAD

You should be aware of the following considerations when using the delimited file format:

- LOAD:
  - When you specify the DELIMITED option, the utility ignores the POSITION keyword. The utility overrides field data type specifications according to the specifications of the delimited format. (For example, length values for CHAR, VARCHAR, GRAPHIC, VARGRAPHIC, CLOB, DBCLOB, and BLOB data are the delimited lengths of each field in the input data set, and the utility expects all numeric types in external format.
  - There is no length field associated with a VARCHAR column when using a delimited input file. Only the actual unload value appears in the input file. When loading the data into a VARCHAR column, DB2 will calculate the length of the field during load.
  - The keyword MIXED can be specified for CHAR, VARCHAR, and CLOB data types to indicate that the input field contains mixed (SBCS and DBCS) data. If MIXED is specified, any required CCSID conversions use the mixed CCSID for the input data. If MIXED is not specified, any such conversions use the SBCS CCSID for the input data.
  - If no field specifications are supplied, the input data is assumed to be in the mixed CCSID if any columns in the table are FOR MIXED. Otherwise it is assumed to be SBCS.
  - For Unicode input, the input data must be in CCSID 1208, UTF-8.
  - CONTINUEIF is not allowed with FORMAT DELIMITED.

– INCURSOR is not allowed with FORMAT DELIMITED.

– The WHEN keyword is not allowed with FORMAT DELIMITED.

– As described above, with FORMAT DELIMITED, multiple INTO TABLE statements are not allowed. FORMAT DELIMITED can only be used to load a single table at a time.

- UNLOAD:

  – For delimited output, UNLOAD does not add trailing padded blanks to variable length columns, even if you do not specify the NOPAD option. For fixed length columns, the normal padding rules apply.

    For example, if a VARCHAR(10) field contains ABC, UNLOAD DELIMITED unloads the field as “ABC”. However, for a CHAR(10) field that contains ABC, UNLOAD DELMITED unloads it as “ABC   “.

    Also note that there is no length field associated with an unloaded VARCHAR column. Only the actual unload value appears in the output file.

  – The default for HEADER is HEADER NONE.

  – HEADER OBID and ROWID are not valid output fields for the delimited output file format. Neither OBID nor ROWID are generated in the delimited output file. Since the header is not allowed, output must be from a single table.

  – When you specify the DELIMITED option, the utility ignores the POSITION keyword. The utility overrides field data type specifications according to the specifications of the delimited format. (For example, length values for CHAR, VARCHAR, GRAPHIC, VARGRAPHIC, CLOB, DBCLOB, and BLOB data are the delimited lengths of each field in the output data set, and the utility unloads all numeric types in external format.)

  – When the delimited output format is used, field POSITION is ignored if you specify it. Field data type specifications, if supplied, are overridden by the requirements of the delimited format; that is, the lengths of CHAR, VARCHAR, GRAPHIC, VARGRAPHIC, CLOB, DBCLOB, and BLOB fields are the delimited lengths of each field in the output data set, and all numeric types are unloaded in external format.

  – A NULL value is indicated by the absence of a cell value where one would normally occur (that is, two successive column delimiters, or missing columns at the end of a record). There is no NULL indicator byte present.

Delimited file data type forms for LOAD and UNLOAD

Table 9-1 shows the acceptable data type forms for the delimited file format.

Table 9-1 Acceptable data type forms for the delimited file format

Data type	Form acceptable to LOAD utility	Form in file created by UNLOAD utility
CHAR, VARCHAR	A delimited or non-delimited character string	Character data enclosed by character delimiters. There are no length bytes preceding the data in the string for VARCHAR.
GRAPHIC- Any Type	A delimited or non-delimited character stream	Data is unloaded as a delimited character string. There are no length bytes preceding the data in the string for VARGRAPHIC.
INTEGER - Any Type	A stream of characters representing a number in external format	Numeric data in external format
DECIMAL - Any Type	A character string that represents a number in external format	A string of characters representing a number
FLOAT (1-21) or REAL	Representation of number in single precision in external format	A string of characters representing a number in floating point notation
FLOAT (22-53) or DOUBLE	Representation of number in double precision in external format	A string of characters representing a number in floating point notation
BLOB, CLOB	A delimited or non-delimited character string	Character data enclosed by character delimiters. There are no length bytes preceding the data in the string.
DBCLOB	A delimited or non-delimited character string	Character data enclosed by character delimiters. There are no length bytes preceding the data in the string.
DATE	A delimited or non-delimited character string containing a date value in external format	Character string representation of a date
TIME	A delimited or non-delimited character string containing a time value in external format	Character string representation of a time
TIMESTAMP	A delimited or non-delimited character string containing a timestamp value in external format	Character string representation of a timestamp

Note: All numeric fields are in EXTERNAL format for LOAD. Field specifications of INTEGER or SMALLINT will be treated as if they were INTEGER EXTERNAL, specifications of DECIMAL, DECIMAL PACKED, or DECIMAL ZONED are treated as DECIMAL EXTERNAL, and specifications of FLOAT, REAL, or DOUBLE are treated as FLOAT EXTERNAL.

Delimited File - Restrictions

A delimiter cannot be binary zero

A default decimal point (.) cannot be a string delimiter

The default delimiter value differs by encoding scheme

- For ASCII / Unicode
  - Double quote (") for character string X'22'
  - Period (.) for decimal X'2E'
  - Comma (,) for column delimiter X'2C'
- For EBCDIC
  - Double quote (") for character string X'7F'
  - Period (.) for decimal X'4B'
  - Comma (,) for column delimiter X'6B'

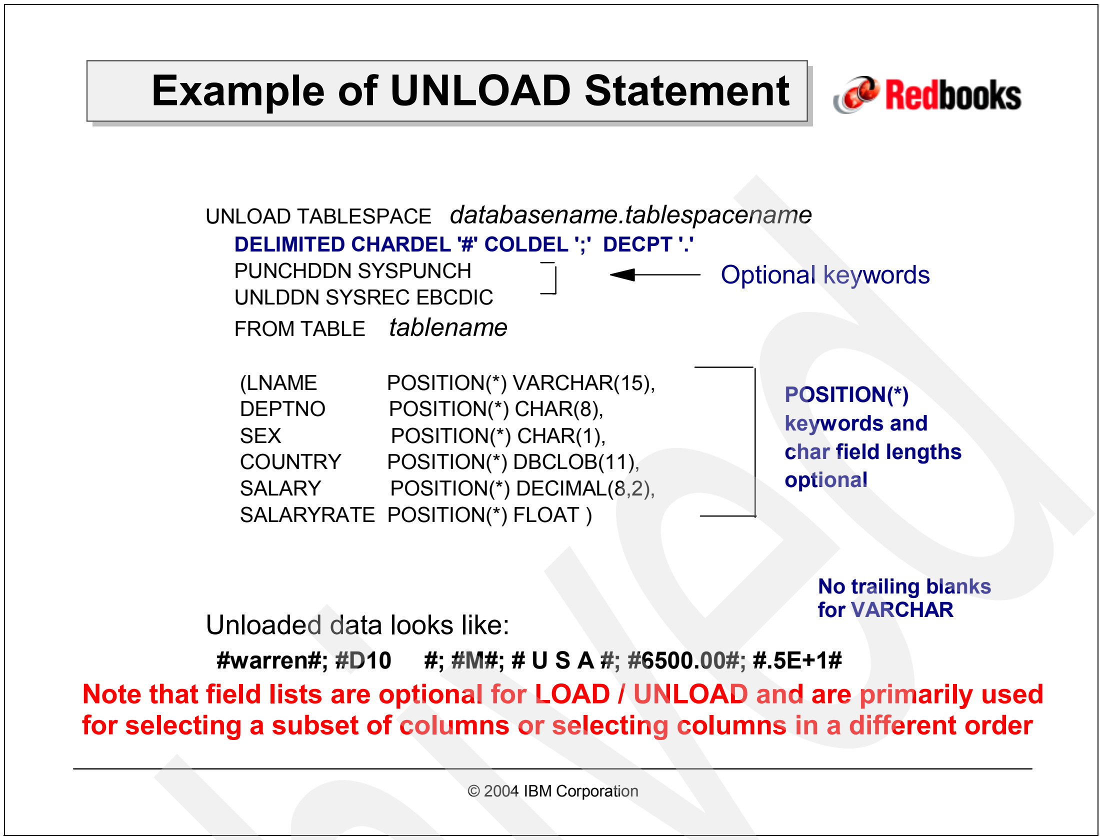
For Unicode input / output, data must be in CCSID 1208, UTF-8

The delimiter must be specified in code page of source / target data
It can be a character or hex constant

9.2.4 Delimited file - restrictions

Note that the default character used for DECPT cannot be specified as either CHARDEL or COLDEL, even if DECPT is specified with some other character.

In most EBCDIC code pages, the hex values that are specified on the visual are a double quotation mark(“) for the character string delimiter, a period(.) for the decimal point character, and a comma(,) for the column delimiter.



9.2.5 Example of UNLOAD statement

Field list is optional for LOAD/UNLOAD. If you don't specify it, the utility loads/unloads all valid columns from the table. Specifying a field list is primarily used to selectively load/unload columns and data in any order you choose.

When delimited is specified for UNLOAD, the NOPAD option is in effect for variable length column output, even if NOPAD is not specified. Therefore, trailing padded blanks are not used for variable length columns. For example, if a VARCHAR (100) field contains the data ABC in it, UNLOAD DELIMITED unloads it as “ABC”.

For fixed length columns, the normal padding rules applies. For example a CHAR(100) column containing ABC, UNLOAD DELIMITED unloads as “ABC <97 blanks>” (with 97 blanks actually following ABC).

The UNLOAD utility generates the delimited LOAD utility control statement in the data set specified in PUNCHDDN.

Delimited Input/Output Case Study

A spreadsheet has 54 cells in each row

Moving data from spreadsheet to DB2 for z/OS

- Save the data in comma separated value (CSV) format
- Upload the saved data to the host, either in binary (that is, no conversion of data), or in text (that is, data is converted)
- Create a DB2 table either in EBCDIC (default option), or in ASCII with the appropriate number of columns and data types
- Use LOAD utility with FORMAT DELIMITED specified

Moving data from DB2 to spreadsheet

- Use UNLOAD utility with DELIMITED specified
- Download the unloaded data from the host to spreadsheet

9.2.6 Delimited input/output case study

A Lotus® 1-2-3® spreadsheet (on a Windows machine) is used in this case study.

To move the data from the spreadsheet to DB2 for z/OS, the following steps are required:

- Save the spreadsheet data in comma separated value (CSV) format.
- Upload the saved data to the host in one of the following formats:
  - Binary (this means the data is stored in ASCII — because we are running the spreadsheet on in a Windows environment — as on the host, without any conversion).
  - Text (this means the data is converted from ASCII to EBCDIC before it is stored on the host).

  To upload the data, you can either use your terminal emulator's file transfer program, or use FTP.
- Create a DB2 table with the appropriate number of columns and data type (consistent with the number of cells and the type of data stored in the spreadsheet), either as an EBCDIC (default option), or an ASCII table.
- Use LOAD utility with FORMAT DELIMITED specified as follows:
  - If the spreadsheet data is uploaded to the host in text format:

    ```
    LOAD DATA INTO TABLE tablename
     FORMAT DELIMITED
    ```

  - If the spreadsheet data is uploaded to the host in binary format:

    ```
    LOAD DATA INTO TABLE tablename
     FORMAT DELIMITED ASCII
    ```

If you do not specify COLDEL, CHARDEL, and DECPT delimiters in the LOAD utility control statement, the data is moved to the table using the default values for COLDEL, CHARDEL, and DECPT delimiters according to the data type you specified in the LOAD utility control statement.

As mentioned before, the default character for COLDEL is a comma (,). In ASCII this is X'2C' and in EBCDIC this is X'6B'.

The default character for CHARDEL is a double quotation mark (“). In ASCII this is X'22' and in EBCDIC this is X'7F'.

The default character for DECPT is a period (.). In ASCII this is X'2E' and in EBCDIC this is X’4B’.

To move the data from DB2 to a spreadsheet, the following steps are required:

- Use the UNLOAD utility with DELIMITED specified as follows:
  - If you want the unloaded data to be in ASCII format:

    ```
    UNLOAD DATA FROM TABLE tablename
    ASCII DELIMITED
    ```

    The data and the delimiters are unloaded in ASCII. Transmit the data to the spreadsheet in binary format.
  - If you want the unloaded data to be in EBCDIC format:

    ```
    UNLOAD DATA FROM TABLE tablename
    EBCDIC DELIMITED
    ```

    The data and the delimiters are unloaded in EBCDIC. Transmit the data to the workstation to be used by the spreadsheet in text format.

RUNSTATS Enhancements

Non-uniform distribution statistics on non-indexed columns

Ability to update statistics history tables with the latest information without updating the statistics used by the optimizer

- Facilitates monitoring and analysis
- RUNSTATS TABLESPACE DB1.TS1
  UPDATE NONE HISTORY ALL

RUNSTATS with UPDATE NONE REPORT NO to invalidate dynamic SQL cache without collecting any statistics

9.3 RUNSTATS enhancements

When the DB2 optimizer can calculate more accurately the filter factor of the predicates of a query, this normally leads to better optimization choices, and helps to improve query performance.

RUNSTATS has been enhanced in Version 8 to gather the following additional statistics, allowing the optimizer to do a better job:

- Frequency value distributions for non-indexed columns or groups of columns
- Cardinality values for groups of non-indexed columns
- LEAST frequently occurring values, along with MOST for both index and non-indexed column distributions

This information could not be gathered by RUNSTATS in previous versions of DB2. More details are discussed in the performance chapter, more precisely in, 10.7, “The need for extra statistics” on page 762.

Version 8 relaxes the Version 7 requirement that history statistics can only be collected if the main catalog statistics (used by the optimizer) are also updated. This greater flexibility allows the user to keep track of statistics changes over time, without the concern that the main optimizer statistics change as well and can result in access path changes, especially for dynamic SQL. Thus, in Version 8, you can specify RUNSTATS UPDATE NONE HISTORY ALL. This way, only all statistics are collected (ALL), but only the history tables are updated (UPDATE NONE).

In Version 7, the easiest way to invalidate statements in the dynamic statement cache that reference a certain object is to use RUNSTATS REPORT YES UPDATE NONE for that object. However, using this statement, the statements in the dynamic statement cache referencing the RUNSTATSed object are invalidated, but RUNSTATS still goes out and collects all the statistics for the report.

In Version 8, you can also specify RUNSTATS object REPORT NO UPDATE NONE. This causes invalidation of those statements in the dynamic statement cache that reference the table space and/or index space specified in RUNSTATS utility control statement, but no statistics are gathered by RUNSTATS. Therefore, using RUNSTATS to invalidate statements in the dynamic statement cache is much cheaper. RUNSTATS will use hardly any CPU, as it only invalidates the referenced objects in the dynamic statement cache.

New Defaults for Better Performance

- RESTART is new default for Utilities (also in V7 with PQ72337)
- SORTKEYS is used by default for LOAD, REORG, and REBUILD
- SORTDATA is used by default for REORG
  - SORTDATA now allowed for 32K records with DFSORT
- REORG will use implicit clustering index
  - if no clustering index, first index defined is used
  - If table space has no indexes, SORTDATA operates as in pre-V8 releases

9.4 New utility defaults for better performance

In this section we discuss some changes to utility defaults, providing more usability and better performance of DB2 utilities.

9.4.1 Automatic utility restart

When a DB2 utility runs into a problem, it normally abends and the utility is put into a stopped state (when you do a -DIS UTILITY(utility-id)). After correcting the problem, you normally want to restart that DB2 utility (from where it left off, using a phase or current restart). Currently, users must modify their job control statements (JCL) or DSNUTILS stored procedure parameters, to indicate that a stopped DB2 utility is to be restarted.

DB2 V8 has been enhanced to allow DB2 for z/OS online utilities to be restarted without having to specify the RESTART, RESTART(PHASE) or RESTART(CURRENT) keyword. When a utility job is started, DB2 will check if it is an initial invocation or a restart, by looking into SYSUTIL for a matching utility-id:

- If it is an initial invocation and the RESTART keyword is specified, the RESTART keyword is ignored.
- If the utility needs to be restarted, and the RESTART keyword is present, the specified keyword is used.

- If the utility needs to be restarted, and no RESTART keyword is present, DB2 will set a default RESTART value based on the utility type, the utility phase, and the specified keywords. As a general rule, DB2 will try to do current restart whenever possible, or revert to phase restart in the cases where current restart is not possible. For more details, please refer to the *DB2 for z/OS Version 8 Utility guide and Reference*, SC18-7427.

Even though with this enhancement the RESTART keyword is optional, you still need to address the problem that initially caused the utility execution to stop, before resubmitting the utility job. For example, if the utility abended because of an out-of-space condition, you must make your data sets larger before restarting the utility.

You also have to make sure to use proper data set disposition parameters that allow for proper restart. To avoid data set disposition problems, the use of TEMPLATEs is highly recommended.

> **Tip:** This enhancement has also been made available through the PTF for APAR PQ72337. In DB2 V7 the implicit utility restart feature is controlled by a new DSNZPARM UTLRSTRT. The default is OFF. The DSNZPARM does not exist in V8. The feature is always enabled.

9.4.2 SORTKEYS and SORTDATA used by default in V8

Starting with Version 5, the SORTKEYS keyword has been added to the LOAD, REORG TABLESPACE and REBUILD INDEX utilities, and SORTDATA and NOSYSREC keywords to the REORG TABLESPACE utility. These keywords activate methods providing better performance. To achieve this better performance, it is necessary to explicitly specify these keywords in the utility control statements. (The only exception to this is REORG TABLESPACE SHRLEVEL CHANGE, which does not require you to specify the parameters SORTDATA, NOSYSREC, or SORTEYS. It always operates as if these keywords are specified.)

However, changing hundreds of utility jobs to add these keywords is impractical in many installations. Also, it was not obvious to new utility users that these keywords should be specified for optimal performance.

In Version 8, SORTKEYS is the default for LOAD, REORG TABLESPACE, and REBUILD INDEX, and SORTDATA is the default for REORG TABLESPACE. NOSYSREC is the default for REORG TABLESPACE (SHRLEVEL CHANGE) as in prior versions.

In Version 8, you can use SORTDATA for 32K records, which was not always possible in prior versions. Using SORTDATA requires adding the clustering key to the record to be sorted. In previous versions, if the row length is close to 32K, adding the clustering key to the sort record can lead to a record that is greater than 32K. You would receive a message (DSNU291I for REORG SHRLEVEL NONE, ignoring the SORDATA keyword, or DSNU294I for REORG SHRLEVEL REFERENCE or CHANGE, ending the utility execution).

Now that DFSORT™ supports larger sort records (provided that the proper maintenance PTF UQ57144 is installed), this is no longer a problem for DB2, and SORTDATA can be used on any 32K table space.

9.4.3 REORG's usage of the implicit clustering index

REORG TABLESPACE no longer requires an index defined CLUSTER YES (explicit clustering index) to order the data. If no clustering index exists, the first index created is used, as is the case for DB2 inserts. Along with this, the Version 8 online schema support provides the ability to alter the cluster attribute of indexes, with an ALTER INDEX statement. Refer to 3.11, "Online schema changes" on page 129 for more information on online schema evolution.

DFSORT is ALWAYS shipped with z/OS, even though you may not be licensed for it. DB2 V8 provides a "special" license for DB2 to use DFSORT, without the user needing to acquire an actual DFSORT license. In V8, users will always see DFSORT messages in DB2 utility outputs rather than other OEM messages. This enhancement allows DB2 to exploit particular functions and features of DFSORT without having to support a more generic sort interface. This means better sort performance and more robust DB2 utilities.

9.4.4 Review of SORTKEYS, SORTDATA, and NOSYSREC

The following paragraphs provide a review of the SORTKEYS, SORTDATA, and NOSYSREC keywords.

SORTKEYS

The SORTKEYS keyword is a performance related option that can improve performance of LOAD and REORG utilities by having impact during both of the following situations:

- The index key sort elapsed time — by reducing I/O and overlapping phases
- The index load elapsed time — by activating index load parallelism

When using SORTKEYS, during the index key sort, index keys are passed in memory rather than written to the SYSUT1 and SORTOUT work files. Avoiding this I/O to the work files improves REORG/LOAD performance. It also reduces disk space requirements for the SYSUT1 and SORTOUT data sets. Using the SORTKEYS option reduces the elapsed time from the start of the reload phase to the end of the build phase.

Of course, if the index keys are already in sorted order, or there are no indexes, SORTKEYS does not provide any advantage. Remember that if SORKEYS is activated and the job abends, during the reload, sort, or build phase, it will always need to restart from the beginning of the reload phase.

More information on the usage and the performance of SORTKEYS for this functionality is reported in the standard DB2 manuals and in the redbooks *DB2 for OS/390 Version 5 Performance Topics*, SG24-2213, and *DB2 for z/OS and OS/390 Version 7 Utilities Suite*, SG24-6289.

You can reduce the elapsed time of a LOAD job for a table space or partition with more than one defined index by invoking parallel index build. We have seen that DB2 V5 introduced the SORTKEYS option to eliminate multiple I/Os to access the keys that are needed to build the indexes. The keys are passed in storage to the sort process, and then directly to the build phase. But, since there is only a single sort and build subtask, the indexes are built serially.

With DB2 V6, when SORTKEYS is specified, DB2 provided multiple pairs of sort and build subtasks so that indexes are built in parallel, thereby improving the elapsed time of LOAD and REORG.

You can use dynamic allocation (SORTDEVT and SORTNUM keywords) to allocate the sort work data sets, or you can allocate them by specifying the DDNAMEs in the form *SWnnWKmm*, where *nn* is the subtask pair number and *mm* is the number of data sets for that subtask pair. Using manual allocation of sort work data sets is a way to control and limit the amount of parallelism, by restricting the number of these data sets.

More information on the usage and the performance of SORTKEYS for this functionality is reported in the standard DB2 manuals and in the redbook *DB2 for OS/390 Version 6 Performance Topics*, SG24-5351.

SORTDATA

The SORTDATA parameter of the REORG utility invokes the execution of an (external) sort (no DB2 sort) on the data that are columns of the clustering index. This allows a consistent execution time for REORG, and better performance. Without the SORTDATA keyword, the data is unloaded through the clustering index. This means that the more the data is disorganized, the better REORG using SORTDATA will perform, since going through the clustering index will take progressively longer with more disorganized rows.

NOSYSREC

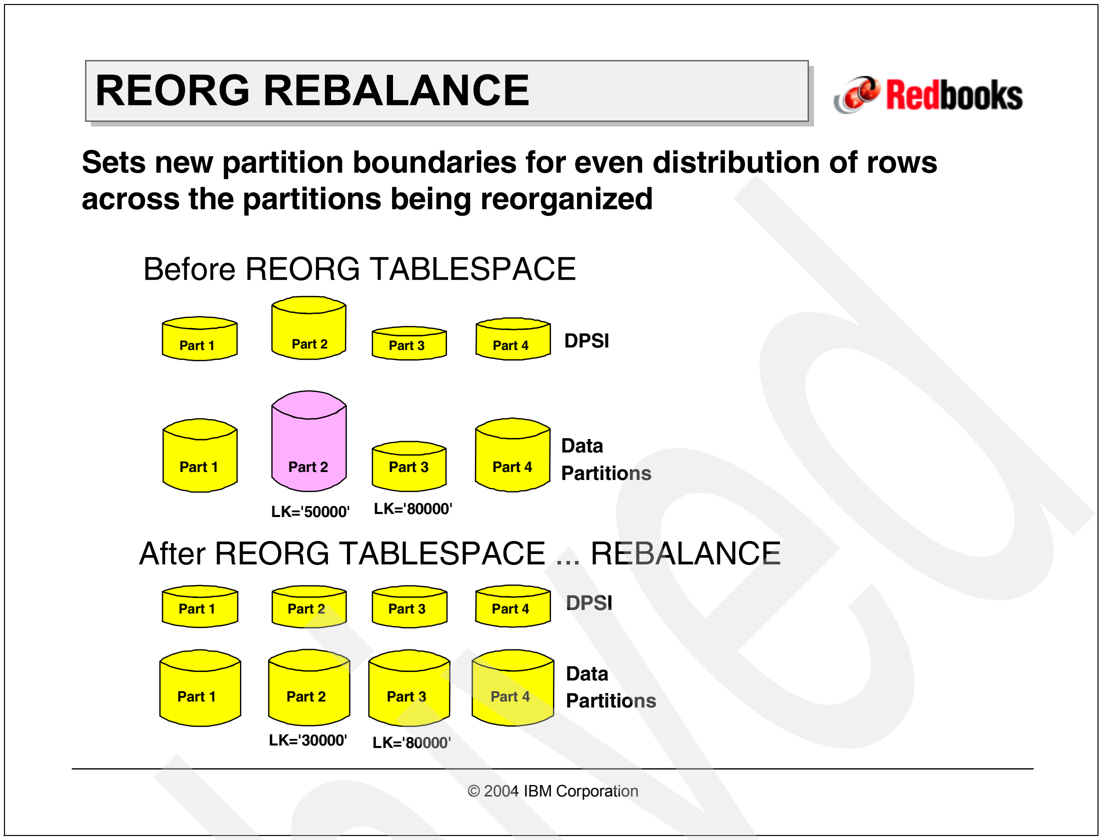
After unloading and sorting the rows by the *UNLOAD* phase, the rows of the table space are normally contained in the unload data set (*SYSREC*). The next phase of the *REORG* utility, the *RELOAD* phase, must retrieve the rows (again) from the unload data set to load them back into the table space. The storing and retrieval of the unloaded rows into and from the SYSREC data set requires I/O operations and raises the question if the two phases could not better cooperate and avoid the intermediate storage of the rows.

The intermediate storage of the rows can be avoided by specifying *NOSYSREC* for the *REORG* utility.

As a consequence of *NOSYSREC*, the *UNLOAD* phase does not use the unload data set (SYSREC). Rows that must be sorted are passed to the *RELOAD* phase via (DF)SORT exits, after they have been sorted by DFSORT (or an equivalent sort utility). Rows that do not need to be sorted, can immediately be passed to the *RELOAD* phase.

NOSYSREC eliminates the I/O for the unload data set. It does not cause additional I/O by DFSORT (or an equivalent sort utility). The fact that different sort work data sets are used does not change the I/O behavior of DFSORT. Thus, *NOSYSREC* represents a true performance improvement.

However, you must be aware that the *REORG* utility **cannot** be restarted if *NOSYSREC* has been specified.



9.5 REORG REBALANCE

In this section we discuss the new REBALANCE option that you can use to have REORG automatically rebalance the data in your partitions.

9.5.1 Rebalancing overview

Since DB2 V6, you can alter the partition boundaries of a partitioned table space by using the ALTER INDEX index-name PART x VALUES (‘new-limit-key’) SQL statement. This puts partition x and the next partition in REORG- pending (REORP) state. A subsequent execution of the REORG utility redistributes the rows between both partitions according to the new limit-key.

In V8, the REORG TABLESPACE utility has a new keyword, *REBALANCE*, which indicates that the rows in the table space or the partition ranges being reorganized, should be evenly distributed for each partition range when they are reloaded.

REBALANCE specifies that REORG TABLESPACE should set new partition boundaries so that all the rows participating in the reorganization are evenly distributed across the partitions being reorganized. The SYSTABLEPART and SYSINDEXPART tables are updated during this process so that they contain the new limit key values. This way, you no longer have to figure out what value to specify as the new limit-key to obtain an even distribution among the partitions involved.

What Does Rebalance Do?

1. **Unload rows from the table space or partition range**
2. **Sort rows by partitioning column(s) and divide by number of parts**
   - Is not perfect if lots of duplicate keys exist
3. **Reload the data**
4. **Update limit key values in the catalog**
5. **Invalidate plans, packages and dynamic statement cache**

When clustering does not match partitioning, REORG must be run twice:

- First to move rows to the right partition
- Second to sort in clustering sequence

9.5.2 What does rebalance do?

As during normal REORG, the data is unloaded. The data is sorted by the partitioning column(s) of the partitioned table space. (Remember that in V8, that partitioning columns no longer have to be supported by an index. Also, the partitioning index does not have to be the clustering index in V8).

Before reloading the data, REORG TABLESPACE with REBALANCE calculates the approximate number of pages expected to be populated across all the partitions, taking into account the percent free space allowed on each page as well as the free pages specified (as that can differ between partitions). During reload, REORG TABLESPACE notes the value of the partitioning columns as each partition reaches the page count threshold. At the end of a successful REORG, the catalog and directory are updated with the new partition boundaries.

Perfect rebalancing is not always possible if the columns used in defining the partition boundaries have many duplicate values within the data row. As all keys with a certain value have to go into the same partition, a key with many duplicates can lead to a partition that is bigger than the other partitions.

You cannot specify the keyword REBALANCE together with SHRLEVEL CHANGE, nor with SCOPE PENDING, OFFPOSLIMIT, INDREFLIMIT, REPORTONLY, UNLOAD ONLY, and UNLOAD EXTERNAL keywords.

When using REBALANCE on a table where the clustering index does not match the partitioning key, REORG must be run twice on the partition range to ensure that the rows are in optimal clustering order. The first reorganization (with the REBALANCE option) moves data rows to the appropriate partition.

After the first reorganization, the table space is put into an AREO* state to indicate that another reorganization is recommended. The second reorganization (without the REBALANCE keyword) orders each data row in clustering order (based on the partitioning index) within the appropriate partition.

You cannot specify the REBALANCE option for partitioned table spaces with LOB columns.

> **Note:** The partition range that you specify on the REORG REBALANCE statement is a range of physical partitions. When rebalancing, DB2 unloads, sorts, and reloads the data based on logical partition numbers. If, because of earlier partition rotations or adding additional partitions, logical and physical partitions no longer match up, REORG REBALANCE may not be possible. When reorganizing, the physical partition numbers associated with the logical partition number that DB2 uses to reload data into, must be within the physical partition range that you specify on the REORG REBALANCE statement; otherwise you receive a DSNU1129I message, and REORG terminates.

REORG Rebalance & ALTER + REORG Comparison

Redbooks

REORG TABLESPACE ... REBALANCE works out new partitioning values for you

- One step automated process
- Supported with SHRLEVEL REFERENCE and NONE
- Data available (read only) almost all of the time
- Not supported for table spaces with LOB columns

ALTER INDEX or ALTER TABLE ALTER PART

- Gives you more control to allow for future skewed growth
- Leaves affected partitions in REORP
- Data unavailable until REORG completes

9.5.3 REORG REBALANCE and ALTER + REORG comparison

Assume that a table space that contains a transaction table named TRANS is divided into 10 partitions, and each partition contains one year of data. Partitioning is defined on the transaction date, and the limit key value is the end of the year.

Manually changing the partition boundaries

Assume that the year 2003 resulted in more data than was projected, so that the allocation for partition 10 almost reaches its maximum of 4 GB. The year 2002, on the other hand, resulted in less data than was projected. Therefore, we want to change the boundary between partition 9 and partition 10 so that some of the data in partition 10 becomes part of the data in partition 9.

To change the boundary, issue the following statement:

```
ALTER TABLE TRANS ALTER PART 9 VALUES (03/31/2003);
```

Now the data in the first quarter of the year 2003 will be part of partition 9. The partitions on either side of the new boundary (partitions 9 and 10) are placed in REORG-pending (REORP) status and are not available until the partitions are reorganized.

Note that we use a table controlled partitioned table space in this example. When using an index controlled partitioned table space, you could have used this statement as well:

```
ALTER INDEX IXTRANS PART 9 VALUES(03/31/2003);
```

Using REORG REBALANCE to spread data

Alternatively, you can rebalance the data in partitions 9 and 10 by using the REBALANCE option of the REORG utility:

```
REORG TABLESPACE dbname.tsname PART(9:10) REBALANCE
```

This method avoids putting the partitions in a REORP state, and making the data unavailable for applications until the REORG has completed. When you use REORG SHRLEVEL REFERENCE to rebalance the partitions, the data is available for readers almost all of the time. Only during the switch phase, the data is not available. It is also during the switch phase that DB2 updates the limit keys to reflect the new partition boundaries.

Note: When altering the partition boundaries manually, it is your job to determine the new partition boundaries. When using REBALANCE, DB2 will figure out the new partition boundaries for you.

REORG TABLESPACE - SCOPE PENDING

Redbooks

Reorganizes only the table space part(s) that are in REORG-pending (REORP) or advisory REORG-pending state (AREO*)

When specifying a partition range, the adjacent high and low parts that are not included in the range must not be in REORP

SYSCOPY records are only written for those partitions that are actually reorganized

9.6 REORG TABLESPACE - SCOPE PENDING

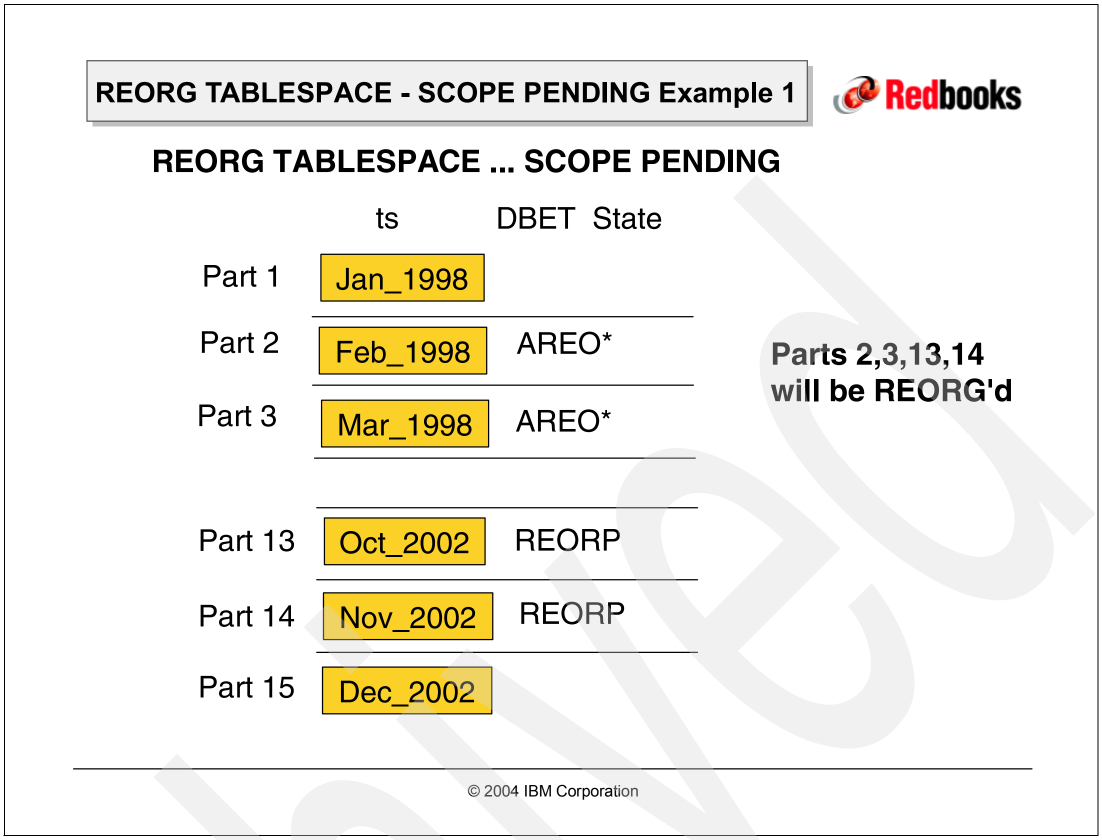
As mentioned before, many enhancements have been implemented across various utilities to improve utility usability. This is another example.

REORG TABLESPACE is extended with the new keyword *SCOPE* to indicate the scope of the reorganization for the table space or partition range. The default is SCOPE ALL, which results in the reorganization of the entire table space or the partition range. When you specify SCOPE PENDING, you indicate that only the partitions in a REORP or AREO* state for a specified table space or partition range are to be reorganized.

If you want to reorganize a range of partitions and specify SCOPE PENDING, make sure that the adjacent partitions, outside the specified range, are not in a REORP state. The REORG terminates with an error otherwise.

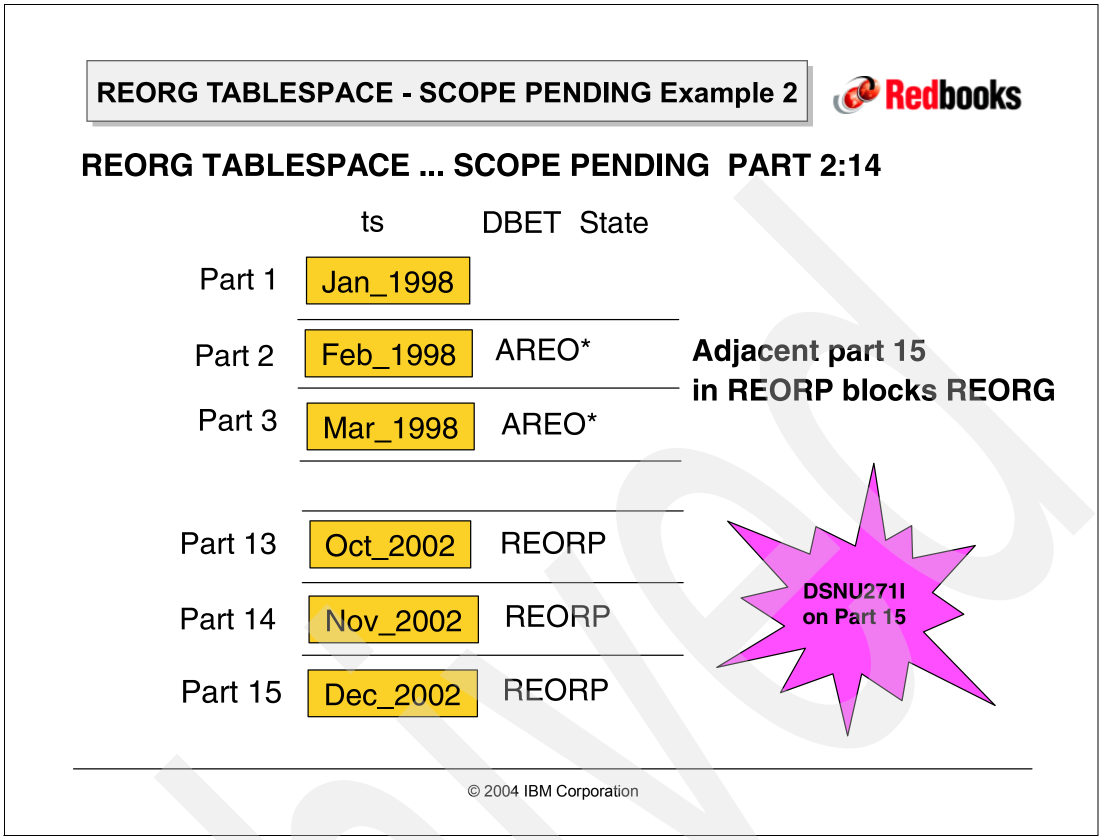
Rows are inserted into the SYSCOPY catalog table only for those partitions that are reorganized.

You cannot specify SCOPE PENDING together with the REBALANCE, OFFPOSLIMIT, INDREFLIMIT, REPORTONLY, UNLOAD ONLY, or UNLOAD EXTERNAL keywords.



9.6.1 REORG TABLESPACE - SCOPE PENDING example 1

In this example, SCOPE PENDING causes only partitions 2, 3, 13, and 14, which are either in REORP or AREO* state, to be reorganized.



9.6.2 REORG TABLESPACE - SCOPE PENDING example 2

In this example, partitions 2, 3, 13, 14, and 15 are either in REORP or AREO* state. Specifying REORG TABLESPACE ... SCOPE PENDING would cause all these partitions to be reorganized. However, we specify SCOPE PENDING, PART 2:14 instead. Since the adjacent partition 15 is not included in the partition range to be reorganized (2:14), but has a REORP state, REORG terminates with DSNU271I message and a return code of 8.

```
DSNU271I ... REORG PENDING ON FOR TABLE SPACE ...PART 15 PROHIBITS PROCESSING
```

The message indicates that an attempt was made to execute a REORG utility to redistribute data in a partitioned table space. The partition number stated in the message was found to have the REORP state on, but was not specified on the PART 2:14 partition range parameter of the REORG utility.

It is of course not required that you should reorganize all partitions that are in AREO* or REORP state at the same time. For example, there are two ranges, partition range 2 to 3 and partition range 13 to 15 which are in AREO* or REORP state. You can specify only the first range, for example REORG TABLESPACE ... SCOPE PENDING PART 2:3, without reorganizing partition range 13 to 15 at the same time.

REORG TABLESPACE ... DISCARD

Now supported by REORG ... SHRLEVEL CHANGE

During the window when discarding the data rows in REORG, these data rows cannot be modified

If a data row that matches the discard criteria gets updated while REORG is in the process of discarding, REORG stops with RC=8 and a DSNU1127I message

9.7 REORG TABLESPACE ... DISCARD

Since DB2 V5, you can specify the DISCARD keyword as part of the REORG TABLESPACE utility control statement, to indicate that rows that meet the specified WHEN conditions should be discarded during the reorganization. However, specifying the DISCARD keyword is only allowed with REORG SHRLEVEL NONE and SHRLEVEL REFERENCE.

In Version 8, you can also specify the DISCARD keyword with REORG SHRLEVEL CHANGE. However, if you use discard processing with a SHRLEVEL CHANGE REORG, while DB2 is discarding data rows (during the UNLOAD phase), modifications to data rows that match the discard criteria are not permitted. When REORG TABLESPACE encounters such a situation, the utility execution is terminated with an error (condition code 8) and a DSNU1127I message.

REBUILD INDEX - SCOPE PENDING

Redbooks

Rebuilds only the index or part(s) that are in rebuild-pending (RBDP), recovery pending (RECP), or advisory REORG pending (AREO*) state

Unlike REORG TABLESPACE, the adjacent high and low parts not included in the range are not checked for RBDP

You can specify the index space name, instead of the index name in the following utilities:

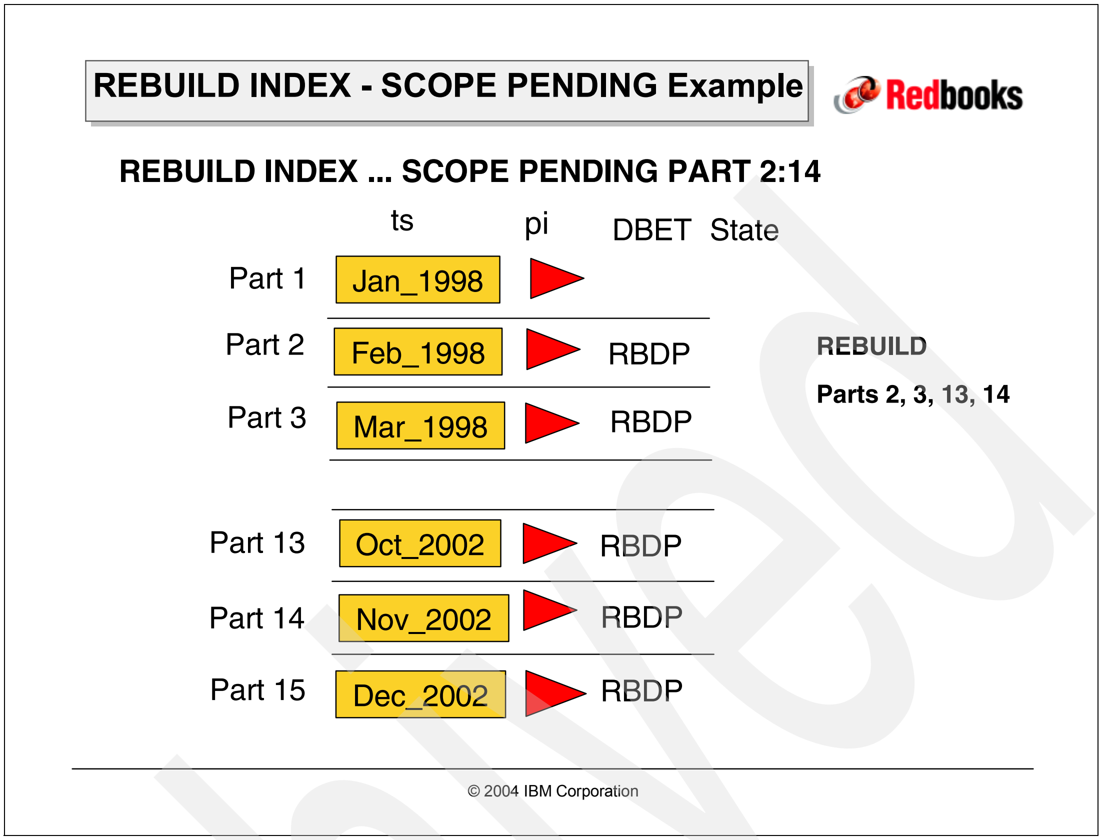
- REBUILD INDEX ... or REBUILD **INDEXSPACE** ...
- REORG INDEX ... or REORG **INDEXSPACE**...
- RECOVER INDEX ... or RECOVER **INDEXSPACE** ...

9.8 REBUILD INDEX - SCOPE PENDING

In DB2 V8, similar to REORG, the REBUILD INDEX syntax has been extended with the new *SCOPE* keyword, to indicate the scope of the rebuild. The default is SCOPE ALL, which results in the reorganization of all the specified indexes. You specify SCOPE PENDING to indicate that the specified indexes should be rebuilt only if they are in a RBDP, RECP, or AREO* state.

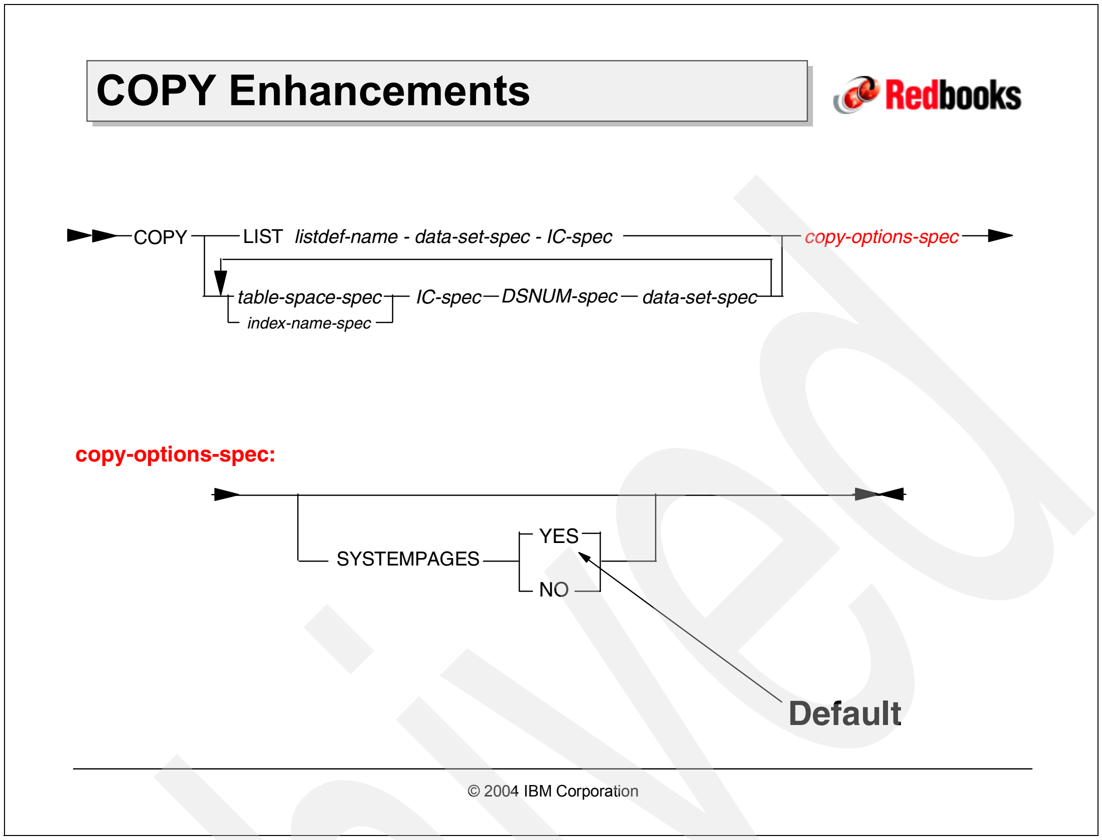
Unlike REORG TABLESPACE, the adjacent high and low parts not included in the range are not checked for the RBDP state.

In V8, REBUILD INDEX also accepts index space names, instead of just index names as the object name to be rebuilt. By the way, this enhancement also applies to RECOVER index and REORG index.



9.8.1 REBUILD INDEX - SCOPE PENDING example

In this example, we use the SCOPE PENDING keyword in the REBUILD INDEX utility. The result is that only the partitions 2, 3, 13, and 14, which are in RBDP state, are rebuilt. Note that, unlike the use of SCOPE PENDING with REORG, REBUILD INDEX does not care that partition 15 is in RBDP, but was not included in the range of partitions to rebuild.



9.9 COPY enhancements

In V8, the DB2 COPY utility (full image copy and incremental image copy) is extended with a new keyword called *SYSTEMPAGES*. You can specify either YES (this is the default) or NO.

The SYSTEMPAGES options applies to both table spaces and indexes (defined with COPY YES).

The visual shows the syntax.

COPY Utility SYSTEMPAGES Option

Indicates whether the dictionary and version system pages are copied at the beginning of the object to be copied. Especially important when

- Copying a piece, or a single data set of a table space or index
- Using incremental image copy and those system pages have not changed

With SYSTEMPAGES YES

- Dictionary pages for the compression dictionary are included
- V8 system pages that contain version information are included
- Both are included at the beginning of the image copy
- Version pages can occur multiple times in the image copy

With SYSTEMPAGES NO

- Copy pages as they appear in the object (pre-V8)

Header page is always included no matter what SYSTEMPAGES option is used

Note that for SYSTEMPAGES YES, the UNLOAD utility can process image copies with data versioning

9.9.1 COPY utility SYSTEMPAGES option

System pages are pages within a table space that describe the data in that table space, partition, or index.

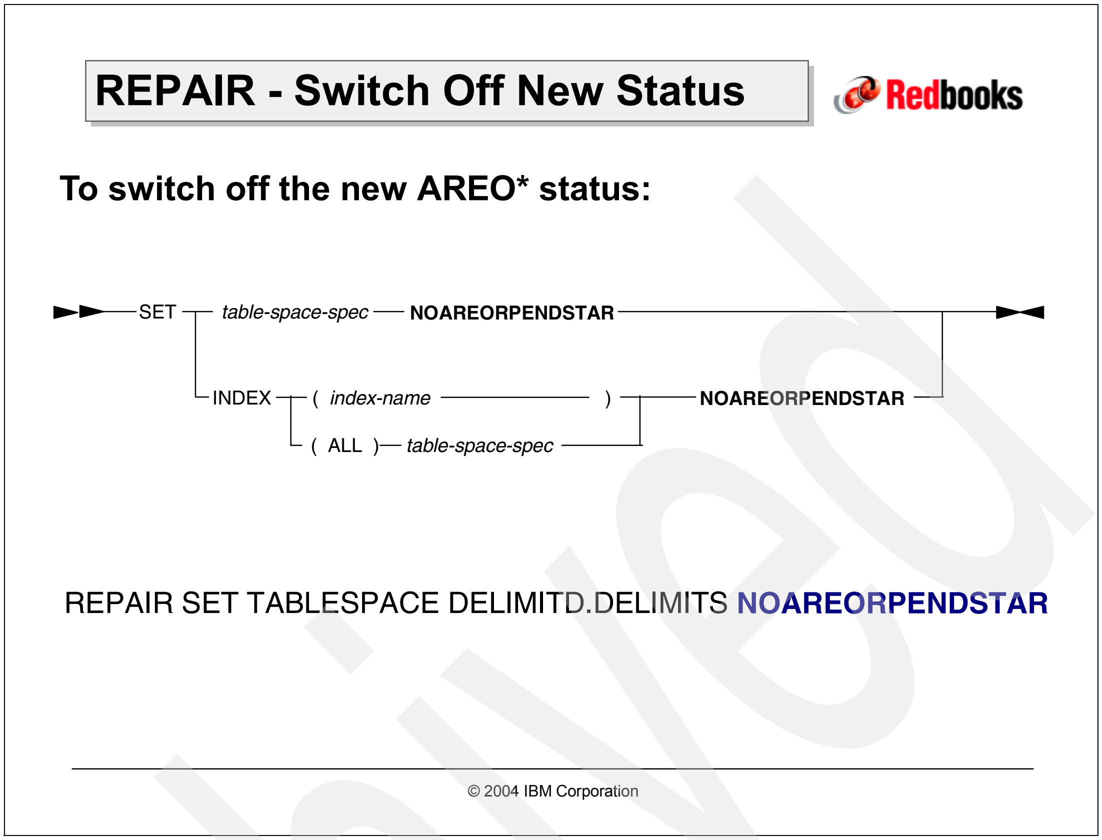
Specifying SYSTEMPAGES YES ensures that any header, dictionary, and version system page is copied at the beginning of the image copy data set. Selecting YES ensures that the image copy contains the necessary system pages for subsequent UNLOAD utility jobs, which want to unload data from an image copy, to correctly format and unload all data rows.

This is especially important when using incremental image copies, in cases where the system pages have not changed, or when copying only a single data set or piece. Specifying SYSTEMPAGES YES guarantees that the image copy contains all system pages in the image copy.

SYSTEMPAGES NO does not ensure that the dictionary and version system pages are copied at the beginning of the image copy data set. The COPY utility copies the pages in the order they occur in the table space or index, including the header pages.

Irrespective of the SYSTEMPAGES option, in V8, the header page is always included in the image copy.

The CHECKPAGE option also validates system pages.

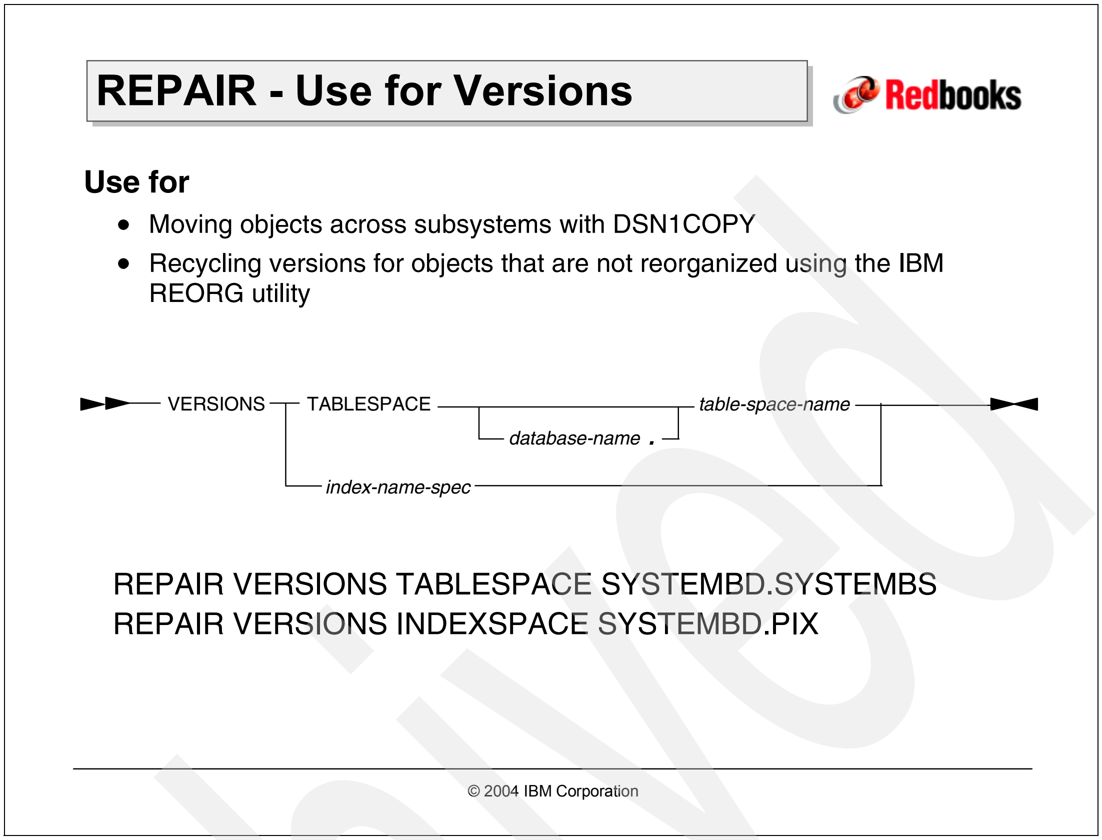


9.10 REPAIR

In this section we discuss the changes to the REPAIR utility.

9.10.1 To switch off new pending status

The REPAIR utility has been enhanced to reset the new advisory REORG pending state (AREO*) for table spaces and index spaces. The new keyword introduced to accomplish this is *NOAREORPENDSTAR*. For more information about AREO*, see 3.15, “Reviewing DBET states used by online schema evolution” on page 189.



9.10.2 REPAIR - use of versions

REPAIR *VERSIONS* updates the version information in the catalog and directory from the information in the table space or index. Use REPAIR VERSIONS when you perform the following tasks:

- After you use DSN1COPY to move objects from one system to another, or within a subsystem.
- As part of version number management for objects that do not use the IBM REORG utility. It depends on how the non-IBM utilities handle versions and what information is recorded in the DB2 catalog tables. You may have to use the REPAIR VERSIONS to update the versions in the catalog and directory in such instances.

REPAIR VERSIONS also writes a SYSCOPY record with an STYPE value of 'V' that the MODIFY utility can use for reclaiming version numbers. In the SYSCOPY entry, the OLDEST_VERSION column contains the lowest version found within the active object.

DSN1COPY processing with versions

Redbooks

1. **Ensure object definitions are the same in source and target**
2. **REORG if necessary**
3. **Ensure there are enough versions available on the target**
4. **Run DSN1COPY with OBIDXLAT**
5. **Run REPAIR VERSIONS on target object**
   - Updates CURRENT_VERSION with
   - MAX(**target.CURRENT_VERSION, source.CURRENT_VERSION**)
   - Updates OLDEST_VERSION with
     MIN(**target.OLDEST_VERSION, source.OLDEST_VERSION**)

9.10.3 DSN1COPY processing with versions

When objects are moved from one system to another and contain version system pages, the version information on the target system's catalog must match the source versions in the physical objects for the data to be accessible. You should follow the process outlined below:

1. Ensure that the current object definitions on the source and target systems are defined the same. For table spaces, each table must have the same number of columns, and each column must be of the same data type. This includes things like compression and SEGSIZE as well. In addition, if a non-version generating ALTER TABLE ADD COLUMN has been executed, an ALTER TABLE ADD COLUMN must also be performed on the target object. (This is no different from V7 DSN1COPY processing.)

   Indexes that are copied may or may not have been altered in V8. If not altered in V8, CURRENT_VERSION and OLDEST_VERSION contain zeros for both the source and target systems.

2. Reorganize any object that has OLDEST_VERSION of 0 and CURRENT_VERSION that is greater than 0 so that the versions match. This is to remove so-called V0 rows from the source table. Their description is not in the system pages, but in SYSOBDS. However, when copying between subsystems using DSN1COPY, only the physical data set is copied, not the SYSOBDS information.
3. Ensure that there are enough versions available on the target. For a table space, the combined active number of versions for the object on both the source and target systems must be less than 255. For an index, the combined active number of versions must be less than 16.

The active number of versions can be calculated as follows:

For the object on both source and target systems:

If the CURRENT_VERSION is less than the OLDEST_VERSION,
add the max number of versions (255 or 16) to CURRENT_VERSION. (This is done to take into account the fact that you just cycled passed the maximum version number).

Calculate the number of active versions as follows:

```
#active_versions = MAX(target.CURRENT_VERSION,source.CURRENT_VERSION) -
                   MIN(target.OLDEST_VERSION,source.OLDEST_VERSION) + 1;
```

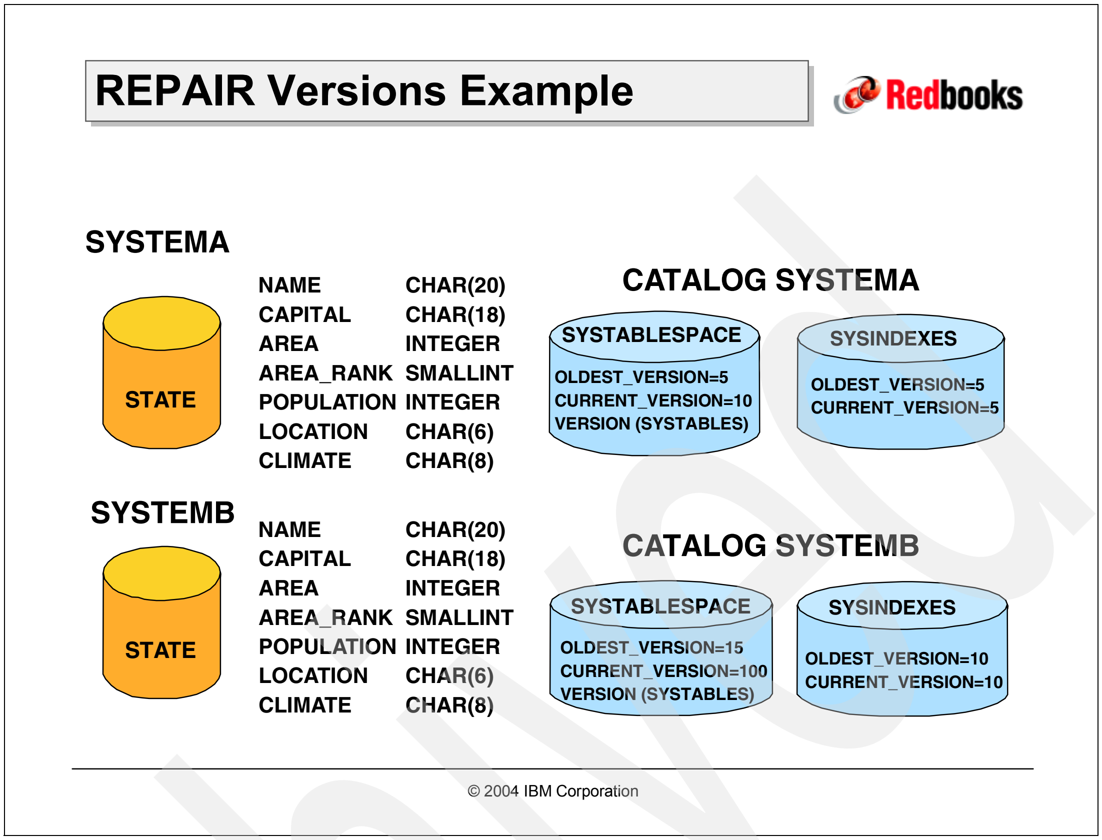
If the number of active versions is too high, first reorganize the entire source and target objects, take image copies, and run MODIFY to reclaim versions.

4. Run DSN1COPY, most likely using the OBIDXLAT option, specifying the proper mapping of table OBIDs from the source to the target system.
5. When on the target system, run REPAIR VERSIONS specifying the object which was copied over.
   - For table spaces, this utility updates:
     - OLDEST_VERSION and CURRENT_VERSION in SYSTABLESPACE
     - OLDEST_VERSION in SYSTABLEPART
     - VERSION in SYSTABLES.
   - For indexes, this utility updates:
     - OLDEST_VERSION and CURRENT_VERSION in SYSINDEXES
     - OLDEST_VERSION in SYSINDEXPART.

   REPAIR uses the following formulas to update the version numbers:

```
CURRENT_VERSION = MAX(target.CURRENT_VERSION,source.CURRENT_VERSION)
OLDEST_VERSION  = MIN(target.OLDEST_VERSION,source.OLDEST_VERSION)
```

More information about versioning can be found in 3.12, “Versioning” on page 156.



9.10.4 REPAIR VERSIONS example

Notice that the current object definitions on both the target and source systems SYSTEMA and SYSTEMB are defined the same.

Utility Changes to Support Informational RI Constraints

LOAD and CHECK DATA do not check informational RI constraints

REPORT TABLESPACESET also reports all table spaces related to the named table space through informational RI constraints

QUIESCE TABLESPACESET also quiesces all table spaces related to the named table space through informational RI constraints

LISTDEF also includes all table spaces related through informational RI constraints when the keyword RI is specified

9.11 Utility changes to support informational RI constraints

Informational RI is introduced in DB2 Version 8 as part of the Materialized Query Tables implementation. It allows you to define referential integrity (RI) relationships that exist in the DB2 catalog, but are not enforced by DB2. You do so by defining foreign key relationships with the *NOT ENFORCED* keyword. For more details, see “Informational RI syntax” on page 730.

Example 9-1 illustrates an informational RI constraint.

Example 9-1 Defining informational RI

```
CREATE DATABASE INFORIDB;
CREATE TABLE INFORIT1(C1 INTEGER NOT NULL PRIMARY KEY,
                      C2 INTEGER)
IN DATABASE INFORIDB;
CREATE UNIQUE INDEX INFORIX1 ON INFORIT1(C1);
CREATE TABLE INFORIT2(C1 INTEGER,
                      C2 INTEGER REFERENCES INFORIT1 NOT ENFORCED)
IN DATABASE INFORIDB;
INSERT INTO INFORIT1 VALUES(10,20);
INSERT INTO INFORIT1 VALUES(30,40);
INSERT INTO INFORIT2 VALUES(50,60);
INSERT INTO INFORIT2 VALUES(70,80);
```

The INSERT statements run successfully because of the keyword NOT ENFORCED.

Instead of using INSERT statements, the LOAD utility can also be used. A LOAD will also run successfully since the LOAD utility does not check informational RI constraints.

Note that if the keyword ENFORCED is specified (the default), the INSERT statements and LOAD utility check for the RI constraints, and our program would not have run successfully, because no primary key entry exists in INFORIT1 for the values 60 and 70.

Now we add a check constraint on each of these two tables as follows:

```
ALTER TABLE INFORIT1 ADD CHECK (C1 > 1)
ALTER TABLE INFORIT2 ADD CHECK (C1 > 1)
```

This causes the table spaces for the two tables to be placed in CHKP state.

Run the CHECK DATA utility with the following control statement:

```
CHECK DATA TABLESPACE INFORIDB.INFORIT1
           TABLESPACE INFORIDB.INFORIT2
```

The CHECK DATA utility runs successfully and resets the CHKP state because it does not check for informational RI constraints. In this case, CHECK DATA only looks at the check constraint, and all rows are OK with that.

When you run the REPORT utility with the following control statement:

```
REPORT TABLESPACESET TABLESPACE INFORIDB.INFORIT1
```

The REPORT utility reports about both tables INFORIT1 and INFORIT2, taking informational RI into account.

You can run the QUIESCE utility with the following control statement:

```
QUIESCE TABLESPACESET TABLESPACE INFORIDB.INFORIT1
```

This establishes the quiesce point for both the table spaces for the two tables.
An application that does the RI checking inside that application can still take advantage of defining informational RI in DB2. Defining informational RI allows you to easily quiesce related objects. Using informational RI and QUIESCE TABLESPACESET gives you an easy way to quiesce a set of application related objects, without having to track them individually and making sure you include all of them in the QUIESCE utility.

LISTDEF can also exploit informational RI. You can run the LISTDEF utility with the following control statement:

```
LISTDEF LIST1 INCLUDE TABLESPACE INFORIDB.INFORIT1 RI
```

And LISTDEF includes the table spaces for the two tables in the generated list.

Utility Changes to Support DPSI - 1

CHECK DATA
- When run against entire partitioned table space scans DPSI, extracts keys, and sorts keys
- When run against a partition scans partition of DPSI corresponding to table partition, extract keys, and skips sort

CHECK INDEX
- When run with PART keyword checks specified partition of DPSI

COPY
- Supports specifying a partition of DPSI with DSNUM keyword

LISTDEF -
- PARTLEVEL keyword specifies the partition granularity for partitioned objects has been extended to DPSIs in V8

TEMPLATE
- Templates created for DPSIs may wish to make use of the &PA OBJECT variable

QUIESCE
- Drain classes and restrictive states for DPSIs mirror PIs:
- WRITE YES: DPSI partitions are DW / UTRO
- WRITE NO: No drains or restrictive states on DPSI partitions

QUIESCE
- NPSIs are DW / UTRO during WRITE YES for table space/partition

9.12 Utility changes to support DPSIs

In most cases DPSIs are treated like partitioning indexes in V7 (or data-partitioned partitioning indexes in V8). We now look at each of the DB2 utilities in more detail.

9.12.1 CHECK DATA

The SCANTAB phase of CHECK DATA extracts foreign keys:

- When run against an entire partitioned table space, if the foreign key index is a data-partitioned secondary index (DPSI), the index is scanned and foreign keys extracted. The extracted foreign keys are then sorted in a SORT phase, before the keys can be used to check for a matching primary key in the parent table.
- When run against a partition of a partitioned table space, if the foreign key index is a data-partitioned secondary index, the partition of the index corresponding to the target data partition is scanned and the foreign keys are extracted. The SORT phase is skipped, as it is not required. All extracted keys are in the correct order since they only came from a singe part of the DPSI.

9.12.2 CHECK INDEX

If the PART keyword is specified with the CHECK INDEX utility, and the index is a data-partitioned secondary index, the specified physical index partition is tested for consistency.

9.12.3 COPY

The DSNUM keyword of the COPY utility may specify a partition of a data-partitioned secondary index (provided that the index is copy-enabled, of course).

9.12.4 LISTDEF

The PARTLEVEL keyword of the LISTDEF utility statement specifies the partition granularity for partitioned objects in LISTDEF's list of objects. With the introduction of data-partitioned secondary indexes, this keyword now extends to those objects as well. The keyword continues to be ignored for non-partitioned objects.

9.12.5 TEMPLATE

This utility statement is unaffected by the introduction of DPSIs, but note that templates created for DPSIs may wish to make use of the &PA OBJECT variable.

9.12.6 QUIESCE

The drain classes and restrictive states used for DPSIs mirror those of PIs. That is, pertinent partition(s) are DW/UTRO during QUIESCE WRITE YES, and there are no drains or restrictive states during QUIESCE WRITE NO against the DPSI parts. For non-partitioned secondary indexes (NPSIs), the entire index is DW/UTRO during QUIESCE WRITE YES operations against the table space or any partition of the table space.

DW (Drain Writers) means draining the write class, permitting concurrent access for SQL readers, but not allowing any updaters to access the object. UTRO represents the utility restrictive state allowing read only access on the target object.

Utility Changes to Support DPSI - 2

Redbooks

LOAD

LOAD loads records into tables and builds or extends any indexes defined on the tables.

- SORT phase is skipped if all of the following apply:
  - No more than one key per table
  - All keys are the same type:
    - Index key
    - Indexed foreign key
    - Foreign key
  - Data is loaded in key order
  - Data being loaded is grouped by table and each input record is loaded into one table only

9.12.7 Changes to LOAD for data-partitioned secondary indexes

The LOAD utility loads records into tables and builds or extends any indexes defined on the tables accordingly. LOAD proceeds in a series of phases. The sort phase is the only phase impacted by the data-partitioned secondary indexes.

SORT PHASE

During the sort phase, temporary file records are sorted in preparation for index maintenance or referential constraint enforcement, if indexes or foreign keys exist. The SORT phase is skipped if all of the following apply:

- There is not more than one key per table
- All keys are of the same type:
  - Index key only
  - Indexed foreign key
  - Foreign key only
- The data being loaded is in key order (if a key exists):
  - If the key in question is an index key only and the index is a data-partitioned secondary index, the data is considered to be in order if the data is grouped by partition, and ordered within partition by key value.
  - If the key in question is an indexed foreign key and the index is a data-partitioned secondary index, the data is never considered to be in order.
- The data being loaded is grouped by table and each input record is loaded into one table only.

Because of the changes to the SORT phase, the calculations to estimate the size of LOAD's work data sets are modified.

LOAD's work data sets:

When calculating the size of work data sets for LOAD, you have to perform the following calculations:

- Calculating the key, **k**
- Calculating the number of keys extracted

The presence of data-partitioned secondary indexes on any table targeted by a given LOAD influences both these calculations.

Calculating the key, k:

If there is a mix of DPSIs and non-partitioned indexes on the table being LOADed, or if there is a foreign key that is exactly indexed by a DPSI, take:

```
Max(longest index key + 15, longest foreign key + 15) x (number of keys extracted)
```

Otherwise, take:

```
Max(longest index key + 13, longest foreign key + 13) x (number of keys extracted)
```

Calculating the number of keys extracted:

For each foreign key that is exactly indexed (that is, where foreign key and index definitions correspond identically):

- Count 0 for the first relationship in which the foreign key participates if the index is not a DPSI. Count 1 if the index is a DPSI.
- Count 1 for subsequent relationships in which the foreign key participates (if any).

Utility Changes to Support DPSI - 3

LOAD - 2

LOAD with an index in PAGE SET REBUILD PENDING

- Page set REBUILD PENDING (PSRBD) on a non-partitioning IX blocks LOAD REPLACE . . . PART on any partition
- PSRBD applies only to non-partitioned indexes, not to DPSIs

Concurrency: DPSIs support complete concurrency between partitions for LOAD PART jobs

Inline statistics

When loading by partition, a separate RUNSTATS INDEX can be avoided by using inline statistics for DPSIs
(Extra RUNSTATS still needed if you have NPSIs)

9.12.8 Changes to LOAD for data-partitioned secondary indexes -2

Other changes brought about by DPSIs that affect LOAD are discussed next.

LOAD with RECOVER PENDING or page set REBUILD PENDING status

Prior to Version 8, a page set REBUILD PENDING status on a non-partitioning index will block the ability to LOAD REPLACE any partition of the table space. This restriction applies only to non-partitioned indexes. Partitioned indexes, including DPSIs, are never placed in a page set REBUILD PENDING status.

Concurrency and compatibility

Today's documented concurrency and compatibility of LOAD with respect to secondary indexes applies only to non-partitioned secondary indexes (NPSIs). DPSIs support total concurrency between partitions. The physical partitions of DPSIs are drained in the same manner as the physical partitions of partitioned partitioning indexes, and there are no logical claims or drains of RR applications from DPSIs during LOAD utility operation. Thus, data-partitioning secondary indexes are a big boon to processing multiple LOAD PART jobs concurrently.

9.12.9 Collecting inline statistics while loading a table

When you run the LOAD utility with the PART keyword specification, you can collect inline statistics on the partitions of the DPSI corresponding to the partitions being loaded, just as you do for partitioning index partitions in V7. There is no need to run RUNSTATS separately. However, you cannot collect inline statistics on NPSIs when you run the LOAD utility with the keyword PART specification, and therefore you still have to run RUNSTATS separately to collect statistics for NPSIs.

Utility Changes to Support DPSI - 4

Redbooks

RECOVER

- DSNUM may specify:
  - a partition of a partitioned table space or partitioned index
  - a data set within a non-partitioned table space
- DSNUM may **not** specify a single data set or logical partition of a non-partitioned index

REPAIR

- PART may now specify a partition of a DPSI

REPORT

- DSNUM may now specify a partition of a DPSI

RUNSTATS

- May be run against single partitions of table spaces or indexes (including DPSIs). Partition-level statistics are used to update aggregate statistics for the entire object

9.12.10 RECOVER

In DB2 Version 8, the DSNUM option of the RECOVER utility may be used to specify:

- A partition of a partitioned table space
- A partition of a partitioned index, including a data-partitioned secondary index (this is new)
- A data set within a non-partitioned table space

However, DSNUM may *not* be specified (at the index level) for:

- A single data set of a non-partitioned index
- A logical partition of a non-partitioned index

Concurrency and compatibility

When partitions of a DPSI are recovered, each partition being recovered is placed in the restrictive state DA/UTUT — where DA (Drain All) causes draining of all claim classes (that is, no concurrent SQL access is possible), and UTUT represents the utility restrictive state whereby the utility has exclusive control on the target object. The concurrency and compatibility characteristics for DPSIs are the same as those for PIs.

9.12.11 REPAIR

You can use the PART keyword with the REPAIR utility to specify a partition of a DPSI.

9.12.12 REPORT

In V8, the DSNUM keyword of REPORT utility statement may specify a partition of a DPSI.

9.12.13 RUNSTATS

There are two formats for the RUNSTATS utility: RUNSTATS TABLESPACE and RUNSTATS INDEX:

- RUNSTATS TABLESPACE gathers statistics on a table space and, optionally, indexes or columns.
- RUNSTATS INDEX only gathers statistics on indexes.

You can run RUNSTATS against a single partition of a partitioned table space or partitioned index (including DPSIs). When run RUNSTATS against a single partition of an object, the partition level statistics that are gathered, are used to update the aggregate statistics for the entire object.

Utility Changes to Support DPSI - 5

REBUILD INDEX

Recreates indexes / index partitions from the table / table partition that they reference

- REBUILD INDEX . . . PART
  - PI or DPSI: Recreates the physical partition
  - NPSI: Recreates the logical partition
- Multiple partitions of DPSI can be rebuilt in parallel
- Concurrency:
  - For a DPSI, each partition being rebuilt is DA / UTUT
  - For a NPSI, each (logical) partition being rebuilt is DR

9.12.14 REBUILD INDEX changes

The REBUILD INDEX utility recreates indexes or index partitions from the table (or table partitions) that they reference.

Physical partitions are recreated when the PART option is used and the target of the REBUILD operation is a (partitioned) partitioning index or a DPSI. Logical partitions are recreated when the PART option is used and the target of the REBUILD operation is a non-partitioned secondary index.

REBUILD INDEX can rebuild one or more partitions of a DPSI through the use of the PART keyword. By using the PART keyword, when only certain partitions of the index need to be rebuilt, REBUILD INDEX avoids unnecessarily scanning the entire table space and unnecessarily rebuilding undamaged partitions.

Estimating the work file size for parallel index build

If you choose to provide work file data sets for the REBUILD INDEX utility (that is, not allow them to be dynamically allocated), you need to estimate the size, and number of keys that are present in all of the indexes, or index partitions being processed by the subtask in order to calculate the size of each sort work file.

When you have determined which indexes or index partitions are assigned to which subtask pairs, use the following formula to calculate the space required:

```
2 x (longest index key + c) x (number of keys extracted)
```

Where:

– Longest index key: Determined as in prior versions

– Value for c: If a mix of DPSIs and non-partitioned indexes are being processed, c is 10; otherwise, c is 8.

– Number of keys extracted: Determined as in prior versions.

REBUILD PENDING statuses

The RBDP* and PSRBD statuses do not apply to DPSIs. They apply only to non-partitioned secondary indexes.

Concurrency and compatibility

When the partitions of a DPSI are rebuilt, each partition being rebuilt is DA/UTUT — where DA (Drain All) causes draining of all claim classes (that is, no concurrent SQL access is possible) and UTUT represents the utility restrictive state whereby the utility has exclusive control on the target object.

When the logical partitions of a NPSI are rebuilt, each partition being rebuilt is DR — where DR (Drain Repeatable Read claimers) causes draining of repeatable read claim classes (that is, limited concurrent SQL access is possible).

Utility Changes to Support DPSI - 6

REORG ... PART

Reorganizes a table space partition (or a range), or indexspace partition

- REORG TABLESPACE PART n
- REORG TABLESPACE PART n:m
- REORG INDEX PART n

REORG phases affected:

- SORT and SORTBLD require more space if there is a mix of DPSIs and NPSIs
- BUILD -- REORG PART SHRLEVEL(NONE)
  - DPSIs are rebuilt -- no contention
  - NPSIs are corrected -- contention is possible
- BUILD2 only for NPSIs, not used with DPSIs

9.12.15 REORG changes related to data-partitioned secondary indexes

REORG TABLESPACE PART n reorganizes the data for part n, reorganizes part n of all partitioned indexes (including all DPSIs), and index entries for logical part n in all non-partitioned indexes.

REORG TABLESPACE PART n:m reorganizes data for part n through part m, reorganizes parts n through m of all partitioned indexes (including all DPSIs), and index entries for logical parts n through m in all non-partitioned indexes.

REORG INDEX PART n reorganizes the part n of the index. REORG INDEX specifying the PART keyword is only allowed when the index is physically partitioned, either a partitioned partitioning index, or a DPSI.

The REORG utility operates in a series of phases. Of these, the SORT, BUILD, SORTBLD and BUILD2 phases are impacted by the use of data-partitioned secondary indexes.

SORT, SORTBLD, and BUILD

The impact on the SORT and SORTBLD phases is with regard to the *size required for the work data sets* used for sorting index entries. To calculate the approximate size required for the work data set, follow these steps:

1. For each table or partition, multiply the number of records in the table or partition by the number of indexes being rebuilt.
2. Add all the products obtained in step 1.
3. Multiply the sum (from step 2) by the largest key length plus a constant, c:
   - If the indexes being rebuilt are a mix of DPSIs and non-partitioned indexes, c is 10.
   - Otherwise, c is 8.

If you choose to provide work file data sets to build indexes in parallel (that is, not allow them to be dynamically allocated), you need to know the size and number of keys that are present in all of the indexes or index partitions being processed by the subtask in order to calculate each sort work file size. When you have determined which indexes or index partitions are assigned to which subtask pairs, use the following formula to calculate the space required:

- 2 x (longest index key + c) x (number of keys extracted)
  - Longest index key: Determined as in prior versions
  - Value for c:
    - If a mix of DPSIs and non-partitioned indexes are being processed, c is 10. Otherwise, c is 8.
    - Number of keys extracted: Determined as in prior versions.

The other impact to the SORTBLD and BUILD phase (depending whether or not multiple indexes are involved and parallel index build is invoked) with respect to the processing of secondary indexes during REORG PART, is *performance*. For DPSIs, the index parts are rebuilt. For non-partitioned secondary indexes, the indexes are corrected (index key update-like processing). Rebuilding a part of a partitioned index (partitioning or DPSI) is much faster than index correction, and avoids any contention between parallel REORG PART jobs.

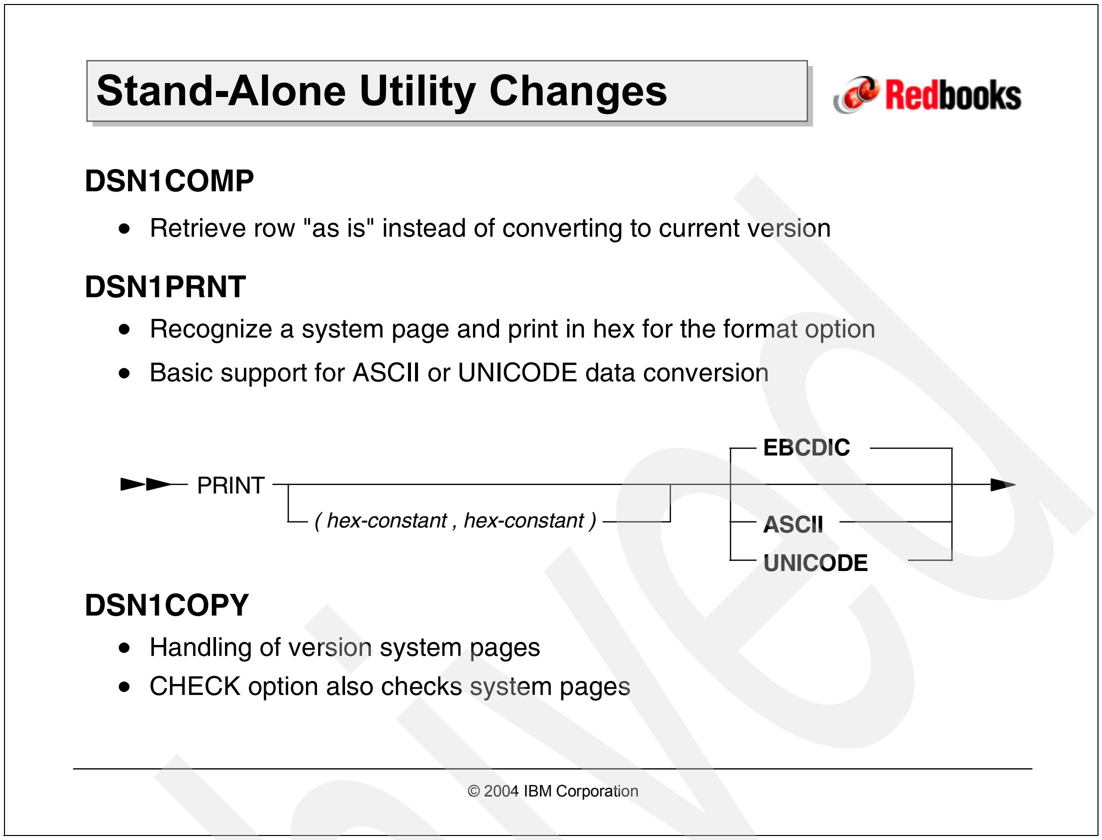
BUILD2

Another very positive effect of using DPSIs is the impact on the BUILD2 phase during online REORG of a partition. When you use REORG TABLESPACE on a partition, or a partition range, with SHRLEVEL(REFERENCE) or SHRLEVEL(CHANGE), and you have a non-partitioned index (or indexes) on the table space, you have a BUILD2 phase.

This phase corrects the index entries for the reorganized part(s) keys in the non-partitioned index(es), and the entire index is not available. For DPSIs, there is *NO BUILD2* phase when doing on-line REORG of a partition, or a partition range. Therefore online REORG of a partition (or range) of a table space with only partitioned indexes is much faster and offers no contention between multiple REORG jobs.

Concurrency and compatibility

The concurrency and compatibility characteristics for DPSIs are the same as those for PIs and for the NPSIs the same as those for former NPIs.



9.13 Stand-alone utility changes

Here we discuss stand-alone utility changes for DSN1COMP and DSN1PRNT.

DSN1COMP

DSN1COMP retrieves a row “as-is”, when estimating the effects of compression on a table space. There is no attempt to convert data to the latest version before compressing rows and deriving a savings estimate.

DSN1PRNT

DSN1PRNT recognizes the table space’s new system pages (related to versioning). When the FORMAT option is specified, details of fields within system pages are not identified with formatted output. Rows on system pages are simply printed in a hex format. Page ranges specified as input identify physical pages and may still be specified even when physical partitions do not match the logical ordering.

In V7, DSN1PRNT (and DSN1COPY) do not support displaying (on the right hand side of the printed report) ASCII or Unicode data. With V8, DB2 has been enhanced to include support for displaying of data in ASCII or Unicode on the right hand side to assist in problem determination. As the goal here is to assist in problem determination, full translation support is not implemented. Accented characters or Unicode characters that are greater than x’80’ are not translated. In addition, the same translate table is used for both ASCII and Unicode (because they are the same for this range of code points). But even with this limited support, rows are much easier to read than before, as illustrated in Example 9-4. In the example we use a Unicode table in which we inserted a few simple rows (Example 9-2).

Example 9-2 A Unicode table with data

```
CREATE DATABASE BSDBUNI;
CREATE TABLESPACE BSTSUNI IN BSDBUNI CCSID UNICODE;
CREATE TABLE  BSDBUNI.TESTA
          (COLA VARCHAR(200) NOT NULL)
          CCSID UNICODE
       IN BSDBUNI.BSTSUNI;
INSERT INTO BSDBUNI.TESTA VALUES(X'C2A7C2A7C2A7C2A7'); -- Paragraph signs in UTF-8
INSERT INTO BSDBUNI.TESTA VALUES('JÜRGEN');            -- Inserting accented character
INSERT INTO BSDBUNI.TESTA VALUES('THIS IS A TEST FOR UPPER CASE');
INSERT INTO BSDBUNI.TESTA VALUES('This is a test for lower case');
SELECT * FROM BSDBUNI.TESTA;
---------+---------+---------+---------+---------+---------+---------+-
COLA
---------+---------+---------+---------+---------+---------+---------+-
§§§§
JÜRGEN
THIS IS A TEST FOR UPPER CASE
This is a test for lower case
```

Then we display the data using DSN1PRNT with the EBCDIC option (this is what you would see with previous DB2 versions). The result is shown in Example 9-3. As you can see, it is not very readable.

Example 9-3 DSN1PRNT PARM=(PRINT,EBCDIC,FORMAT)

```
PAGE: # 00000002 -----------------------------------------------------------------------------------------
DATA PAGE:  PGCOMB='10'X  PGLOGRBA='00004D5E2DA9'X  PGNUM='00000002'X  PGFLAGS='00'X  PGFREE=3961
            PGFREE='0F79'X  PGFREEP=125  PGFREEP='007D'X  PGHOLE1='0000'X  PGMAXID='04'X  PGNANCH=4
PGTAIL:  PGIDFREE='00'X  PGEND='N'
ID-MAP FOLLOWS:
01  0014 0024 0033 0058

RECORD:  XOFFSET='0014'X  PGSFLAGS='02'X  PGSLTH=16  PGSLTH='0010'X  PGSOBD='0003'X  PGSBID='01'X
0008C2A7 C2A7C2A7 C2A7                                                 ..B.B.B.B.

RECORD:  XOFFSET='0024'X  PGSFLAGS='02'X  PGSLTH=15  PGSLTH='000F'X  PGSOBD='0003'X  PGSBID='02'X
00074AC3 9C524745 4E                                                   ...C....+

RECORD:  XOFFSET='0033'X  PGSFLAGS='02'X  PGSLTH=37  PGSLTH='0025'X  PGSOBD='0003'X  PGSBID='03'X
001D5448 49532049 53204120 54455354 20464F52 20555050 45522043 415345     ..................|...&&.......

RECORD:  XOFFSET='0058'X  PGSFLAGS='02'X  PGSLTH=37  PGSLTH='0025'X  PGSOBD='0003'X  PGSBID='04'X
001D5468 69732069 73206120 74657374 20666F72 206C6F77 65722063 617365     ........../.......?..%?...../..
```

However, using the Unicode option on the DSN1PRNT execution, the result looks much better (Example 9-4).

Example 9-4 DSN1PRNT PARM=(PRINT,UNOCODE,FORMAT)

```
PAGE: # 00000002
-----------------------------------------------------------------------------------------
DATA PAGE:  PGCOMB='10'X  PGLOGRBA='00004D5E2DA9'X  PGNUM='00000002'X  PGFLAGS='00'X  PGFREE=3961
            PGFREE='0F79'X  PGFREEP=125  PGFREEP='007D'X  PGHOLE1='0000'X  PGMAXID='04'X  PGNANCH=4
PGTAIL:  PGIDFREE='00'X  PGEND='N'
ID-MAP FOLLOWS:
01  0014 0024 0033 0058

RECORD:  XOFFSET='0014'X  PGSFLAGS='02'X  PGSLTH=16  PGSLTH='0010'X  PGSOBD='0003'X  PGSBID='01'X
0008C2A7 C2A7C2A7 C2A7                                                   ..........@@

RECORD:  XOFFSET='0024'X  PGSFLAGS='02'X  PGSLTH=15  PGSLTH='000F'X  PGSOBD='0003'X  PGSBID='02'X
00074AC3 9C524745 4E                                                     ..J..RGEN@@@

RECORD:  XOFFSET='0033'X  PGSFLAGS='02'X  PGSLTH=37  PGSLTH='0025'X  PGSOBD='0003'X  PGSBID='03'X
001D5448 49532049 53204120 54455354 20464F52 20555050 45522043 415345    ..THIS IS A TEST FOR UPPER CASE@

RECORD:  XOFFSET='0058'X  PGSFLAGS='02'X  PGSLTH=37  PGSLTH='0025'X  PGSOBD='0003'X  PGSBID='04'X
001D5468 69732069 73206120 74657374 20666F72 206C6F77 65722063 617365    ..This is a test for lower case@
```

The result is not perfect, but definitely much better than before.

> **Tip:** To be as helpful as possible, DB2 tries to determine the best formatting option by itself. If the first page in the input data set is a header page, DSN1PRNT uses the format information in the header page as the default format. Therefore, if you do not specify the EBCDIC, ASCII, or Unicode option, DB2 tries to determine the option itself (based on the header page information). However, if you specify the option, it is honored.

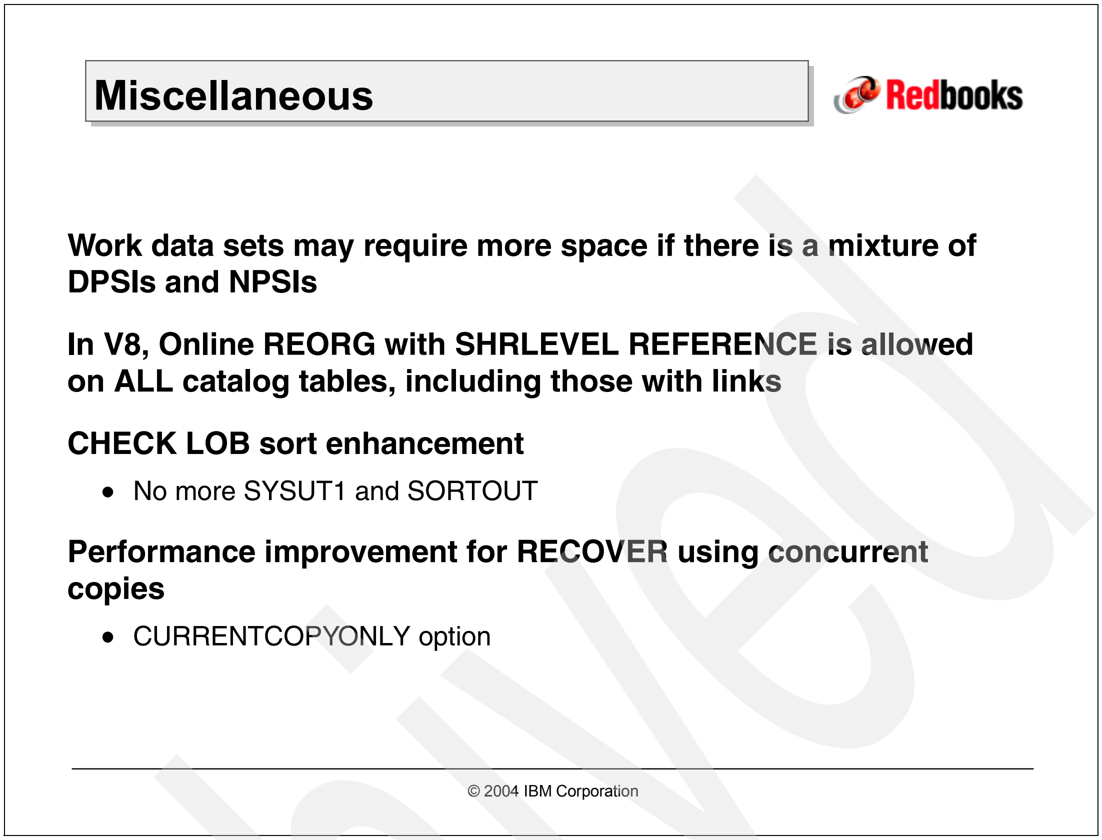
The DSN1COPY print option is identical to the one used by DSN1PRNT. Therefore, this enhancement also applies to DSN1COPY.

DSN1COPY

DSN1COPY tolerates the existence of table space system (versioning) pages. When the PRINT option is specified, the pages are printed in hexadecimal format.

When you use DSN1COPY (with the OBIDXLAT option) to copy data between objects or subsystems, you must use the REPAIR utility with the VERSIONS keyword to update the version information in the DB2 catalog of the target object.

The CHECK option also validates system pages.



9.14 Miscellaneous enhancements

Note the requirement for more space for work data sets if you have a mixture of DPSIs and NPSIs.

9.14.1 Online REORG of all catalog table spaces

In Version 7, the following catalog table spaces cannot be specified in REORG SHRLEVEL CHANGE or REFERENCE, although they can be REORGed with SHRLEVEL NONE:

- DSNDB06.SYSDBASE
- DSNDB06.SYSDBAUT
- DSNDB06.SYSGROUP
- DSNDB06.SYSPLAN
- DSNDB06.SYSVIEWS
- DSNDB01.DBD01

As part of the V8 migration process, the DB2 catalog is converted to Unicode. To do that, DB2 uses online REORG SHRLEVEL REFERENCE. To allow this, online REORG had to be enhanced to be able to also deal with those DB2 catalog tables that contain links. DB2 allows everybody to benefit from this, and in V8 users (so not just the DB2 migration process) can run online REORG on all catalog tables including those with links, which you could not do in prior versions.

In V8, as in previous versions, you cannot REORG DSNDB01.SYSUTILX.

9.14.2 CHECK LOB sort enhancement

In V8, the SYSUT1 and SORTOUT DD statement for sort input and output are no longer needed. (The WORKDDN keyword will remain, but is ignored.) The CHECK LOB utility now uses a sort subtask. In the CHECKLOB phase, all active pages of the LOB table space are scanned. During the scan, up to four records can be generated per LOB page. These records are now passed directly into the SORTIN phase (via a so-called sort pipe). After the sorting, the sorted records are passed to the REPRTLOB phase by the SORTOUT.

9.14.3 Performance improvements for RECOVER with concurrent copies

DB2 Version 8 introduces a new keyword for the RECOVER utility to enable faster recovery when the image copies were taken using the concurrent copy feature. The new keyword is *CURRENTCOPYONLY*.

It specifies that RECOVER is to improve the performance of restoring concurrent copies (copies that were made by the COPY utility with the CONCURRENT option) by using only the most recent primary copy for each object in the list. When you specify CURRENTCOPYONLY for a concurrent copy, RECOVER builds a DFSMSdss™ RESTORE command for each group of objects that is associated with a concurrent copy data set name.

To avoid the 255 object limit per DFSMSdss RESTORE command, the FILTERDDN option is used “under the covers” whenever required. For that purpose, a temporary file is allocated by the RECOVER utility, taking advantage of the new VOLTDEVT DSNZPARM, a unit name that is to be used for temporary allocations.

If the RESTORE fails, RECOVER does not automatically use the next most recent copy or the backup copy, and the object fails. If you specify DSNUM ALL with CURRENTCOPYONLY and one partition fails during the restore process, the entire utility job on that object fails. If you specify CURRENTCOPYONLY and the most recent primary copy of the object to be recovered is not a concurrent copy, DB2 ignores this keyword.

Utility Unicode Statements

Utility control statements may be specified in Unicode or EBCDIC

- DB2 detects which encoding scheme is being used
- Must be all UTF-8 or EBCDIC -- no mixing!
- Object names in messages will be in EBCDIC

New utility stored procedure interface DSNUTILU for Unicode

9.14.4 Utility Unicode statements

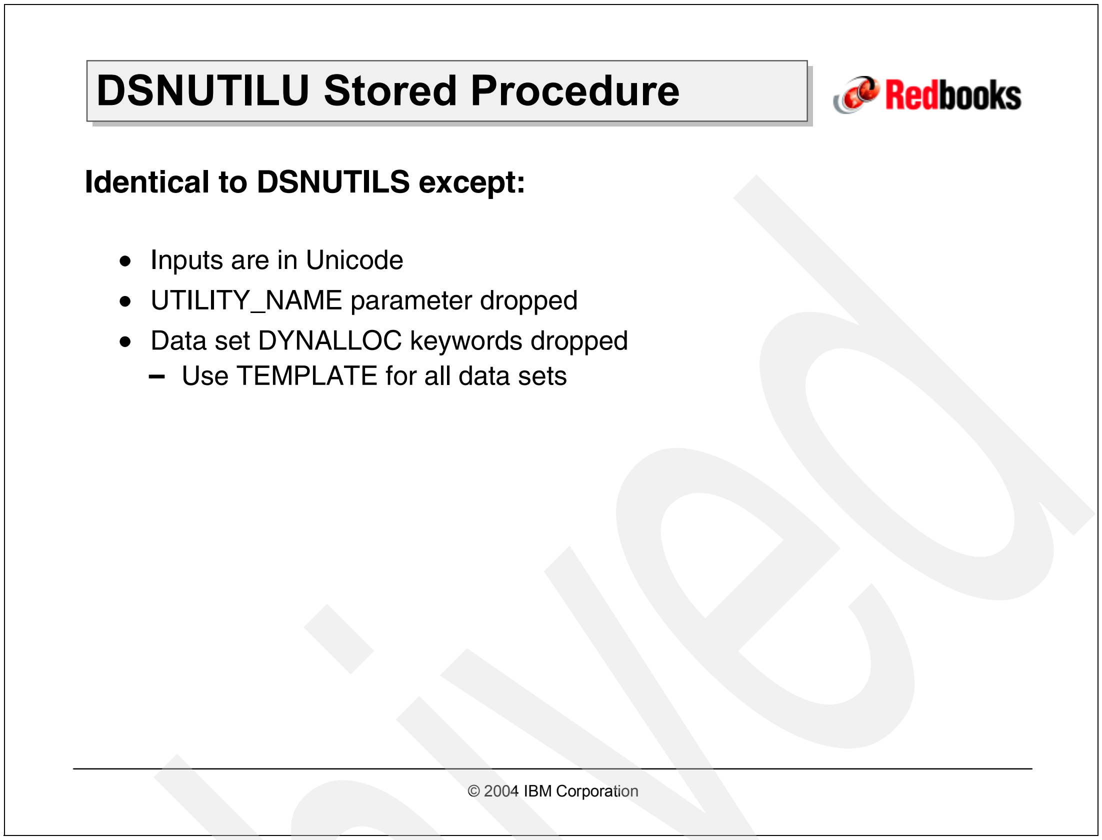
In V8, the utility control statement parser can parse Unicode control statements, specifically UTF-8. You can provide utility control statements, either entirely in EBCDIC characters or entirely in Unicode characters.

All utility control statement input data sets which begin with any of these characters are processed as Unicode.

- '20'x - Unicode blank
- '2D'x - Unicode dash (the utility comment delimiter)
- '41'x through '5A'x inclusive - upper case Unicode letters

Utility control statement input data sets are those provided to the DSNUTILB program with DD names SYSIN, SYSLISTD or SYSTEMPL, or the contents of the UTSTMT field passed to the DSNUTILU stored procedure created for this purpose.

All output to the SYSPRINT data set and the MVS console continue to be in EBCDIC with translation taking place as required.



9.15 DSNUTILU Unicode utility stored procedure

The DSNUTILU stored procedure is identical to DSNUTILS stored procedure introduced in Version 7 with two exceptions:

- All input parameters to the procedure are in Unicode. UTILITY_ID and RESTART inputs are translated to EBCDIC by the stored procedure for processing. UTSTMT input is stored in a temporary SYSIN data set and is processed in Unicode as outlined above.
- The dynamic allocation of data sets is removed. As of Version 7, this function is performed by the TEMPLATE control statement. In order to eliminate dynamic allocation, the following DSNUTILS keywords are not supported by DSNUTILU:
  - UTILITY, xxxxDSN, xxxxDEVT, xxxxSPACE for all values of xxxx.

CREATE PROCEDURE DSNUTILU

```
CREATE PROCEDURE DSNUTILU
  ( IN UTILITY_ID VARCHAR(16) CCSID UNICODE
  , IN RESTART VARCHAR(8) CCSID UNICODE
  , IN UTSTMT VARCHAR(32704) CCSID UNICODE
  , OUT RETCODE INTEGER)
  EXTERNAL NAME DSNUTILU
  LANGUAGE ASSEMBLE
  WLM ENVIRONMENT WLMENV1
  COLLID DSNUTILU
  RUN OPTIONS 'TRAP(OFF)'
  PROGRAM TYPE MAIN
  MODIFIES SQL DATA
  ASUTIME NO LIMIT
  STAY RESIDENT NO
  COMMIT ON RETURN NO
  PARAMETER STYLE GENERAL
  RESULT SETS 1
  SECURITY USER;
```


9.15.1 CREATE PROCEDURE DSNUTILU

The DSNUTILU stored procedure enables you to provide control statements in Unicode UTF-8 characters instead of EBCDIC characters to execute DB2 utilities from a DB2 application program.

When called, DSNUTILU performs the following actions:

- It translates the values specified for utility-id and restart parameters into EBCDIC.
- It creates the utility input (SYSIN) stream for control statements that use Unicode characters.
- It deletes all the rows currently in the created temporary table (SYSIBM.SYSPRINT).
- It captures the utility output stream (SYSPRINT) into a created temporary table (SYSIBM.SYSPRINT).
- It declares the following cursor to select from the temporary SYSPRINT table, as follows:

```
DECLARE SYSPRINT CURSOR WITH RETURN FOR
   SELECT SEQNO, TEXT FROM SYSPRINT ORDER BY SEQNO
```

- It opens the SYSPRINT cursor and returns.

The calling program then fetches from the returned result set to obtain the captured utility output. All output to SYSPRINT, and to the operator console, is in EBCDIC format.

DSNUTILU always uses DFSORT “under the covers”.

As DSNUTILS, DSNUTILU must also run in a WLM environment using TCB=1.

10

Performance enhancements

List of Topics

- Materialized query tables
- Indexing enhancements
- Stage 1 and indexable predicates
- Table UDF cardinality option and block fetch
- Trigger enhancements
- Distribution statistics on non-indexed columns
- Cost-based parallel sort for single and multiple tables
- Performance of multi-row operations
- Volatile table support
- Data caching and sparse index for star join
- Miscellaneous performance enhancements
- Visual Explain enhancements

This chapter describes the performance enhancements. It consists of the following topics:

- Materialized query tables
- Index only access for VARCHAR columns
- Stage 1 and indexable predicates
- Table UDF cardinality and block fetch
- Trigger enhancements
- Distribution statistics on non-indexed columns
- Cost-based parallel sort for single and multiple tables
- Volatile table support
- Data caching and sparse index for star join

For more information about the DB2 V8 performance enhancements, see also *DB2 UDB for z/OS Version 8 Performance Topics*, SG24-6465.

Data Warehousing Issues

To improve query performance especially for Data Warehousing

Summary tables are often created manually for users

- To improve the response time
- To avoid redundant work of scanning, aggregation and joins of the detailed base tables (e.g. history)
- To simplify SQL to be coded

User needs to be aware of summary tables and know whether to use them or the base tables depending on the query.

10.1 Data warehousing issues

Decision-support system queries typically operate over huge amounts of data. They perform multiple joins and complex aggregation operations. In addition to that, those decision-support queries become increasingly interactive. This implies that the demand on response times has also reached a very high standard. Traditional optimization techniques often fail to meet these new requirements. In some cases, the only solution is to pre-compute the whole or parts of each query in order to avoid redundant work and to simplify SQL to be coded, and use these pre-computed results to be able to provide a timely answer when queries are submitted to the system.

In the past, this had to be done manually. The disadvantage of this manual solution is that users must be aware of the existence of the summary tables and know exactly whether they can be more helpful than the base tables depending on the query.

In DB2 for z/OS Version 8, such pre-computed results are known as materialized query tables (MQTs). A materialized query table is called an automatic materialized query table (AMQT), if it is *automatically* considered by the database management system (DBMS) to answer a query. This means that the user no longer has to be aware of its existence. The DBMS takes care of that for them. The success of this approach is based on the fact, that expected queries usually share a number of common sub-operations.

What is a Materialized Query Table (MQT)?

Redbooks

Table containing materialized data derived from one or more source tables specified by a fullselect

- Source tables can be base tables, views, table expressions or user-defined table expression
- MQTs can be accessed directly via SQL or
- Chosen by the optimizer (through **automatic query rewrite**) when a base table or view is referenced

Two types: Maintained by system or by user

Synchronization between base table(s) and MQT

- Using 'REFRESH TABLE' statement
- Batch update, triggers, etc. for user-maintained MQTs

Can provide significant query performance improvement

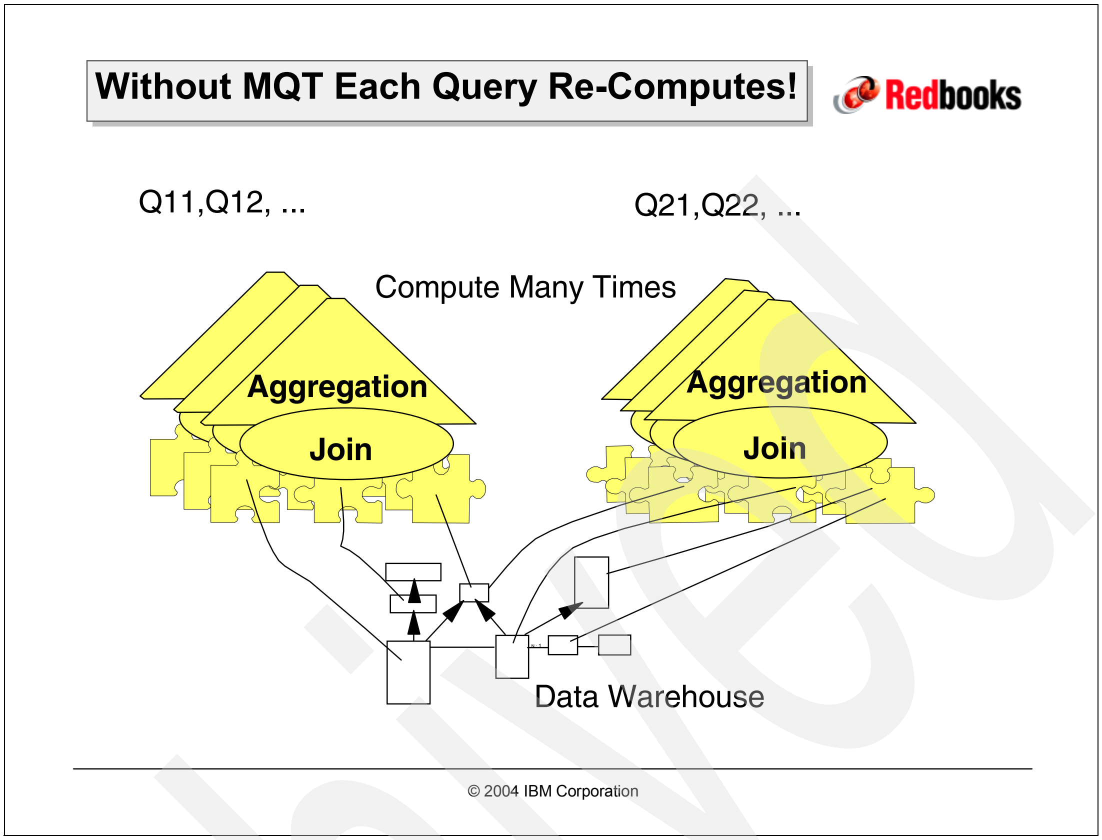
10.2 What is a materialized query table?

As mentioned above, a materialized query table (MQT) contains pre-computed data. The pre-computed data is the result of a query, that is a fullselect associated with the table, specified as part of the CREATE/ALTER TABLE statement.

The source for a materialized query table can be base tables, views, table expressions, or user-defined table functions.

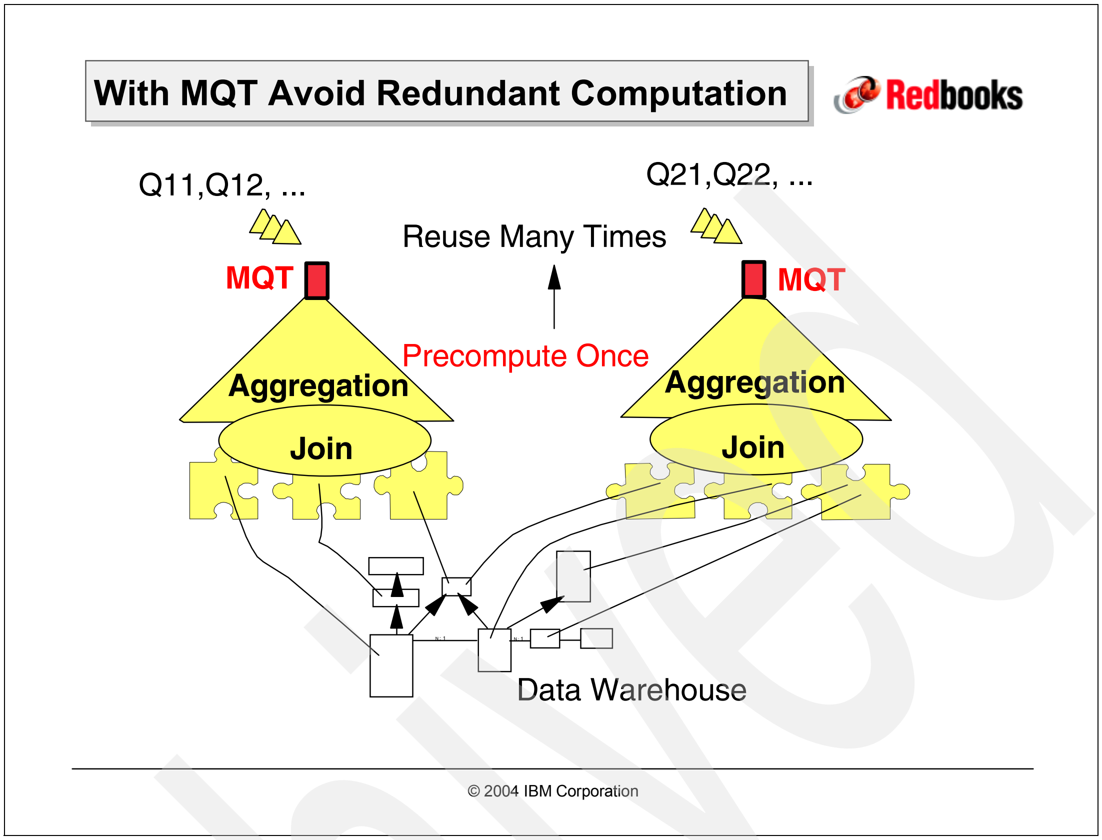
MQTs can either be accessed directly via SQL, which is what people are used to in data warehouse systems for a long time, or can be chosen by the optimizer (through *automatic query rewrite*). The second case is using the DBMS's optimization technology to determine whether or not to use the MQT, and is the real power of the feature, as users no longer have to be aware of the existence of the MQT. The DBMS takes care of that. In the second case, we are talking about automatic materialized query tables, which are also known as *automatic summary tables* (ASTs) or *materialized views*. Throughout the rest of the publication we use the term *materialized query tables* or MQTs.

MQTs can be classified in a couple of different ways. One classification is between so-called *system-maintained* and the *user-maintained* MQTs. The default is MAINTAINED BY SYSTEM (system-maintained MQT), which means that these MQTs cannot be updated by the LOAD utility, nor by INSERT, UPDATE, or DELETE SQL statements. The only way to update system-maintained MQTs is through the (new) REFRESH TABLE SQL statement.
MQTs created as MAINTAINED BY USER (user-maintained MQTs), can be updated via the LOAD utility, INSERT, UPDATE or DELETE SQL statements, or any other way to update a table, for example, using triggers.



10.2.1 Without MQT each query recomputes

As mentioned before, if you are running queries against your data warehouse, each time that you submit a query against the DBMS, it has to (re)compute the result set. This can mean scanning an enormous amount of data, doing all kinds of groupings and calculations, consuming the system's resources every single time you request complex information from your data warehouse. This usually results in a lot of system resources (CPU and I/O) being used, and prolonged response times.



10.2.2 With MQTs, avoid redundant computation

As shown on the visual above, if you use MQTs, the costly computations do not have to be performed every single time you run a query against your warehouse tables. The query to build the MQT is executed only once. From then on, the data is available in the MQT, and can be used instead of the underlying tables.

Creating an MQT

Redbooks

CREATE TABLE or ALTER TABLE (to register an existing table)

- Specify a fullselect clause
- DATA INITIALLY DEFERRED, REFRESH DEFERRED

Two types of MQTs can be defined in the CREATE/ALTER TABLE

- MAINTAINED BY SYSTEM
  - Data can only be updated by REFRESH TABLE statement
- MAINTAINED BY USER
  - SELECT/INSERT/UPDATE/DELETE and LOAD operations are permitted

Automatic Query Rewrite exploitation

- ENABLE/DISABLE QUERY OPTIMIZATION
- May wish to create user-maintained tables with DISABLE option and alter to ENABLE once table has been populated
- Optimizer only uses system-maintained if a REFRESH has occurred
- New special registers to govern MQT usage as well

10.2.3 Creating a materialized query table

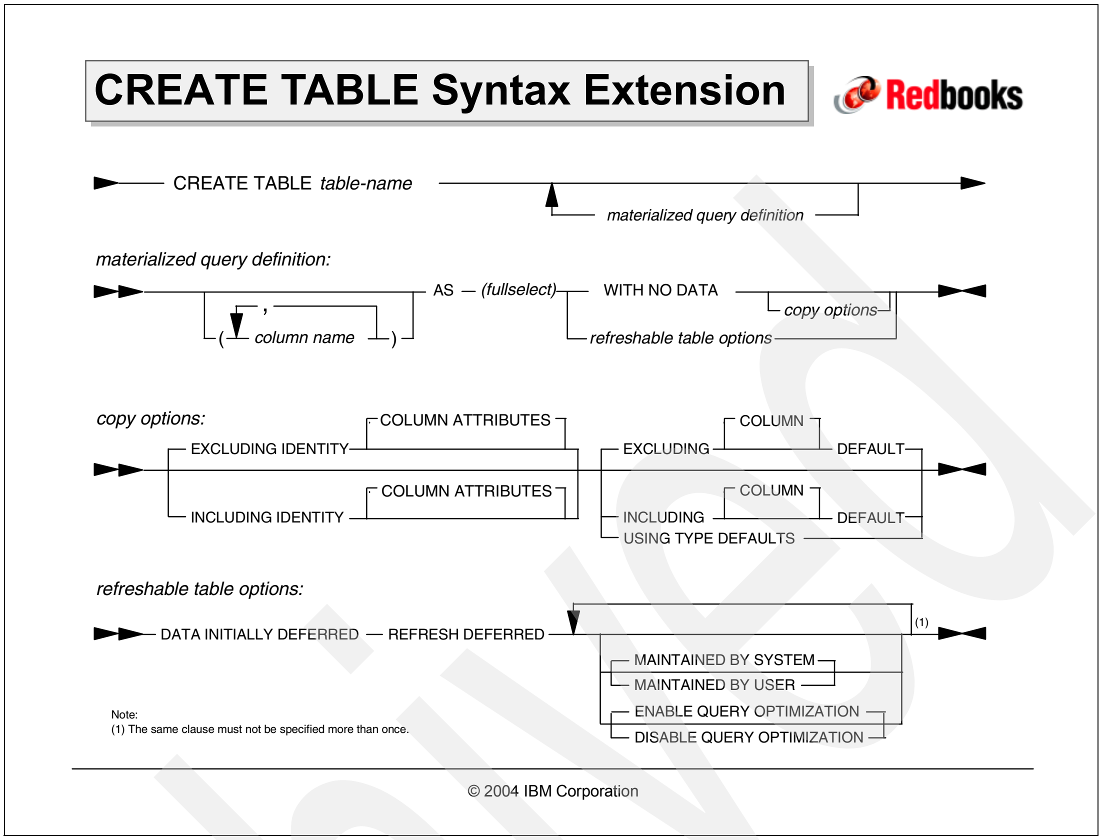
Creating a materialized query table is similar to creating a view. The difference is that a view is only a logical definition, while a materialized query table contains the materialized data of the query result. Because of this similarity, some IT books and other vendors also use the term *materialized views*.

You can either create an MQT from scratch using the CREATE TABLE statement, or register existing tables, which you previously used as “manual” MQTs, as “official” MQTs (by using the ALTER TABLE statement), and enable them for automatic query rewrite. Refer to 10.2.9, “The basics of automatic query rewrite” on page 718 for a more detailed consideration of the query rewrite.

Note that the CREATE TABLE syntax has also been enhanced to allow you specify the WITH NO DATA clause (see syntax diagram on the next visual). This enables you to create a table where the column definitions are inherited from the columns specified in the fullselect. This clause was already available when creating a declared temporary table. In V8, the syntax is extended to the CREATE TABLE statement.

Using the WITH NO DATA clause leads to creation of a (normal) table that is not populated with data. The specified fullselect is not executed. With this option, you do not create an MQT. The object created is registered in the DB2 catalog table SYSIBM.SYSTABLES as type ‘T’. (Instead of using WITH NO DATA, you can also use DEFINITION ONLY. The effect of both syntax options is exactly the same. Using WITH NO DATA is preferred as it is the SQL standards syntax.)

The copy options shown in the syntax diagram specify how the column identity and default attributes are inherited. The rules are the same as those in DECLARE GLOBAL TEMPORARY TABLE WITH NO DATA.



CREATE TABLE syntax extension

The visual above shows extracts of the CREATE TABLE statement syntax, which is used to create MQTs. As you can see from this diagram, a few new syntax blocks are available that are related to the definition of MQTs. When creating an MQT, it is mandatory to specify DATA INITIALLY DEFERRED followed by REFRESH DEFERRED, as shown in the *refreshable table options* block. These two parameters define a table as a materialized query table.

DATA INITIALLY DEFERRED means that when a materialized query table is created, the MQT is not populated instantly. In order to have the MQT populated, you must use the REFRESH TABLE statement (for system-maintained MQTs) or any other allowed SQL statement (for user-maintained MQTs).

REFRESH DEFERRED means that the data in the MQT is not refreshed immediately when its underlying base tables are updated and can be refreshed at any time using the REFRESH TABLE statement (for system-maintained MQTs — MAINTAINED BY SYSTEM) or via any other allowed SQL statement (for user-maintained MQTs — MAINTAINED BY USER). Note that (unlike DB2 for LUW) you can populate user-maintained MQT using the REFRESH TABLE statement as well.

Another choice you have to make when creating an MQT is between ENABLE QUERY OPTIMIZATION and DISABLE QUERY OPTIMIZATiON. If you choose to ENABLE QUERY OPTIMIZATION, you allow that this MQT can be exploited by automatic query rewrite. In contrast to that, specifying DISABLE QUERY OPTIMIZATION will cause this MQT not to be considered by the automatic query rewrite process. In addition to specifying this option at CREATE/ALTER TABLE time, there are also two new special registers that govern the selection of the MQT by automatic query rewrite at run time. For more information, see 10.2.7, “Controlling MQT for automatic query rewrite” on page 713.

Creating an MQT - Example

```
CREATE TABLE MQT1 AS (
 SELECT T.PDATE, T.TRANSID,
        SUM(QTY * PRICE) AS TOTVAL,
        COUNT(QTY * PRICE) AS CNT
 FROM SCNDSTAR.TRANSITEM TI, SCNDSTAR.TRANS T
 WHERE TI.TRANSID = T.TRANSID
 GROUP BY T.PDATE, T.TRANSID)
DATA INITIALLY DEFERRED
REFRESH DEFERRED
MAINTAINED BY SYSTEM
ENABLE QUERY OPTIMIZATION
IN MYDBMQT.MYTSMQT;
```


Creating an MQT - example

The visual above shows a simple example of the creation of an MQT. As you can see, the AS keyword followed by a fullselect preludes the definition of an MQT. The other four parameters that are special for the creation of an MQT in this example are:

- DATA INITIALLY DEFERRED
- REFRESH DEFERRED
- MAINTAINED BY SYSTEM (default)
- ENABLE QUERY OPTIMIZATION (default)

MQTs are registered in the DB2 catalog table SYSIBM.SYSTABLES. The identifier in column TYPE is ‘M’ for MQTs.

As you can see from the sample above, as for any other table that physically exists, you can decide which table space you want to use to store your MQT data.

Changing the Attributes of a MQT

ALTER TABLE statement to update the attributes of a MQT

To register/unregister MQT, use

- ADD/DROP MATERIALIZED QUERY clause

Can alter between system-maintained and user-maintained

- Control the types of operations permitted on MQT
- System-maintained during online day to prevent updates and User maintained during off-line data loading

To enable or disable automatic query rewrite option on a MQT, use

- Enable/disable query optimization clause
- Use to temporarily disable query optimization during table maintenance

10.2.4 Changing the attributes of an MQT

As for almost all other DB2 objects, you can change the attributes of your MQTs using an ALTER SQL statement. Since your MQT is considered to be a table, the appropriate statement is ALTER TABLE.

The next visual shows the enhanced ALTER TABLE syntax diagram. You can perform the following changes regarding MQTs using the ALTER TABLE statement:

- ALTER TABLE ADD MATERIALIZED QUERY

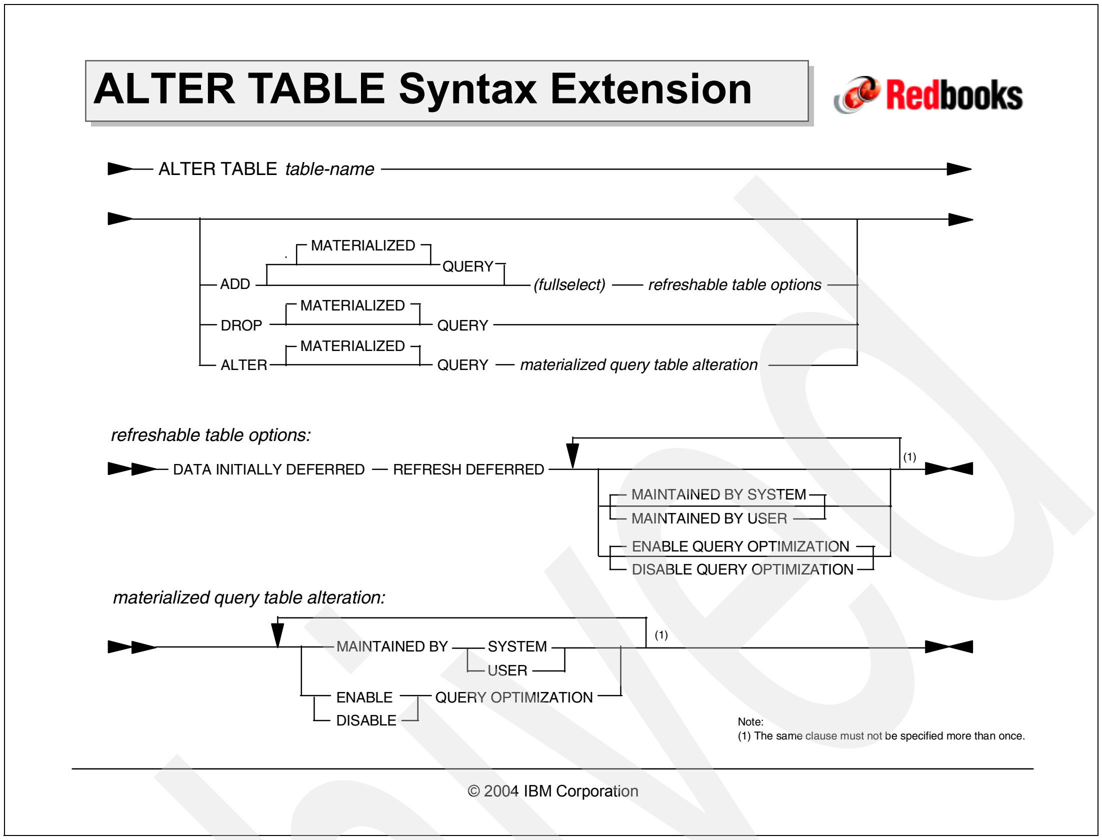
  This option converts a base table to an MQT. This option is of interest if today you are already working with tables that hold aggregated, pre-computed data, and you want to register these tables as MQTs.

- ALTER TABLE DROP MATERIALIZED QUERY

  This option converts an MQT into a (normal) base table.

- ALTER TABLE ALTER MATERIALIZED QUERY

  This option lets you change all parameters which can be used to define the characteristics of an MQT, as we discussed in 10.2.3, “Creating a materialized query table” on page 703.



ALTER TABLE syntax extension

The visual above shows the enhanced ALTER TABLE syntax. As you can see, you can change an existing table into an MQT or vice versa.

You can also switch an MQT between a system-maintained and user-maintained MQT, or enable or disable the MQT for query optimization.

Alter Base Table to MQT - Example

```
ALTER TABLE T1 ADD MATERIALIZED QUERY (
 SELECT T.PDATE,
        SUM(QTY * PRICE) AS TOTVAL,
        COUNT(QTY * PRICE) AS CNT
 FROM SCNDSTAR.TRANSITEM TI, SCNDSTAR.TRANS T
 WHERE TI.TRANSID = T.TRANSID
 GROUP BY T.PDATE)
DATA INITIALLY DEFERRED
REFRESH DEFERRED
MAINTAINED BY USER
ENABLE QUERY OPTIMIZATION;
```


Alter base table to MQT - example

The example assumes that there is a table T1 that already exists. The data in table T1 was generated using the fullselect shown above. That is, the data stored in T1 is the result of a pre-computation, which performs some scalar functions, a join, GROUP BY, and so on, which make up the SELECT statement. When altering an existing table into an MQT, it is the user's responsibility to make sure that the data in the table matches the result of the query that makes up the MQT.

If a user is aware of table T1 and what data it contains, they can use this table or a subset of data from tables TI (TRANSITEM) and T (TRANS), which is stored in T1. However, if — and this is very likely the case — the user does not know that this pre-computation has been done already, and submits the fullselect against TI and T instead of T1, all the computation has to be done again, which is a great waste of system resources. In addition, it is very often not a trivial task to figure out whether the data you are looking for is actually fully available from T1.

For these reasons, you may want to turn an existing table (T1) into an MQT, managed by DB2.

Starting with DB2 V8, you can ALTER existing tables into MQTs. As shown in the example above, we use the ENABLE QUERY OPTIMIZATION option. This means that when we submit a select statement against TI and T instead of selecting from T1, the optimizer knows that the subset of data that you need already exists in table T1, and uses T1 instead of accessing TI and T.

Altering a WITH NO DATA table into an MQT

Assume that you have created the following definition-only table, using the following syntax:

```
CREATE TABLE MQT1
AS(SELECT * FROM NORMAL_TAB)
WITH NO DATA
```

If you want to change the characteristics of this table using the ALTER TABLE statement and turn it into an MQT, you *cannot* use either of the following statements:

```
ALTER TABLE MQT1
ADD MATERIALIZED QUERY DATA INITIALLY DEFERED REFRESH DEFERRED
```

Nor these:

```
ALTER TABLE MQT1
ALTER MATERIALIZED QUERY MAINTAINED BY USER
```

Instead, you must code:

```
ALTER TABLE MQT1
ADD MATERIALIZED QUERY (SELECT * FROM NORMAL_TAB)
DATA INITIALLY DEFERRED REFRESH DEFERRED
```

After you use the last of the statements above, the WITH NO DATA table is converted to an MQT, which you can also verify from the entry in TYPE column of SYSIBM.SYSTABLES, because it changes from ‘T’ to ‘M’.

Fullselect Considerations

Redbooks

Base table or views can be referenced subject to a number of conditions

No references to ROWIDs, LOB types or remote objects

To enable query optimization (which is done at the QBLOCK level)

- Fullselect must be a subselect
- Single select after view and table expression merge
- No outer joins or subselects are allowed

Has other specific rules regarding the contents of the fullselect

- Error message returned when fullselect does not satisfy the rules

10.2.5 Fullselect considerations

There are a few considerations for the fullselect you specify as part of the *materialized-query-definition* block.

When you specified WITH NO DATA or DEFINITION ONLY (which does NOT define an MQT):

- The fullselect must not refer to host variables, or include parameter markers.
- The fullselect must not reference a remote object
- The fullselect must not result in a column having a ROWID data type, because a ROWID requires a generated attribute
- The fullselect must not result in a column having a LOB data type, because a ROWID is needed for LOB data types, but ROWID is restricted.
- The fullselect must not contain PREVIOUS VALUE FOR and NEXT VALUE FOR expressions.

When you create an MQT, by specifying REFRESH DEFERRED, and use the DISABLE QUERY OPTIMIZATION option, the following additional restrictions apply:

- The fullselect cannot contain a reference to a created global temporary table or a declared global temporary table.
- The fullselect cannot reference another materialized query table

When you specify ENABLE QUERY OPTIMIZATION, your fullselect must adhere to the following additional restrictions:

- The fullselect must be a subselect.

- The subselect cannot reference to a user-defined scalar or table function with the EXTERNAL ACTION or NON-DETERMINISTIC attributes, or built-in function RAND.
- The subselect *cannot* contain:
  - Any predicates that include subqueries
  - A nested table expression or view that requires temporary materialization
  - A join using the INNER JOIN syntax
  - An outer join
  - A special register
  - A scalar fullselect
  - A row expression predicate
  - Sideway references
  - Table objects with multiple CCSID sets

When the fullselect does not satisfy these restrictions, an error is returned. Note also that when defining a materialized query table, the column attributes, such as DEFAULT and IDENTITY, are *not* inherited from the fullselect.

Populating and Refreshing a MQT

Redbooks

Ensure data currency meets user requirements to avoid out-dated results

Issue REFRESH TABLE statement periodically

- Deletes all rows in the MQT
- Executes the fullselect as defined in the original MQT definition
- Inserts calculated data into the MQT
- Updates the catalog with refresh timestamp and cardinality information
- Executes in a single UOW
- Consider logging and performance impact

For user-maintained MQTs, other population methods can also be used, for example, LOAD, Inserts, Updates, DPROP Apply

10.2.6 Populating and refreshing an MQT

You can use the REFRESH TABLE *mq_table* SQL statement to populate a system-maintained, or user-maintained MQT.

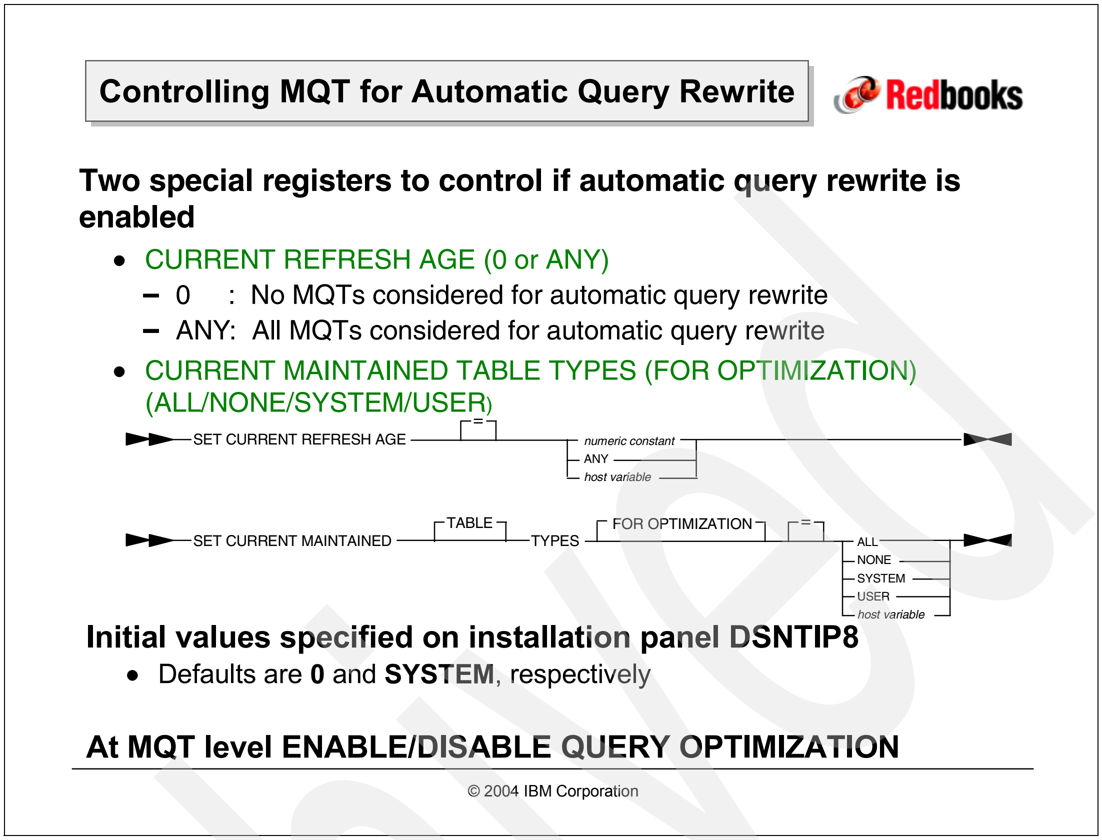
Whenever you issue the REFRESH TABLE statement, the following actions are performed:

1. All rows are deleted from the MQT. This is a mass delete when the MQT physically resides in a segmented table space. As for regular tables, mass deletes are much faster if the data is stored in a segmented table space instead of a simple table space.
2. The MQT's fullselect is executed to recalculate the data from the tables that are specified in this fullselect. The isolation level used for this execution is the one that belongs to the MQT (the isolation level that was in effect when the MQT was created). Access to the MQT itself is blocked during the execution of the REFRESH TABLE statement.
3. The calculated data is then inserted into the MQT.
4. The catalog is updated with the refresh timestamp and cardinality of the MQT. After successful execution of a REFRESH TABLE statement, the SQLCA field SQLERRD(3) also contains the number of rows inserted into the materialized query table.

The four steps described above are all done within a single commit scope. In DB2 for z/OS V8, only full refresh is supported.

The REFRESH TABLE statement is an explainable statement. The EXPLAIN output contains rows for INSERT with the fullselect in the MQT definition.

Query rewrite avoids using locked MQTs. It will use those MQTs not locked, or if no MQTs are available, base tables instead.



10.2.7 Controlling MQT for automatic query rewrite

The process of recognizing whether an MQT can be used in answering a query and rewriting the query accordingly, is called *automatic query rewrite*.

Two new special registers, CURRENT REFRESH AGE and CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION, control whether or not an MQT is considered by automatic query rewrite for a dynamically prepared query. The default value for both parameters is specified on panel DSNTIP8 during installation time. The default value for CURRENT REFRESH AGE ends up as DSNZPARM parameter REFSHAGE, and the one for CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION goes into DSNZPARM parameter MAINTYPE.

You can think of the refresh age of an MQT, as the duration between the current timestamp and the time when the MQT was last refreshed using the REFRESH TABLE statement. In V8, the CURRENT REFRESH AGE can only have two possible values:

- 0 (zero)

  A value of 0 means that no MQTs are considered for automatic query rewrite by this application.

- ANY

  This means that all MQTs are considered for automatic query rewrite by this application.

If you check the current setting of the special register after you have set it to ANY, you see a value of 99999999999999.000000. This represents 9999 years, 99 month, 99 days, 99 hours, 99 minutes, 99 seconds. The six zeros after the decimal point represent microseconds, which are ignored.

Because user-maintained MQTs can be updated by using INSERT, UPDATE, or DELETE SQL statements, or the LOAD utility, the refresh age of a user-maintained materialized query table cannot truly represent the “freshness” of the data in a user-maintained MQT. This is why a second new special register is used in conjunction with MQTs, called CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION. You can set this special register to ALL, NONE, SYSTEM, or USER. Only those MQTs which belong to the group that matches the value of the special register, are eligible for automatic query rewrite. That is, if you decide to set this value to NONE, no MQT is considered for automatic query rewrite.

Note that in addition to these special registers that govern whether or not MQTs are considered by automatic query rewrite at the application level, you also have the ENABLE/DISABLE QUERY OPTIMIZATION option on the CREATE/ALTER TABLE statement. Specifying DISABLE QUERY OPTIMIZATION disables the selection of the MQT by automatic query rewrite. The MQT must be defined with ENABLE QUERY OPTIMIZATION, before automatic query rewrite can consider the MQT.

If a system-maintained materialized query table has not been populated with data (when the REFRESH_TIME column in SYSIBM.SYSVIEWS is equal to the default timestamp ’0001-01-01.00.00.00.000000’), the MQT is not considered by automatic query rewrite. For a user-maintained materialized query table, the refresh timestamp in the system catalog table is not maintained, therefore users should not use the value.

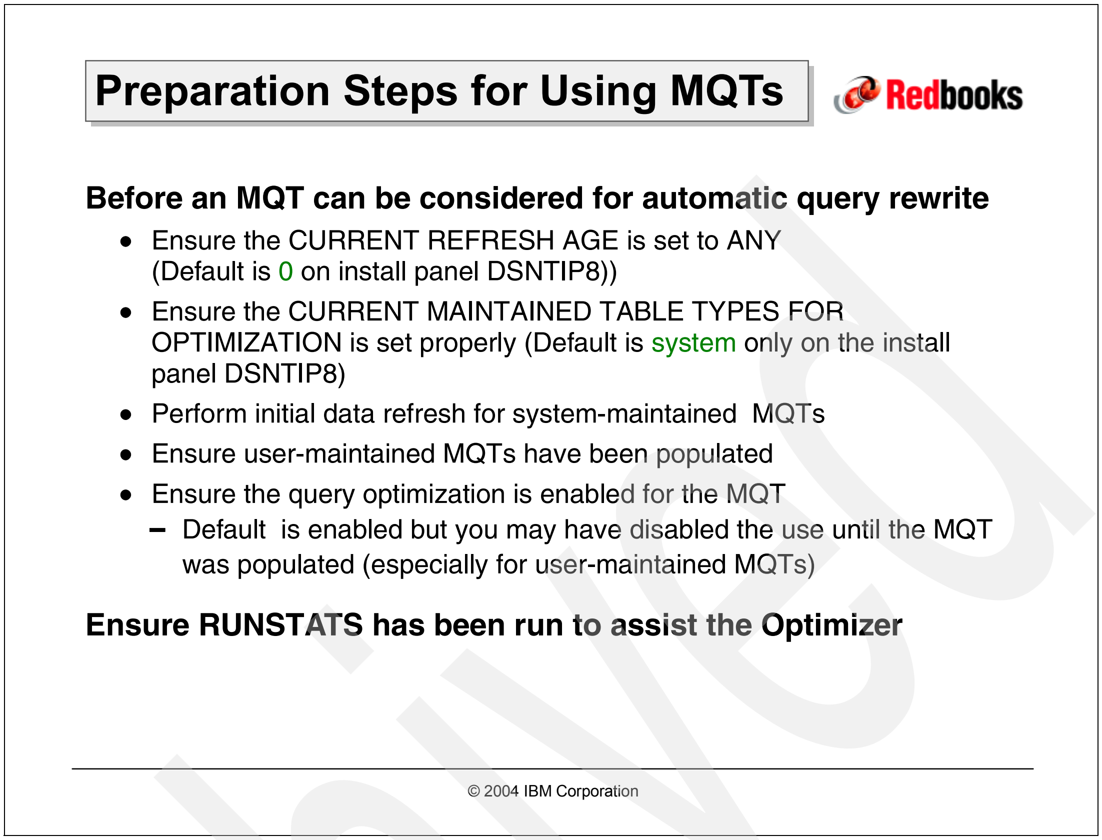
Relationship Between Two Special Registers for MQTs

		CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION			
		SYSTEM	USER	ALL	NONE
CURRENT REFRESH AGE	ANY	All system-maintained query optimization enabled MQTs	All user-maintained query optimization enabled MQTs	All query optimization enabled MQTs	None
	0	None	None	None	None

Relationship between two special registers for MQTs

Use the table above to find out which types of MQTs are considered for automatic query rewrite, depending on the actual settings of the special registers, CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION and CURRENT REFRESH AGE in your application, or as defaulted by their respective DSNZPARM values.

As you can see, setting the special register CURRENT REFRESH AGE to 0, or just accepting the default value for DSNZPARM REFSHAGE (0-zero), prevents DB2 from using MQTs during automatic query rewrite.



10.2.8 Preparation steps for using MQTs

The visual above summarizes the steps that you must take care of, in order to prepare your MQT for being considered by automatic query rewrite.

If you just accept the system wide defaults, which are set during the installation/migration of your DB2 subsystem, no MQT will ever be considered during automatic query rewrite. You must set the value for CURRENT REFRESH AGE to ANY. This can either be done system wide, by specifying REFSHAGE=ANY in your DSNZPARM, or as a session parameter by using the SET CURRENT REFRESH AGE = ANY statement.

Once you have enabled automatic query rewrite, you may also have to change the default for the CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION special register. If you accept the default, only system-maintained MQTs are taken into account.

Apart from setting the appropriate values for the aforementioned special registers, you must also make sure that your MQTs themselves have been created or altered specifying the ENABLE QUERY OPTIMIZATION clause. If QUERY OPTIMIZATION is DISABLED, your MQTs are not affected by the special registers described above.

As mentioned before, system-maintained MQTs that have not been populated with data are not considered for automatic query rewrite.

Also make sure to run RUNSTATS against the MQT after refreshing the content of the MQT, to ensure that the optimizer has accurate statistics when determining the access path for your queries. Otherwise, DB2 uses default or out-of-date statistics. The estimated performance of queries that are generated by automatic rewrite might inaccurately compare less favorably to the original query. When you run the REFRESH TABLE statement, the only statistic that DB2 updates for the MQT is the cardinality statistic (CARDF).

Another thing you must be aware of, is that only MQTs with an isolation level equal to, or higher than the query isolation level, will be considered by automatic query rewrite.

The slide below is printed inside a box on the page:

The Basics of Automatic Query Rewrite

State of the MQT (that is, must be ready for query optimization and not locked)

Whether the query result can be derived from or directly use the MQT

- There are no predicates in the MQT subselect that are not in the query
- Each column in the result table can be derived from one or more columns in the MQT
- The GROUP BY clauses are compatible

Estimated cost of the queries

- MQTs are not used for short-running queries
- The original and rewritten costs are compared and the lower cost chosen

Only considered for read-only queries

- Dynamically prepared queries only
- Not for static SQL or static SQL with REOPT(VARS)

Note that optimization is considered at the query block level

10.2.9 The basics of automatic query rewrite

Automatic query rewrite is the general process that examines an SQL statement which references one or more base tables, and, if appropriate, rewrites the query so that it performs better. This process also determines whether to rewrite a query so that it refers to one or more materialized query tables that are derived from the source tables. (Another form of query rewrite is the generation of additional predicates that can be derived from predicates coded in the SQL statement; this is also known as predicate transitive closure.)

A user query can contain multiple query blocks. Examples for query blocks are, subselect of UNION or UNION ALL statement, temporarily materialized views, materialized table expressions, and subquery predicates. Automatic query rewrite is generally considered at query block level.

The qualified query block in the user query and the subselect in the MQT definition are analyzed to determine whether the query can be rewritten to an equivalent one using the MQT and provide the same results with better performance. Here the overriding principle is that the MQT must contain the source table data needed to satisfy the query.

Furthermore, the following conditions are checked:

- In general, there cannot be any predicates in the MQT subselect, which are not in the query, because in this case, it can be assumed that these predicates may have resulted in discarded rows as the MQT was refreshed.
- GROUP BY clauses are compared to determine if a query GROUP BY clause will result in a subset of the rows that are in the MQT.
- The query select list is examined to determine if each column in the result table can be derived from one or more columns from the MQT.

If the characteristics described above apply for both the MQT and the query, the query is rewritten so that all, or parts of the references to base tables are replaced by references to the MQT. If the query rewrite process is successful, DB2 determines the cost and the access path of the new (rewritten) query. Only if the cost of the new query is less than the cost of the original query, will the new query be submitted.

In addition, MQTs are not used for short-running queries. It would take the optimizer more time to figure out whether or not the MQT can be used, than to actually execute the query.

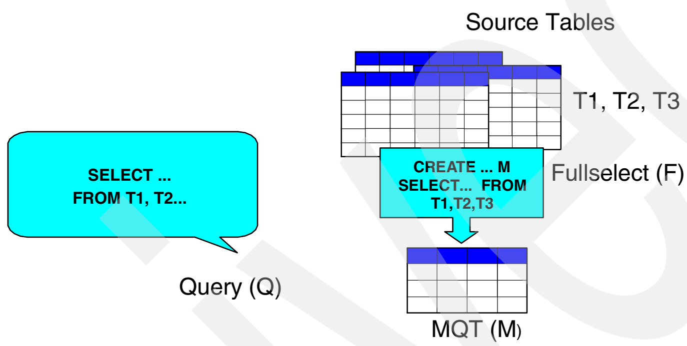
Apart from all the things mentioned above, automatic query rewrite is only supported for *dynamically prepared queries that are read-only*. It is not supported for statically bound queries, including REOPT(VARS) packages. You can, however, use an MQT in either a statically bound query, or a dynamically prepared query to improve the response time, if appropriate, by coding them directly in the SQL statement (“bypassing” automatic query rewrite done by the optimizer).

To determine whether an MQT is used instead of a base table, you can use the EXPLAIN statement. The EXPLAIN output shows the plan for the rewritten query. When used, the name of the MQT is shown in the TNAME column, and a TABLE_TYPE of 'M' is used.

Automatic Query Rewrite Table Comparison

Source tables referenced in the fullselect are compared to the set of source tables referenced by the query

If there are source tables in common, the materialized query table is a possible candidate for query rewrite



Automatic query rewrite table comparison

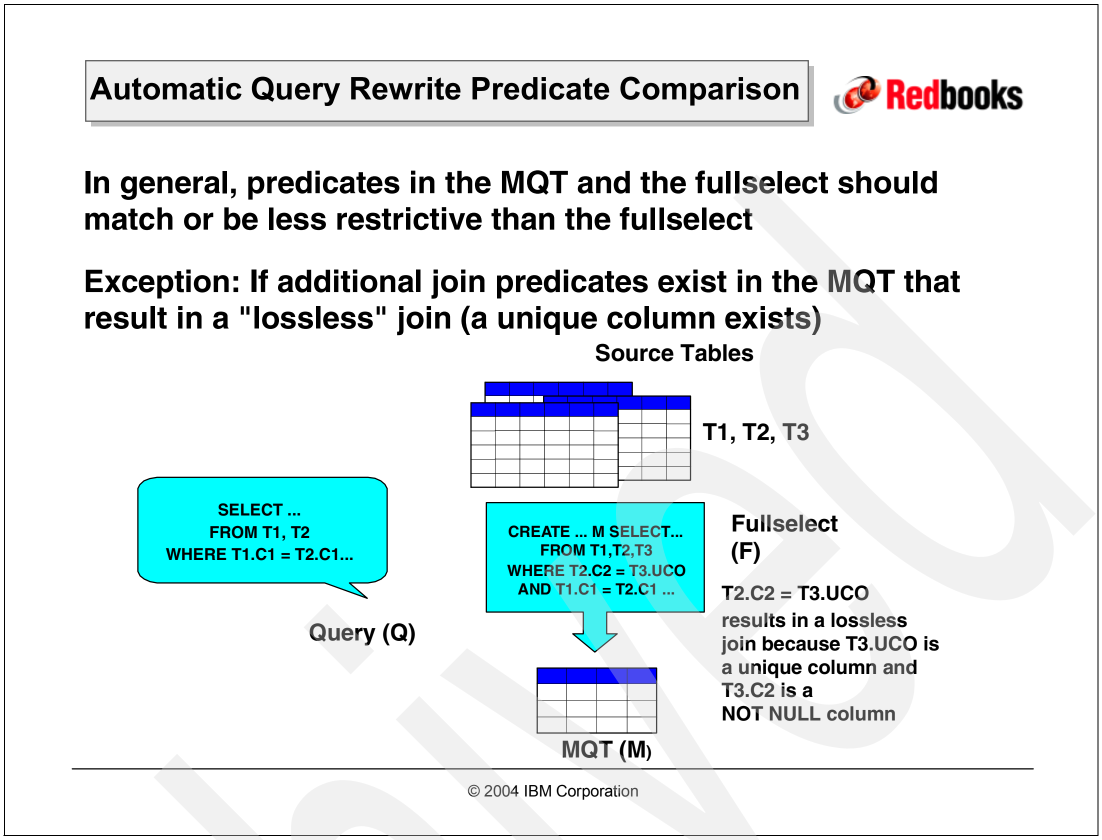
The following pages give a high level step-by-step overview of the rules used by automatic rewrite to determine eligibility of MQTs.

We use the following abbreviations:

- M to denote the MQT
- Q to reference the user query
- F for the fullselect that makes up the MQT’s content

The first and probably easiest checking that is done pertains the source tables that are referenced in the query.

As you can see from the visual above, tables T1 and T2, which are referenced in the SELECT statement, have been used in the definition of the MQT.



Automatic query rewrite - predicate comparison

In general, the materialized query table fullselect (F) should contain the same predicates as the user query (Q). If the MQT fullselect (F) contains predicates that are not in the user query (Q), DB2 assumes that these predicates may result in discarded rows when the materialized query table was refreshed. Thus, any rewritten query that makes use of the materialized query table might not give the correct results. The query is not a candidate for query rewrite.

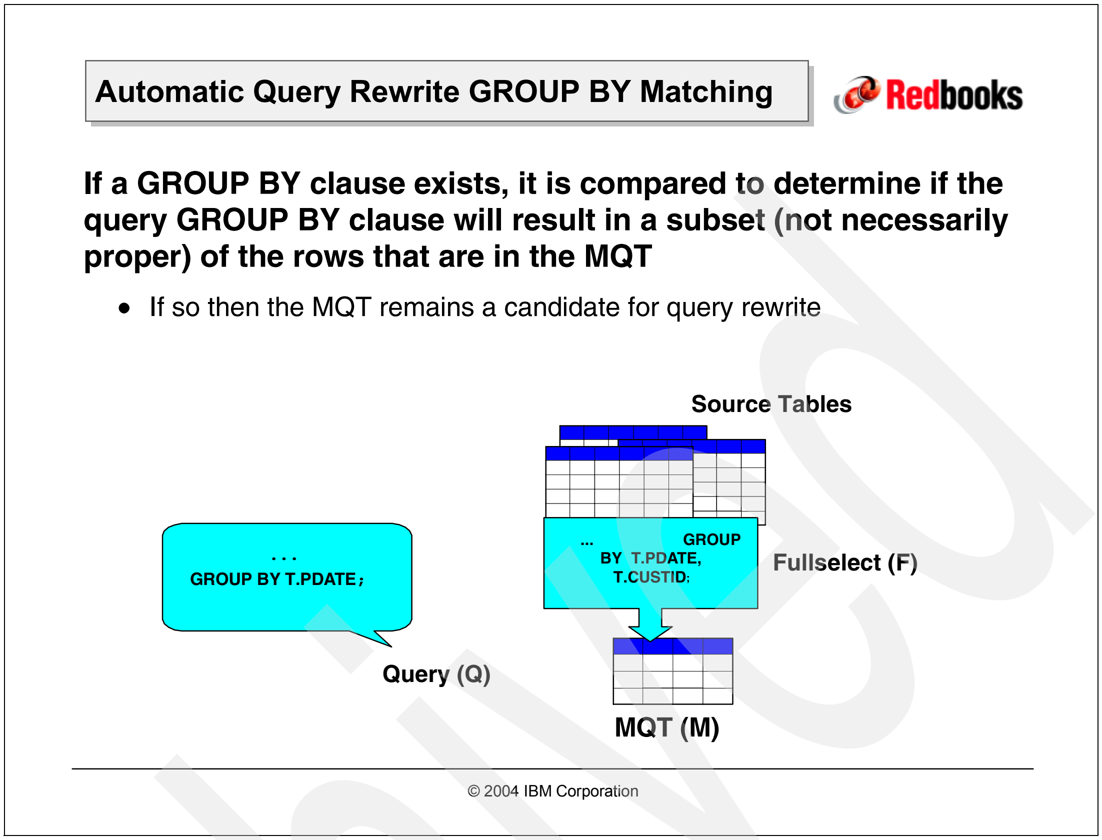
However, DB2 behavior differs if a predicate joins a common base table (in both Q and F) to an extra table that is unique to the materialized query table fullselect (F). The predicate does not result in discarded data if DB2 can determine that the join is lossless, for example, through the existence of a referential constraint (PK-FK) between the two base tables. However, the materialized query table fullselect must not have any local predicates that reference this extra table; otherwise, it could again reduce the number of rows returned by the MQT.

In the visual above, the T2.C2 = T3.UCO predicate only occurs in the MQT, not in the fullselect. However, assuming that T2.C2 is a NOT NULL column and T3.UCO is a unique column, having this extra join predicate (and no additional filtering predicates on T3) does not reduce the number of rows in the result set of the MQT. It is a so-called lossless join, and therefore the MQT can be substituted for the underlying base tables. The optimizer can determine that this is a lossless join when an RI relationship is defined between T2.C2 and T3.UCO. More information on the importance of RI constraints when using MQTs can be found in 10.2.11, “The role of constraints” on page 729.

It is recommended that the predicates in the query be coded in exactly the same way as they are in the MQT subselect, because otherwise the matching may fail on some complex predicates.

For example, the matching between the simple equal predicates such as COL1=COL2 and COL2=COL1 will be successful. Extra blanks are also ignored. In contrast to that, the matching between (COL1+3)*10=COL2 and COL1*10+30=COL2 will fail.

The IN-list predicates are an exception; the order of the items in the IN-list need not be in exactly the same order.



Automatic query rewrite GROUP BY matching

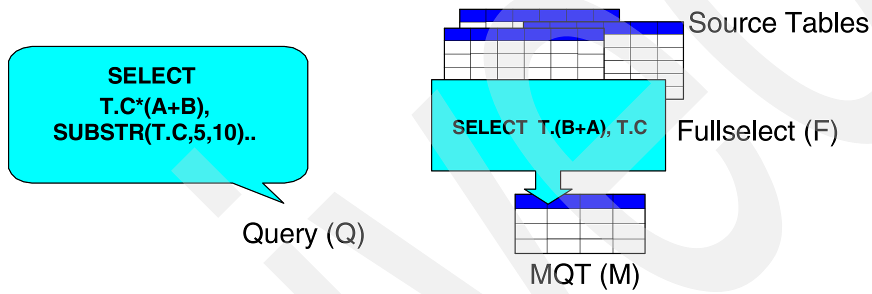
In a subsequent step, DB2 checks if the GROUP BY clause used in the query will result in a subset of the rows that are in the MQT.

The visual above shows an MQT which fulfills this requirement. T.PDATE is used in the query and in the MQT's fullselect. As the MQT's GROUP BY clause is more granular (at a lower level, since it also includes the T.CUSTID column), grouping by T.PDATE can be derived from the MQT's content.

Automatic Query Rewrite Expression Derivation

Determine if the column(s) in the result table requested by the query can be derived from one or more columns in the MQT

- The columns must be equivalent, i.e.T1.C1 = T2.C1 or
- Column references to the base table can be derived from the MQT
  - For example, SUBSTR(C, 5, 10) can be derived from column C in the MQT (M)
  - C*(A+B) can be derived from (B+A) column and C column in M



Automatic query rewrite expression derivation

Automatic query rewrite also checks whether the columns in the select list of the query can be derived from the data in the MQT.

In the visual, C*(A+B) in the user query can be derived from the result column of the expression B+A and column C, and SUBSTR(*C,5,10) can be derived from column C in the MQT.

Automatic Query Rewrite Heuristics of Selection

If there are multiple MQTs that match the query(Q), heuristic rules are applied

- **1st choice:** MQTs that involve no regrouping, no residual joins (that is, table left in Q), and no rejoins
- **2nd choice:** MQTs that involve no regrouping and no residual joins
- **Final criteria**: The MQT with the largest reduction power: |T1|*...*|Tn| / |M|, where T1, ..., Tn are base tables in F

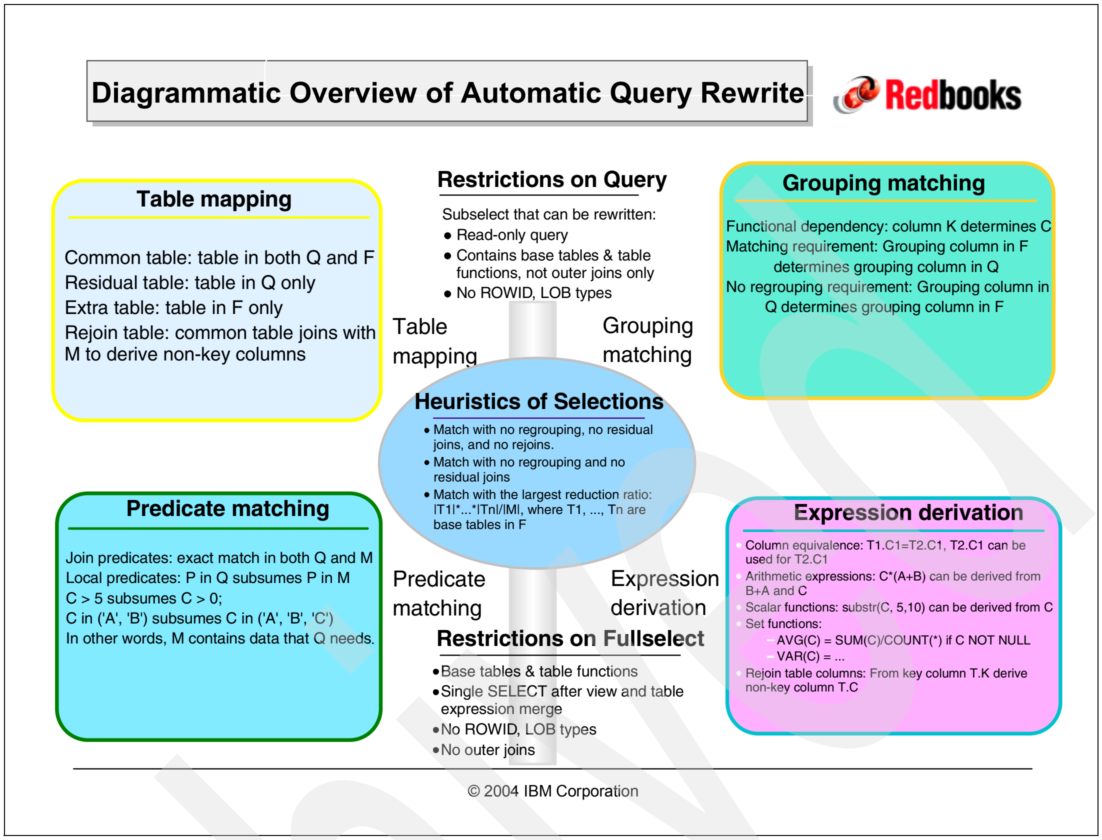
Certain restrictions apply on the Query (Q) and the MQT fullselect (V) for automatic query rewrite

- See next visual for 'restrictions on F' and 'Restrictions on Q'

Automatic query rewrite MQT selection heuristics

If there are multiple MQTs that match the query but cannot be used simultaneously, heuristic rules are used to choose one of them.

In general, if multiple MQTs apply, the optimizer chooses the one(s) that require the least amount of additional processing after using the data from the MQT.



Summary of automatic query rewrite for materialized query tables

The visual above summarizes the checking that is performed for a query, if automatic query rewrite is generally enabled in your DB2 subsystem.

MQT Exploitation - Simple Example

```
QUERY:
SELECT T.PDATE, AVG (QTY * PRICE) AVGAMT
FROM SCNDSTAR.TRANSITEM TI, SCNDSTAR.TRANS T
WHERE TI.TRANSID = T.TRANSID AND
      T.PDATE >= '2001-01-01'
 GROUP BY T.PDATE;

AFTER QUERY REWRITE:
SELECT PDATE, CASE WHEN SUM(CNT) = 0 THEN NULL
              ELSE SUM(TOTVAL)/SUM(CNT) END AVGAMT
FROM MQT1
WHERE PDATE >= '2001-01-01'
GROUP BY PDATE;
```


10.2.10 MQT exploitation - simple example

The example above is used to give you an impression of what a rewritten query may look like.

The example refers to the sample MQT that we introduced earlier in this chapter. For ease of use, the CREATE TABLE statement is repeated in Example 10-1.

Example 10-1 Sample MQT creation statement

```
CREATE TABLE MQT1 AS (
 SELECT T.PDATE, T.TRANSID,
        SUM(QTY * PRICE) AS TOTVAL,
        COUNT(QTY * PRICE) AS CNT
 FROM SCNDSTAR.TRANSITEM TI, SCNDSTAR.TRANS T
 WHERE TI.TRANSID = T.TRANSID
 GROUP BY T.PDATE, T.TRANSID)
DATA INITIALLY DEFERRED
REFRESH DEFERRED
MAINTAINED BY SYSTEM
ENABLE QUERY OPTIMIZATION
IN MYDBMQT.MYTSMQT;
```

The prerequisites for automatic query rewrite, described in 10.2.9, “The basics of automatic query rewrite” on page 718, are fulfilled for the initial query. Let us look at it in detail now:

- The query uses the same tables as the MQT.
- The only predicate in the MQT is TI.TRANSID = T.TRANSID. This one is also present in the query. The query has additional predicates, which is fine, as it does not preclude the use of the MQT.
- The way the GROUP BY clause is coded in the query indicates that the query will result in a subset of the rows that are in the MQT. The GROUP BY in the MQT is more granular than the query.
- Each column of the query can be derived from, or is a column of, the MQT. The AVG(QTY * PRICE) can be derived from SUM(QTY * PRICE)/COUNT(QTY * PRICE).

The Role of Constraints

Constraints are very important in determining whether a materialized query table can be used for a query

- Referential constraints (to allow extra lossless joins in MQT, or some dimensions missing in a query)
- Functional dependencies (primary keys and unique indexes, to allow more group-by columns in a query)

V8 allows informational referential constraints to be specified for base tables, and avoids the enforcement overhead (in a data warehouse environment)

10.2.11 The role of constraints

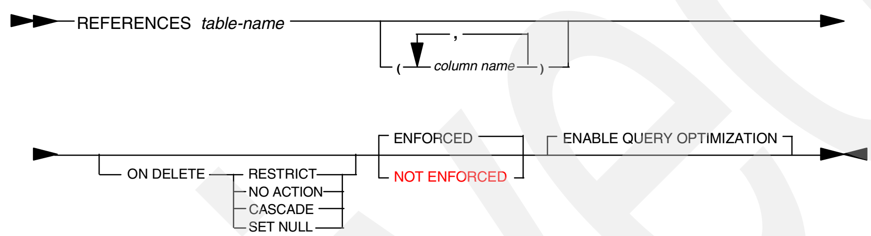
Referential constraints between base tables are an important factor in determining whether an MQT can be used for a query. For this reason, informational referential integrity constraints are introduced in DB2 V8. They allow you to declare referential constraints (primary - foreign key relationships), but avoid the overhead of enforcing the referential constraints by DB2 at the same time. DB2 can take advantage of the referential constraints during automatic query rewrite.

Informational Constraints

Informational RI is not enforced by database manager and is ignored by most utilities

- Except LISTDEF RI, QUIESCE/REPORT TABLESPACESET

Informational RIs are always used by the optimizer in query rewrite



Informational RI syntax

The extract of the CREATE or ALTER TABLE syntax diagram above shows that V8 allows you to use the NOT ENFORCED clause to define informational RI on a table.

As stated in the visual, most utilities are not affected by the new possibility to define informational constraints on your tables. For more details on the use of informational RI by utilities, see 9.11, “Utility changes to support informational RI constraints” on page 675.

Informational RI Example

```
CREATE TABLE SCNDSTAR.TRANS
  (TRANSID  CHAR(10) NOT NULL PRIMARY KEY,
   ACCTID   CHAR(10) NOT NULL,
   PDATE    DATE NOT NULL,
   STATUS   VARCHAR(15),
   LOCID    CHAR(10) NOT NULL,
   CONSTRAINT ACCTTRAN  FOREIGN KEY (ACCTID)
      REFERENCES SCNDSTAR.ACCT NOT ENFORCED,
   CONSTRAINT LOC_ACCT FOREIGN KEY (LOCID)
      REFERENCES SCNDSTAR.LOC NOT ENFORCED
  )
 IN DBND0101.TLND0101;
```


Informational RI example

The example in the visual above demonstrates the usage of the NOT ENFORCED parameter in a CREATE TABLE statement, to define two informational constraints ACCTTRAN and LOC_ACCT on table SCNDSTAR.TRANS.

MQT Administration and Considerations

Redbooks

Information about MQTs is stored in SYSIBM.SYSVIEWS

No primary keys, unique indexes or triggers are allowed on MQTs

All the MQTs and their indexes are dropped if their associated base tables are dropped

Specialized MQTs or generalized MQTs?

- Design issue as to whether to have a few generic MQTs or lots of more specialized MQTs to help with particular queries
- Trade off between performance and maintenance

10.2.12 MQT administration and considerations

After the successful creation of an MQT, the fullselect used to define it is stored in SYSIBM.SYSVIEWS.

The REFRESH_TIME column contains the default timestamp at creation time.
It contains the value of the CURRENT TIMESTAMP special register after the REFRESH TABLE statement finishes successfully (or after altering the MQT to a system-maintained MQT).

Since you cannot create a primary key for an MQT and no unique index can be created, an MQT can never be a parent table in a referential constraint. Note that you can create a primary key and unique indexes on the underlying base tables of an MQT. Index uniqueness is derived from definition when processing a query that references an MQT.

Consider creating your MQTs using the DISABLE QUERY OPTIMIZATION option. Otherwise, it is possible that queries are rewritten to use the empty MQT when using user-maintained MQTs.

If you drop a base table, all associated MQTs and their indexes are dropped as well.

The design of MQTs involves trade-offs between conflicting design objectives. On the one hand, automatic MQTs that are specialized to a particular query or a set of queries can lead to the greatest performance benefits. This approach can also lead to a proliferation of MQTs, since many are needed to support a wide variety of queries. Since automatic MQTs can be expensive to define and keep current, this approach can be expensive

On the other hand, MQTs whose purpose is more general, that is, which support a large number of submitted queries will often tend to provide less performance improvement, but easier maintenance, because there will be fewer of them.

In order to be able to make accurate decisions on which MQTs are needed, you need to fully understand the query workload that is run against the underlying base tables in your system.

Summary of Changes

Redbooks

SQL Statements

- CREATE TABLE / ALTER TABLE (extended)
- REFRESH TABLE (new)

Special Registers

- CURRENT REFRESH AGE (new)
- CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION (new)

Catalog Changes

- New table/view type 'M' for MQT in many catalog tables
- New columns for informational RIs in SYSIBM.SYSRELS
- New column NUM_DEP_MQTS in SYSIBM.SYSTABLES and SYSIBM.SYSROUTINES
- New columns REFRESH_TIME, REFRESH, ENABLE, ISOLATION, MAINTENANCE, and SIGNATURE in SYSIBM.SYSVIEWS

10.2.13 Summary of changes

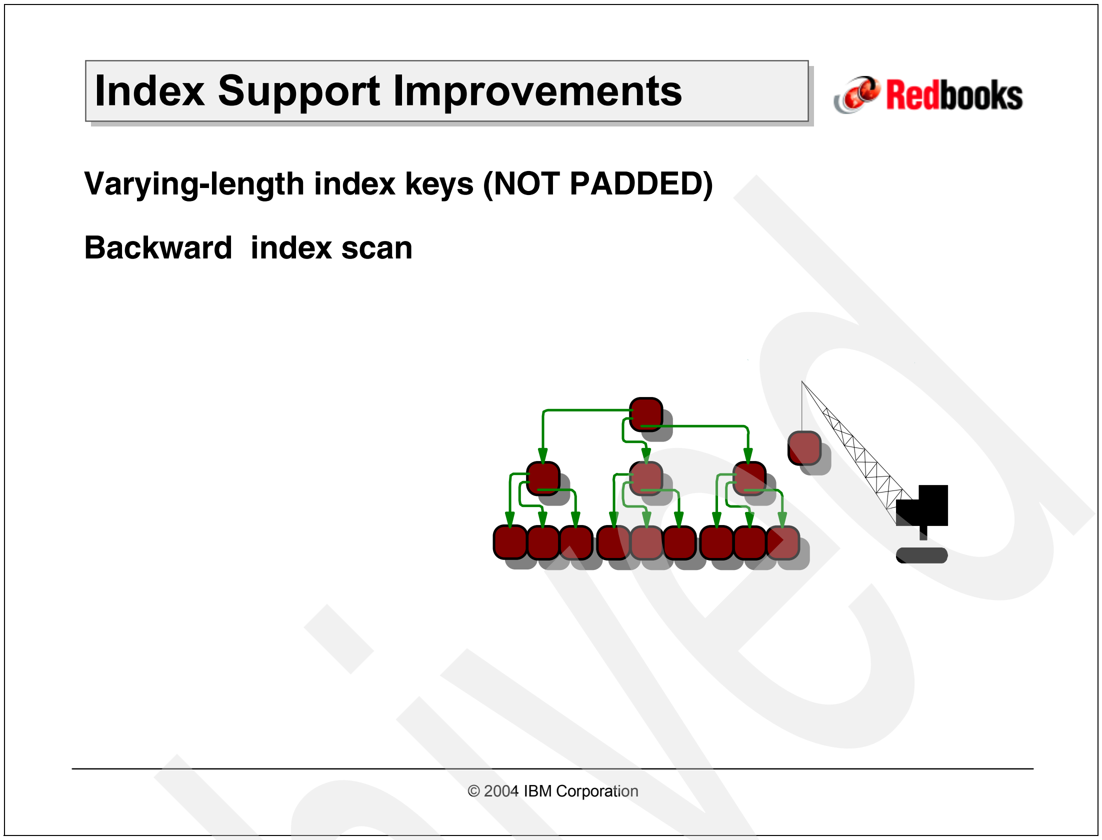
Type ‘M” for MQTs is stored in SYSIBM.SYSTABLES, SYSIBM.SYSVIEWS, SYSIBM.SYSVIEWDEP, SYSIBM.SYSPLANDEP, SYSIBM.SYSPACKDEP, SYSIBM.SYSVTREE, and SYSIBM.SYSVLTREE.

There are two new columns in catalog table SYSIBM.SYSRELS; they are ENFORCED and CHECKEXISTINGDATA. If the value of ENFORCED is set to N, the entry belongs to an informational RI constraint. CHECKEXISTINGDATA basically contains the same information. If ENFORCED is set to N, CHECKEXISTINGDATA is also always set to N.

Both SYSIBM.SYSTABLES and SYSIBM.SYSROUTINES are expanded by column NUM_DEP_MQTS, which contains the information about how many MQTs are dependent on a table or a table UDF respectively.

In addition to that, table SYSIBM.SYSVIEWS has six new columns that contain information related to MQTs:

- REFRESH - ‘D’ for deferred refresh mode; or blank, which means the row does not belong to an MQT.
- ENABLE - ‘Y’ or ‘N’ for QUERY OPTIMIZATION enablement, or blank for a view.
- MAINTENANCE - ‘S’ for system-maintained, ‘U’ for user-maintained or blank for view.
- REFRESH_TIME - only used by system-maintained MQTs. It indicates the timestamp of last REFRESH TABLE statement.
- ISOLATION - Isolation level when MQT is created or altered from a base table.
- SIGNATURE - Contains an internal description of the MQT.



10.3 Index support improvements

In this section we discuss two new functions:

- Varying-length index keys
- Backward index scan

Varying-Length Index Keys

Prior to V8, data is varying length in the table but padded to the maximum length in indexes

In V8, padding or not padding varying-length columns to the maximum length in index keys is an option that you can choose

Performance Benefits:

- "Index-only access" for VARCHAR data
- Normally reduces the storage requirements for indexes since only actual data is stored

10.3.1 Varying-length index keys performance benefits

Prior to DB2 V8, VARCHAR and VARGRAPHIC columns are padded to their maximum lengths when they are part of an index, but remain in their variable length format in the tables. This facilitates fast key comparisons since these comparisons are between equal length columns. The disadvantage to this approach is that index only access is not allowed when retrieving a varying length key column.

Prior to V8, you can use the RETVLCFK=YES DSNZPARM (Panel DSNTIP4 Install DB2 - Application Programming Defaults Panel 2, field VARCHAR FROM INDEX). This allows you to use VARCHAR columns of an index and still have index only access. However, when one of the columns in the SELECT list of the query is retrieved from the index when using index-only access, the column is padded to the maximum length, and the actual length of the variable length column is not provided. Therefore, an application must be able to handle these “full length” variable length columns. DB2 V7 enhances this feature by allowing index-only access against variable length columns in an index, even with RETVLCFK=NO, if no variable length column is present in the SELECT list of the query.

DB2 V8 supports true varying-length key columns in an index. Varying-length columns are not padded to their maximum lengths, if you choose that option. This reduces the storage requirements for this type of index, since only actual data is stored. Furthermore, this allows for index-only access to index key columns of varying-length in all cases, and since the length of the variable length column is stored in the index, it can potentially improve performance.

Indexes can be created or altered to contain true varying-length columns in the keys. Padding of both VARCHAR and VARGRAPHIC data to their maximum length can now be controlled.

You can continue to use existing indexes that contain padded varying-length columns. However, with DB2 V8, you have the ability to convert padded indexes to varying-length indexes and also to convert varying-length indexes back to padded indexes.

Varying-Length Index Keys - 2

New Keywords in CREATE/ALTER INDEX

- NOT PADDED
  - Require additional bytes to store the length information for the variable length columns within the key
- PADDED
  - Varying-length columns are padded to maximum length
  - May be a better option for performance at the expense of storage because of fast key comparisons since all keys have same length

Indexes are not automatically converted to NOT PADDED in V8

- ALTER INDEX is required

The default is controlled by the PADIX DSNZPARM

```
CREATE UNIQUE INDEX INDEX1
        ON TABLE1(COL01,VARCOL02) NOT PADDED
```


Varying-length index keys - CREATE/ALTER INDEX changes

The new keywords NOT PADDED and PADDED on CREATE INDEX and ALTER INDEX statements specify how varying-length columns are stored in the index.

- NOT PADDED specifies that varying-length columns are not padded to their maximum length in the index. If there exists at least one varying-length column within the key, length information is stored with the key. For indexes composed of only of fixed length columns, there is no length information added to the key.

  The default on the CREATE INDEX statement can be controlled through the new DSNZPARM PADIX (Panel DSNTIPE Install DB2 - Thread Management, field PAD INDEXES BY DEFAULT). A sample create of a non-padded index is:

```
CREATE UNIQUE INDEX DSN8810.XDEPT1
ON DSN8810.DEPT (DEPTNO ASC)
NOT PADDED
USING STOGROUP DSN8G810
PRIQTY 512
SECQTY 64
ERASE NO
BUFFERPOOL BP1
CLOSE YES
PIECESIZE 1 M;
```

- PADDED specifies that varying-length columns within the index are always padded with the default pad character to their maximum length. All indexes prior to DB2 V8 new-function mode are padded by default. A sample create of a padded index is:

```
CREATE UNIQUE INDEX DSN8810.XDEPT1
ON DSN8810.DEPT (DEPTNO ASC)
PADDED
USING STOGROUP DSN8G810
PRIQTY 512
SECQTY 64
ERASE NO
BUFFERPOOL BP1
CLOSE YES
PIECESIZE 1 M;
```

When comparisons are made between keys with varying-length columns, the keys have to match in length. This requires that like columns of different sizes have the smaller column padded to the size of the larger column. Key comparison is left to right and column by column. The following example illustrates this using a single column index:

```
Key entry 1 length=4 Value= x'F1F2F3F4'
Key entry 2 length=3 Value= x'F1F2F3'
Pad character = x'40'

After padding, Key 2 = x'F1F2F340'
When Key entry 1 and Key entry 2 are compared
Key value 1 > Key value 2
```

Indexes are not automatically converted to NOT PADDED, except for the system defined indexes on the DB2 catalog. They are converted to NOT PADDED as part of the enable new function mode processing.

ALTER INDEX Changes

Alter index from PADDED to NOT PADDED

- Index placed in RBDP state
- No access to data through this index
- Reset by REBUILD INDEX

ALTER INDEX INDEX1 NOT PADDED

Alter index from NOT PADDED to PADDED

- Index placed in RBDP state
- No access to data through this index
- Reset by REBUILD INDEX

ALTER INDEX INDEX1 PADDED

ALTER INDEX changes

Indexes from a prior release do not automatically convert to NOT PADDED, even if an ALTER TABLE ALTER COLUMN SET DATATYPE statement is executed and the altered column is part of an the index. You have to use the ALTER INDEX statement to change a PADDED index to NOT PADDED.

After an index has been altered to NOT PADDED, the index is placed in rebuild pending state, if there exists at least one varying-length column in the index. A REBUILD of the index is necessary to realize the full benefit of a NOT PADDED index.

Altering a PADDED index to a NOT PADDED index can be done as shown in the following example:

```
ALTER INDEX DSN8810.XDEPT1 NOT PADDED
```

When altering a NOT PADDED index to a PADDED index, the index is placed in rebuild pending state, if there exists at least one varying-length column in the index.

Altering a NOT PADDED index to a PADDED index can be done as shown below:

```
ALTER INDEX DSN8810.XDEPT1 PADDED;
```

Note that DB2 tried to mitigate the effect of having an index in RBDP status. See 3.14.4, “RBDP Index avoidance” on page 185.

Performance Expectation

Factors affecting performance of PADDED / NOT PADDED index:

NOT PADDED indexes can provide a true index-only access path

Size of the index pages

- In general, number of index pages for NOT PADDED index are fewer than for the PADDED index, thus less index getpages and index page I/Os
- Potentially fewer index levels
- Smaller index size favors the index to be chosen for a query (Index-only access)

Extra comparison

- NOT PADDED indexes that contain multiple VARCHAR columns require more than one comparison

Performance expectation

NOT PADDED indexes can provide a number of *advantages* over PADDED indexes:

- NOT PADDED indexes allow for true index-only access. If you have applications that today cannot take advantage of index-only access because they contain VARCHAR columns in the index, NOT PADDED indexes can allow for index-only access and may improve performance for those queries.
- NOT PADDED indexes are smaller in size than their PADDED counterparts. Smaller indexes can mean fewer index pages to scan, and potentially fewer index levels.

The *disadvantage* of using NOT PADDED indexes is that they are more complex to process. DB2 has to read the length of the NOT PADDED column in the index in order to determine where the next column start. With fixed length columns, this is much easier. Processing a NOT PADDED index with VARCHAR keys can require a significant amount of additional CPU time, just like VARCHAR columns in a data record.

NOT PADDED index performance is heavily dependent on the number and size of VARCHAR columns in the index key, as column comparisons are more expensive.

Backward Index Scan

In V8, DB2 selects an ascending index and can use a backward scan to avoid the sort for the descending order

In V8, DB2 uses the descending index to avoid the sort and can scan the descending index backwards to provide the ascending order

To be able to use an index for backward scan,

- Index must be defined on the same columns as ORDER BY and
- Ordering must be exactly opposite of what is requested in ORDER BY
  - If index defined as DATE DESC, TIME ASC, can do:
    - Forward scan for ORDER BY DATE DESC, TIME ASC
    - Backward scan for ORDER BY DATE ASC, TIME DESC
- But must sort for
  - ORDER BY DATE ASC, TIME ASC
  - ORDER BY DATE DESC, TIME DESC

10.3.2 Backward index scan

With the enhancements introduced to support dynamic scrollable cursors, DB2 also provides the capability for backward index scans. This allows DB2 to avoid a sort and/or allows you to define fewer indexes. With this enhancement it is no longer necessary to create an ascending and descending index on the same table columns. The visual shows an example.

For another example, if you create an ascending index (the default) on the ACCT_NUM, STATUS_DATE and STATUS_TIME columns of the ACCT_STAT table, DB2 can use this index for backward index scanning for the following SQL statement:

```
SELECT STATUS_DATE, STATUS
FROM ACCT_STAT
WHERE ACCT_NUM = :HV
ORDER BY STATUS_DATE DESC, STATUS_TIME DESC
```

DB2 can use the same index for forward index scan for the following SQL statement:

```
SELECT STATUS_DATE, STATUS
FROM ACCT_STAT
WHERE ACCT_NUM = :HV
ORDER BY STATUS_DATE ASC, STATUS_TIME ASC
```

This is true also for static scrollable cursors and non-scrollable cursors. In V7 you have to create two indexes for the above to avoid a sort for both queries.

To be able to use the backward index scan, you have to create the index on the same columns as the ORDER BY and the ordering must be exactly opposite of what is requested in the ORDER BY.

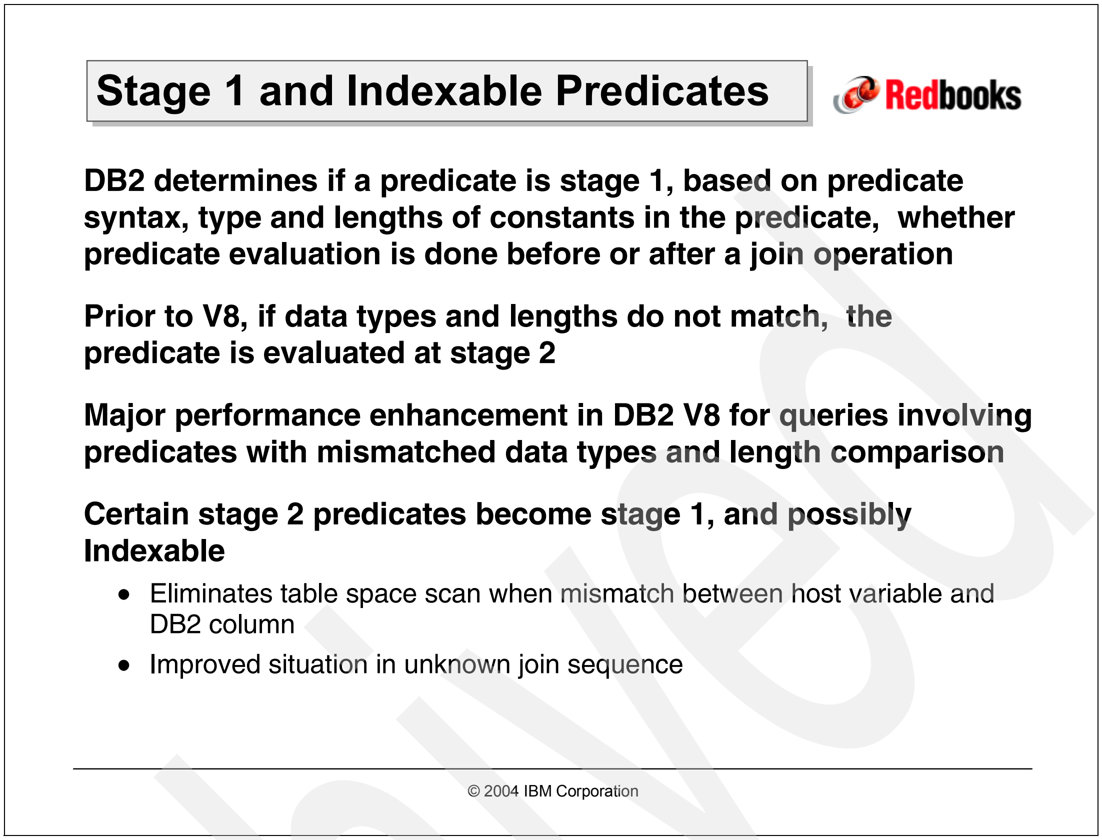
For example, if you create the index as ACCT_NUM, STATUS_DATE DESC, STATUS_TIME ASC, then DB2 can do a:

- Forward index scan for ORDER BY on STATUS_DATE DESC, STATUS_TIME ASC
- Backward index scan for ORDER BY STATUS_DATE ASC, STATUS_TIME DESC.

DB2 has to perform a sort for:

- ORDER BY STATUS_DATE DESC, STATUS_TIME DESC
- STATUS_DATE ASC and STATUS_TIME ASC

A backward index scan takes advantage of sequential detection to trigger dynamic prefetch to read 32 index pages backward as needed. This can improve I/O performance by an order of magnitude compared to a synchronous read of index pages one page at a time.



10.4 Stage 1 and indexable predicates

Stage 1 predicates are simple predicates evaluated by the Data Manager (DM). They are evaluated first to reduce processing cost and eliminate the number of rows to be evaluated by the complex predicates. They are also known as “sargable” predicates.

Stage 2 predicates are complex predicates evaluated by the Relational Data System (RDS). They are also known as “nonsargable” or residual predicates.

An indexable predicate is a predicate that can use a (matching) index access as an access path. Indexable predicates are always stage 1, but not all stage 1 predicates are indexable.

DB2 determines if a predicate is stage 1 based on:

- Predicate syntax
- Predicate type and lengths of constants
- Predicate evaluation done before or after join operation

DB2 V8 has introduced enhancements to facilitate major performance enhancement for queries involving predicates with mismatched data types and length comparison and also in joins.

Mismatched Data Types

Redbooks

Becoming more common as not all programming languages support the full range of SQL data types, for example

- C/C++ has no DECIMAL data type
- Java has no fixed character data type

The following predicate types become stage 1 and also indexable for unlike data types

- Col op expression
- Expression op col
- Col BETWEEN expression 1 and expression 2

10.4.1 Mismatched data types

When you do a database and application design, you normally make sure that the data type of the columns in your tables match with the data types used by the host variables in your programs. This has always been a good design rule (and still is) because it allows DB2 to use certain techniques (like using an index) to boost performance.

However, it has become more and more difficult to apply this rule in all situations, especially when you build new applications to access existing data (with an existing design) on a DB2 for z/OS system. For example, when your application is coded in C, that language does not have a DECIMAL data type, although some of the existing tables might have columns defined as DECIMAL(p,s). Note that the C/C++ compiler for z/OS supports fixed-point (packed) decimal data type.

Another case is Java. The Java language does not have a fixed length character string data type; every string is variable length. In DB2, on the other hand, in most cases, fixed length character columns defined as CHAR(n) are used.

Prior to DB2 V8, for many types of predicates, if the data types of the predicate operands do not match, then the predicate is considered residual, also known as stage 2, and its treatment can have a negative effect on the performance of the query.

So when you run a simple SELECT statement in a Java application as shown below:

```
SELECT RESOURCE_GROUP,RESOURCE_OPTION,INTVAL,CHARVAL
FROM Q.RESOURCE_TABLE
WHERE RESOURCE_GROUP = :hv_res_gr
```

DB2 cannot use an index on RESOURCE_GROUP because of the mismatch in data type of the column and the host variable. The data type of the RESOURCE_GROUP column is CHAR, and that of the :hv_res_gr host variable is VARCHAR (since that is the only string data type supported by Java).

In addition, it is sometimes also necessary to join tables on columns with different data types, also resulting in not maximizing performance. (Joining on CHAR and VARCHAR columns does not cause performance problems.)

DB2 V8 provides improved performance of queries that involve predicates with mismatched data types. Now those predicates can be processed at stage 1, and can possibly also use an index (subject to certain restrictions).

Processing the following types of predicates is improved by this enhancement:

- *col* op *expression*
- *expression* op *col*
- *col* BETWEEN *expression1* AND *expression2*
- *col* IN (*list*)

In these expressions:

- *'col'* is the column name of a table.
- *'expression'* is any expression. It may contain constants, host variables, special registers, parameter markers or columns. The expression can be a simple column. For example, T1.col = T2.col, or T1.col > T2.col. If it contains a column, the column must not be in the same table as the other predicate operand.
- 'op' is either =, <, <=, >, >= or <> (note that '<>' is not indexable, but can be processed at stage 1).
- '*list*' items have to meet all of the following criteria:
  - list items are only elements from the following list:
    - Constants
    - Host variables
    - Special registers
    - Session variables
    - Parameter markers
  - The predicate that contains the list is not in the WHEN clause of a trigger
  - For every element in list, column = list-element must be stage 1 and indexable

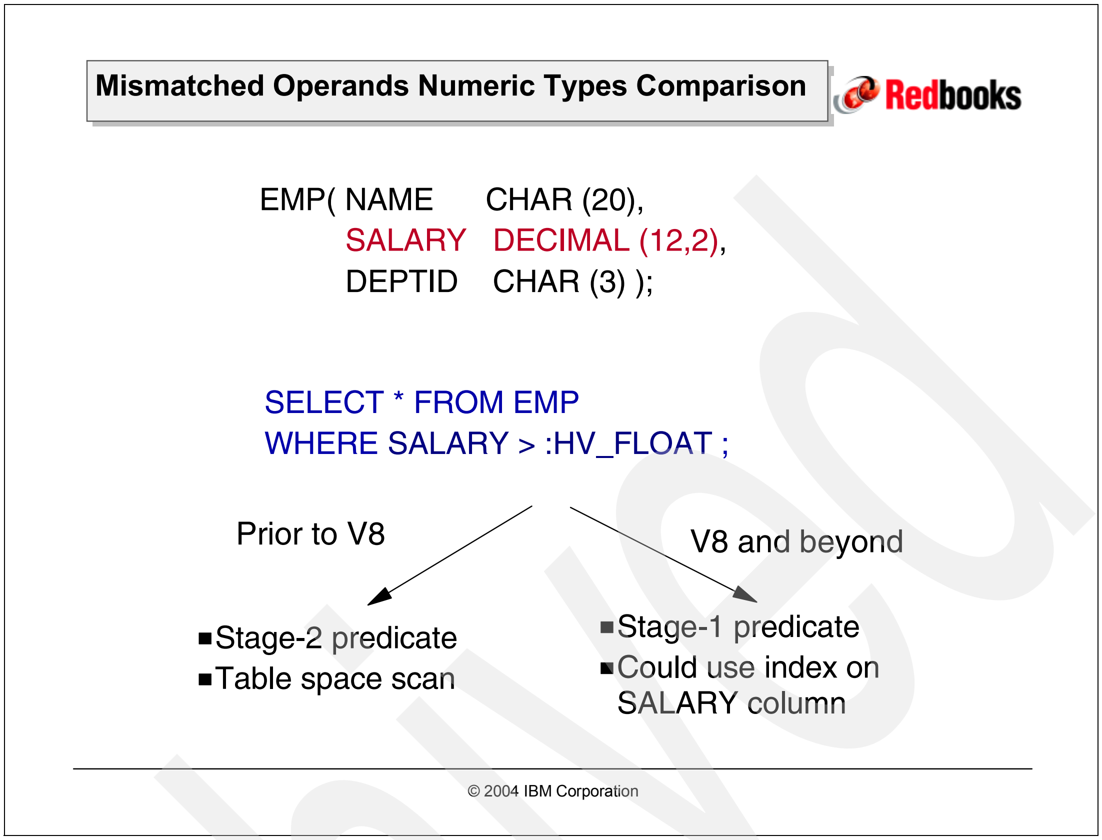
When each predicate operand is a simple column from different tables (for example T1.col = T2.col), then the join sequence determines which predicate operand is considered the 'column' and which is considered the 'expression'. The inner table is considered to be the 'column' and the outer table in the join the 'expression'.

For example, consider the following predicate:

- T1.col > T2.col

If T1 is the inner table of the join, then T1.col is considered the 'column' and T2.col is considered the 'expression'. Likewise, if T2 is the inner table of the join, then T2.col is considered the 'column' and T1.col is considered the 'expression'.

All predicates of the form listed above are now indexable and processed during stage 1, subject to certain conditions. Let us now look at a few examples to illustrate these enhancements.



10.4.2 Mismatched operands numeric types comparison

Assume that we have a table EMP defined as shown in the visual.

This example shows how the SALARY column (decimal data type) is compared with a float host variable. In this case the predicate can be processed during stage 1 and, assuming an index exists on SALARY, is also indexable. Note that salary has a precision less than 16.

This would not be the case if SALARY is defined as DECIMAL(16,2).

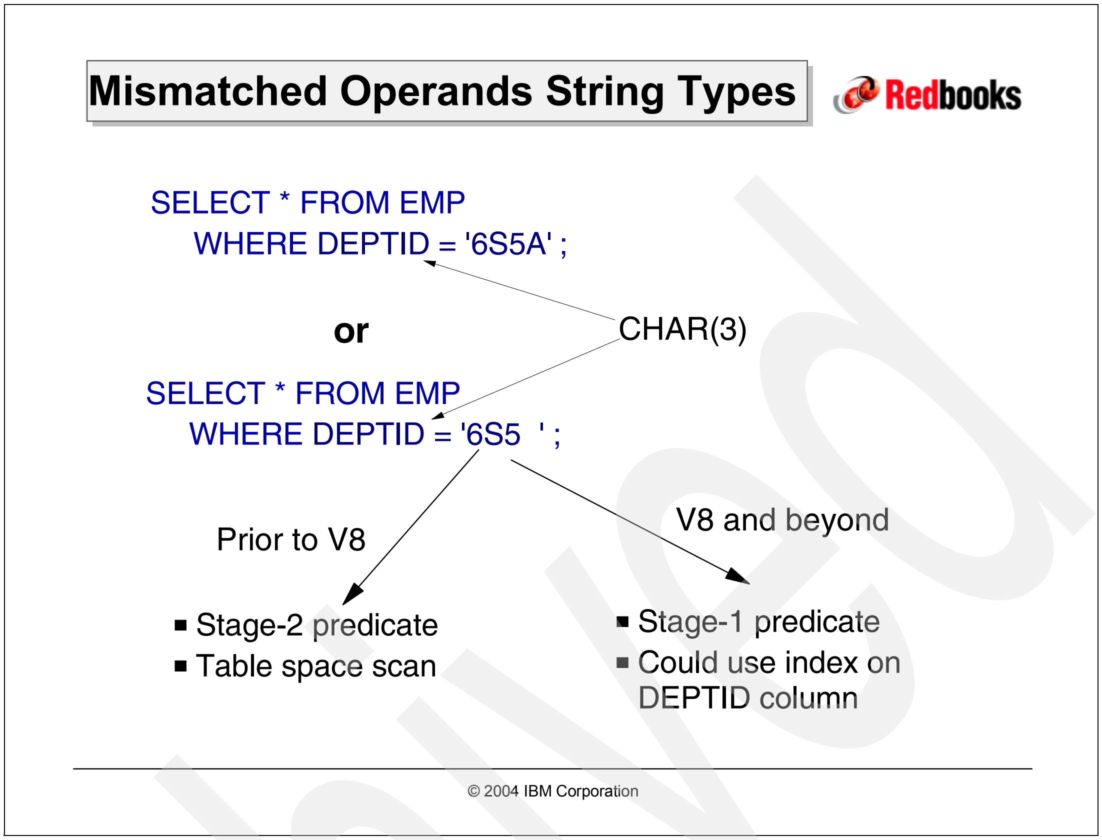
Numeric types comparison

All numeric type comparisons are stage 1 and indexable except the following ones:

- REAL -> DEC(p,s) where p > 15
- FLOAT -> DEC(p,s) where p > 15

Note that in the comparison notation above, the REAL or FLOAT “value” refers to the “right-hand side” of the predicate, or the outer table in a join. For example, the restriction applies to the following predicate: DEC_column > REAL_hostvar (if the precision of the DEC_column is greater than 15).

In the case above, the decimal value is the indexed value, so the comparison must be done on the decimal value. However, REAL and FLOAT values cannot be converted to decimal with precision > 15 without possibly changing the collating sequence. Consequently, these are stage 2 (residual) predicates.



10.4.3 Mismatched operands string types

This example shows how the DEPTID column (character data type) is compared with a character host variable of longer length. In this case the predicate can be processed during stage 1 and, assuming an index exists on DEPTID, is also indexable.

String types comparison

We now consider several types of string comparisons.

Same CCSID string comparisons

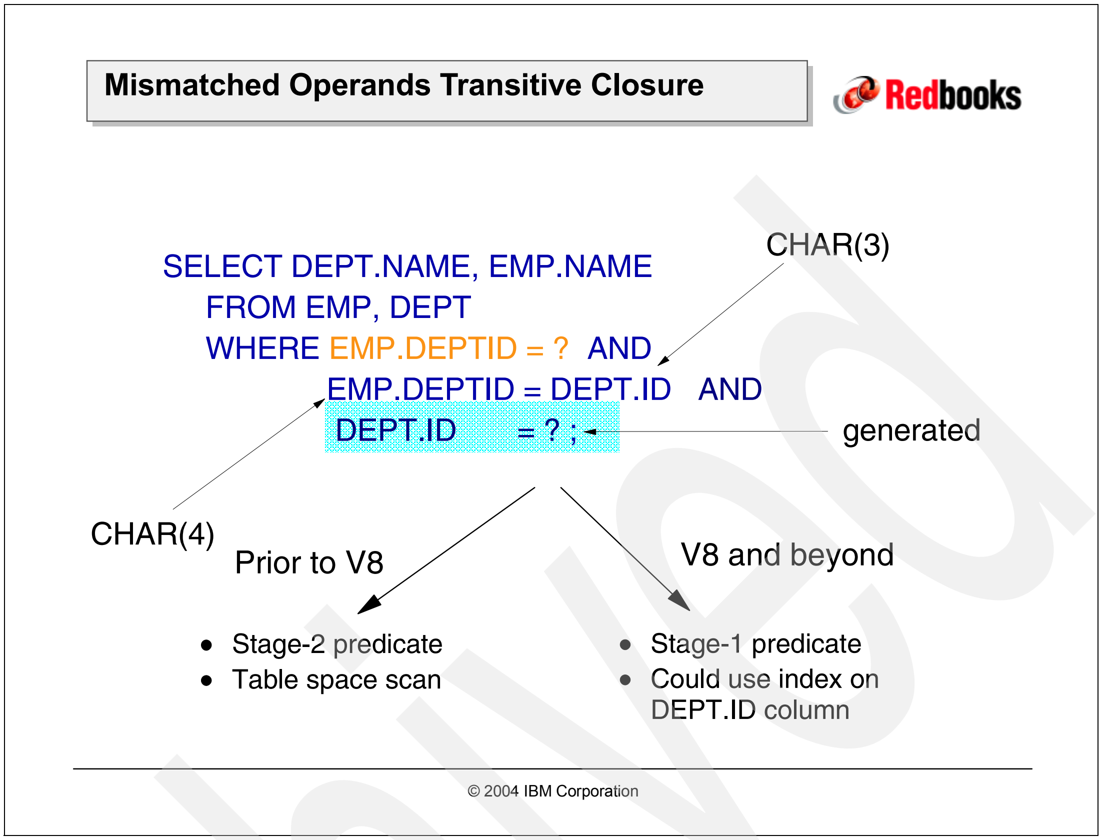
All predicates comparing string types with the same CCSID are stage 1 and indexable except the following ones:

- graphic/vargraphic -> char/varchar

  In general, predicates comparing graphic/vargraphic to char/varchar are not indexable. However, if the char/varchar is Unicode mixed and the predicate is an '=' predicate, then the predicate is indexable.

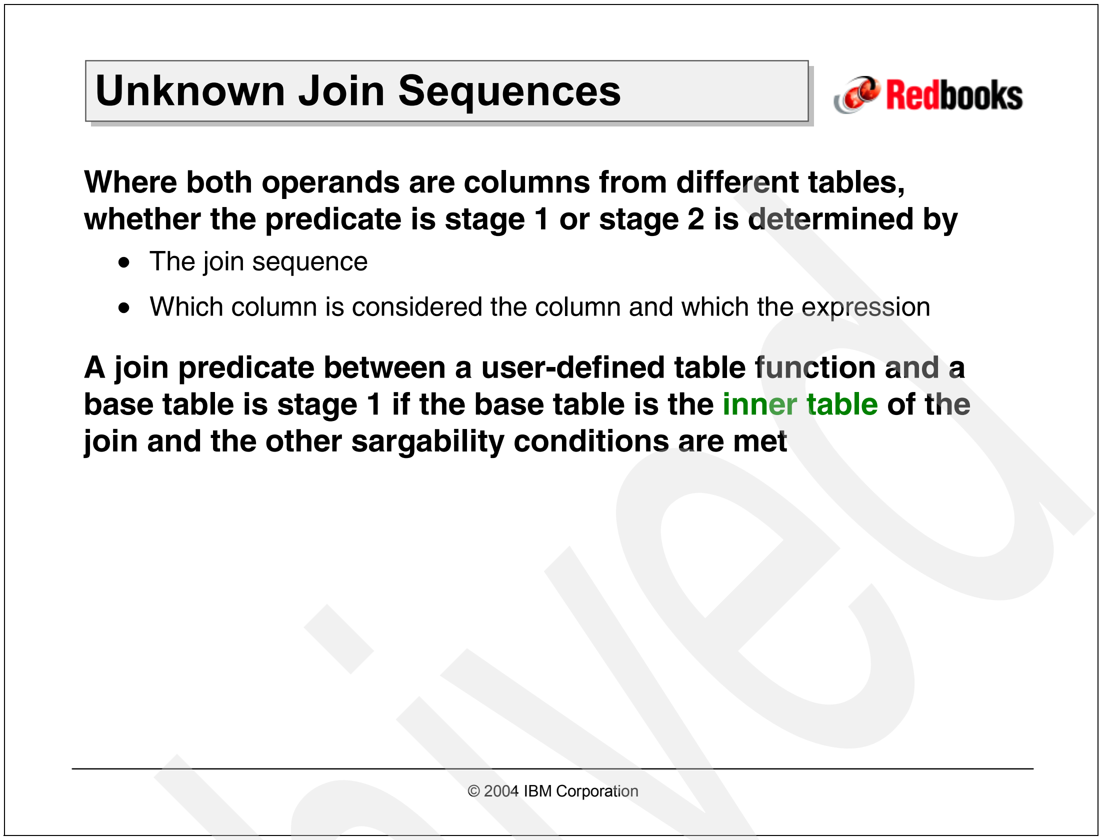
- char/varchar(n1) -> char/varchar(n2) n1 > n2 and not '=' pred
- graphic/vargraphic(n1) -> graphic/vargraphic(n2) n1 > n2 and not '=' pred
- char/varchar(n1) -> graphic/vargraphic(n2) n1 > n2 and not '=' pred

Here the indexed value is the right hand side of “->”, and so the comparison must be done with that data type and length. However, when the left hand side value in these cases is cast to the right hand side data type and length, truncation may occur. Consequently, these cases are stage 1 but not indexable.



10.4.4 Mismatched operands transitive closure

Assume that table DEPT exists with column ID defined as CHAR(3). Remember EMP.DEPTID is defined as CHAR(4). In DB2 V8, predicate transitive closure is done even though EMP.DEPTID and DEPT.ID are of different lengths. The new predicate is stage 1 and is indexable. The generated parameter marker has the same size as the parameter marker in the EMP.DEPTID predicate.



10.4.5 Unknown join sequence

In the case of a join where the operands are columns from different tables, whether the predicate is stage 1 or not is determined by the following:

- Whether DB2 evaluates the predicate before or after a join operation:

  A predicate that is evaluated after a join operation is always a stage 2 predicate.

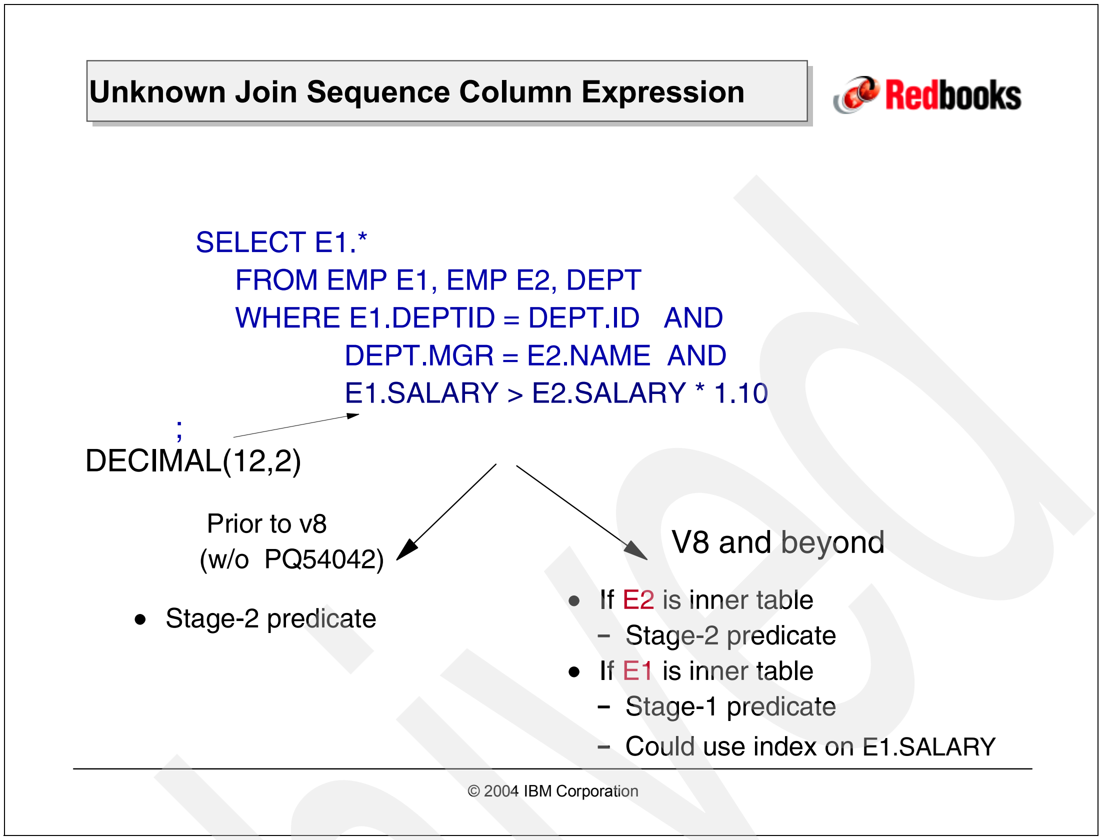
- Join sequence:

  The same predicate might be stage 1 or stage 2, depending on the join sequence. Join sequence is the order in which DB2 joins tables when it evaluates a query. This is not necessarily the same as the order in which the tables appear in the predicate.

  For example, the following predicate might be stage 1 or stage 2:

  T1.C1=T2.C1+1

  If T2 is the first table in the join sequence, the predicate is stage 1, but if T1 is the first table in the join sequence, the predicate is stage 2.

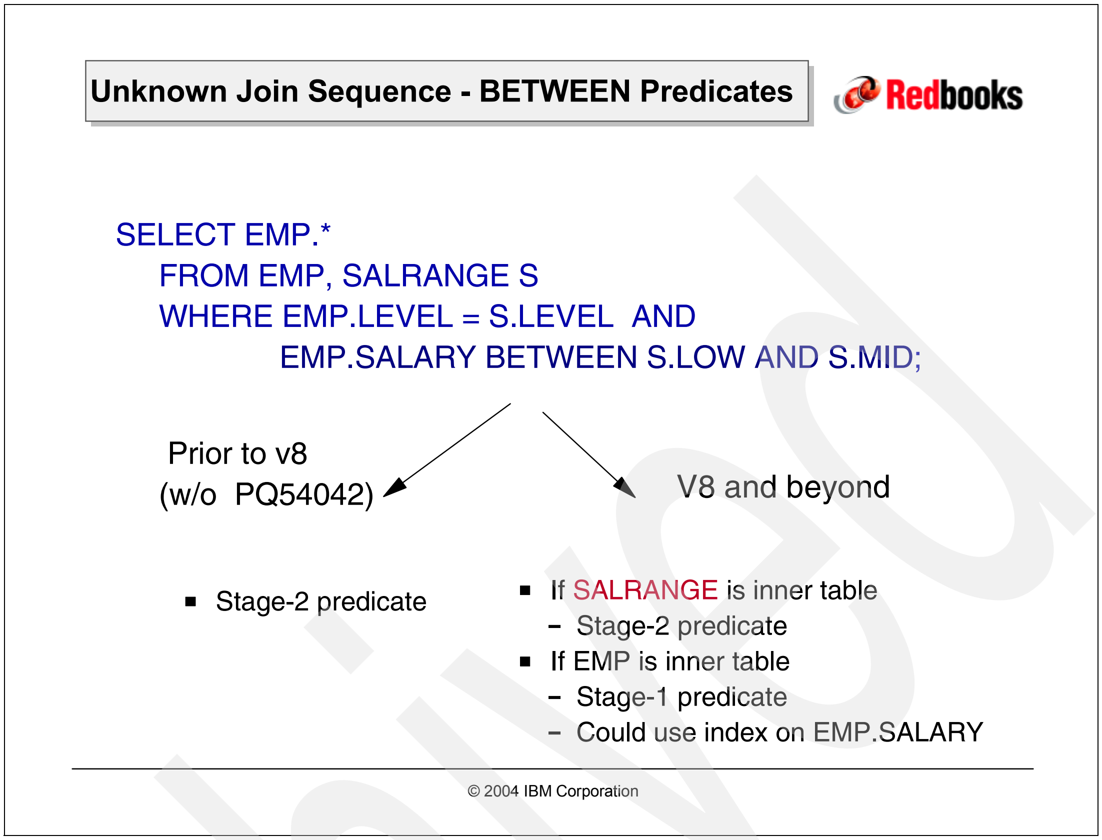


10.4.6 Unknown join sequence - column expression

A performance improvement that impacts joins when column expressions are used in predicates was introduced in DB2 V7 with the APAR PQ54042 and extended to V8.

Assume that table DEPT with column ID exists.

Whether the predicate E1.SALARY > E2.SALARY * 1.10 is considered stage 1 or not is determined by the sequence in which the tables are joined.

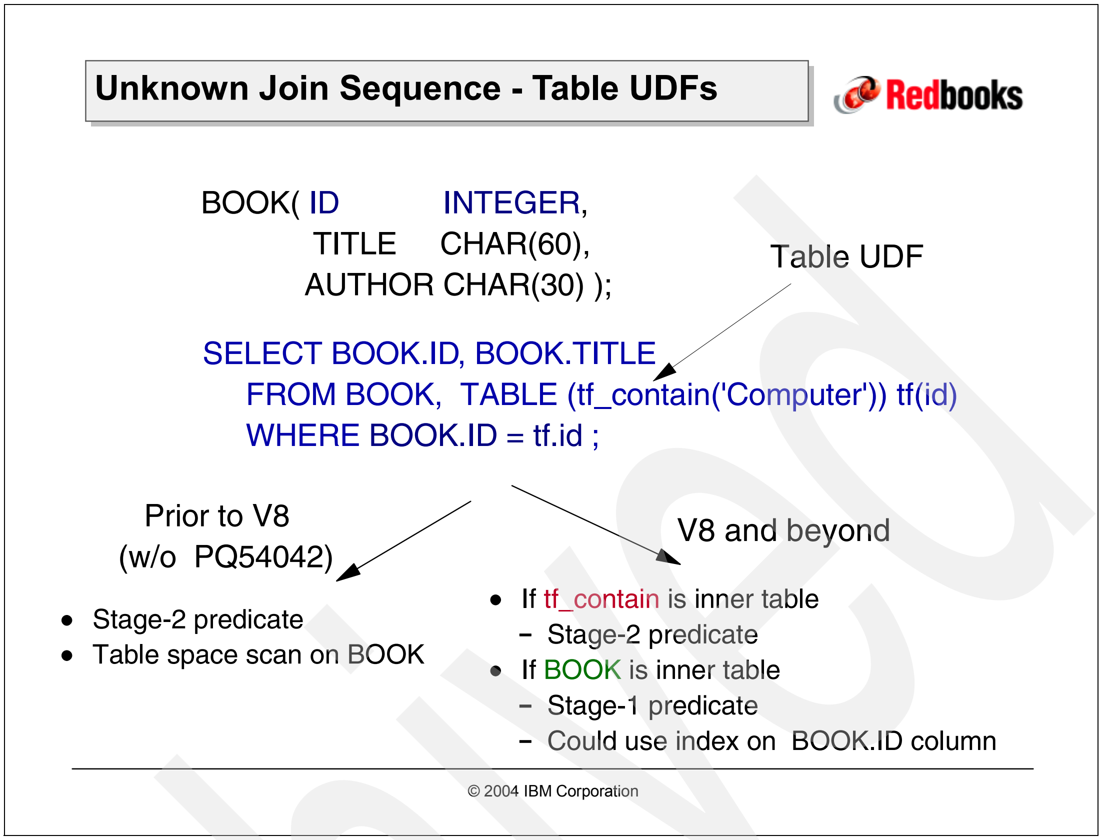


10.4.7 Unknown join sequence - BETWEEN predicates

A performance improvement that impacts joins when BETWEEN is used with column names in predicates was introduced in DB2 V7 with the APAR PQ54042 and extended to V8.

Assume that table EMP has been altered to include column LEVEL and table SALRANGE containing columns LOW and MID exists.

Whether the predicate EMP.SALARY BETWEEN S.LOW AND S.MID is considered stage 1 or not is determined by the sequence in which the tables are joined.

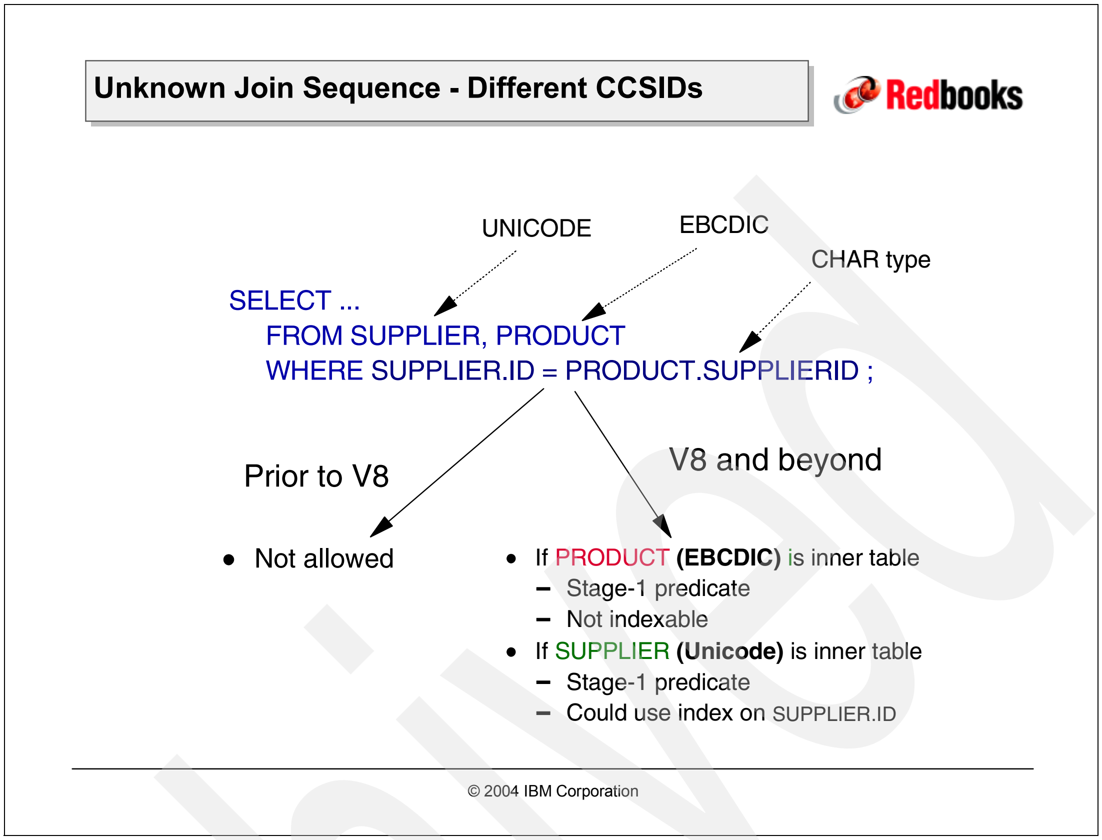


10.4.8 Unknown join sequence - table UDFs

A performance improvement that impacts table UDFs was introduced in DB2 V7 with the APAR PQ54042 and extended to V8 to unlike data types.

With this enhancement, the book.ID=tf.ID predicate is stage 1 and indexable provided the table UDF is accessed first (outer table).

If you join a base table with a user-defined table function, the sequence in which the tables are joined determines whether the predicate is stage 1 or not.



10.4.9 Unknown join sequence - different CCSIDs

DB2 UDB for z/OS and OS/390 is increasingly being used as a part of large client server systems, for example, in data centers of multinational companies and e-commerce. DB2 V7 introduced Unicode support to store data in Unicode. However, being able to store data in Unicode is not enough to solve all code page related problems. For example, DB2 V7 does not allow joining tables with different encoding schemes. Thus, an EBCDIC table cannot be joined with a Unicode table.

DB2 V8 enhances support for Unicode and allows joining of tables with different encoding schemes. Thus, an EBCDIC table can be joined with a Unicode table.

The join predicate in this situation is considered stage 1. However, if the join predicate is indexable or not depends on the sequence in which the tables are joined.

The same restrictions as for string comparisons between the same CCSID also apply here. Besides that, in order to be stage 1 and indexable, the inner table column, or the “col” side of the predicate, has to be Unicode. Otherwise, the predicates are stage 1 but not indexable. The reason is that all predicates comparing unlike CCSID are evaluated in the Unicode encoding scheme.

In all cases involving “column = column” comparisons (same or unlike CCSID) with columns from different tables, the optimizer considers a merge scan join as a potential access path.

Table UDF Cardinality and Materialized FETCH

What is it? .....

- User-defined table function cardinality option indicates the total number of rows returned by a user-defined table function reference
  - Used by DB2 at bind time to evaluate table function access cost
  - A host variable or parameter marker cannot be used
- Using materialized FETCH, rows returned from a user-defined function are pre-fetched into a work file in its first invocation
  - Feature is enabled by DB2 based on the access cost estimation

Benefits .....

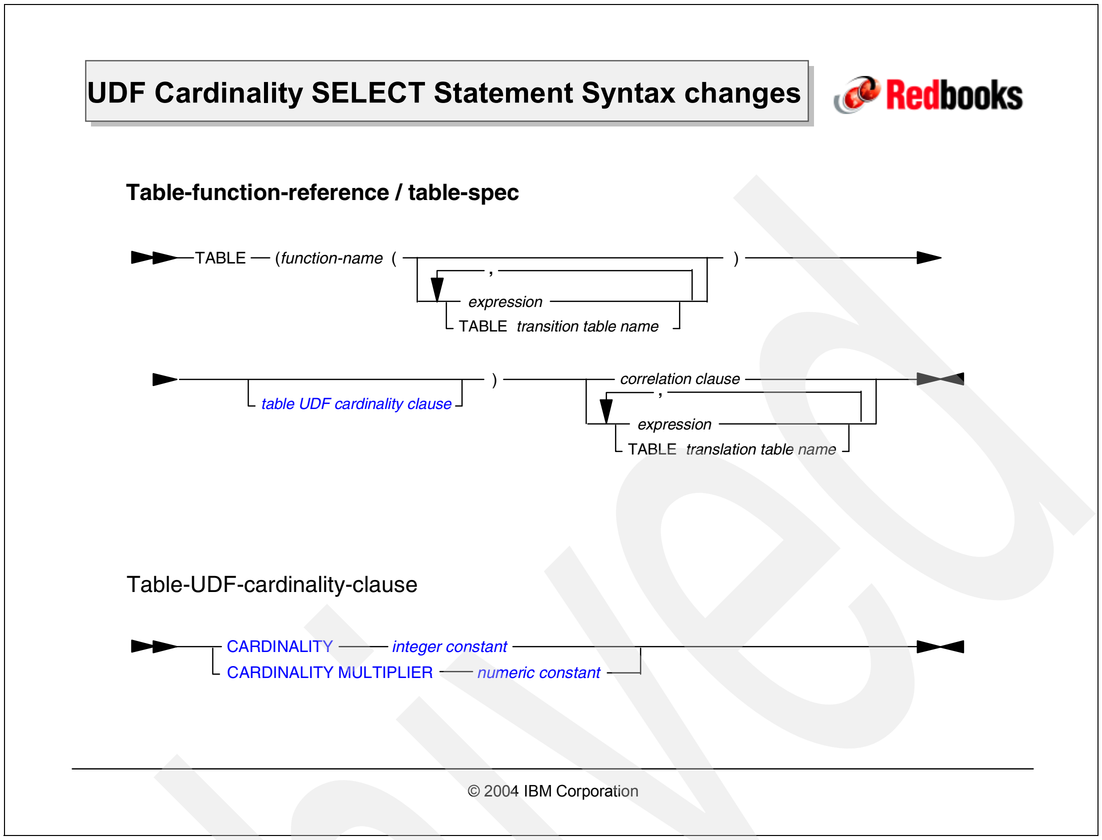
- Better performance for text search with Text Extender using UDF
- Enhances ability to tune performance of queries containing user-defined table function
- Performance improvement to move data between table functions and DB2 using block data movement
- Best performance when combined with V8 capability to make predicates with unlike data types stage1

10.5 Table UDF cardinality and materialized FETCH

Today, for a user-defined table function, you can specify the CARDINALITY option to specify an estimate of the expected number of rows that the function returns. The number is used for optimization purposes. This is fine as long as each invocation of the UDF returns more or less the same number of rows. However, that is not always the case. Subsequent invocations of the table UDF, depending on the input parameters, can return a totally different answer set size.

In Version 8, DB2 allows you to specify the cardinality option when you reference a user-defined table function in an SQL statement, for example, in a SELECT. With this option, users have the capability to better tune the performance of queries that contain user-defined table functions.

The user-defined table function cardinality option indicates the total number of rows returned by a user-defined table function reference. The option is used by DB2 at bind time to evaluate the table function access cost.



10.5.1 UDF cardinality SELECT statement syntax changes

A cardinality clause can be specified to each user-defined table function reference within the table specification of the FROM clause in a subselect. This option indicates the expected number of rows to be returned by referencing the function in a particular query. The cardinality clause comes in two flavors, as shown in the visual.

- The CARDINALITY keyword, followed by an integer that represents the expected number of rows returned by the user-defined table function.

  This keyword specifies an estimate of the expected number of rows returned by user-defined table function reference.

  Example: DB2 expects number of rows to be returned by a user-defined table function is 30 regardless of value in CARDINALITY column in SYSIBM.SYSROUTINES for this function.

  ```
  SELECT * FROM TABLE (TUDF(1) CARDINALITY 30) AS X;
  ```

- The CARDINALITY MULTIPLIER keyword, followed by a numeric constant.

  The expected number of rows returned by the table function is computed by multiplying the given number with the reference cardinality value that is retrieved from the CARDINALITY column of SYSIBM.SYSROUTINES for the corresponding table function name, that was specified when the user-defined table function was created.

  Example 1: If SYSIBM.SYSROUTINES.CARDINALITY = 1 for the user-defined table function, DB2 assumes the expected number of rows to be returned is 30 (30 * 1) for **this** invocation of the function.

  ```
  SELECT * FROM TABLE(TUDF(2) CARDINALITY MULTIPLIER 30) AS X;
  ```

Cardinality Option and Cardinality in Catalog

Cardinality-clause is not SQL standard syntax

Cardinality option

- Cardinality and cardinality multiplier cannot co-exist
- Cardinality is integer only
  - Value of integer_constant must range from 0 to 2147483647
- Cardinality multiplier can be integer, decimal, floating point

Cardinality in catalog

- Table UDF cardinality in table SYSIBM.SYSROUTINES
- Column CARDINALITY populated by CREATE TABLE UDF statement

Cardinality option and cardinality in catalog comparison

- Cardinality multiplier is multiplied with cardinality in catalog
- Cardinality option applies to current query only, it does not overwrite cardinality in catalog

10.5.2 Cardinality option and cardinality in catalog

The cardinality clause is a non-standard SQL feature, specific to DB2 for z/OS implementation.

The cardinality clause allows you to specify either the CARDINALITY option or the CARDINALITY MULTIPLIER option. These keywords are mutually exclusive.

Specifying the CARDINALITY option when referencing a user-defined table function in a SELECT statement does not change the corresponding CARDINALITY column value in SYSIBM.SYSROUTINES. When you specify the CARDINALITY option when referencing a user-defined table function, the value only applies for that particular query, and the value in the CARDINALITY column in SYSIBM.SYSROUTINES is ignored for that particular query.

Specifying the CARDINALITY MULTIPLIER option when referencing a user-defined table function in a SELECT statement does not change the CARDINALITY column value in SYSIBM.SYSROUTINES. When you specify the CARDINALITY MULTIPLIER option when referencing a user-defined table function, the value only applies for that particular query. However, the value in the CARDINALITY column in SYSIBM.SYSROUTINES is not ignored.

The CARDINALITY column value in SYSIBM.SYSROUTINES can only be initialized by the CARDINALITY option in the CREATE FUNCTION statement when the user-defined table function is created. It can be changed by the CARDINALITY option in the ALTER FUNCTION statement.

The following example illustrates a case where the CARDINALITY MULTIPLIER option for a user-defined table function can influence the query optimization process of DB2.

```
SELECT *
FROM BOOKS B,
TABLE(CONTAINS(1,'cs') CARDINALITY MULTIPLIER 15.0) AS X1(ID),
TABLE(CONTAINS(2,'database') CARDINALITY MULTIPLIER 2.0) AS X2(ID),
TABLE(CONTAINS(3,'Clark') CARDINALITY MULTIPLIER 0.03) AS X3(ID)
WHERE B.ID = X1.ID AND B.ID = X2.ID AND B.ID = X3.ID;
```

In this example, we assume that, for a user-defined table function CONTAINS, the CARDINALITY column in SYSIBM.SYSROUTINES is 1000. The table function CONTAINS searches a string in a column of the BOOKS table and returns ID numbers of the matching BOOKS rows. The first argument of CONTAINS indicates the column number of BOOKS and the second argument is the search string.

The first reference to CONTAINS searches a string 'cs' in the category of books (which is column 1 of the BOOKS table). We expect that 15000 books will meet the condition.

The second reference indicates that we expect to find 2000 books that contain a string 'database' in their abstracts (column 2).

The third reference indicates that there are probably around 30 books written by authors called 'Clark' (the authors column is column 3 in the BOOKS table).

The following example shows that, instead of using the CARDINALITY MULTIPLER option, the same query can be written using the CARDINALITY option.

```
SELECT *
FROM BOOKS B,
TABLE(CONTAINS(1,'cs') CARDINALITY 15000) AS X1(ID),
TABLE(CONTAINS(2,'database') CARDINALITY 2000) AS X2(ID),
TABLE(CONTAINS(3,'Clark') CARDINALITY 30) AS X3(ID)
WHERE B.ID = X1.ID AND B.ID = X2.ID AND B.ID = X3.ID;
```

When you estimate the number of rows returned by each reference of the CONTAINS function, DB2 can evaluate the access cost more accurately based on the specified cardinality option, and a more appropriate join sequence and join type can be chosen by the query optimization process. The effectiveness of the option depends on the access cost of the user defined table function computed by DB2, relative to the access costs of the other tables in the query.

Sequential Fetch vs. Materialized Fetch

Sequential fetch

- Each returned row needs a UDF fetch call
- Each call needs a context switch, aka. suspend/resume

Materialized fetch

- All rows are returned in one UDF (open) call and stored in work file
- Saves (#rows -1) context switch
- Adds one-time create work file, insert rows into work file and deallocate work file cost
- Plan_table: Materialized fetch, as a new table access type for table UDF is displayed as 'RW'.

Sequential fetch compared to materialized fetch

- When the number of rows reaches a threshold, materialized fetch performs better than sequential fetch
- Depends on other UDF parameters, such as CPU time per row, I/O cost per row,
- Depends also on system parameters, such as MIPS
- DB2 decides which is better, based on access cost estimation
- The feature of materialized fetch is independent of the cardinality option. If no cardinality option is specified, the cost estimation is done using the cardinality data in the catalog (or using the default value)

10.5.3 Sequential fetch and materialized fetch comparison

The performance improvement of queries using table UDFs is achieved in conjunction with another new feature introduced in DB2 V8, called “Materialized fetch” or “Table UDF block fetch”.

Before this enhancement, each returned row from a user-defined table function needs a UDF fetch call, and each call needs a context switch, since a UDF runs in a WLM managed address space, like a stored procedure. Context switches can become expensive, especially when the table UDF returns many rows.

To reduce the amount of context switches required, DB2 V8 uses a technique called materialized fetch or block fetch.

All rows are returned during the first invocation of the UDF, prefetched and stored in DB2 workfile. By using this technique, the savings are #rows_in_table_UDF_result - 1 context switches, at the cost of a one-time workfile creation, the cost to insert the rows into the workfile and the cost of deallocating the workfile.

Note that this type of block fetch is not to be confused with block fetching in a distributed environment.

During access path selection, the optimizer decides whether or not to use this new block fetching technique, depending on the following criteria:

- The estimated #rows returned by the user-defined table function is considered.
- SYSIBM.SYSROUTINES information: When available, the values in the IOS_PER_INVOC, INSTS_PER_INVOC, INITIAL_IOS, INITIAL_INSTS columns are taken into consideration. Note that you have to supply this information by manually updating the catalog. RUNSTATS has no way of collecting this information. When the information is not available, default values are used.
- The processor speed of the machine you are running on is also taken into consideration, since the INST_PER_INVOC and INITIAL_INSTS columns are expressed in number of instructions, and not in service units or CPU seconds.

You can determine if DB2 has used the table UDF block fetch feature by using EXPLAIN. When used, the ACCESSTYPE field in the PLAN_TABLE contains “RW.” Equivalent information is available in the mini-plan trace record IFCID 22.

Trigger Enhancements

TRIGGER work file creation avoided

- When the WHEN clause (conditional trigger) evaluates false (and the trigger is not invoked)
- When old/new transition variables/tables can fit in a memory buffer

Very significant performance enhancement

- When few or no triggers fired
- When the transition variable information fits into the memory buffer

Applies to both BEFORE and AFTER triggers

10.6 Trigger enhancements

Prior to DB2 V8, each time an AFTER trigger with a WHEN clause (also known as a conditional trigger) is invoked, a work file is created for the old and new transition variables. The work file is always created, even when the trigger is not activated because the WHEN condition evaluates false.

For example, let us say that you insert 1000 rows into a table that has a trigger. Assume that only three of these rows actually invoke the trigger (satisfying the WHEN clause) that is defined on the table. Since a transition table (work file) is created for each change/insert, the transition table is created 1000 times and only used by the trigger manager three times. So for 997 times, the workfile is created and deleted needlessly.

In the following example, the insert of the first row causes the row to be inserted into a transition table, but the trigger is not invoked, because the values NAME = 'TASHA' and POUNDS = 10 do not match the WHEN clause for trigger NEWCAT. The transition table is deleted after the statement is completed.

The trigger is not invoked for the second row also, because the values NAME = ‘BLACKIE’ and POUNDS = 9 do not match the WHEN clause for trigger NEWCAT. The transition table is deleted after the statement is completed.

However, the trigger is invoked for the third row because the values NAME = ‘SUNSHINE’ and POUNDS = 12 match the WHEN clause for trigger NEWCAT. So the row that is inserted into the transition table is actually used to process the AFTER trigger.

Here is the coding for our example:

```
CREATE TRIGGER NEWCAT
       AFTER INSERT ON CATS
       REFERENCING NEW AS NROW
   FOR EACH ROW   MODE DB2SQL
   WHEN (NROW.NAME = 'SUNSHINE' AND NROW.POUNDS = 12)
INSERT INTO PETS(COL1,COL2,COL3,COL4)
                  VALUES (0, 1, NROW.NAME, 'INSERTED SUNSHINE');

INSERT INTO CATS(ID,NAME,POUNDS,C4,C5,C6,C7,C8,C9,C10)
       VALUES (1, 'TASHA', 10, '001',4, 2, 2, 4342, 'PURINA CAT CHOW', 'ANN')
INSERT INTO CATS(ID,NAME,POUNDS,C4,C5,C6,C7,C8,C9,C10)
       VALUES (2, 'BLACKIE', 9, '001', 4, 2, 2, 3023, 'KAL KAN', 'BETH')
INSERT INTO CATS(ID,NAME,POUNDS,C4,C5,C6,C7,C8,C9,C10)
       VALUES (3, 'SUNSHINE', 12, '001', 4, 2, 2, 1000, 'FRISKIES BUFFET', 'BETH')
```

In addition, V8 uses a memory buffer to store a small numbers of rows with the transition variables to avoid creating and deleting the work files. If more information needs to be stored than fits into the buffer, a workfile is created.

Note that this enhancement applies to ALL triggers, not just AFTER triggers.

This enhancement can represent a very significant performance enhancement, especially when the trigger is only fired a few times compared to the number of time is evaluated, or when only a small amount of transition variable information needs to be kept, and can fit in a memory buffer, so no workfile needs to be created.

The Need for Extra Statistics

Distribution statistics are currently collected for indexed columns only

Non-uniform distribution statistics for non-leading indexed columns are not collected by RUNSTATS, which can result in non-optimal performance

- Less efficient join sequences
- Inappropriate table join method
- Increase in the number of rows that need to be processed

10.7 The need for extra statistics

With DB2 V7, RUNSTATS may only collect frequencies on the leading column and leading concatenated column groups of indexed columns. Data correlation and skew may occur on any column or column group. When a query contains predicates on columns which are correlated and/or skewed, and the optimizer does not have correlation and/or skew statistics, the optimizer may incorrectly estimate the filtering for those predicate(s).

Inaccurate filter factors can manifest themselves in many ways — inefficient join sequence, join method, single table access. Also, access paths for queries where insufficient statistics information is available tend to be more unstable. Inaccurate filter factor estimation can result in inaccurate query cost, so efficient and inefficient access paths may end up having similar cost estimate. This increases the probability that insignificant costing change results in access path change and possibly severe regression. When the optimizer is accurately estimating filter factors, the estimated costs more accurately reflect actual costs. So inefficient and efficient access paths are unlikely to be close in cost estimate — which should result in more efficient and stable access paths.

With DB2 V8, RUNSTATS can collect the most and least frequently occurring frequencies on almost any column or column group. RUNSTATS can also collect multi-column cardinality on almost any column group. In many cases, customers have predicates on some matching and some screening columns. Now customers can collect statistics to better reflect the total index filtering. When customers have correlation and skew on some indexed and some non-indexed columns, customers can collect statistics so the optimizer more accurately estimates the number of rows which qualify from the table. The number of rows qualified from a table can be critical in accurately costing joins — such as how many probes will be done to the inner table for nested loop join, and in deciding which table should be the outer table.

10.7.1 DSTATS (Distribution statistics for DB2 for OS/390) tool

Before DB2 V8, users could download a program called DSTATS (Distribution Statistics for DB2 for OS/390). DSTATS is a standalone DB2 application program containing embedded dynamic and static SQL statements. This tool is aimed to address the issue by collecting additional statistics on column distributions that were not being collected by RUNSTATS before DB2 V8. The DSTATS tool can be downloaded using the following link:

`http://www-1.ibm.com/support/docview.wss?uid=swg24001598`

The RUNSTATS utility is enhanced to collect additional distribution statistics on virtually any column (or group of columns) that would likely be used in a predicate. The collection of this information is especially important when the data distribution in those columns is skewed.

> **Note:** This enhancement to the RUNSTATS utility eliminates the need to use the DSTATS tool for DB2 V8. In addition, the DSTATS program has not been enhanced to support DB2 V8.

RUNSTATS Enhancement

Can collect distribution statistics on any column, or group(s) of columns, indexed and non-indexed, specified at the table level

Frequency distributions for (non-indexed) columns or groups of columns

Cardinality values for groups of (non-indexed) columns

LEAST frequently occurring values, along with MOST for both indexed and non-indexed column distributions

New keywords : COLGROUP MOST LEAST BOTH

SORTNUM SORTDEVT

10.7.2 RUNSTATS enhancement

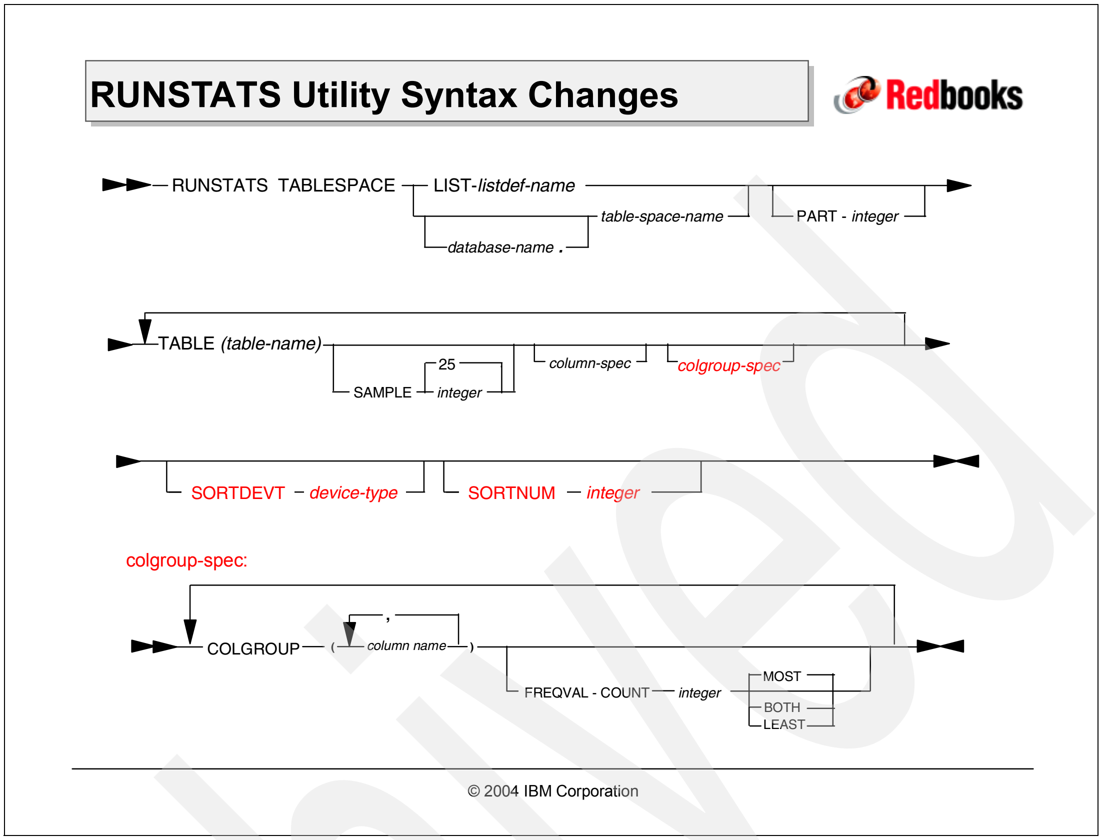
In DB2 Version 8, the RUNSTATS utility has incorporated the functionality of the DSTATS program. You can now use RUNSTATS to collect distribution statistics on any column in your tables, whether or not they are part of an index, and whether or not they are the leading columns of an index (when collecting statistics for column groups).

When customers have correlation and skew on certain columns, collecting these additional statistics can help the optimizer to more accurately estimate the number of rows which qualify from the table (filter factor). More accurate filter factor computations should lead to better optimization choices. Thus the query performance improves with better filter factor information in the DB2 catalog.

These additional statistics can only be gathered by the “stand-alone” RUNSTATS utility. They cannot be gathered when collecting statistics as part of another utility’s execution, so-called inline statistics.

To summarize, RUNSTATS enhancements provide the following functionality:

- Frequency value distributions for non-indexed columns or groups of columns.
- Cardinality values for groups of non-indexed columns.
- LEAST frequently occurring values, along with MOST frequently occurring values, for both index and non-indexed column distributions. (DB2 V7 only gathers the most frequently occurring values, and therefore does not require you to specify a keyword to indicate which statistics you want.)



10.7.3 RUNSTATS utility syntax changes

The visual shows the changes to the RUNSTATS utility syntax to collect cardinality and distribution statistics on any column or group of columns in a table.

The *colgroup-spec* allows you to specify the COLGROUP and associated keywords to collect cardinality and distribution statistics on any column or group of columns in a table.

The *sort-spec* specifies the device type that allows DFSORT to dynamically allocate the sort work data sets that are required.

The *correlation-stats-spec* block (Figure 10-1) in V7 allows you to specify what distribution statistics to gather at the index level. This block is enhanced in DB2 V8 to include the keywords MOST, BOTH, and LEAST. Their meaning in this block is the same as in the colgroup-spec block. The same block can also be used in the specification of a RUNSTATS INDEX statement.

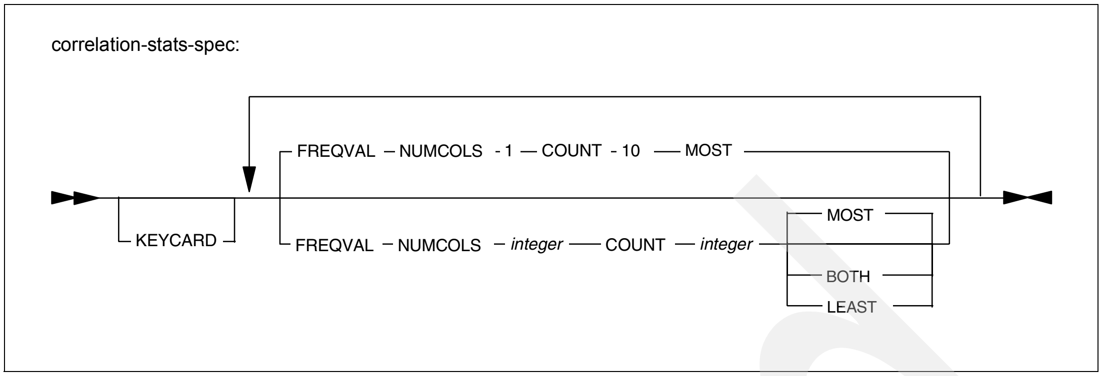


Figure 10-1 Correlation-stats-spec syntax block

Collecting cardinality and distribution statistics

To enable the collection of cardinality and distribution statistics on any table column, a new colgroup-spec block is introduced. New keywords COLGROUP, LEAST, MOST and BOTH are introduced in this block. In addition, the existing keywords FREQVAL, and COUNT can also be used. Cardinality and distribution statistics are collected only on the columns explicitly specified. Cardinality and distribution statistics are not collected if you specify COLUMN ALL.

COLGROUP

When the keyword COLGROUP is specified, the set of columns specified within the COLGROUP keyword is treated as a group. The cardinality values are collected on the column group. You can specify the COLGROUP keyword multiple times (with different sets of columns) for the same table in your RUNSTATS statement.

The cardinality statistics for the specified group(s) of columns are collected in SYSCOLDIST, and if the table space is partitioned also in SYSCOLDISTSTATS, catalog tables.

The COLUMN keyword works the same as in previous releases. The cardinality statistics of individual columns are collected in the SYSCOLUMNS catalog table, as in versions prior to DB2 V8.

FREQVAL

This keyword controls the collection of frequent value statistics. These are collected either on the column group or on individual columns, depending on what you specified on the COLGROUP keyword. If FREQVAL is specified, then it must be followed by the keyword COUNT. When specified for table-level statistics, FREQVAL can only be specified together with the COLGROUP keyword. (No frequent value statistics are collected for columns specified together with the COLUMN keyword.)

If the FREQVAL keyword is specified together with the COLGROUP keyword, the distribution statistics are collected in SYSCOLDIST and if the table space is partitioned also in SYSCOLDISTSTATS catalog tables.

COUNT integer

COUNT indicates the number of frequent values to be collected. Specifying an integer value of 20 means to collect 20 frequent values for the specified columns. No default value is assumed for COUNT. The keyword COUNT *integer* is followed by the keyword MOST, LEAST, or BOTH.

MOST
The most frequent values are collected when the keyword MOST is specified.

LEAST
The least frequent values are collected when the keyword LEAST is specified.

BOTH
The most frequent values and the least frequent values are collected when the keyword BOTH is specified.

Collecting column correlation statistics on indexed columns

The keywords FREQVAL, NUMCOLS, and COUNT in the correlation-stats-spec block can be used to collect, by default, the most frequent value statistics for the specified index as in prior versions. In V8, this block is enhanced to include specification of the keyword MOST, LEAST, or BOTH, as shown before, in Figure 10-1 on page 766.

Using work data sets when using the COLGROUP keyword

When collecting statistics on non-indexed columns, DB2 cannot rely on the index to provide an easy way to calculate the requested statistics. Therefore DB2 will have to invoke DFSORT to get the data in the correct sequence for determining the requested statistics. You need to specify temporary work data sets and a sort message data set. You can use the SORTDEVT option to specify the device type for temporary data sets that DFSORT can use for sorting.

You can also use the SORTNUM option to specify the number of temporary data sets to use. The DD name STATWKnn defines the sort work data sets that are used during utility processing. The value of nn identifies one or more data sets that are to be used by the subtask invocation of DFSORT. You can dynamically allocate the work data sets by using the TEMPLATE utility, or you can define the data sets through JCL statements.

Note that RUNSTATS may also require sorting when gathering statistics on data-partitioned secondary indexes, but in this section we focus on the gathering non-indexed column statistics.

Estimating the size of the STATWKnn data sets

If you define the data sets through JCL, you need to determine the size and number of records that RUNSTATS is to create and process. You can use the following formula to calculate the size of the data sets:

```
(longest_record_length + prefix) × sum from 1 to N (#colgroups-tabn × #rows-tabn)
```

The variables in the preceding formula have the following values:

N	Number of tables for which distribution statistics are collected
#colgroups-tabn	Number of column groups that are specified for the nth table
#rows-tabn	Number of rows for the nth table

Example 1

- RUNSTATS TABLESPACE DSN8D81A.DSN8S81E TABLE(DSN8810.EMP) COLGROUP(EDLEVEL,JOB,SALARY)

SYSIBM.SYSCOLDIST

NAME	TYPE	NUMCOLUMNS	COLGROUPCOLNO	CARDF
EDLEVEL	C	3	00090008000C	+0.330000E+02

SYSIBM.SYSCOLDISTSTATS

PARTITION	NAME	TYPE	NUMCOLUMNS	COLGROUPCOLNO	CARDF
1	EDLEVEL	C	3	00090008000C	+0.320000E+02
2	EDLEVEL	C	3	00090008000C	+0.000000E+00
3	EDLEVEL	C	3	00090008000C	+0.100000E+02
4	EDLEVEL	C	3	00090008000C	+0.000000E+00
5	EDLEVEL	C	3	00090008000C	+0.000000E+00

EDLEVEL,JOB,SALARY are non-indexed columns
Cardinality value of the column group is stored in the catalog tables

10.7.4 Example 1

In this example, the cardinality is collected for the column group (EDLEVEL, JOB, SALARY) by specifying the COLGROUP keyword.

The cardinality value is stored in column CARDF in SYSCOLDIST catalog table. Table space DSN8D81A.DSN8S81E has five partitions to hold the data for DSN8810.EMP and therefore the cardinality values are stored for each partition in column CARDF in SYSCOLDISTSTATS catalog table.

The name of only the first column (EDLEVEL) in the column group is recorded in column NAME in the catalog tables. The value 3 in column NUMCOLUMNS indicates that there are three columns in the column group. COLGROUPCOLNO shows the column numbers for the three columns in the colgroup.

The CARDF value in SYSCOLDIST catalog table indicates that there are 33 distinct values in DSN8810.EMP table for the colgroup (EDLEVEL,JOB,SALARY).

The CARDF value in SYSCOLDISTSTATS catalog table indicates that there are 32 distinct values in partition 1 and 10 distinct values in partition 2 in DSN8810.EMP table for the colgroup (EDLEVEL,JOB,SALARY). Partitions 2, 4, and 5 do not have any rows in the table.

Example 2

- **RUNSTATS TABLESPACE DSN8D81A.DSN8S81E TABLE(DSN8810.EMP)
  COLGROUP(EDLEVEL, JOB, SALARY)
  FREQVAL COUNT 10 MOST**

SYSIBM.SYSCOLDIST

NAME	TYPE	NUM COLUMNS	COL GROUPCOLNO	CARDF	COL VALUE	FREQUENCYF
EDLEVEL	C	3	00090008000C	+0.33000E+02	-	-
EDLEVEL	**F**	3	00090008000C	-0.10000E+01	.Ø....	+0.9523809523809523E-01
EDLEVEL	**F**	3	00090008000C	-0.10000E+01	.Ø....	+0.4761904761904762E-01
EDLEVEL	**F**	3	00090008000C	-0.10000E+01	.Ø....	+0.2380952380952381E-01
EDLEVEL	**F**	3	00090008000C	-0.10000E+01	.Ø....	+0.2380952380952381E-01
EDLEVEL	**F**	3	00090008000C	-0.10000E+01	.Ø....	+0.2380952380952381E-01
EDLEVEL	**F**	3	00090008000C	-0.10000E+01	.Ø....	+0.2380952380952381E-01
EDLEVEL	**F**	3	00090008000C	-0.10000E+01	.Ø....	+0.2380952380952381E-01
EDLEVEL	**F**	3	00090008000C	-0.10000E+01	.Ø....	+0.2380952380952381E-01
EDLEVEL	**F**	3	00090008000C	-0.10000E+01	.Ø....	+0.2380952380952381E-01
EDLEVEL	**F**	3	00090008000C	-0.10000E+01	.Ø....	+0.2380952380952381E-01

The cardinality value of the column group and 10 most frequent values are collected for the column group specified (EDLEVEL, JOB, SALARY)

10.7.5 Example 2

In this example, the cardinality is collected for the column group (EDLEVEL, JOB, SALARY) by specifying the COLGROUP keyword.

In addition to this, the 10 most frequently occurring values for the colgroup are also collected by specifying FREQVAL COUNT 10 MOST.

The values are stored in SYSCOLDIST and also in SYSCOLDISTSTATS catalog tables since the table space DSN8D81A.DSN8S81E is partitioned. The entries in SYSCOLDISTSTATS catalog table are not shown on the visual.

Column TYPE indicates whether the row has the cardinality value or frequency value.

- If TYPE has the value ‘C’, CARDF contains the cardinality value, and FREQUENCYF is not relevant.
- If TYPE has the value ‘F’, FREQUENCYF has the percentage of rows in the table with the value specified in column COLVALUE when the number is multiplied by 100, and CARDF is not relevant.

Note that columns COLVALUE and COLGROUPCOLNO are VARCHAR columns FOR BIT DATA. Moreover, if the value has a non-character data type, the data might not be printable.

Performance Expectation

Performance gain in certain queries (such as queries that use non-leading indexed columns and/or where non-indexed columns are used as predicates, and that have skewed (non-uniform) data)

CPU and elapsed times for RUNSTATS utility increases if new functions are invoked

- Actual amount of increase depends on the number of columns specified in the COLGROUP keyword
- Number of COLGROUPs specified
- Amount of sorting that needs to be done be RUNSTATS

Note:

Do not forget to allocate sort workfiles for RUNSTATS when collecting statistics on non-indexed columns

10.7.6 Performance expectation

When the new RUNSTATS functionality is invoked, the user can expect to see an increase in CPU and elapsed time for the utility. The amount of increase depends on the number of columns specified with the COLGROUP keyword at the table level, as well as the number of column groups (multiple COLGROUP keywords) specified.

More information about other RUNSTATS enhancements can be found in 9.3, “RUNSTATS enhancements” on page 651.

Cost-Based Parallel Sort

DB2 V8 introduces cost-based consideration to determine if parallel sort for single or multiple (composite) tables should be disabled

- Sort data size < 2 MB (500 pages)
- Sort data per parallel degree < 100 KB (25 pages)

Hidden DSNZPARM OPTOPSE control - default is ON

- ON - cost-based parallel sort considerations for both single table and multi-table
- OFF - not cost-based - sort parallelism behavior same as V7, i.e.
  - Single (composite) table : parallel sort
  - Multiple tables : sequential sort only

Considerations:

- Elapsed time improvement
- More usage of workfiles and virtual storage

10.8 Cost-based parallel sort

Currently (V7), DB2 tries to do as much sorting in parallel as possible. However, there are instances where it may not be cost effective to execute the sort process in parallel, one typical case being a small data sort. In DB2 V8 not all sorts are done in parallel. A cost model is used to decide whether or not to run the sorts in parallel, for both single-table as well as multi-table sorts.

The hidden DSNZPARM OPTOPSE is provided so that an installation can go back to the prior behavior of sort parallelism.

- ON — cost-based — parallel sort for both single table and multi-table

  Parallel sort is disabled under the following conditions:

  – Sort data size < 2MB (500 pages)

  – Sort data per parallel degree < 100KB (25 pages)

- OFF — not cost-based — sort parallelism behavior same as V7

Query Example - Plan Table

EXPLAIN output to determine if parallel sort is executed or not

```
SELECT * from Ta, Tb, Tc   where Ta.a2 = Tb.b2 and Tb.b3 = Tc.c3;

        SMJ_1          SMJ_2
 Ta   --------> Tb  ---------> Tc  (Merge scan join used to join the tables)
```

```
       Prior to DB2 V8                  DB2 V8 (with Hidden DSNZPARM OPTOPSE set
                                        to ON -> Post-Optimizer decides to do
                                        parallel sort)
+--------------------------------+      +--------------------------------+
|PLAN|TNAME|ACC|SORTC|SORTN|JOIN|      |PLAN|TNAME|ACC|SORTC|SORTN|JOIN|
|    |     |PGR|_PGR |_PGR |_PGR|      |    |     |PGR|_PGR |_PGR |_PGR|
|    |     |ID |_ID  |_ID  |_ID |      |    |     |ID |_ID  |_ID  |_ID |
+--------------------------------+      +--------------------------------+
|  1 | Ta  | 1 | ?   | ?   | ?  |      |  1 | Ta  | 1 | ?   | ?   | ?  |
|  2 | Tb  | 2 | 1   | 2   | 3  |      |  2 | Tb  | 2 | 1   | 2   | 3  |
|  3 | Tc  | 4 | ?   | 4   | 5  |      |  3 | Tc  | 4 | 3   | 4   | 5  |
+--------------------------------+      +--------------------------------+
```

SORTC is executed in sequential mode   SORTC is executed in parallel mode

© 2004 IBM Corporation

10.8.1 Parallel sort of multi-table composite

In data warehousing environments, it is often good practice to utilize as many resources as are available, in order to reduce the elapsed time of critical queries. Prior to DB2 V8, in some situations it is not possible to fully utilize the CPU when parallel sort is involved. This is mainly due to the fact that sort-composite is not pushed down for parallelism if the composite involves more than one table.

We use the following terminology:

- **Single composite table:** For example, a 2-way merge scan join sorts the table(s) in join column sequence before joining. The result of the sort is a single composite table. The sort of single composite table can be done in parallel in V7.
- **Multi-table composite:** This is the case, for example, in a 3-way merge scan join (see the visual). After TB1 and TB2 have been joined, they are resorted before joined to TB3. The result of that sort is a multi-table composite. We cannot use parallel sort in V7 for a multi-table composite table.

In DB2 V8, the sort process has been enhanced to be able to run the multi-table sorts in parallel. In this case, CPU resources can be better exploited and elapsed time can be reduced. As mentioned in the previous section, the optimizer decides whether the sort is done in parallel or not.

To illustrate this enhancement, let us assume that the access path for the following query (a three-table join) uses two merge scan joins, also known as sort merge join (SMJ):

```
SELECT * from Ta, Tb, Tc where Ta.a2 = Tb.b2 and Tb.b3 = Tc.c3;
```

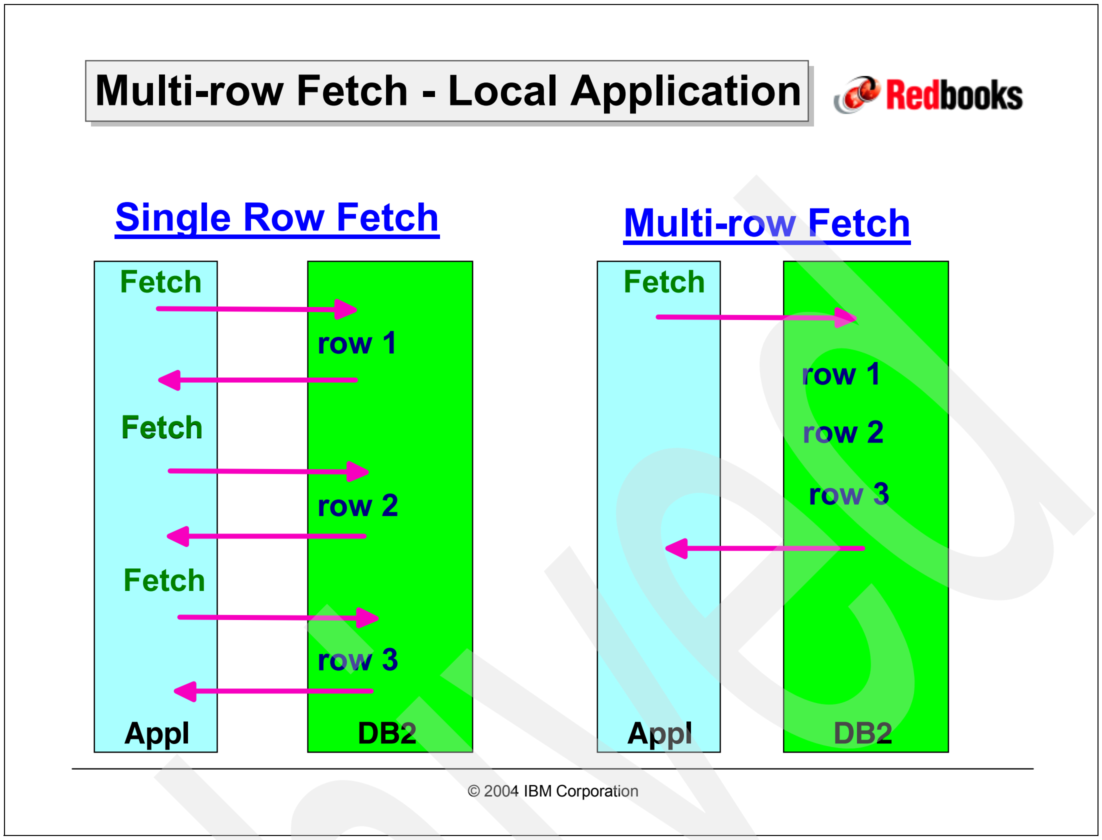
Prior to DB2 V8, the “sort composite” for SMJ_2 (output of SMJ_1 involving Ta and Tb and input to SMJ_2), is executed in the parent task (performed as a sequential sort). In DB2 V8, this sort may be pushed down to the child task and performed in parallel.

You can tell whether a sort is executed in parallel or not by examining the EXPLAIN output. In PLAN_TABLE output, SORTC_PGR_ID reflects the parallel_group_id if a "sort composite" is executed in parallel. Similarly, SORTN_PGR_ID reflects the parallel_group_id if a "sort new" is executed in parallel.

For PLAN 2, SORTC_PGR_ID, which involves only Ta, reflects parallel_group_id of 1 prior to DB2 V8 and in DB2 V8.

For PLAN 3, SORTC_PGR_ID, which involves Ta and Tb, reflects parallel_group_id of 3 — that is, this SORTC is executed in parallel in DB2 V8. Prior to this enhancement, SORTC_PGROUP_ID is "?", meaning that this SORTC is executed in sequential mode.

Note that in the PLAN_TABLE entries, “?” indicates null value.



10.9 Performance aspects of multi-row operations

DB2 Version 8 introduces multi-row operations. This way you reduce the number of trips over the API from the application into DB2. Multi-row fetch and insert is described in great detail in 4.4, “Multi-row FETCH and INSERT” on page 239.

This section discusses some of the performance aspects and early performance measurements of using multi-row operations. We discuss the performance impact of:

- Multi-row fetch in local applications
- Multi-row insert in local applications
- Positioned update and delete with multi-row operations in local applications
- Multi-row operations in a distributed environment
  - When using rowsets in your DB2 for z/OS applications
  - Exploitation by DB2 clients on Linux, UNIX and Windows
  - Automatic exploitation of multi-row operations by DDF

10.9.1 Multi-row fetch in local applications

When using multi-row fetch operations, you normally fetch a set of rows, a rowset in a single fetch operation, as shown below:

```
FETCH NEXT ROWSET FROM my-cursor FOR 3 ROWS INTO :hva1, :hva2, :hva3
```

In the example we fetch 3 rows (to be in line with the figure above) in a single API (Application Program Interface) crossing (going from the application to DB2 and back) instead of 3 API crossings without rowsets.

Using multi-row fetch, local applications can reduce their CPU time up to 50% by avoiding API overhead for each row fetch. The percentage of improvement that is to be expected is lower if more columns (making the fetch more expensive) and/or fewer rows (making the reduction of the number of API crossings less significant) are fetched per call.

Remember also that fetch intensive workloads experience a higher impact of turning on the class 2 accounting trace. Therefore, a higher percentage of CPU time improvement can be expected by using multi-row fetch operations, if accounting trace class 2 is active.

10.9.2 Multi-row insert in a local environment

When using multi-row insert operations, you insert a set of rows, a rowset, in a single insert operation, as shown below:

```
INSERT INTO my-table FOR 20 ROWS VALUES(:hva1,:hva2,...)
```

In this example we insert 20 rows in a single insert operation. Again this saves trips across the API between the application and DB2. Up to 30% CPU time reduction has been achieved in lab performance measurements by avoiding the overhead of crossing the API for each individual row insert. Percentage-wise, the improvement is less for multi-row insert than for multi-row fetch, because in general an insert operation is more expensive than a fetch, and therefore the overhead of crossing the API is less significant.

The percentage of improvement that can be obtained is higher on tables with fewer indexes, fewer columns, and/or when you insert more rows per call.

Note that when using multi-row insert, you have an ATOMIC clause (the default) that specifies that, if the insert of any row fails, all changes made by the multi-row insert are undone. In order to provide this functionality, DB2 takes a SAVEPOINT at the start of a multi-row insert using the ATOMIC clause, which typically takes about 15μs on a z900 processor. This contributes less than 5% overhead when using a 2-row insert, and is completely negligible for a many-row insert operation.

10.9.3 Multi-row update and delete operations in a local environment

Similar improvement can be expected for multi-row cursor update and delete operations. However, remember that in most cases you want to delete or update individual rows from a rowset and not the entire rowset. In that case, no performance improvement is to be expected.

10.9.4 Multi-row operations in a distributed environment

Here we discuss multi-row operations in several situations.

Between DB2 for z/OS systems

Using multi-row operations, a dramatic reduction in network traffic and response time is possible by avoiding a message send/receive operation for each row in the following cases:

- Fetch when not [read-only or [CURRENTDATA NO and ambiguous cursor]]
- Update and/or delete with cursor
- Insert

These cases prevent DB2 from using block fetch in a distributed environment. An updateable cursor, or an ambiguous cursor when using CURRENTDATA NO, as well as an INSERT statement, always send a single row across the wire.

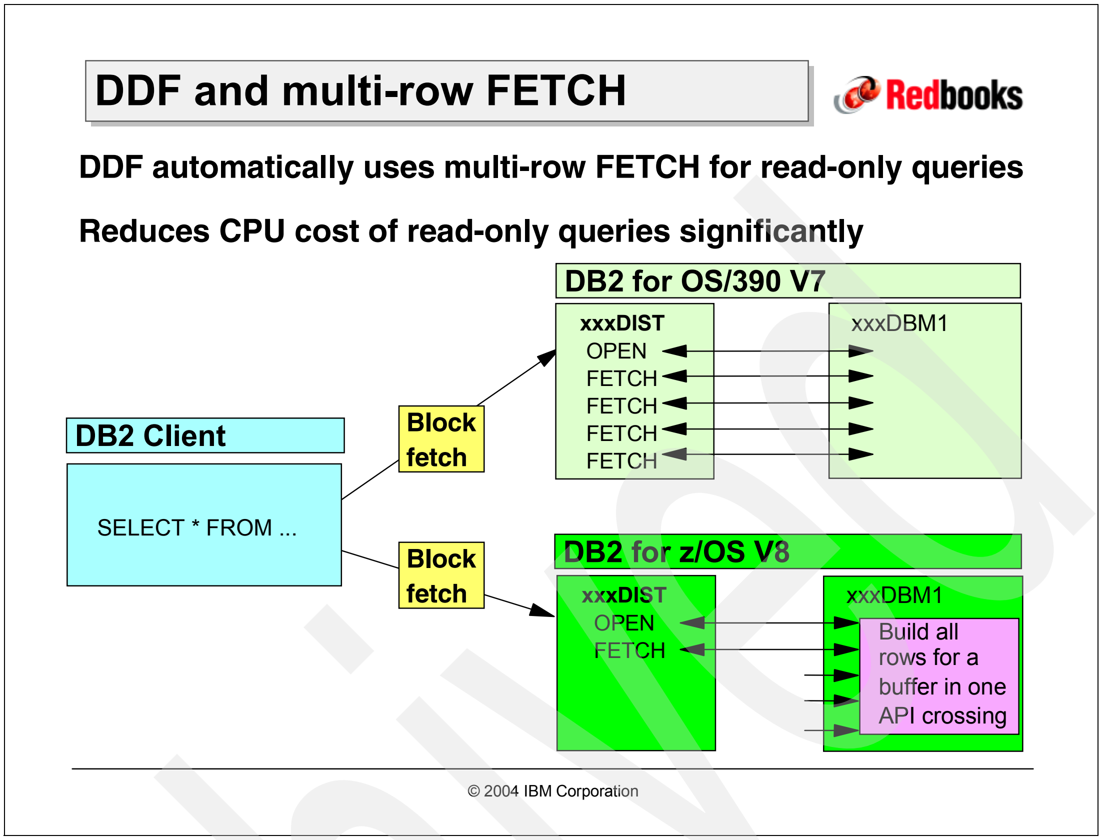
When using multi-row operations, DB2 always sends a rowset in a single network message, even for an updateable cursor, an ambiguous, or a (multi-row) insert operation. Therefore by using multi-row operations, you can have a similar effect on network traffic than block fetching, and reduce the network time dramatically. (However, you should be aware that you potentially hold locks all rows of the rowset when doing rowset operations.)

Between DB2 distributed clients and DB2 for z/OS

When executing INSERT statements on the distributed platform, DB2 ODBC clients on Linux, UNIX and Windows, already for so-called “array input”. When talking to a DB2 for z/OS and OS/390 V7, all INSERT statements are bundled by the driver in a single network message, when they are send to the DB2 for z/OS system. At the server side, the message is taken apart and multiple INSERT statements are executed to insert all the rows that make up the input array.

This functionality saves on network traffic, as fewer messages are sent. When going to DB2 for z/OS V8, “array input” can take advantage of DB2’s capabilities to use multi-row INSERT. In this case a single message is sent (as before) with a single (multi-row) INSERT statement. In this case, we see a reduction in the number of API crossings between DDF and DBM1 because it is only one INSERT statement, and therefore a reduction in CPU and elapsed time.

In addition, when using dynamic scrollable cursors with a DB2 ODBC/CLI client on the distributed platforms, it also use multi-row fetch operations when communicating with a DB2 for z/OS V8 server. This way, it is possible to retrieve multiple rows (a rowset) in a single network operation. Remember that non-rowset dynamic scrollable cursors only send single rows across the wire because of the semantics of the “dynamic” keyword. (This also applies to sending rows between DB2 for z/OS systems when using non-rowset dynamic scrollable cursors.)



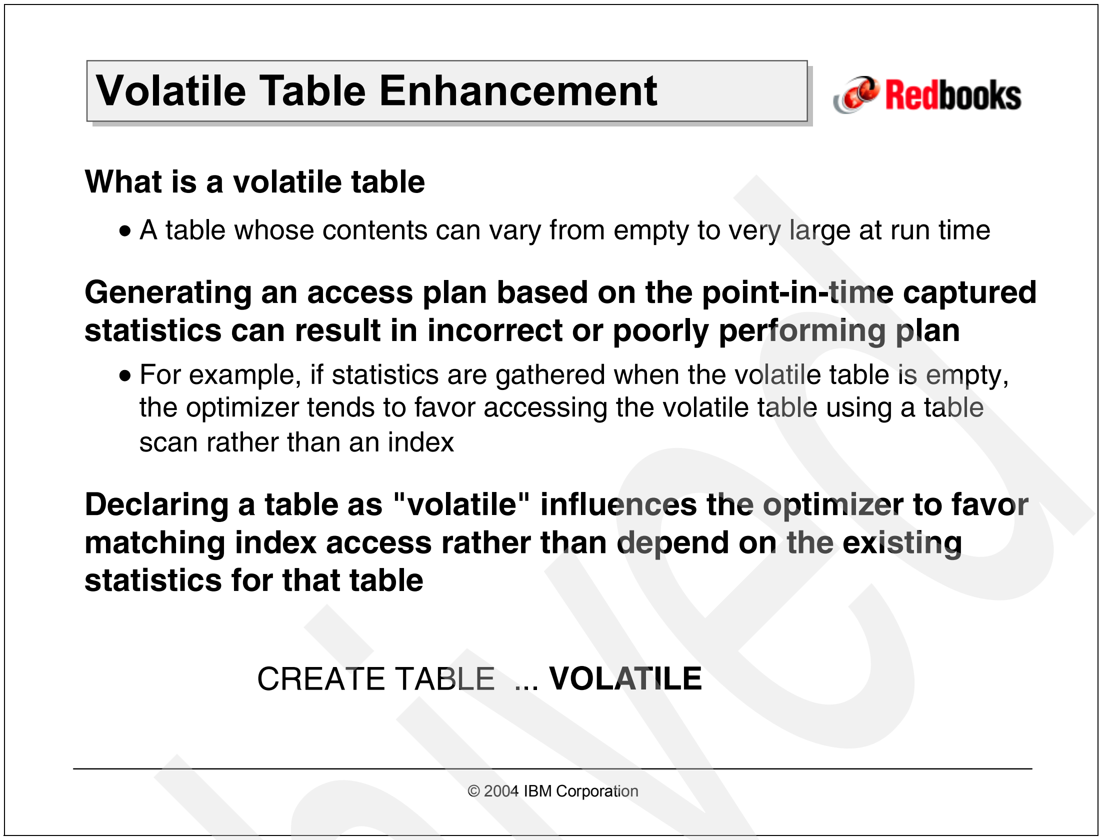
10.9.5 Automatic multi-row FETCH with DDF

In most cases, you must change your applications in order to exploit multi-row fetch operations. For example, you have to set up your applications to use host variable arrays. However, when you use a remote client to connect to DB2, for example, a Java application using a Type 4 connection with the Universal Driver, DB2 will automatically use multi-row fetch "under the covers" when fetching rows from the tables while building a block that will be sent back to the client.

In order for DB2 to be able to automatically exploit multi-row fetch for distributed applications, the cursor has to be read-only or you must be using CURRENTDATA NO with an ambiguous cursor. When these conditions are satisfied, DB2 can enable block fetching. When DB2, is putting together a block of data to be sent to the client inside the DDF address space, DDF issues FETCH statements (like any other application). When you are using block fetching against a V8 DB2 system, the DDF address space will use multi-row fetch to build the blocks of data to be sent to the client. This is completely transparent to the requester.

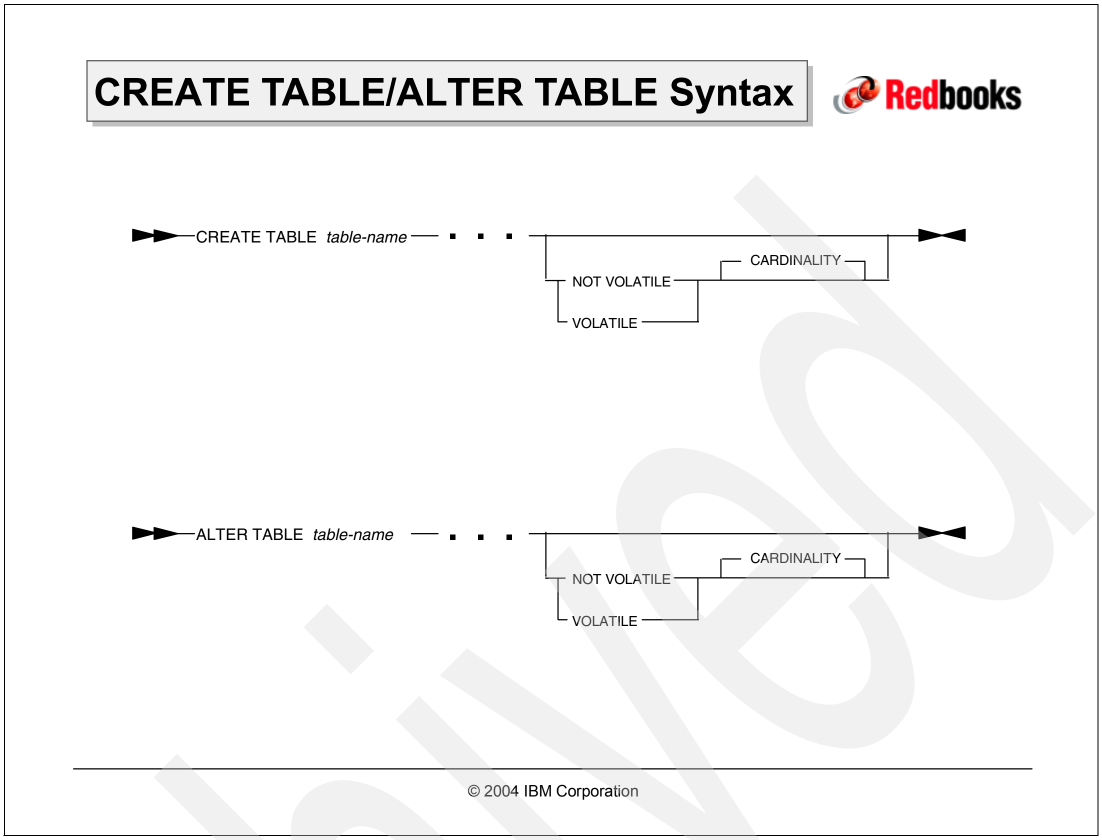
Early measurements have shown significant CPU savings by using this feature, up to 50% in a case where many rows are fetched.

Note that this enhancement does not require the client application to use the FOR n ROWS clause, or host variable arrays, and that this enhancement has no effect on the blocking done by DDF. It only affects the number of API crossings between DDF and DBM1.



10.10 What is a volatile table

A volatile table is a table whose contents can vary from zero to very large at run time. DB2 often does a table space scan or non-matching index scan when the data access statistics indicate that a table is small, even though matching index access is possible. This is a problem if the table is small or empty when statistics are collected, but the table is large when it is queried. In that case, the statistics are not accurate and can lead DB2 to pick an inefficient access path. Favoring index access may be desired for tables whose size can vary greatly.



10.10.1 CREATE TABLE/ALTER TABLE syntax

DB2 V8 adds two new keywords to the CREATE TABLE statement: VOLATILE (to favor index access whenever possible), and NOT VOLATILE (to allow any type of access to be used). Here are their effects:

VOLATILE: Specifies that for SQL operations, index access is to be used on this table whenever possible. However, be aware that by specifying this keyword, list prefetch and certain other optimization techniques are disabled.

NOT VOLATILE: Specifies that SQL access to this table should be based on the current statistics. This is the default.

CARDINALITY: An optional keyword expressing the fact that the table can have frequently changing cardinality; it can have only a few rows at times, and thousands or millions of rows at other times. This keyword is allowed for DB2 family compatibility, but will serve no additional function in DB2 for z/OS.

SAP R/3 Cluster Table Concurrency Issue

Cluster tables contain groups - or clusters - of rows

- The rows in a cluster logically belong together
- The clusters are defined by the primary key of the index on the table

There is a high incidence of lock contention when the same cluster table is being accessed concurrently due to access path patterns

VOLATILE option can be used to force index access for these tables to reduce chance of contention

Significant performance improvement for some SAP applications

10.10.2 SAP R/3 cluster table concurrency issue

One common database design involves tables that contain groups of rows that logically belong together. Within each group, the rows should be accessed in the same sequence every time. The sequence is determined by the primary key on the table. Lock contention can occur when DB2 chooses different access paths for different applications that operate on a table with this design.

To minimize contention among applications that access tables with this design, specify the VOLATILE keyword when you create or alter the tables. A table that is defined with the VOLATILE keyword is known as a volatile table. When DB2 executes queries that include volatile tables, DB2 uses index access whenever possible. As well as minimizing contention, using index access preserves the access sequence that the primary key provides.

Defining a table as volatile has a similar effect on a query to setting the NPGTHRSH subsystem parameter (introduced in DB2 V7) to favor matching and non-matching index access for all tables.

The effect of NPGTHRSH is subsystem-wide, whereas volatile is at table level. In DB2 V8, you can forget about DSNZPARM NPGTHRSH, instead use volatile tables.

Review of DSNZPARM NPGTHRSH

In DB2 V7 the best solution to the problem is to run RUNSTATS after the table is populated. However, if it is not possible to do that, you can use subsystem parameter NPGTHRSH to cause DB2 to favor matching index access over a table space scan and over non-matching index access.

The value of NPGTHRSH is an integer that indicates the tables for which DB2 favors matching index access. Values of NPGTHRSH and their meanings are:

- 0 DB2 selects the access path based on cost, and no tables qualify for special handling. This is the default.
- n The value you set depends on the following:
  - DB2 favors matching index access for tables for which the total number of pages on which rows of the table appear (NPAGES) is less than n. This is the situation when data access statistics have been collected for all tables and NPAGES is updated. The recommended value for n is a small value, such as 10.
  - DB2 favors matching index access for tables for which NPAGES=-1. This is the situation when data access statistics have not been collected for some tables, (NPAGES=-1 for those tables).

  The recommended value for n is a high value, such as 500.
- -1 DB2 favors matching index access for all tables; this is not recommended.

Cluster Table Concurrency Example

First name, Last name, Seq. # form the primary key

George Bush, Gray Davis, & Ron Gonzales are each clusters

Lock contention is minimized when rows are always accessed in Sequence # order within each cluster

Specifying VOLATILE favors index access and reduces chance of deadlock

First Name	Last Name	Sequence #	City
George	Bush	1	D.C.
George	Bush	2	Houston
George	Bush	3	Austin
Gray	Davis	1	Sacramento
Gray	Davis	2	Los Angeles
Ron	Gonzales	1	San Jose

10.10.3 Cluster table concurrency example

The volatile table enhancement provides a way in DB2 to indicate that a given table is made up of logical rows, with each logical row consisting of multiple physical rows from that table. A logical row is identified by the primary key with a “sequence number” appended, to provide the logical ordering of the physical rows. When accessing this type of table, the logical rows are intended to be accessed in this order (primary key + sequence number). This reduces the chance of deadlocks occurring when two applications want to access the same logical row but touch the underlying physical rows in a different order. To illustrate this enhancement, we create the following table and populate it with data:

```
CREATE TABLE VOLTABLE(
  SEQ#          SMALLINT GENERATED ALWAYS AS IDENTITY,
  FIRSTNAME     CHAR(10),
  LASTNAME      CHAR(10),
  CITY          CHAR(20));

CREATE UNIQUE INDEX IX1 ON VOLTABLE(FIRSTNAME,LASTNAME,SEQ#);

INSERT INTO VOLTABLE(FIRSTNAME,LASTNAME,CITY)
VALUES('GEORGE','BUSH','D.C.');
INSERT INTO VOLTABLE(FIRSTNAME,LASTNAME,CITY)
VALUES('GEORGE','BUSH','HOUSTON');
INSERT INTO VOLTABLE(FIRSTNAME,LASTNAME,CITY)
VALUES('GEORGE','BUSH','AUSTIN');
INSERT INTO VOLTABLE(FIRSTNAME,LASTNAME,CITY)
VALUES('GRAY','DAVIS','SACRAMENTO');
INSERT INTO VOLTABLE(FIRSTNAME,LASTNAME,CITY)
VALUES('GRAY','DAVIS','LOS ANGELES');
INSERT INTO VOLTABLE(FIRSTNAME,LASTNAME,CITY)
VALUES('RON','GONZALES','SAN JOSE');
```

Notice that the table VOLTABLE is created without the keyword VOLATILE. The table is therefore created as a non-volatile table. The PLAN_TABLE entry reveals that the access strategy is a table space scan for the following query:

```
SELECT * FROM VOLTABLE
```

The PLAN_TABLE entry reveals that the access strategy is through index IX1 with list prefetch enabled, if the query includes a WHERE clause as in the following query:

```
SELECT * FROM VOLTABLE where FIRSTNAME = 'GEORGE' and LASTNAME = 'BUSH'
```

Thus, if two applications try to access the data concurrently from this table, they do not necessarily retrieve the rows in the same sequence and there is the possibility of deadlock.

Use the ALTER TABLE statement to alter the table VOLTABLE to be volatile as follows:

```
ALTER TABLE VOLTABLE VOLATILE
```

The PLAN_TABLE entries reveal that the access strategy is through index IX1 with no list prefetch enabled for both the following queries:

```
SELECT * FROM VOLTABLE
SELECT * FROM VOLTABLE where FIRSTNAME = 'GEORGE' and LASTNAME = 'BUSH'
```

Thus, if two applications try to access the data concurrently from this table, they retrieve the rows in the same sequence, thus eliminating or at least reducing the possibility of deadlock.

Volatile Table Implications

Column SPLIT_ROWS in SYSIBM.SYSTABLES indicates if the table is a VOLATILE table

For VOLATILE tables, index access is chosen, whenever possible, regardless of the efficiency of the available indexes

Certain types of optimization techniques are disabled when using VOLATILE

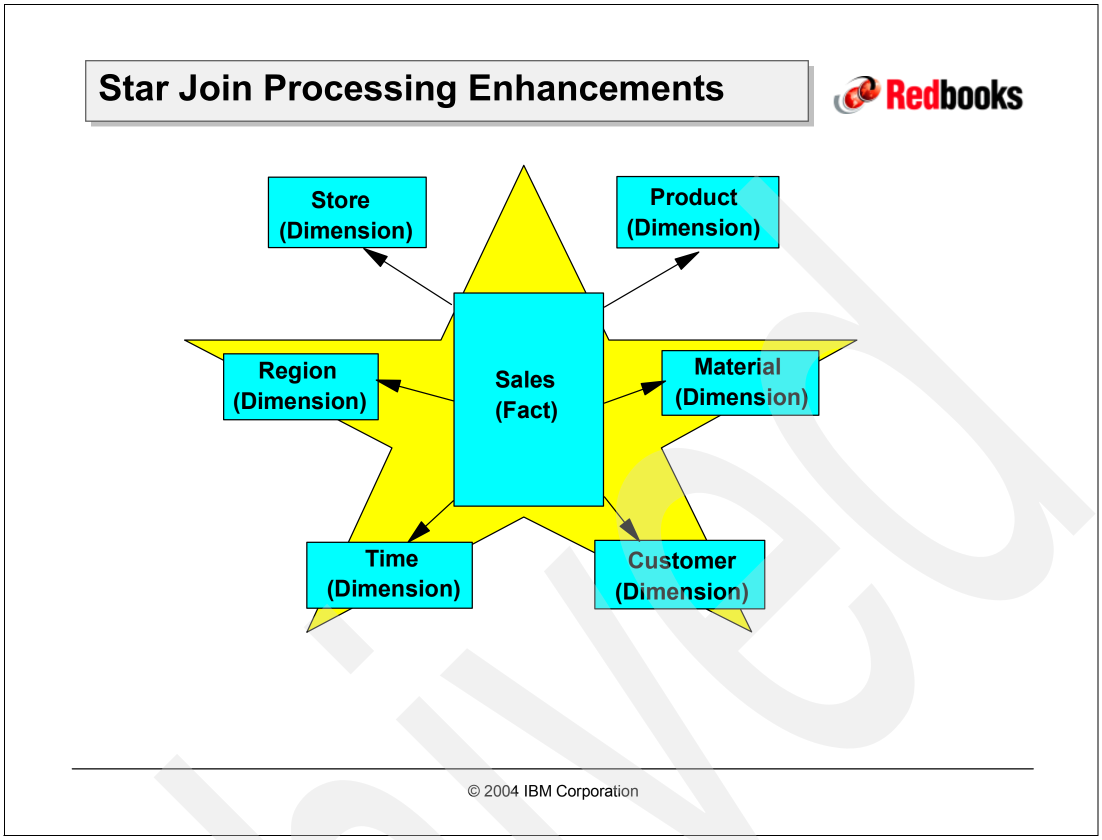
10.10.4 Volatile table implications

If you create a table with keyword VOLATILE, a new column SPLIT_ROWS in SYSIBM.SYSTABLES contains 'Y", otherwise the value is blank.

A non-volatile table can be changed to a volatile table using keyword VOLATILE with ALTER TABLE and the value in column SPLIT_ROWS in SYSIBM.SYSTABLES is set to 'Y'. A volatile table can be changed to a non-volatile table using keyword NOT VOLATILE with ALTER TABLE and the value in column SPLIT_ROWS in SYSIBM.SYSTABLES is set to ' '.

In both these situations, the plan or package is not invalidated. Instead, the value 'A' is stored in column VALID in SYSIBM.SYSPLAN and SYSIBM.SYSPACKAGE to indicate that the access strategy needs to be evaluated. The plan can continue to be executed with the existing access strategy. An explicit rebind is necessary to change the access strategy and at that time, the value in column VALID is changed from 'A' to 'Y'.

Certain types of access paths are disabled for volatile tables. They are: list prefetch, hybrid join and multi-index access.



10.11 Star join processing enhancements

There exist many kinds of data models to support today's business requirements, but typically for data warehousing, the design usually uses a number of highly normalized tables (known as dimensions or snowflakes) surrounding a centralized table (known as the fact table). This model is known as a *star schema*, named as such because the dimension tables appear as points of a star, surrounding the central fact table. The visual shows a simple star schema design.

Star Schema Data Model

Redbooks

Star schema data model

- Often used in data warehouses, data marts and OLAP
  - To minimize data redundancy
  - Has highly normalized tables (dimensions or snowflakes)
  - Has a centralized table (fact table)

Star schema query characteristics

- Join predicates between the fact and dimension tables are equi-join predicates
- Existence of local predicates on the dimension tables
- Large number of tables (dimensions/snowflakes and fact tables) participating in the query

10.11.1 Star schema data model

The increased demand for Decision Support Systems (DSS) and Online Analytical Processing (OLAP) for business increases the complexity of database design, query construction and query optimization.

The concept of a data warehouse or data mart is evolving and expanding as the business grows. Typically, the central data warehouse needs to support several data marts, which look at a subset of the data in their own way.

The attributes that generally dictate a star schema database model are as follows:

- Large fact table:
  - The fact table often contains sales type transactions that can be in the order of hundreds of millions, or billions of data rows.
- Highly normalized design:
  - The dimension tables are highly normalized to avoid maintaining redundant descriptive data in the central fact table. Redundant data introduces opportunities for data inconsistencies. In addition, storing descriptive information in the fact table would make the size of the fact table even bigger than it already is.
- Relatively small dimensions:
  - Highly normalized dimension tables contain a finite number of descriptions and detailed information for the codes stored in the fact table.

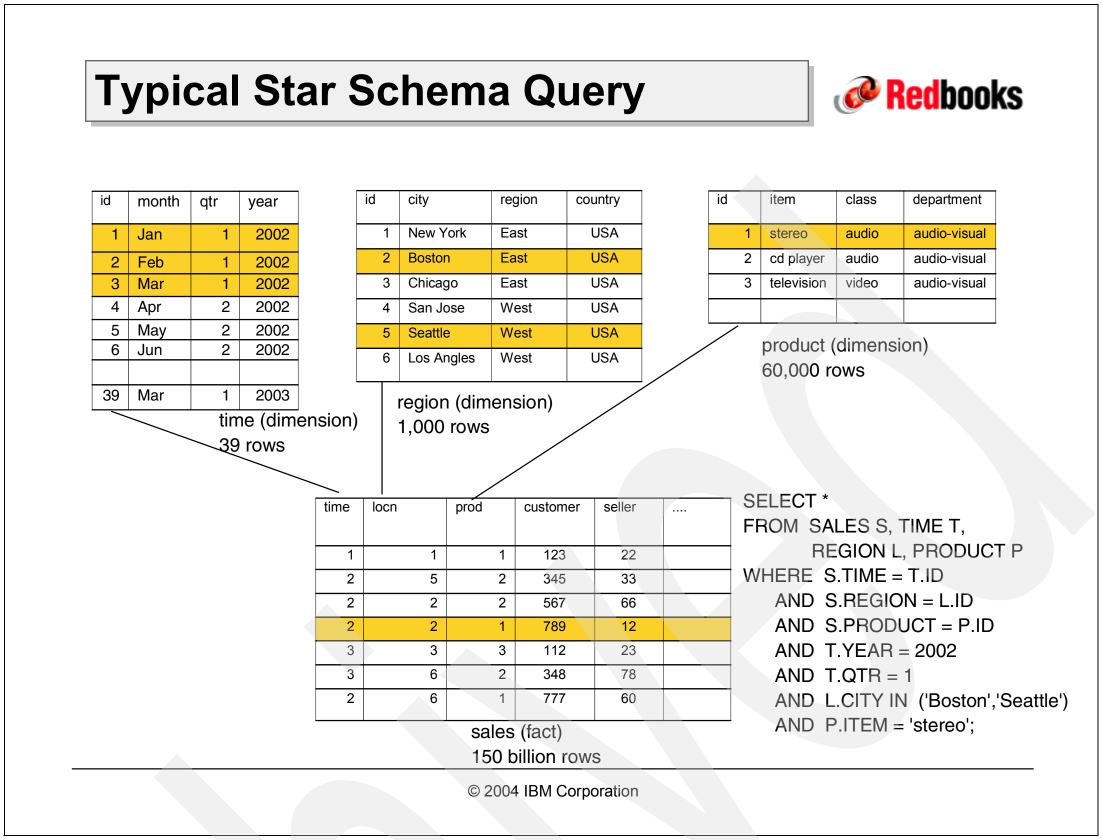
- Sparse "Hyper Cube":
  - There is a high correlation among dimensions, leading to a sparse nature of data in the fact table. For example, product sales are dependent on the climate, therefore the sale of shorts is more likely in a state enjoying hot weather.
- Fact table is dependent on the dimension tables:
  - If the dimension tables exist due to normalization of repetitive fact table data, then there exists a parent-child relationship, where the dimension table is the parent, and the fact table is the child. There is no requirement however for an explicit foreign key relationship to be defined.

Unlike OLTP queries, where a large number of short duration queries execute, OLAP queries involve a large number of tables and immensely large volumes of data to perform decision making tasks. Hence, OLAP queries are expected to run much longer than OLTP queries, but are less frequent.

In a purely normalized star schema design, the fact table does not contain attribute information, and merely contains event occurrences. To decode the fact table data, each row must be joined to the relevant dimensions to obtain the code descriptions.

Consequently, we can characterize a typical star schema query as having the following properties:

- Join predicates between the fact and dimension tables are equi-join predicates:
  - Decoding a fact table column requires an equi-join to the relevant dimension table, matching on the fact code value.
- Existence of local predicates on the dimension tables:
  - Assuming the attribute information has been normalized within the dimension tables, selective predicates are applied to the dimensions rather than the fact table.
- Large number of tables participating in the query:
  - The star schema (and the snowflake schema) dictates that many tables participate in a single star join query.
- No join predicates that cross dimensions.



10.11.2 Typical star schema query

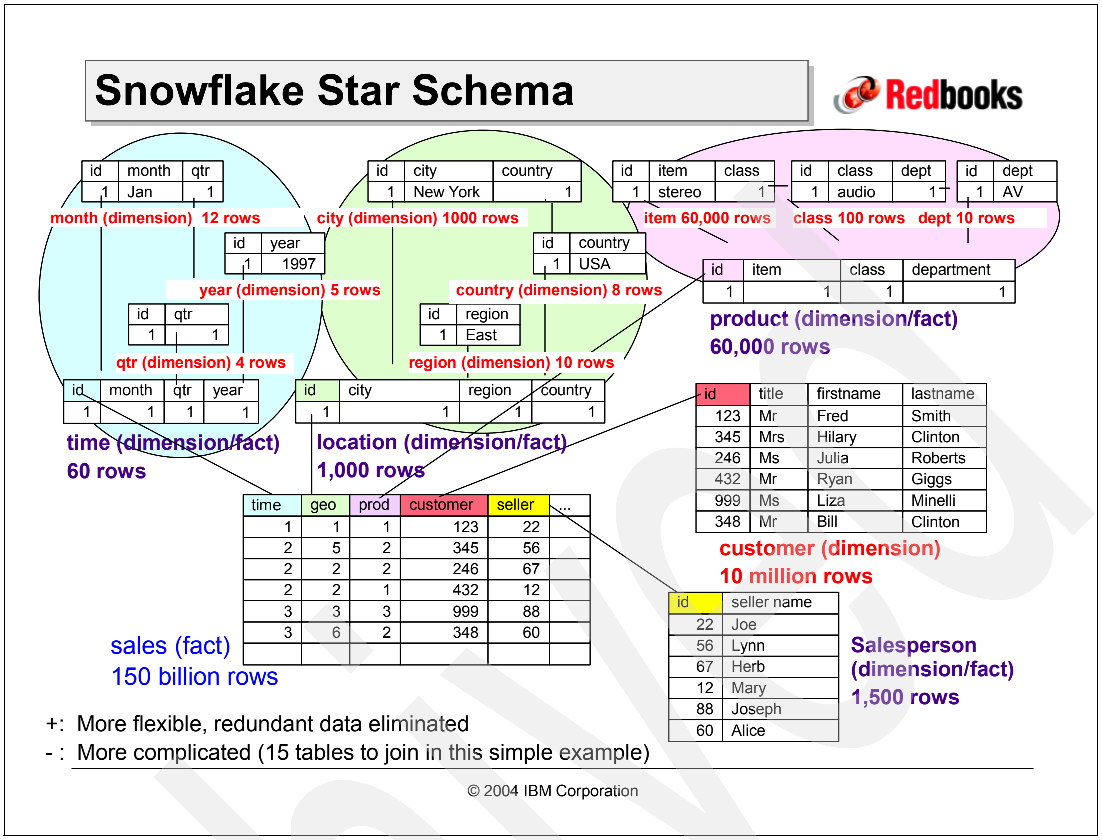
The sample star join query against the sample set of tables is shown on the visual.

Assume that a very expensive stereo is one of 20,000 products that are only sold at a few of the 600 store locations, including Boston and Seattle, and these 600 locations have a 1 in 3 month rotation for a single sale.

Considering the attributes of a star schema query, and the complexity of the associated data model, an efficient access path for such a query has three major objectives:

- Encourage a matching index scan of the fact table.
  - The large fact table cannot be scanned in its entirety, unless a large percentage of fact table rows are to be retrieved. Therefore matching index access must be available on as many selective join predicates as possible.
- Access dimension tables before the fact table to minimize the search space.
  - With the filtering provided by the intersection of dimensions, matching index access on as many fact and dimension join columns as possible reduces the range of data rows that must be retrieved for each fact table access. This implies that there are selective local predicates applied to these dimensions to narrow down the search space.
- Determine the balance between increased fact table filtering and excessive cartesian result.
  - Because there are no join predicates between dimensions (all join predicates are between dimensions and the fact table — unless snowflakes are involved), joining dimensions before accessing the fact table to reduce the search space, results in a cartesian product of those tables.

– Although increasing the selective dimensions may limit the qualifying fact table rows, it will increase the cartesian dimensions that must be joined to the fact table. For a cartesian result of 10,000 rows, adding a further dimension with as little as 2 table rows doubles the cartesian result, thus doubling the number of rows to be joined to the fact table.

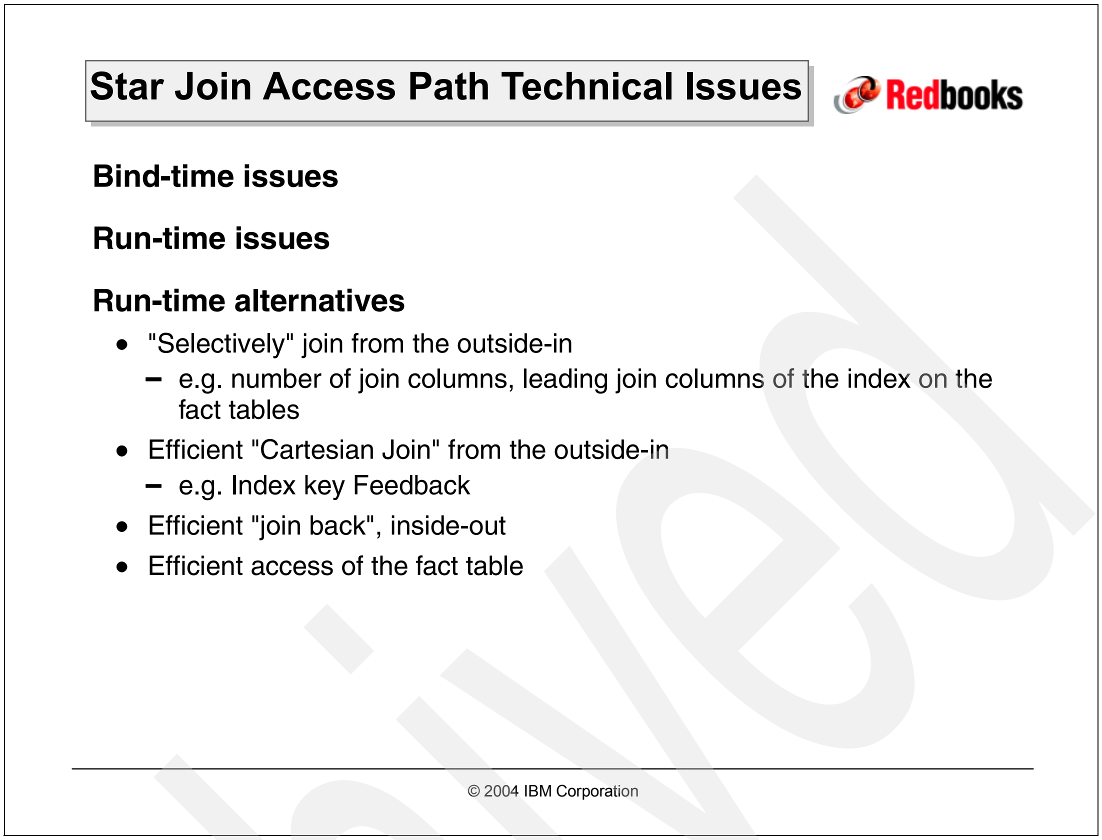


Snowflake star schema design

The snowflake star schema takes the star schema concept to the next level, with further normalization within the dimensions.

Probably the majority of star schema implementations include snowflakes in at least some of their dimensions. Note that an alternative design might take different levels within what is shown here as a single dimension (for instance, time) and implement them as separate dimensions. Such a design would still qualify as a star schema, although its capabilities for performance and flexibility are likely to be different from the design shown here.

In the figure above, the time, location, and product dimensions are further normalized than they were in the previous visual. This is what we call a snowflake design. It is the further normalization of a simple star schema. Note that in the time dimension for example, month, qtr (quarter) and year, are only represented by a "code number" in the time dimension. The actual values, for example, the actual year, is one more level down, in the YEAR table. The same is true for month and qtr.



10.11.3 Star join access path technical issues

The technical issues surrounding the optimization of decision support and data warehousing queries against a star schema model can be broken down into bind-time and run-time issues.

The first consideration generally is the sheer size of the fact table, and also the large number of tables that can be represented in the star schema.

- Bind-time issues:

  For non-star join queries, the optimizer uses the pair-wise join method to determine the join permutation order. However, the number of join permutations grows exponentially as the number of tables being joined increases. This results in an increase in the bind time, because of the time required to evaluate all possible join permutations.

- Run-time issues:

  Star join queries are also challenging at run-time. DB2 uses either cartesian joins of dimension tables or pair-wise joins. The pair-wise joins has worked quite effectively in the OLTP world. However, in the case of star join, since the only table directly related to other tables is the fact table, the fact table is most likely chosen in the pair-wise join, with subsequent dimension tables joined based on cost.

  Even though the intersection of all dimensions with the fact table can produce a small result, the predicates supplied to a single dimension table are typically insufficient to reduce the enormous number of fact table rows. For example, a single dimension join to the fact table may find:

  – 100+ million sales transactions in the month of December
  – 10+ million sales in San Jose stores
  – 10+ million sales of Jeans, but
  – Only thousands of rows that match all three criteria

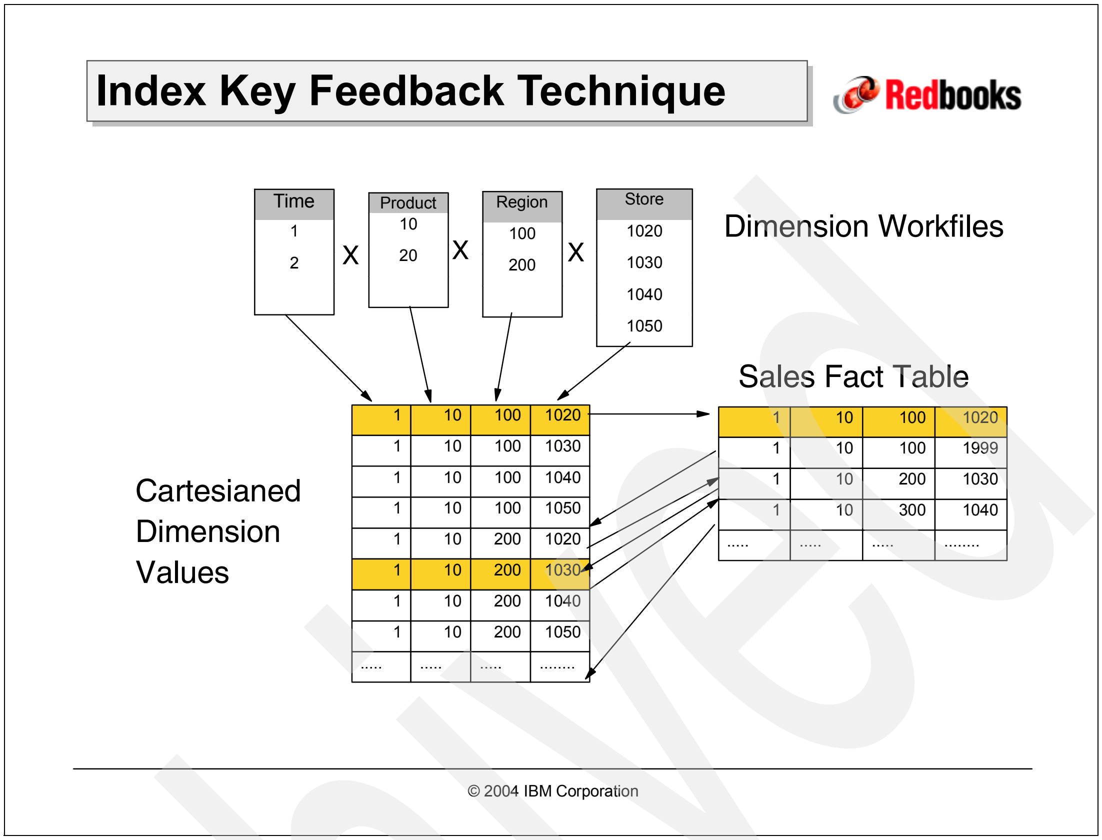
Hence there is no one single dimension table which could be paired with the fact table as the first join pair to produce a manageable result set.

With these difficulties in mind, the criteria for star join can be outlined as follows:

- **"Selectively" join from the outside-in:**
  - Purely doing a cartesian join of all dimension tables before accessing the fact table may not be efficient if the dimension tables do not have filtering predicates applied, or there is no available index on the fact table to support all dimensions. The optimizer should be able to determine which dimension tables should be accessed before the fact table to provide the greatest level of filtering of fact table rows.
  - For this reason, outside-in processing is also called the filtering phase. V8 enhances the algorithms to do a better job at selecting which tables to join during outside-in processing.
- **Efficient "Cartesian Join" from the outside-in:**
  - A physical cartesian join generates a large number of resultant rows based on the cross product of the unrelated dimensions. A more efficient cartesian type process is required as the number and size of the dimension tables increase, to avoid an exponential growth in storage requirements. Index key feedback technique is useful for making cartesian joins efficient.
- **Efficient "join back", inside-out:**
  - The join back to dimension tables that are accessed after the fact table must also be efficient. Non-indexed or materialized dimensions present a challenge for excessive sort merge joins and workfile usage. DB2 V8 enhancements introduce support for in-memory workfiles and sparse indexes for workfiles to meet this challenge.
- **Efficient access of the fact table:**
  - Due to the generation of arbitrary (or unrelated) key ranges from the cartesian process, the fact table must minimize unnecessary probes and provide the greatest level of matching index columns based on the pre-joined dimensions.

Of course, the very first step is to determine if the query qualifies for a star join. You can refer to the whitepaper *The Evolution of Star Join Optimization* available from the Web site:

`http://www.ibm.com/software/data/db2/os390/techdocs/starjoin.pdf`



Index key feedback technique

For an efficient cartesian join process, DB2 employs a “logical”, rather than “physical” cartesian join of the dimension tables. Each dimension covered by the chosen fact table index is accessed independently before the fact table. Each qualifying dimension has all local predicates applied, with the result sorted into join column order, and finally materialized into it’s own separate workfile.

Rather than requiring the physical workfile storage involved in a physical cartesian product, DB2 simulates a cartesian join by repositioning itself within each workfile to potentially join all possible combinations to the central fact table. The sequence of this simulated cartesian join respects the column order of the selected fact table index.

The sparseness of data within the fact table implies a significant number of values generated by the cartesian process are not to be found by a join to the fact table. To minimize the CPU overhead of joining unnecessarily derived rows to the fact table, DB2 introduces an index key feedback loop to return the next highest key value whenever the fact table is accessed.

A *miss* returns the next valid fact table index key so that DB2 can reposition itself within the dimension workfiles, thus skipping composite rows with no possibility of obtaining a fact table match.

A *hit* on the fact table index returns the matching fact table row. When a matching row is found in the fact table, the next fact table key is checked to see if there is a duplicate key. The position is moved forward in the fact table index until no more duplicates are found. When this "not found" condition is met, the next highest key is returned.

The visual demonstrates the index key feedback technique.

This technique, used during the outside-in stage, is a *push-down star join*, which involves an index scan on the fact table. The star join is pushed down to the Data Manager component of DB2 (that is, stage 1 — non-star joins are handled by DB2's Relational Data System (RDS) component, that is stage 2). In the PLAN_TABLE, the column JOIN_TYPE has the value "S" to indicate star join.

Star Join Workfile Challenges

Redbooks

Where do they come from?

- Materialization of snowflakes
- Sort of dimension tables

Typically small (hundreds or thousands)

Today, RSCAN is the only option

- Outside-in processing
- Inside-out processing (sparse index can also be used in V7)

Version 8 enhancements

- In-memory workfiles
- Sparse index on workfiles (also during outside-in processing)

10.11.4 Star join workfile challenges

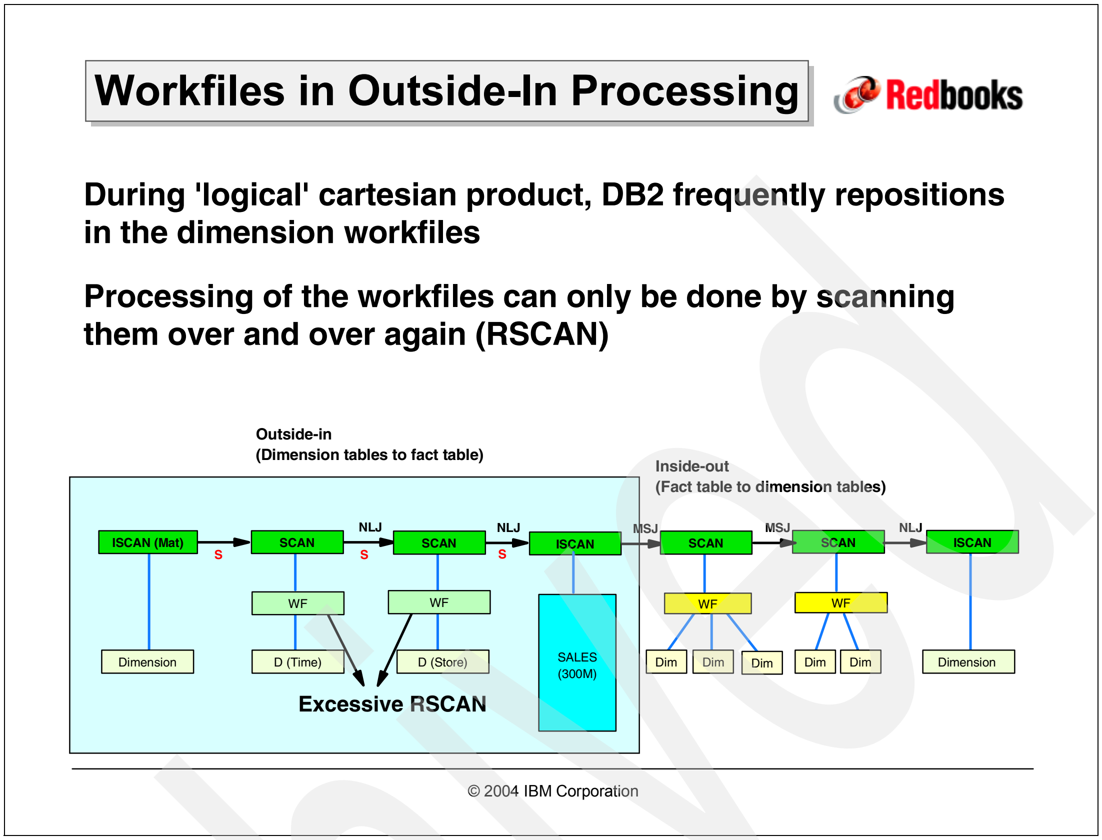
Processing star join queries usually involves a lot of workfile processing. They are mainly created because:

- The star schema uses a snowflake design. Currently (V7), snowflakes are always materialized in workfiles before being processed.
- Dimension tables need to be sorted in the selected fact table index order. After local predicates have been applied, the qualifying rows of dimension tables are sorted in the join column order, and stored in a workfile.

Workfiles are used during outside-in and inside-out processing, as illustrated in the next sections.

These workfiles are usually scanned multiple times while computing the result table for the query.

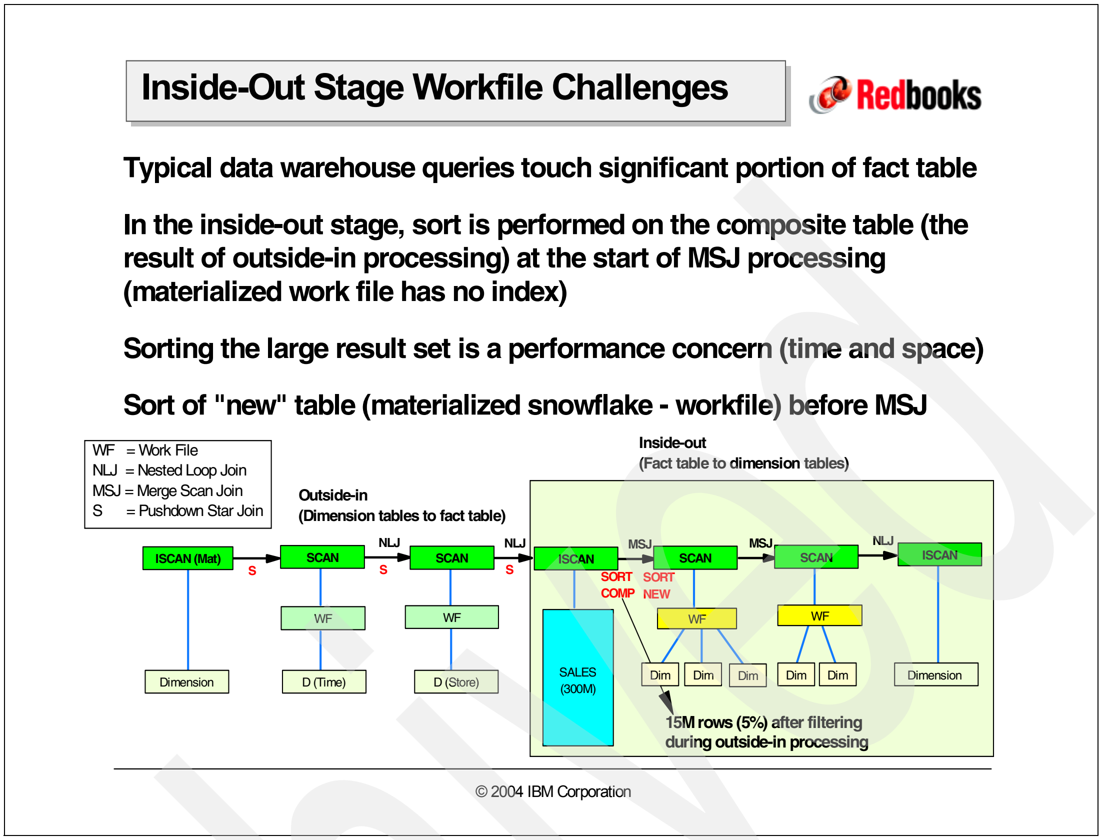
DB2 Version 8 dramatically enhances star join workfile processing by introducing in-memory workfiles and sparse indexes on workfiles.



Workfiles in outside-in processing

Workfiles are frequently used during the outside-in phase of star join processing. Dimension tables that are snowflakes are materialized in a workfile. In addition, also non-snowflake dimensions are materialized in workfiles after applying local predicates, and sorted in the key columns of the index chosen by the optimizer to access the fact table.

As mentioned before, DB2 does not actually perform a physical cartesian product when filtering the dimensions. It “simulates” the cartesian product by repositioning in the dimension tables. When these dimension tables are workfiles, finding matching entries involves a relational (sequential) scan of the workfiles (as there are no indexes on workfiles). Depending on when the dimension is joined, DB2 may have to reposition and re-scan the workfiles multiple times. These workfile scans can negatively impact the performance of your star join queries.



Inside-out stage workfile challenges

In the inside-out stage, the fact table (already joined with some dimension tables in the outside-in stage) is joined with the remaining dimension tables.

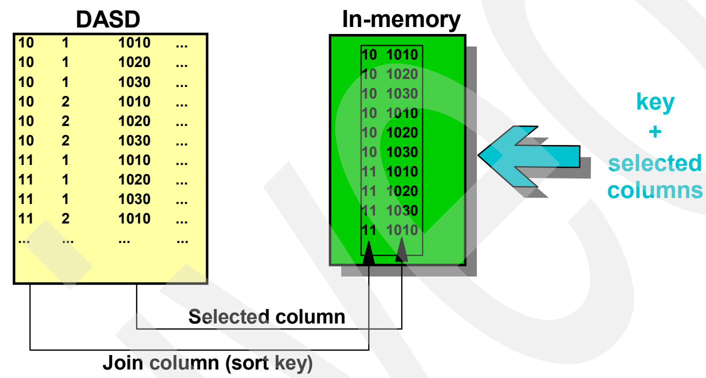
Extensive workfile processing is often required in the inside-out stage, because the intermediate result rows may still be large after the outside-in join. The lack of an index on workfiles may lead the optimizer to select a merge scan join. In addition, the tables that need to be joined back during inside-out processing (very often snowflakes materialized in a workfile) need to be sorted in the merge scan join column order.

This causes increased workfile space consumption, excessive CPU and I/O consumption, increased parallelism overhead due to merge activity. This is a critical storage/performance issue for large intermediate results or short running queries.

In-memory Workfiles

Redbooks

- In-memory data structure - above the bar
- Sorted in the join column order
- Containing only the join column and the selected columns
- Binary search for the target row
- More beneficial for large join composite
- Ideal for scanning dimension workfiles



10.11.5 In-memory workfiles

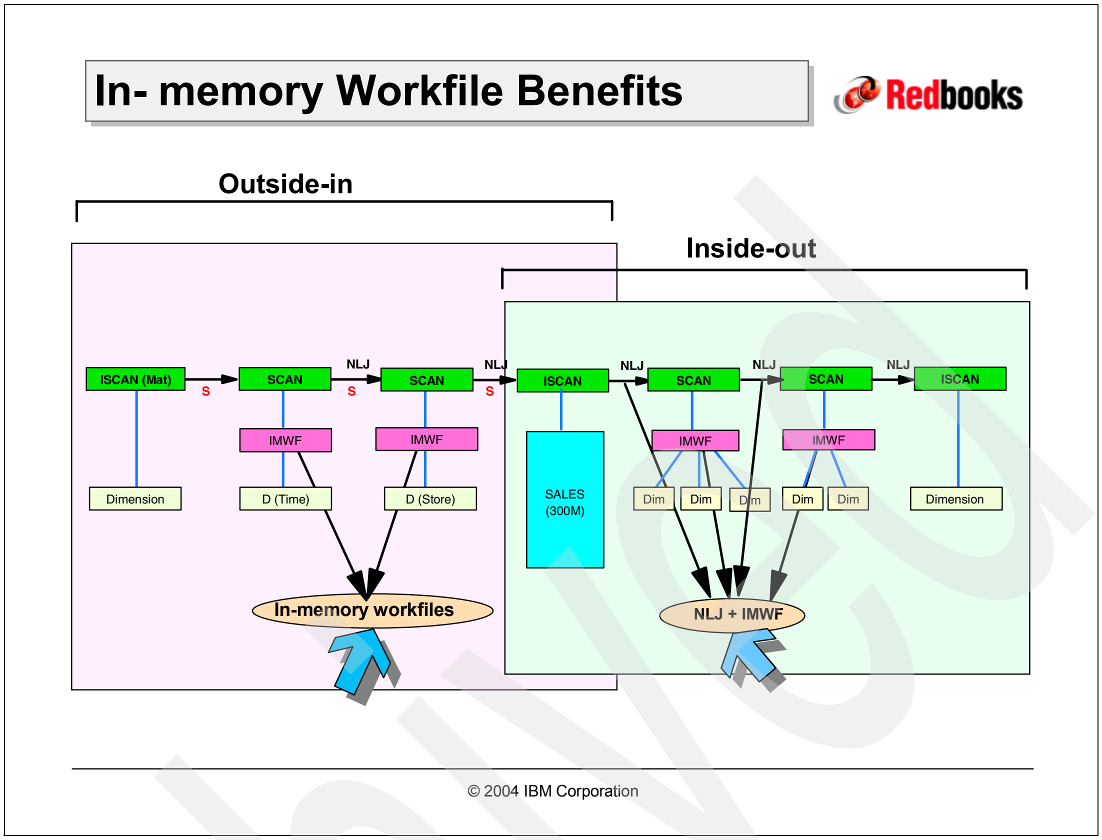
DB2 V8 supports in-memory workfiles for star join queries. This means that the normal workfile database is not used. The complete workfile is stored in memory instead.The in-memory workfiles contain all the columns of a workfile that are necessary to satisfy a query. These columns are the join column and the selected columns. The rows are sorted in join column sequence. The in-memory workfile is dense, in the sense that it contains an entry for each workfile record. DB2 performs a binary search to find the target row. (The entries are not stored in a B-tree structure like “normal” indexes.) As in-memory workfiles potentially save a large number of I/O operations against workfiles, they promise a considerable performance gain.

In-memory workfiles are stored in a new dedicated storage pool that is called a star join pool. The DB2 DSNZPARM SJMXPOOL specifies its maximum size, which defaults to 20 MB (maximum 1GB). It resides above the 2 GB bar and is only in effect when star join processing is enabled through DSNZPARM STARJOIN. When a query that exploits star join processing finishes, the allocated blocks in the star join pool to process the query are freed. More information on how to size SJMXPOOL can be found in the “Dedicated virtual memory pool for star join operations” section in the *DB2 Administration Guide,* SC18-7413.

The use of the star join pool is not mandatory. If it is not created, star join processing takes place without using in-memory workfiles. Also, if the allocation of space for a workfile in the star join pool fails, because SJMXPOOL is reached, then processing falls back to using the new sparse index, discussed in the next topic.

The possibility of caching workfiles in memory does not only help star join performance, but also other concurrently running jobs which normally use physical workfiles, such as sort, merge join, view materialization, nested table expression materialization, trigger, created temp table, non-correlated subquery, table UDF, etc. Since the star join is now capable of caching the data in memory, instead of using workfiles, these jobs do not have to compete for workfile storage in the buffer pool and table spaces used for workfile processing, resulting in fewer workfile access contention.

The use of in-memory workfiles, instead of traditional workfiles in the work database (usually DSNDB07), is also known as workfile caching or data caching.



In-memory workfile benefits

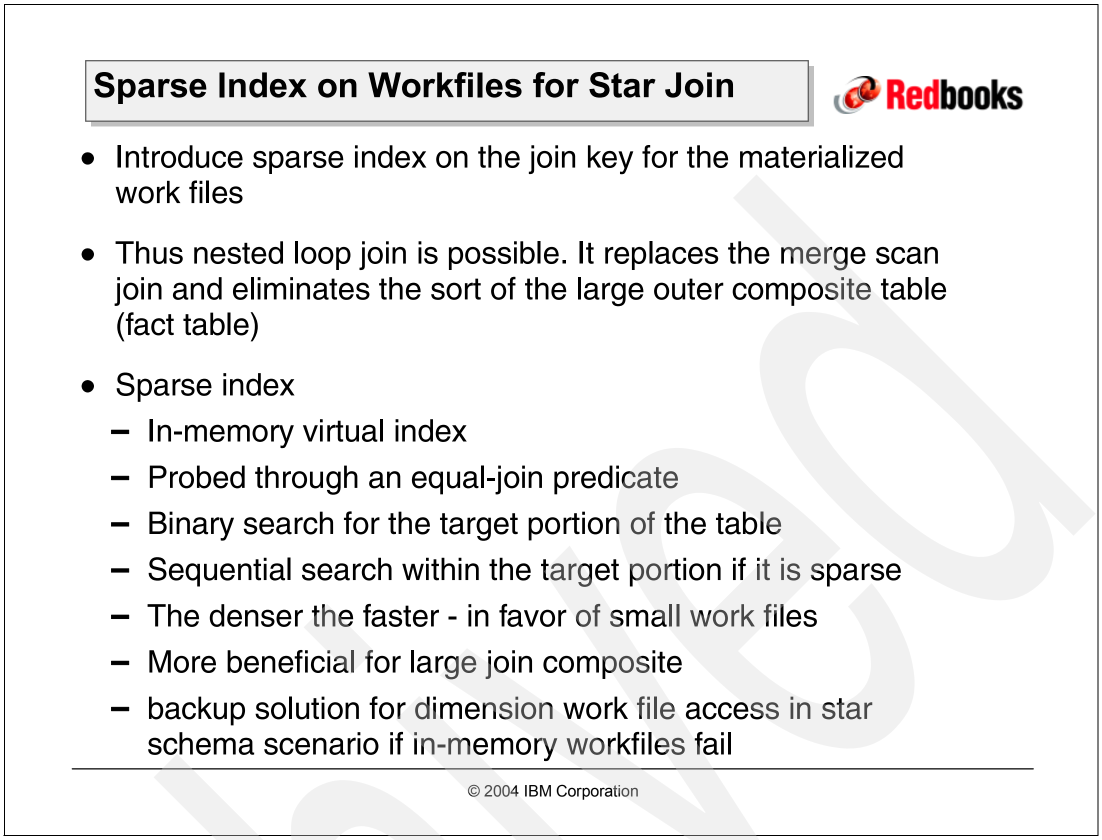
Both outside-in and inside-out processing benefit from the use of in-memory workfiles.

During outside-in processing, repositioning inside the workfiles, when doing the “logical” cartesian join, can be done using a binary search inside an in-memory workfile, which is much more efficient than repositioning using a relational (table space) scan. These are the benefits:

- Significant I/O cost reduction.
- Slight CPU cost reduction. More CPU reduction can be expected when large sorts or multiple sorts are involved.

During inside-out processing, the optimizer can now choose a nested loop join instead of a merge scan join, and avoid having to sort the large composite (the result of outside-in processing). In addition, access to the workfiles, usually representing materialized snowflakes, will be more efficient as the workfiles are now in-memory structures, and DB2 can use a binary search to access them. These are the benefits:

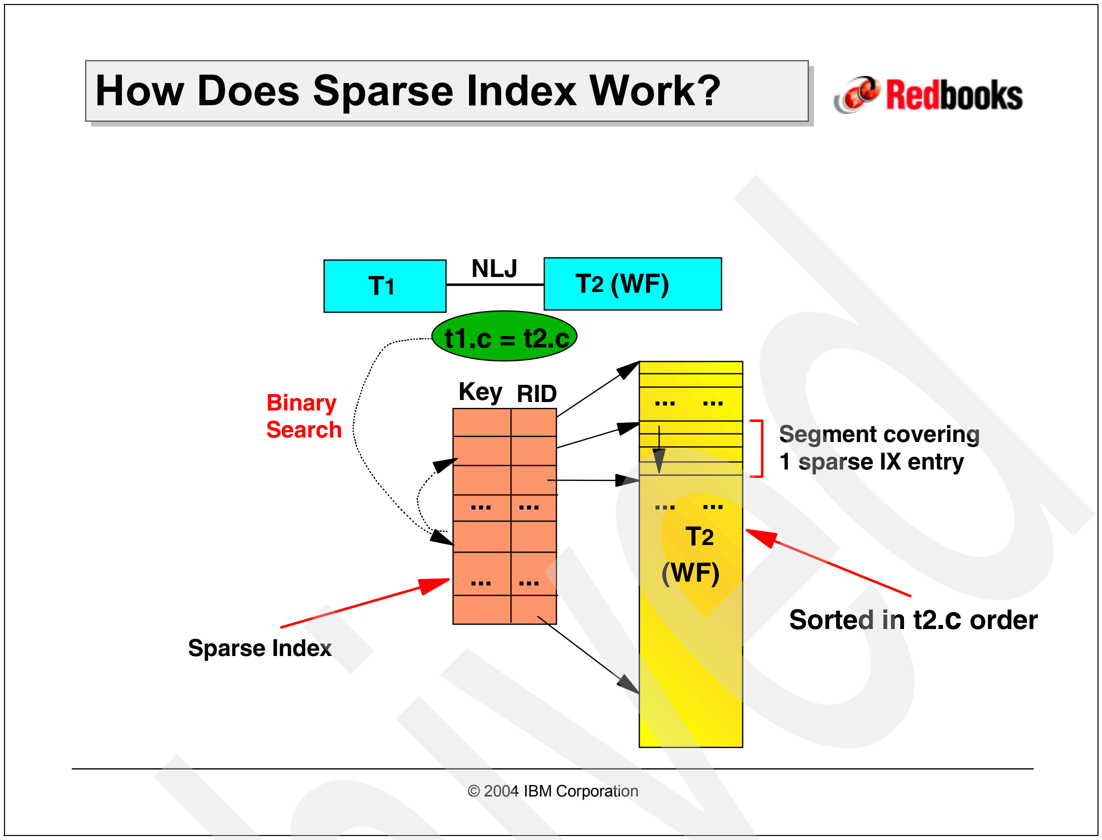
- The sort of a large composite (result of the outside-in joining phase) table is avoided (no sort work file space requirements) because the query no longer uses a merge scan join.
- Reduction of parallelism overhead (merge and repartition).
- Greater exploitation of parallelism.



10.11.6 Implementation of sparse index on workfiles for star join

When a star join query is executed, DB2 first tries to cache the workfile in-memory, as described in the previous section. DB2 tries to allocate the required space in the new in-memory workfile storage pool. If the allocation is successful, the sorted records for the workfile are cached in memory, and the physical workfile is not created. If the pool is not created or the allocation fails because no more space is available in the pool, only the sort key is saved in-memory as a sparse index, and the data records are stored in a physical workfile.

The sparse index is implemented as a memory structure within DB2's virtual storage. Each entry of the index contains the sort key and the 5-byte RID. If the total number of work file entries is small enough, then all entries can be represented in the index, thus providing a one-to-one relationship between the index and work file. The index itself only becomes sparse if the number of index entries cannot be contained within the space allocated. The maximum size of the structure is 240 KB.



How does sparse index work?

The workfile is sorted in the join column sequence (t2.c). While sorting, the (sparse) index is built (containing the key (join) columns and the RID. The sparse index structure is flat rather than a B-tree structure used by normal indexes on data tables. The index is probed through an equi-join predicate (t1.c = t2.c). A binary search of the index is utilized to find the target row (or segment of rows). In case the index is sparse (not all entries in the workfile have an index entry), a sequential search of the work file is subsequently initiated within the target segment to find the corresponding join row.

Note that the query will try to use data caching (in-memory workfiles) first, and that the use of the sparse index is only a "fallback" plan.

Tip: APAR PQ61458 (6/02) provides support for sparse indexes on the snowflake workfiles that are used in the inside-out join phase for DB2 V7.

V7 cannot use in-memory workfiles at all, or sparse indexes during outside-in processing.

Other star join enhancements

Snowflake handling - Controlled materialization

- V7 - Snowflakes are always materialized
- For large or non-filtering snowflakes, snowflake materialization may dominate the overall query time
- V8 - Controlled materialization
  - Materialization only done when adequate filtering is expected
  - When no materialization is done, indexes of the underlying tables making up the snowflake can be used

Better selection of filtering dimensions

- Improved cost estimation algorithm that better estimates the filtering effect of dimensions

10.11.7 Other star join enhancements

Here we discuss some other enhancements related to the star join configuration.

Snowflake handling - Controlled materialization

Prior to DB2 V8, all snowflakes were materialized. This provided the benefit of simplified access path selection by reducing the overall number of tables joined.

For the inside-out join phase (post fact table), relatively small snowflakes, or snowflakes that provide adequate filtering are good candidates to be materialized. With the introduction of in-memory workfiles and/or sparse index on workfiles, the snowflake, which may contain many tables, is resolved once and fact table rows are joined to a much smaller result set using an efficient join method that can take advantage of the in-memory or sparse index.

For large or non-filtering snowflakes, the materialization overhead may dominate the overall query time, and is therefore detrimental to query performance. For in-memory workfile and sparse index on workfile, the result must be sorted to allow a binary search to locate the target row. Sorting a large result can be expensive. If the memory is available for a very large result, the binary search for in-memory workfiles may result in multiple iterations to find the target row. If fallback occurs to sparse index, then the index may be too sparse, and therefore each locate in the workfile may still require a large sequential scan.

V8 introduces controlled materialization. The filtering of each snowflake is ranked, and only those snowflakes that provide adequate filtering compared to the base table size will be materialized.

The choice not to materialize can overcome the sort and workfile allocation overhead, and rather than requiring an index to be built on the workfile, the indexes on the underlying snowflake tables can be used for efficient joins after the fact table.

Selection of filtering dimensions

Besides the star join enhancements already described, DB2 V8 provides an improved cost estimation algorithm that better estimates the filtering effect of dimensions. This results in a better table join sequence (which tables are processed during outside-in and which tables are processed during inside-out processing) and can yield a significant performance improvement.

Example of a Star Join Access Path

Query#	Pln#	Corr Name	Table Name	Join Mtd	Join Type	Acc Type	Access Name	Sort New
11001	1	DP	/BI0/D0SD_C01P	0	S	I	/BI0/D0SD_C01P~0	N
11001	2	DT	/BI0/D0SD_C01T	1	S	T		Y
11001	3	DU	/BI0/D0SD_C01U	1	S	T		Y
11001	4	F	/BI0/F0SD_C01	1	S	I	/BI0/F0SD_C01~0	N
11001	5	D5	/BI0/D0SD_C015	1		I	/BI0/D0SD_C015~0	N
11001	6	D3	/BI0/D0SD_C013	1		I	/BI0/D0SD_C013~0	N
11001	7		DSN_DIM_TBLX(02)	1		T		Y
11001	8	D2	/BI0/D0SD_C012	1		I	/BI0/D0SD_C012~0	N

Access_type T indicates either "sparse index" or "data caching" is used
The final decision is taken at runtime and cannot be shown by EXPLAIN

10.11.8 Example of a star join access path

You can determine if the in-memory workfiles or sparse index enhancement is used from the PLAN_TABLE. The visual shows an example of a pushdown star join access path. When you see ACCESS_TYPE = 'T', this indicates that either data caching or a sparse index is used for this query. The decision to use data caching or sparse index is made at execution time, when the work file data is sorted. Therefore the EXPLAIN output in the PLAN_TABLE cannot show which one of these features is actually used when running the query.

REOPT(ONCE)

Bind option that controls when the Optimizer builds the access path information for dynamic SQL applications

- By default, access path is built at PREPARE
- REOPT(ONCE)
  - Defers access path selection until OPEN
  - Values of host variables on OPEN are used to build access path
  - Resulting access path is cached in the global prepared cache

New terminology

- REOPT(NONE) equivalent to NOREOPT(VARS)
- REOPT(ONCE)
- REOPT(ALWAYS) equivalent to REOPT(VARS)

10.12 REOPT(ONCE)

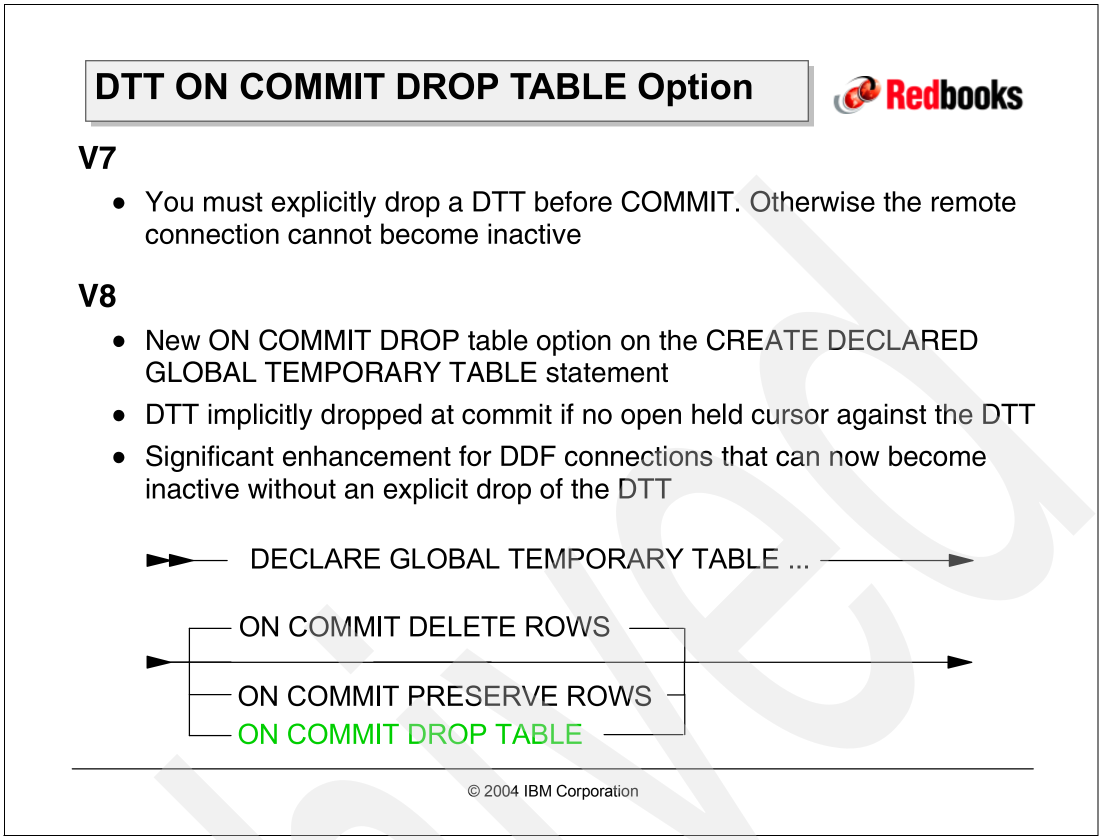
REOPT(ONCE) is a new bind option that tries to combine the benefits of REOPT(VARS) and dynamic statement caching. For an SQL statement with input host variables, static or dynamic, the access path chosen by the optimizer during bind time (before the values of host variables are available) may not always be optimal.

The bind option, REOPT(VARS), solves this problem by (re)preparing the statement at run time when the input variable values are available, so that the optimizer can re-optimize the access path using the host variable values. However, for frequently called SQL statements that take very little time to execute, re-optimization using different input host variable values at each execution time is expensive, and it may affect the overall performance of applications.

The idea of REOPT(ONCE) is to re-optimize the access path only once (using the first set of input variable values) no matter how many times the same statement is executed. The access path chosen based on the set of input variable values is stored in the dynamic statement cache and used for all later executions (as with normal dynamic statement caching). This solution is based on the assumption that the chosen set of host variable values at run time are better than the default ones chosen by optimizer at bind time.

Three new options, REOPT(ONCE), REOPT(NONE), and REOPT(ALWAYS) can be specified in the BIND and REBIND commands for plans and packages. REOPT(NONE) is the default option. NOREOPT(VARS) is a synonym for REOPT(NONE). REOPT(VARS) can be specified as a synonym for REOPT(ALWAYS). REOPT(ONCE) is valid only in the new-function mode.

REOPT(ONCE) only applies to dynamic SQL statements and is ignored if you use it with static SQL statements.DB2 for z/OS caches only dynamic statements. If a dynamic statement in a plan or package that is bound with REOPT(ONCE) runs when dynamic statement caching is turned off (DSNZPARM CACHEDYN=NO), the statement runs as if REOPT(ONCE) is not specified.



10.13 DTT ON COMMIT DROP TABLE option

Prior to Version 8, a declared global temporary table (DTT) is treated like a base table with an open held cursor. So, a remote connection is not eligible to become inactive (and the thread being returned to the pool) at commit time.

Scrollable cursors use DTTs. Therefore, connections that use scrollable cursors that are open are not eligible to become inactive at a COMMIT time. (Note that when you CLOSE a scrollable cursor, the DTT gets cleaned up.)

DTTs are created as “pseudo” release at deallocate structures. Any data inserted into the DTT is removed at commit (unless it is created with ON COMMIT PRESERVE ROWS). The DTT remains active across a commit. The DTT must be dropped explicitly if the user wants it to go away at commit. If the thread reuse does not go through a "New User" process (for example, a DDF inactive connection), then the DTT does not go away until the thread is terminated and deallocated. The lock on the TEMP table space remains as long as the DTT exists.

This has caused a number of problems in the past.

For example, when a Java bean accesses a stored procedure that creates and populates a DTT and passes the values of that temp table as a result set back to the JAVA program, after the bean issues the `close.statement`, `close.resultset`, and `close.connection`, the locks on the TEMP table spaces are left in place, even with AUTOCOMMIT ON, and DB2 explicitly closes everything, the cursor, the result set, the connection.

Any application that uses connection pooling will always leave all temporary tables out there for the duration the Web application server is up. And all locks on that table will be preserved for this entire duration, even past COMMITS.

In DB2 for z/OS V8, you can define a declared global temporary table with a new option ON COMMIT DROP TABLE, so the application does not have to explicitly delete the DTT before committing.

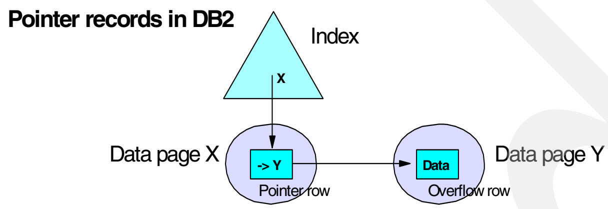
This change is a significant change for distributed work that can now be switched from active to inactive. Temporary tables that do not have any HELD cursors open are dropped automatically at COMMIT. This will allow the connection to be switched to INACTIVE. When the DTT is defined using the ON COMMIT DROP TABLE clause, the declared global temporary table is implicitly dropped at COMMIT if there are no open cursors on the table that are defined as WITH HOLD.

Note that you will have to change your applications to take advantage of this feature, as a DTT is defined within the program to specify the ON COMMIT DROP TABLE clause.

Locking Enhancements

Redbooks

Locking enhancement for relocated rows



Isolation level **Cursor Stability** with **CURRENTDATA NO or YES**	V7	V8
Lock and unlock request for the **pointer** record	YES	YES
Lock and unlock request for the **overflow** record	YES	NO

Lock avoidance for singleton select when using ISO(CS) and CURRENTDATA(YES)

10.14 Locking enhancements

This section describes locking related enhancements.

10.14.1 Eliminate locking of overflow records for programs using ISO(CS)

When you update a variable length row on a page, it may be that after the update, the row no longer fits on its original page. In that case, the new row is stored on a different page. In order to avoid updating all the indexes that point to that row (remember that an index entry contains a RID, which contains the page number), a pointer record is created on the original page. The pointer record then points to the actual row.

If a later update of the row decreases its size, it can be put back into its original (home) page, again without any updates to the index.

You can have variable length rows when:

- Using variable length fields like VARCHAR and VARGRAPHIC
- Using DB2 data compression
- Altering a table to add a column, but you have not performed a REORG to materialize all the rows to have the new column
- Using the new V8 online schema enhancements to enlarge the a columns data type, but you have not ran REORG to materialize all the rows to the latest format

The good thing about using the overflow pointer is that we avoid updating the index(es). The disadvantage is that we potentially double the number of data I/Os and getpage requests, and increase the number of lock and unlock requests.

You can check whether or not overflow records exist by looking at the FARINDREF and NEARINDREF information in SYSIBM.SYSTABLEPART. This information is updated by running RUNSTATS and provides information about the number of rows that have been relocated far (> 16 pages) or near(< 16 pages) their "home" page. It is recommended to run a REORG when (FARINDREF +NEARINDREF) exceed 5 to 10%.

To reduce the amount of overflow rows, and hence the number of double I/Os, getpages, and lock/unlock requests, you can use a higher PCTFREE value (FREEPAGE will not help in this case).

In V7, there is no lock avoidance on both the pointer record (or page when using page locking) and the overflow record (or page) itself. With DB2 Version 8, only the pointer (or page)is locked (Table 10-1).

Table 10-1 Lock avoidance of overflow record

Isolation Cursor Stability CURRENTDATA No or Yes	Version 7	Version 8
Lock and unlock for the pointer record	YES	YES
Land and unlock for the overflow record	YES	**NO**

In a V8 data sharing environment, this enhancement is only enabled when using row level locking.

10.14.2 Lock avoidance for singleton select using ISO(CS) and CD(YES)

Semantically, a program that executes a SELECT INTO statement and is bound with ISOLATION(CS) does not need to obtain an S page or row lock (provided the data is committed), since only a single row can be returned. This is the case regardless of the setting of the CURRENTDATA(CD) bind option. The cursor that DB2 uses under the cover to process the singleton select is closed after the SELECT is processed.
Before this enhancement, lock avoidance only occurs for plans and packages using ISO(CS) and CD(NO). Lock avoidance did not occur for programs bound with ISO(CS) and CD(YES). With this enhancement, introduced by APAR/PTF PQ89070/UQ91470, DB2 will use lock avoidance for singleton select statements bound with ISO(CS) irrespective of the CURRENTDATA bind option.

Generally speaking, if DB2 can determine that the data that it is reading is committed, DB2 can avoid taking the lock for plans and packages bound with ISO(CS) in the following cases:

- For rows that do not satisfy the search condition, lock avoidance is possible with CURRENTDATA(YES) or CURRENTDATA(NO).
- For a row that satisfies the search condition on a *singleton SELECT*, lock avoidance is possible with CURRENTDATA(YES) or CURRENTDATA(NO).
- For rows that satisfy the search condition from a CURSOR, lock avoidance is only possible when you use the CURRENTDATA(NO) bind option.

In a lab environment, a test case, executing a simple select statement 100,000 times in a loop, showed up to 25% better elapsed time in V8 when compared to a run without lock avoidance. When compared to V7, V8 performs 7% better with this lock avoidance enhancement.

Non-correlated EXISTS subquery

Prior to V8, ALL qualifying rows of a non-correlated EXISTS subquery are retrieved and stored in a workfile

- Potentially a lot of processing to find all qualifying rows for the subquery
- Potentially a lot of workfile space to store all these rows

DB2 V8 stops evaluating the non-correlated EXISTS subquery as soon as ONE qualifying row is found

- One qualifying row is sufficient to evaluate the predicate as true
  - Example:

```
SELECT EMPNO,LASTNAME
  FROM DSN8810.EMP
  WHERE EXISTS (
          SELECT *
            FROM DSN8810.PROJ
            WHERE PRSTDATE > ’2005-01-01’);
```


10.15 Non-correlated EXISTS subquery enhancement

When you use the keyword EXISTS, DB2 simply checks whether the subquery returns one or more rows. Returning one or more rows satisfies the condition; returning no rows does not satisfy the condition.

For example, list all employees, if any project that is represented in the project table has an estimated start date that is later than 1 January 2005:

```
SELECT EMPNO,LASTNAME
  FROM DSN8810.EMP
  WHERE EXISTS (
                SELECT *
                  FROM DSN8810.PROJ
                  WHERE PRSTDATE > ’2005-01-01’);
```

Because this example is using a non-correlated subquery, the result of the subquery is always the same for every row that is examined for the outer SELECT. Therefore, either every row appears in the result of the outer SELECT, or none appears.

Prior to DB2 Version 8, all qualifying rows are retrieved and stored in a workfile. If many rows qualified, that could mean a lot of work and workfile space. However, the EXISTS predicate is not interested in the actual qualifying values from the subselect. It only wants to know if any rows qualify.

In V8, DB2 will only find the first qualifying row for the non-correlated EXISTS subselect. If DB2 finds one, the predicate is true and DB2 stops processing the subselect. Depending on the number of qualifying rows in the subselect and the amount of work it takes to find all qualifying rows, this enhancement can be a big performance boost.

IN List Processing Enhancements

Redbooks

V7 Enhancements to IN list processing

- Predicate pushdown for IN list predicates (V7 APAR PQ73454)
- Correlated subquery transformation enhancement (V7 APAR PQ73749)
- Activated by INLISTP DSNZPARM (default value 0 in V7 = disabled)

By default enabled in V8 (INLISTP=50)

- IN list predicate pushdown into nested table expression (NTE) or materialized view (MV)
  - Better filtering inside NTE and MV ; fewer resulting rows
  - Potential index usage on columns when resolving NTE or MV
- Correlated subquery transformation enhancement
  - IN list predicates generated by predicate transitive closure "pulled up" to parent query block
    - ➡ Filtering can be done at parent level, resulting in fewer invocations of subquery executions
    - ➡ IN list predicate on parent query block can take advantage of existing indexes and provide a better access path for parent query block

10.16 IN list processing enhancements

DB2 Version 7 has introduced numerous enhancements to IN-list processing, some of which are mentioned in the following sections:

- Predicate pushdown for IN list predicates (V7 APAR PQ73454)
- Correlated subquery transformation enhancement (V7 APAR PQ73749)
- Select INLIST improvements (V7 APAR PQ68662)

The reason for mentioning them here is that they are not well known, as well as the fact that some of them are activated via a DSNZPARM called INLISTP. The default value in V7 for INLISTP was 0 (zero), which means that the feature is not active. In V8, the default value is INLISTP=50, which means that the feature is now active by default. (This also expresses confidence in these new features, as they are now enabled by default.)

Predicate pushdown for IN list predicates (V7 APAR PQ73454)

Performance problems have been observed with complex vendor generated queries involving IN list predicates and materialized views or table expressions. This performance problem was due to the large workfile that resulted from a materialized view or table expression. If qualified IN list predicates can be pushed down into the materialized view or table expression, the materialized result set becomes much smaller. (This enhancement is activated through the new DSNZPARM INLISTP).

Example 10-2 demonstrates the effect of this change.

Example 10-2 INLIST predicate pushdown

```
SELECT *
         FROM (
                 SELECT * FROM T1
                UNION
                 SELECT * FROM T2
              ) X(C1,C2)
        WHERE X.C1 IN ('AAA','CDC','QAZ');

--The transformed query is equivalent to the following:

SELECT *
         FROM (
                 SELECT * FROM T1 WHERE T1.C1 IN ('AAA','CDC','QAZ')
                UNION
                 SELECT * FROM T2 WHERE T2.C1 IN ('AAA','CDC','QAZ')
              ) X(C1,C2)
        WHERE X.C1 IN ('AAA','CDC','QAZ');
```

This transformation can result in an order of magnitude of performance improvement in some cases, due to:

- Filtering at the early level to reduce the intermediate workfile size
- Possible exploitation of indexes on T1 and/or T2

Overall query performance should improve after setting the INLISTP parameter.

Correlated subquery transformation enhancement (V7 APAR PQ73749)

Other performance issues have been observed with complex vendor generated queries involving IN list predicates in correlated subqueries. These IN list predicates are constructed in such a way that the elements are drawn in from a previously constructed table, that is, the size of the list for a constructed IN list predicate depends on the size of such a table.

Allowing a correlated IN list predicate to be generated by the transitive closure process, and pulling the generated IN list predicate to the parent query block, can improve performance. The addition of the IN list predicate to the parent query block allows filtering to occur earlier and results in a reduction in the number of subquery executions. Additional performance improvements may be achieved if the IN list predicate on the parent query block allows a better access path as a consequence of indexability of the IN list predicate.

The following conditions must all be met in order for an IN list predicate to be generated from transitive closure and bubbled up from the subquery to its parent query block.

- Boolean term predicates, COL1 IN (Lit1, Lit2,...) and COL1 = COL2, both appear in the query.
- This subquery is on the right-hand side of a Boolean term predicate of type EXISTS, IN, or ANY.
- COL2 is correlated (to the parent query block or some ancestor of the parent query block).
- Parent query block in inner join, a single base table, or a view. This feature is disabled if the parent query block is an outer join.
- The subquery does not contain COUNT or COUNT_BIG.

An IN list predicate of more than one literal satisfying all of the above conditions will be generated in the parent query block and will not appear in the subquery. Other predicates that used to participate in transitive closure (including singleton IN list) will still be generated in the subquery as before; if these predicates can be bubbled up, then they will appear in both the parent query block and in the subquery.

Let us look at Example 10-3 in order to show the behavior prior to this enhancement.

Example 10-3 IN list query before transformation

```
SELECT *
   FROM A
  WHERE (EXISTS (SELECT 1
                  FROM B,C
                 WHERE B1 = 5
                   AND B1 = C1
                   AND C1 = A.A1
                   AND B2 IN (10, 20)
                   AND B2 = A.A2
               )
        ) ;
```

The SQL statement in Example 10-3 will generate the following three predicates within the EXISTS subquery:

```
C1 = 5
A.A1 = 5
A.A1 = B1
```

This enhancement, combined with an appropriate setting for the INLISTP DSNZPARM value, will cause a qualified IN list and equal predicates to be candidates for cross query block transitive closure process. This process, when conditions allow, will result in an IN list predicate being deduced by transitive closure and generated in the parent query block rather than in the subquery.

The predicates generated, at the parent query level, by this process are:

```
A.A1 = 5
A.A2 IN (10,20)
```

This is equivalent to the transformation of the query shown in Example 10-4.

Example 10-4 Transformed IN list query

```
SELECT *
         FROM A
        WHERE
              A.A1 = 5
          AND A.A2 IN (10,20)
          AND (EXISTS (SELECT 1
                        FROM B,C
                       WHERE B1 = 5
                         AND B1 = C1
                         AND C1 = A.A1
                         AND B2 IN (10, 20)
                         AND B2 = A.A2
                     )
              ) ;
```

INLISTP installation parameter

It is evident that strategic placement of IN list predicates in an appropriate query block, whether pushing down or pulling out, is an extremely effective way to achieve better performance for some queries. These optimizations are focused on popular vendor generated queries, so such performance improvement are expected to benefit a broad set of customers. The effectiveness of these optimizations have already been tested at the IBM Teraplex site, as well as at a customer site.

The reasons for proposing this system parameter INLISTP with respect to IN list predicate optimization are as follows:

- Allowing customer to tune the INLISTP value to their workload
- Prevention of SQLCODE -101
- Prevention of potential degradation in the access path for some queries

In DB2 V8, the default value for INLISTP is 50. The parameter INLISTP will remain as a hidden keyword installation parameter.

Notice that for the predicate pushdown optimization, the INLISTP value of 1 will have the same effect as the value of 0. This is because an IN list predicate of a single element is transformed into an equal predicate internally, and a qualified equal predicate is already being pushed down into a materialized view or a table expression. However, for the cross query block transitive closure optimization, the INLISTP value of would have a different effect than the value of 0. This is because no predicate has ever been considered for a cross query block transitive closure. For a positive INLISTP value, equal predicates, in addition to IN list predicates of more than 1 element, will also be considered as candidates to be pulled out to the parent query block position.

Select INLIST improvements (V7 APAR PQ68662)

This enhancement does not change any behavior in DB2 V8. It is mentioned here merely because it is not well known that this enhancement was introduced in V7.

The problem that is addressed here is poor access path selection on a single table query with IN predicate on index columns. The optimizer struggles between I-scan and R-scan and as the number of entries in the INLIST increases the query will tend to select the R-scan. This enhancement improves the index scan cost model to more accurately reflect the chance that the needed index or data page may already be cached in the buffer pool when the last INLIST item is processed.

Using dynamic prefetch during IN list index access to for non-contiguous data

This is another IN list processing enhancement introduced in V6 and V7, via APAR PQ71925.

When index access is used with an IN-list predicate and the list items are scattered in the index key column's domain, the index access may not be able to benefit from sequential prefetch.

This enhancement turns off sequential prefetch at bind time so that runtime can kick off dynamic prefetch whenever a sequential pattern is detected. When doing an SQL EXPLAIN, the PLAN_TABLE may contain a new value in the PREFETCH field. 'D' may be observed in the PREFETCH field, meaning that dynamic prefetch is expected by the DB2 optimizer.

DSC Short Prepare Enhancements

Redbooks

V7 Problem

- When a matching statement is found in cache, it is copied into thread related storage before it can be used
- If no storage a GETMAIN is performed
- To keep size of local pool limited initiate storage contraction regularly
- **GETMAIN/FREEMAIN make up large part of cost of short prepare**

V8 Solution

- Use a number of shared storage pools (assigned via hashing to minimize contention) to copy statements in
- Potentially fewer storage required (not each thread needs its own pool)
- Using "best fit" algorithm to find space in assigned pool
- Result: Fewer GETMAIN/FREEMAIN operations, faster short prepare

10.17 Faster Dynamic Statement Cache short prepares

When global caching is active (DSNZPARM CACHEDYN=YES), DB2 maintains “skeleton” copies of each prepared statement in the EDM pool. This storage is not an issue for the short prepare focus. However, whenever a thread issues a prepare, this indicates that thread is going to need its own copy of the prepared statement, in order to execute the statement.

When a thread finds a previously executed identical statement in the global cache, DB2 acquires storage and makes a copy of the statement for that thread. Prior to DB2 Version 8, this storage comes from a thread-based storage pool. So each thread that gets anything from the cache will have one of these pools.

In a system with many long-running threads, such as an SAP system, these pools can get quite large and fragmented if left uncontrolled. The current design (prior to DB2 V8) “contracts” the pool quite frequently, almost at every commit. The contraction logic frees most of the unused space back to the OS, keeping the size to a minimum. This means that when a thread performs a prepare after a commit, DB2 most probably has to go back to the OS to GETMAIN a new piece of storage for the pool, so it has somewhere to put the statement. This effectively means DB2 has to perform a GETMAIN and FREEMAIN pair for almost every prepare. Measurements showed that this was the dominant cost of the short prepare path (although the FREEMAIN part really happens at commit).

The new approach aims to virtually eliminate the cost of the OS level GETMAINs and FREEMAINs by going to a centralized storage approach. With this approach, DB2 will use a number of storage pools that are owned by the system, not a particular thread. To avoid latch contention, the new implementation uses a fixed number of pools.

When a new piece of storage is needed, a hash function is used, to “randomly” assign a pool. The hash uses the statement identifier as input. Each of the pools is a “bestfit” pool and it extends its size as needed, in 1 MB chunks. With this approach, we rely on the best fit logic to keep the pools at a minimum size. With the thread-based storage model, further reductions in contractions would have lead to some storage increase.

Example

Assume we have three threads. Sequentially, that is, at discrete different times, each one prepares and executes a statement that is 10K in size, then it commits, freeing the storage back to the pool.

With the thread-based approach, at the end of the sequence, each thread will have 10 KB allocated for a total of 30 KB. With the centralized approach, the storage would get reused at each thread’s statement prepare, so there would be only ever be 10 KB allocated to the pool. If the three threads’ statement executions in fact do not occur at different times, there are still benefits. Assume, for example, that all three executed concurrently. The threads will have used 30 KB in the centralized pool as well. But assume that after they all commit, one thread subsequently prepares and executes a statement that is between 10 KB and 30 KB in size. This could be satisfied from the existing centralized pool. But with the previously used thread-based pool, DB2 would need to GETMAIN a new 10 - 30 KB chunk to its unique pool.

The centralized approach may use as much storage as the thread-based approach in the worst case, but most likely the timing will be such that it will use much less. The major benefit is then the removal of the majority of OS calls to GETMAIN and FREEMAIN storage, and this benefit occurs regardless of the level of storage usage improvement.

Another benefit occurs in the case of short-running threads. When a thread that connects, prepares, and executes only a few statements, then ends, the centralized approach is clearly better. In this case the thread does not need to create its storage pool or GETMAIN any storage. The central pool is highly likely to already contain sufficient free storage with the normal coming and going of other activity to satisfy this small number of executed statements.

The more centralized management of the available storage means that we can use the available storage to the best advantage among all threads. Idle threads will also not be holding storage that they are not using, leading to more efficient usage of storage resources, and reduced induced overhead in a busy system.

EXPLAIN Enhancements

EXPLAIN STMTCACHE

- Describe the access path of a statement in the DSC in the PLAN_TABLE

EXPLAIN via a stored procedure

- DSN8EXP allows you to EXPLAIN statements without the privilige to execute the statement
- Also available in DB2 V7
- Can be invoked by Visual Explain

New PLAN_TABLE columns

PLAN_TABLE access via an alias

Completely redesigned Visual Explain

10.18 Explain enhancements

DB2 V8 makes a number of changes to the EXPLAIN function, greatly enhancing the ways to obtain and look at DB2 access path information.

10.18.1 EXPLAIN STMTCACHE

The STMTCACHE option of the EXPLAIN statement allows you to obtain information about the current access path being used by statements in the global dynamic statement cache. For more information, see 4.20, “EXPLAIN STMTCACHE” on page 368.

10.18.2 EXPLAIN using a stored procedure

The PTFs for APARs PQ90022 and PQ93821 introduce a stored procedure, DSN8EXP, that allows you to invoke the EXPLAIN statement using a stored procedure. The main purpose is to allow you to run the SQL EXPLAIN statement against objects for which you do not have the authority to execute SQL statements against. Remember that to run an EXPLAIN, you need the necessary authorization to execute the SQL statement you are explaining. For example, a developer may have responsibilities for SQL performance on a payroll application, but may not have SELECT, INSERT, UPDATE, and DELETE access to the object, because the data is very sensitive. The developer now can use the stored procedure to execute an EXPLAIN against the object. The developer does not need access to the actual objects referenced in the statement being explained, but only needs the authority to execute the stored procedure.

This stored procedure can also be invoked by Visual Explain. For more information about the enhanced Visual Explain, see 10.19, “Visual Explain enhancements” on page 823.

New PLAN_TABLE Columns

TABLE_ENCODE CHAR(1)

Indicates the encoding scheme of the table. If the table has a single CCSID set, then the column will contain 'E' for EBCDIC 'A' for ASCII or 'U' for Unicode. If the table contains multiple CCSID sets, then the column will be set to 'M' for multiple CCSID sets.

TABLE_SCCSID FIXED(16)

The SBCS CCSID value of the table. If the TABLE_ENCODE column is 'M', the value is zero.

TABLE_MCCSID FIXED(16)

The Mixed CCSID value of the table. If the TABLE_ENCODE column is 'M', the value is zero.

TABLE_DCCSID FIXED(16)

The DBCS CCSID value of the table. If the

TABLE_ENCODE column is 'M', the value is zero.

ROUTINE_ID INTEGER

Used to pinpoint the table function record from SYSIBM.SYSROUTINES.

10.18.3 PLAN_TABLE changes

The visual above shows the new columns that have been added to the plan_table.

In addition to that, the explanation of two columns has been changed in order to be able to store MQT related information. Those columns are:

TNAME The name of a table, materialized query table, created or declared temporary table, materialized view, table expression, or an intermediate result table for an outer join that is accessed in this step, blank if METHOD is 3.

TABLE_TYPE M for materialized query table.

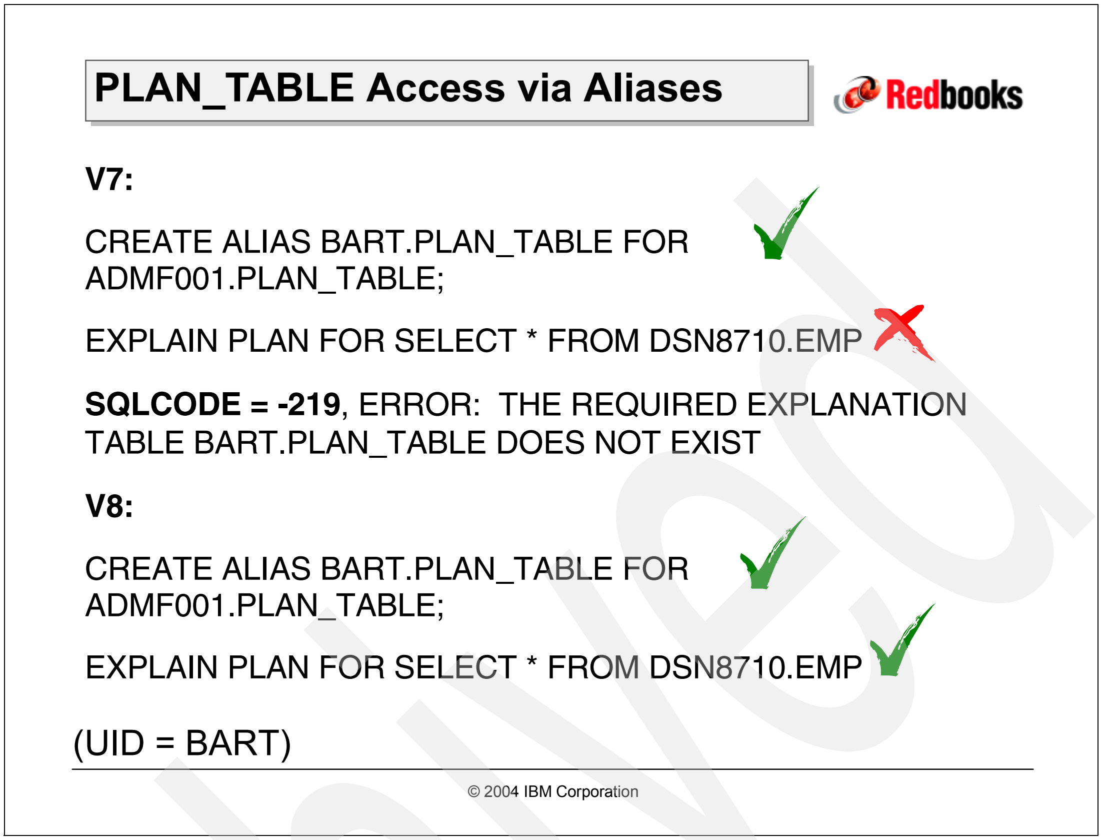
If automatic query rewrite occurs and the final query plan comes from a rewritten query, the PLAN_TABLE shows the name of the matched MQTs and the access path using the MQTs. The TABLE_TYPE column of the PLAN_TABLE for the MQT rows show value ‘M” for an MQT. For more information about MQTs, refer to 10.2, “What is a materialized query table?” on page 700.

You can use the CREATE TABLE statement shown in Example 10-5 to create the V8 PLAN_TABLE.

Example 10-5 DB2 V8 PLAN_TABLE definition

```
CREATE TABLE userid.PLAN_TABLE (
  QUERYNO INTEGER NOT NULL,
  QBLOCKNO SMALLINT NOT NULL,
  APPLNAME CHAR(8) NOT NULL,
  PROGNAME VARCHAR(128) NOT NULL,
  PLANNO SMALLINT NOT NULL,
```

```
   METHOD SMALLINT NOT NULL,
   CREATOR VARCHAR(128) NOT NULL,
   TNAME VARCHAR(128) NOT NULL,
   TABNO SMALLINT NOT NULL,
   ACCESSTYPE CHAR(2) NOT NULL,
   MATCHCOLS SMALLINT NOT NULL,
   ACCESSCREATOR VARCHAR(128) NOT NULL,
   ACCESSNAME VARCHAR(128) NOT NULL,
   INDEXONLY CHAR(1) NOT NULL,
   SORTN_UNIQ CHAR(1) NOT NULL,
   SORTN_JOIN CHAR(1) NOT NULL,
   SORTN_ORDERBY CHAR(1) NOT NULL,
   SORTN_GROUPBY CHAR(1) NOT NULL,
   SORTC_UNIQ CHAR(1) NOT NULL,
   SORTC_JOIN CHAR(1) NOT NULL,
   SORTC_ORDERBY CHAR(1) NOT NULL,
   SORTC_GROUPBY CHAR(1) NOT NULL,
   TSLOCKMODE CHAR(3) NOT NULL,
   TIMESTAMP CHAR(16) NOT NULL,
   REMARKS VARCHAR(254) NOT NULL,
   PREFETCH CHAR(1) NOT NULL WITH DEFAULT,
   COLUMN_FN_EVAL CHAR(1) NOT NULL WITH DEFAULT,
   MIXOPSEQ SMALLINT NOT NULL WITH DEFAULT,
   VERSION VARCHAR(64) NOT NULL WITH DEFAULT,
   COLLID VARCHAR(128) NOT NULL WITH DEFAULT,
   ACCESS_DEGREE SMALLINT ,
   ACCESS_PGROUP_ID SMALLINT ,
   JOIN_DEGREE SMALLINT ,
   JOIN_PGROUP_ID SMALLINT ,
   SORTC_PGROUP_ID SMALLINT ,
   SORTN_PGROUP_ID SMALLINT ,
   PARALLELISM_MODE CHAR(1) ,
   MERGE_JOIN_COLS SMALLINT ,
   CORRELATION_NAME VARCHAR(128) ,
   PAGE_RANGE CHAR(1) NOT NULL WITH DEFAULT,
   JOIN_TYPE CHAR(1) NOT NULL WITH DEFAULT,
   GROUP_MEMBER CHAR(8) NOT NULL WITH DEFAULT,
   IBM_SERVICE_DATA VARCHAR(254) NOT NULL WITH DEFAULT,
   WHEN_OPTIMIZE CHAR(1) NOT NULL WITH DEFAULT,
   QBLOCK_TYPE CHAR(6) NOT NULL WITH DEFAULT,
   BIND_TIME TIMESTAMP NOT NULL WITH DEFAULT,
   OPTHINT VARCHAR(128) NOT NULL WITH DEFAULT,
   HINT_USED VARCHAR(128) NOT NULL WITH DEFAULT,
   PRIMARY_ACCESSTYPE CHAR(1) NOT NULL WITH DEFAULT,
   PARENT_QBLOCKNO SMALLINT NOT NULL WITH DEFAULT,
   TABLE_TYPE CHAR(1) ,
   TABLE_ENCODE CHAR(1) NOT NULL WITH DEFAULT,
   TABLE_SCCSID SMALLINT NOT NULL WITH DEFAULT,
   TABLE_MCCSID SMALLINT NOT NULL WITH DEFAULT,
   TABLE_DCCSID SMALLINT NOT NULL WITH DEFAULT,
   ROUTINE_ID INTEGER NOT NULL WITH DEFAULT
 IN database-name.table-space
```



10.18.4 Access the PLAN_TABLE via a DB2 alias

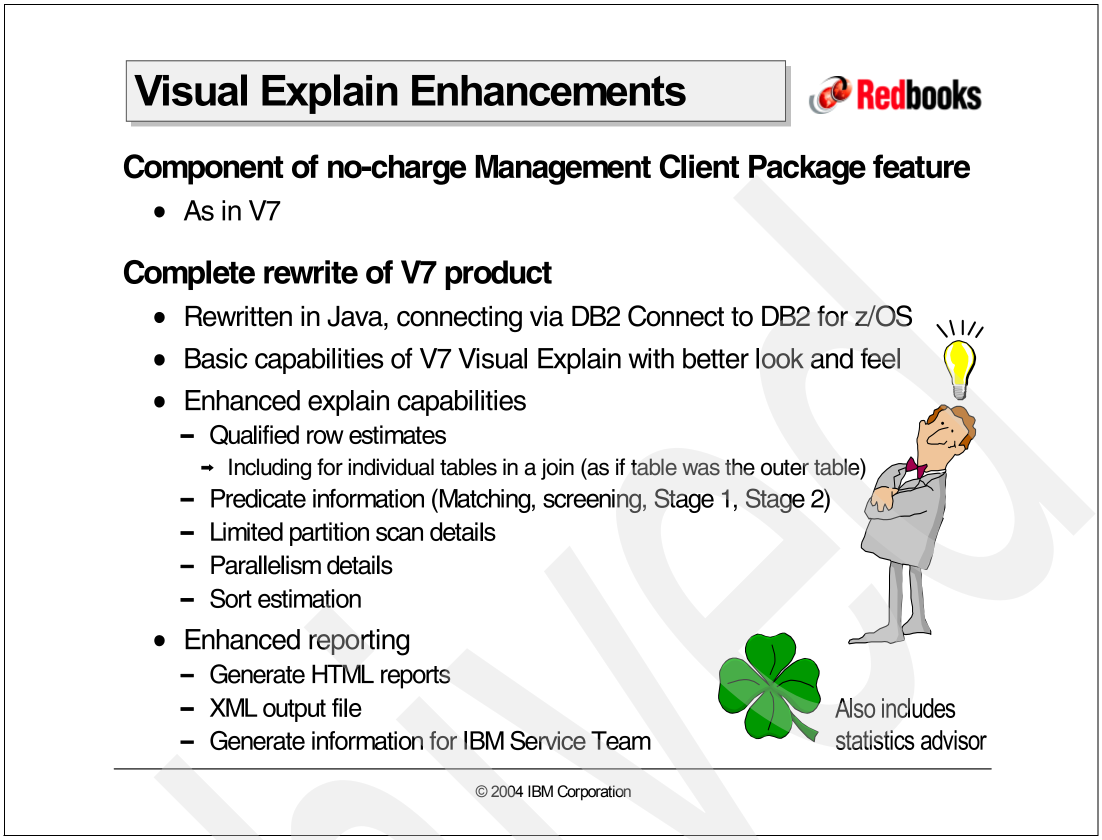
Up to V7, you never had the possibility to access explain tables with different OWNER and QUALIFIER names. (OWNER refers to the creator of the explain table; QUALIFIER refers to the user that issues the EXPLAIN command.) A limitation of our EXPLAIN command has been that only the owner of the explain tables can issue the EXPLAIN command to populate his/her tables. This has prevented you from being able to consolidate your plan tables under a single AUTHID.

Starting with V8, can use the ALIAS mechanism to populate explain tables created under a different AUTHID. The following external explain tables can now have aliases defined on them:

- PLAN_TABLE
- DSN_STATEMNT_TABLE
- DSN_FUNCTION_TABLE

As you can see from the visual above, prior to this change, when you used the alias feature, you got an error message saying that the required explain table does not exist.

The alias name must have the format userid.table_name where tablename can be PLAN_TABLE, DSN_STATEMNT_TABLE or DSN_FUNCTION_TABLE.



10.19 Visual Explain enhancements

For those of you that know the Visual Explain product that shipped as part of the no-charge Management Client Package with Version 7, the new Visual Explain that ships with Version 8 has almost nothing in common with the V7 product, other than its name. It is a complete rewrite. This does not mean that the V7 Visual Explain is not a useful product, but its Version 8 follow-on just does so much more.

First of all, it is still free, and comes as part of the no-charge Management Client Package feature with DB2 for z/OS Version 8. It can be used to explain queries against V7 and V8 systems, but you will only get to use the gems of the product with DB2 Version 8.

As mentioned before, the V8 Visual Explain is a completely re-write of the V7 product. It is now completely written in Java, and it is designed in such a way that it is easy to make new features available as plug-ins. It uses DB2 Connect to connect to DB2 for z/OS.

10.19.1 Overview of the new features in Visual Explain

First of all, the V8 Visual Explain has all the functions the “old” Visual Explain has. It allows you to:

- Explain SQL statements that you enter, and display the access path graphically.
- Explain SQL statement from the DB2 catalog, and display the access path graphically.
- Generate and print a report on those statements.
- Graphically display your DSNZPARMs (using the DSNWZP stored procedure).

However, it provides much, much, more detail about the access paths than you ever had before. When using Visual Explain against a DB2 Version 8 system, it not only uses the:

- PLAN_TABLE
- DSN_STATEMNT_TABLE
- DSN_FUNCTION_TABLE
- DB2 catalog statistics

It also uses a number of additional explain tables. Their content is not externalized and can change without notice, but the information is exploited by Visual Explain (VE) for your benefit. These tables contain a wealth of information that was not available to you in the past, and can be of great help when analyzing access path selection problems. They are automatically created by VE in a database/table space of your choice when enabling the tool.

The following new information is now available in Visual Explain:

- Single predicate filter factor estimates
- Determine whether a predicate is sargable (stage 1) or not (stage 2)
- Estimated number of rows at different stages during the query execution:
  - From a single table perspective (as though the table were the outer table)
  - How many rows are estimated after first n tables have been joined?
- At what time during the processing is a predicate applied? Is a predicate? For example:
  - A matching index predicate
  - An index screening predicate
  - A stage 1 non-indexable predicate
  - A stage 2 non-indexable predicate
- Limited partition scan details; what partitions will be accessed during the limited partition scan
- Index filter factor estimates:
  - The filter factor of matching index predicates
  - The total index filtering (matching and screening filter factors combined)
- Parallelism details:
  - Key range or page range partitioning
  - What each parallelism task does:
    - What pages are scanned by each task
    - What key ranges are scanned by each task
- Sort key length and sort data length:

  This information can be useful for input into formulas for sort pool sizing, and sort workfile sizing, where before you had to turn on a trace (IFCID 95 and 96) at run-time.

  - VE also provides the estimated number of records to be sorted.

In addition to displaying all this information on a screen, Visual Explain also allows you to generate reports, as html documents. In addition, you can also save your “explain analysis output” as an XML document for later reuse, or send to others for analysis, like your colleagues or IBM service people. When saved as an XML document, you can later reload the file and continue your interactive analysis.

When you suspect that the access path problem you are investigating is an actual DB2 problem (and not just a lack of statistics information that gives the optimizer incomplete information to determine the best access path), you normally open a problem ticket with IBM, and provide the necessary information in order for the service team to analyze the problem.

This is often a painful process of gathering, sending, gathering additional information or resending files. VE provides you with a “Service SQL” option, that gathers all the required information about your suspect query, and allows you to FTP it immediately to IBM for investigation, by pressing only a few buttons.

Describing all these enhancements in great detail is beyond the scope of this book. However, we want to give you some examples of how much more information the new Visual Explain provides on the following pages. If you want to learn more about Visual Explain, the tool comes with extensive help and a complete tutorial.

10.19.2 Setting up Visual Explain

As mentioned before, VE comes as part of the DB2 Management Client packages on a CD-rom. However, it can also be downloaded from the Web at:

`http://www.ibm.com/software/db2zos/dld.html`

After you install the VE code, you must do some initial setup:

- You must set up your connection to the DB2 system you want explain queries on.
- You must set up the VE explain tables.

Cataloging a DB2 system

When connecting to your DB2 systems, Visual Explain uses DB2 Connect. (As in V7, a restricted-use copy of DB2 Connect Personal Edition V8 for Windows is provided with the Management Client Package).

Instead of having to use the Configuration Assistant tool that comes with DB2 Connect, you can also set up your connection via Visual Explain. (Make sure the DB2 Connect packages have been bound previously to that DB2 system and that you are using a TCP/IP connection to the host, which most people are). The configuration is very simple and straightforward. All required information can be entered on a single window (see Figure 10-2).

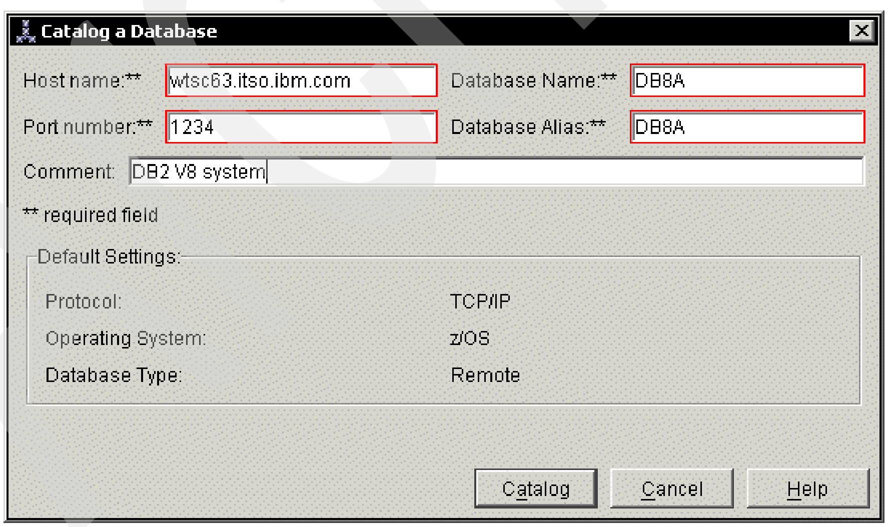


Figure 10-2 Catalog a database

Enabling Visual explain

After the connection is established, you have to enable VE. To do so, you must use the **Subsystem —> Enable Visual Explain** menu option (Figure 10-3).

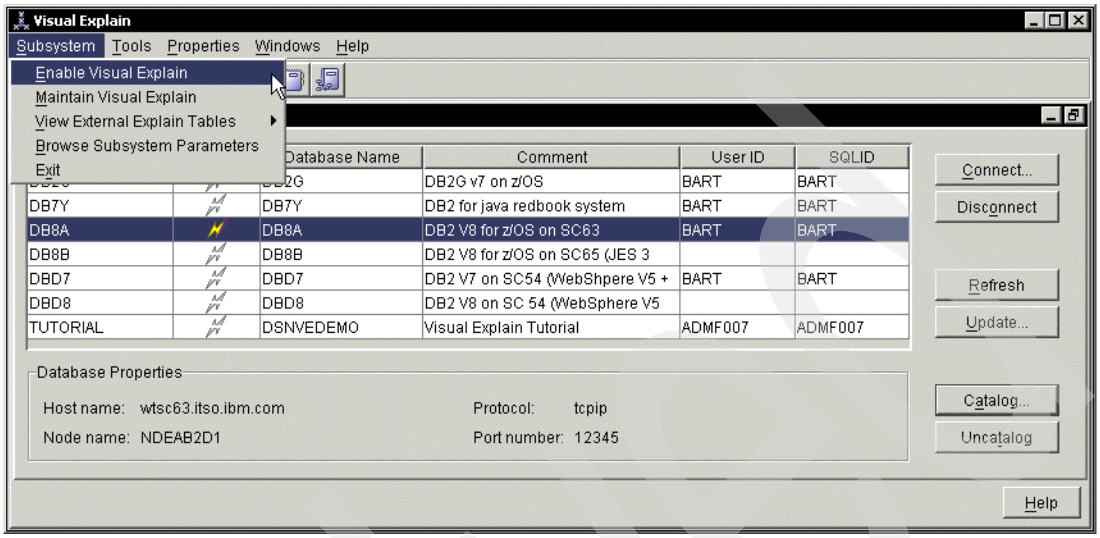


Figure 10-3 Enable Visual Explain

You are asked to provide a database and table space name, and table qualifier. During the enablement, VE creates the necessary explain tables that are used by the tool. In case, the database and table space does not exist, VE will ask whether you want to create them before continuing. Based on your input parameters, VE will then create these objects.

As you can see, you have some additional options within the subsystem menu. You can also use the option to **Maintain Visual Explain**. This option allows you to delete entries from the explain tables used by VE. The **View External Explain Tables** option allows you to look at rows in the external explain tables, PLAN_TABLE, DSN_STATEMNT_TABLE and DSN_FUNCTION_TABLE.

This is a good option if you are familiar with the “cryptic” output that the SQL EXPLAIN statement produces, but for most people, the graphical representation is much easier to understand. Lastly, you have the **Browse Subsystem Parameters** option, that shows all the values of the currently active DSNZPARM module. It uses the DSNWZP stored procedure to obtain this information (Figure 10-4).

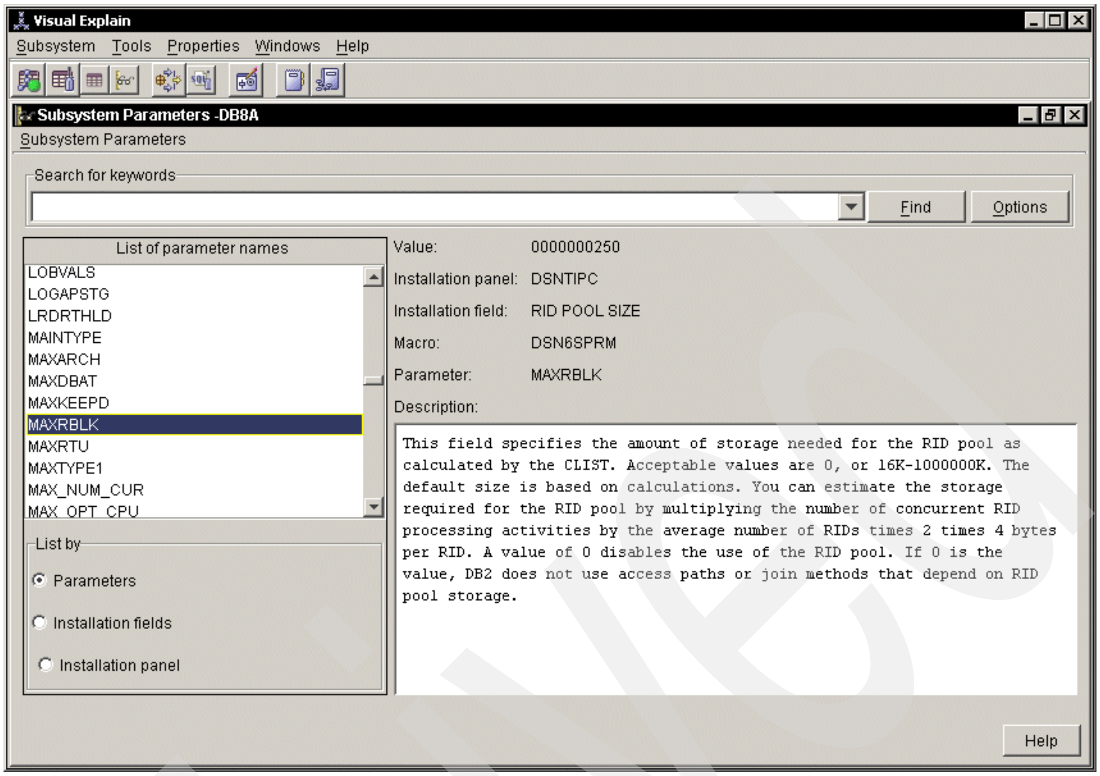


Figure 10-4 Display DSNZPARM

10.19.3 Entering SQL statements to analyze

Here you have two options: You can retrieve the SQL statements you want to explain from the catalog tables SYSSTMT and SYSPACKSTMT (option List Static SQL Statements), or you can type in your own statements (Tune SQL). Both are available from the Tools menu option.

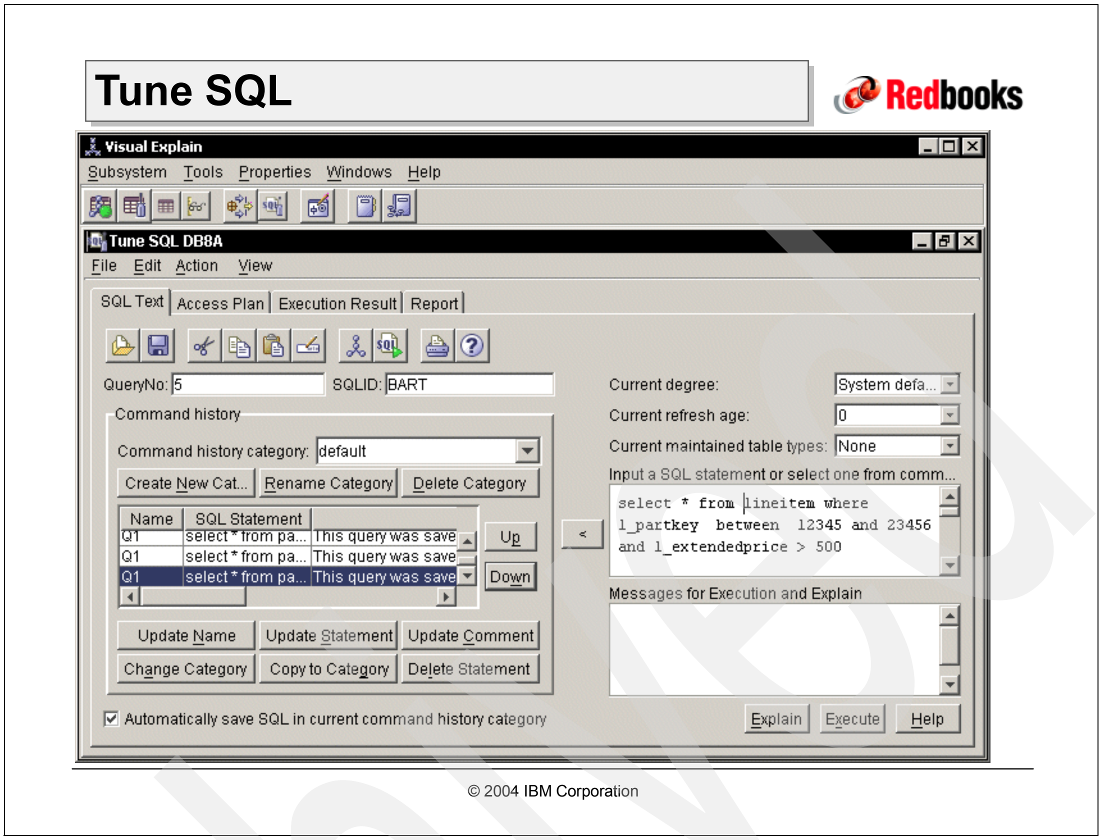
List Static SQL Statements

As mentioned before, this option allows you to list static SQL statements that are stored in the DB2 catalog. You have the option to list all statements (declares, open, fetch, etc.), or only the explainable SQL statements.

Listing all statements can be useful when analyzing problem SQL statements. An SQL trace often only contains the statement number of the executing statement that takes a lot of time, for example an OPEN statement, whereas you need to find the corresponding DECLARE CURSOR statement to find out what the root of the problem is.

In addition, remember that the columns in SYSPACKSTMT and SYSSTMT that contain the SQL statement text are marked FOR BIT DATA, and contain Unicode data, once you are in DB2 V8 new-function mode. This means that they are not automatically converted to the CCSID of the application's ENCODING BIND parameter. However, when using Visual Explain, VE takes care of this, and you can still read your SQL statement text information.

After retrieving the statements, you can explain them, provided they are explainable statements, of course.



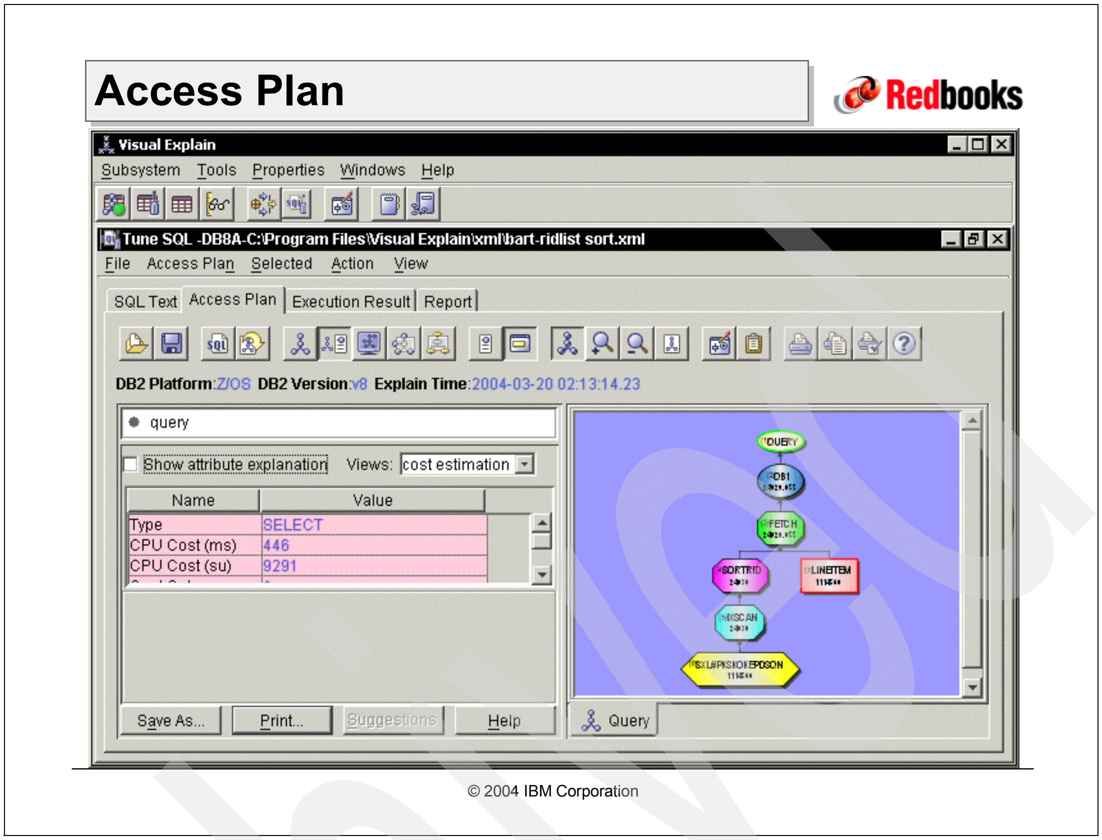
Tune SQL

This option allows you to type in your own statements, or import a saved SQL statement. You can then explain the statement by clicking the Explain button. You can also execute the statement, in case you would like to see the result.

Note that you can also specify:

- Current degree (for parallelism)
- Current refresh age (for MQTs)
- Current maintained table types (for MQTs)

When you click the Explain button, the tool starts demonstrating its true power.



10.19.4 Access plan

When VE is done explaining the statement and gathering all related statistics information, the result is displayed in the Access Plan pane as shown in the visual above.

On the right hand side, you see the graphical representation of the query's execution plan, the *access plan graph*. An access plan graph consists of nodes and lines that connect those nodes. The nodes represent data sources, operators, queries, and query blocks. The arrows on the lines indicate the direction of the flow. Different node types have a different color and/or shape (that are customizable).

If you move your mouse over a node, additional information is displayed in a pop-up window, as shown in Figure 10-5. Note that for this node, the cardinality at that node is shown. This means that the optimizer thinks that at this stage during the processing (SORTRID), DB2 expects to sort 24930 RIDs. Looking at these cardinality numbers can give you an easy way to validate whether the number is similar to what you would expect, knowing the data that the query processes.

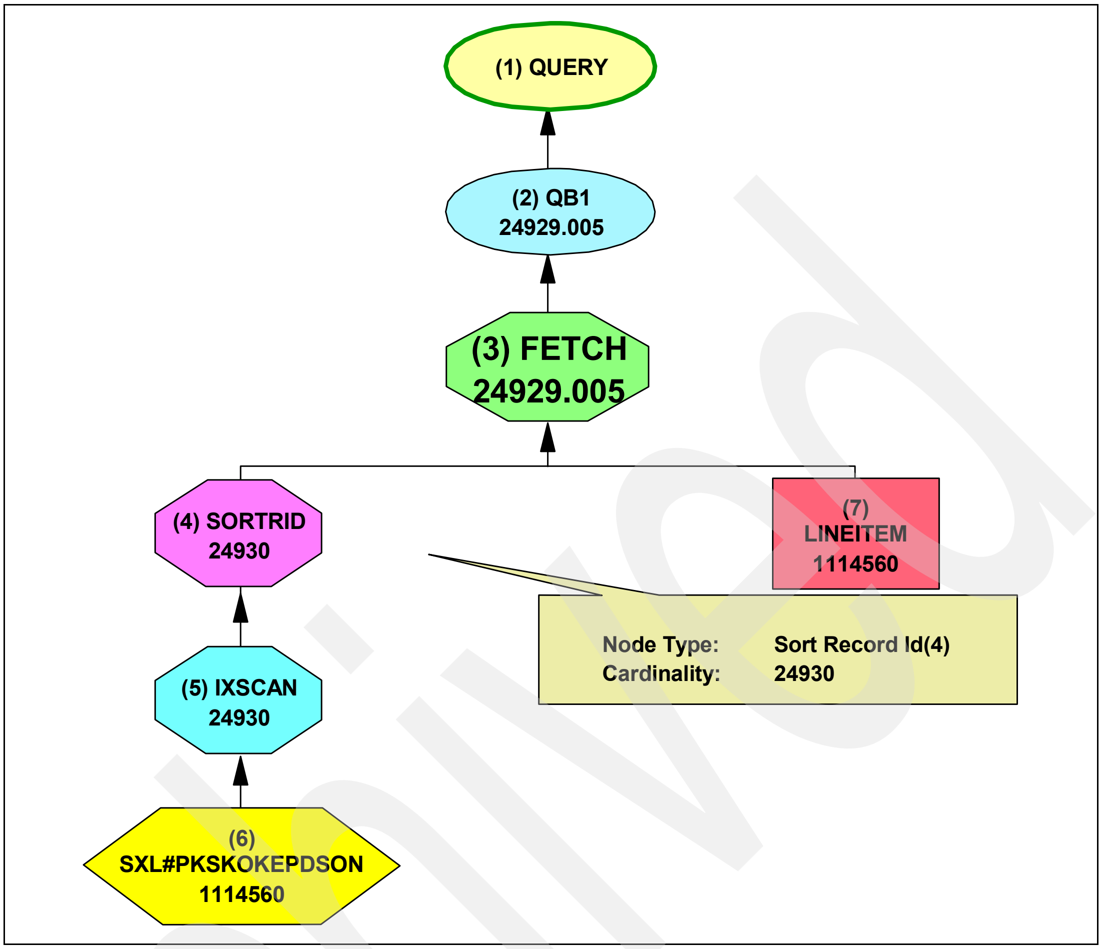
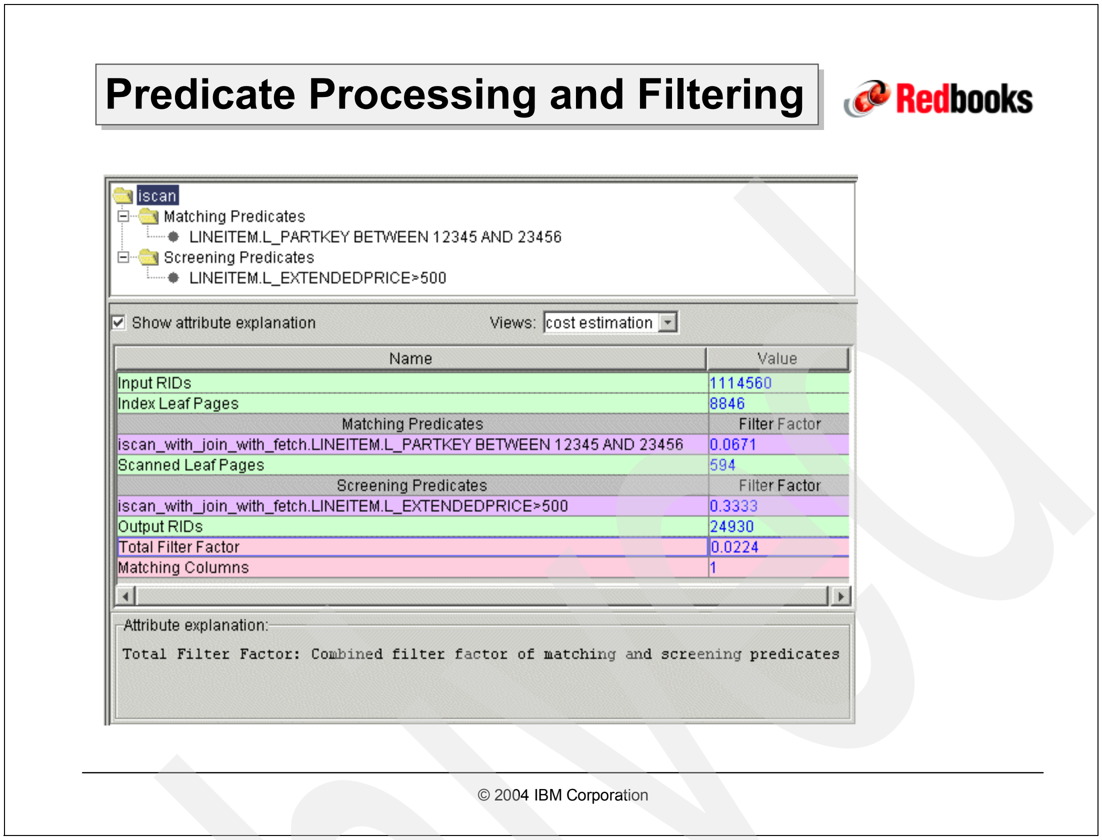


Figure 10-5 Access plan graph

On the left-hand side, you have additional information on the currently selected node. In the visual above the top node called query has the focus (this is also indicated on the left-hand side). The information displayed about the query is the type of SQL statement (in our case a SELECT), CPU cost (in ms and service units), the statement category ('A' in our case) and the reason (in case of a category 'B' statement).



10.19.5 Predicate processing and filtering

When we select the IXSCAN node, the left-hand side brings up a very helpful window related to the processing that goes on in the IXSCAN node. It is shown in the figure above. The following information is presented for our query:

Input RIDs This is the number of RIDs that goes into the IXSCAN process. As this IXSCAN is accessing the first table (index), the number of RIDs is the same as the number of rows in the table, or keys in the index (as the index is a unique index).

Index leaf pages This is the number of leaf pages in the index.

When dealing with an index, we have two types of predicates, index matching and index screening predicates.

Filter factors For the *matching predicate*:

```
l_partkey  between  12345 and 23456
```

the estimated filter factor is 0.0671 (the percentage of rows left after applying this individual predicate).

For the *screening predicate*:

```
l_extendedprice > 500
```

the estimated filter factor is 0.333 (a default).

The *Total (combined) Filter Factor* (of the matching and screening index predicates) is 0.0224.

Scanned Leaf Pages As only the index matching predicates determine the number of leaf pages scanned, this field appears under the matching predicates section.

Output RIDs The number of RIDs left after applying both matching and screening predicates.

Matching Columns Indicates the number of columns the matching predicate used for matching.

> **Note:** As you can see, Visual Explain provides you with information about whether a predicate is an index matching or index screening predicate, (whether a predicate is stage 1 non- indexable or stage 2 — not shown here because we are looking at an IXSCAN node), as well as the individual and combined filter factors. So, from now on — no more guesses; in V8 you have the facts.

You can obtain other information by clicking on another node. A table node, for example gives you all the information about the table itself, as well as all its related objects, like columns, indexes, table space and partition, as shown in Figure 10-6.

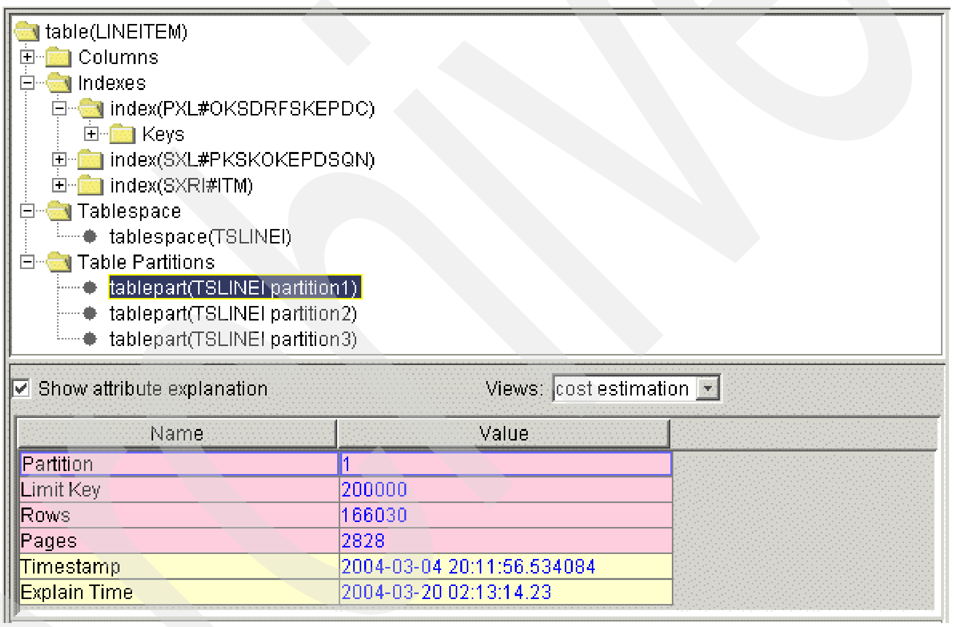
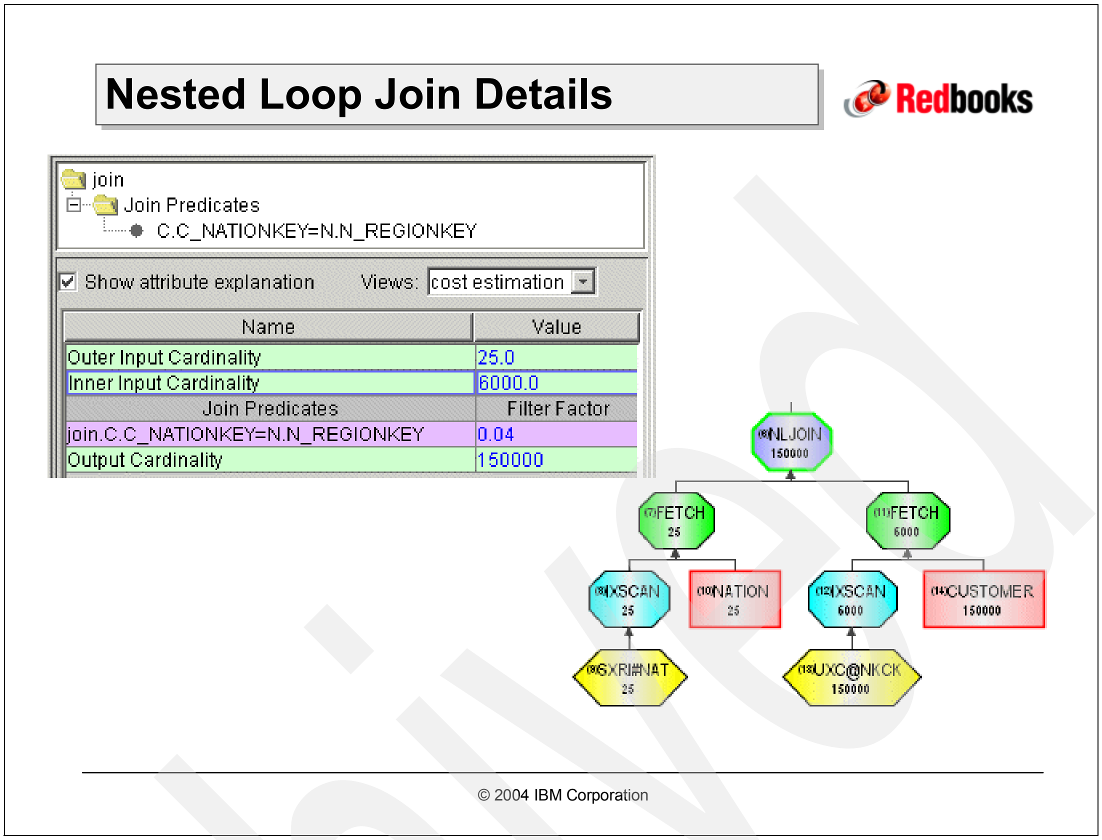


Figure 10-6 Object tree information

By exploring the object tree, you can obtain all pieces of information, including statistics about the objects involved in the query.



10.19.6 Nested loop join details

The graph in the figure above represents a nested loop join. A nested loop join construct consists of one NLJOIN node and two subtrees. The right subtree represents the inner table and the left subtree represents the outer table. Both tables can be accessed with either a table space scan, single-index access, or multiple-index access. The left subtree can also include another join operation (nested loop join, merge scan join, hybrid join, or star join).

The way to read the diagram is as follows:

- The outer table is accessed via a non-matching index scan (you can see this by clicking on the IXSCAN node, with “views all” selected (instead of the default “cost estimation”, in the node descriptor on the left side). The Nation table is also accessed since it appears in the diagram (for an index-only access the node representing the Nation table would not appear in the graph). The result is that all 25 rows are fetched (one at a time).
- The inner table is accessed via a matching index scan (on the join predicate). Note that you interpret the information about the inner table access on a per row of the outer table basis. In this case, for each row from the outer table, the inner table will use a matching index scan with data access, and is expected to return 6000 rows.

When you select the NLJOIN node, you see the information on the left-hand side of the figure. It contains interesting information that we can only obtain using Visual Explain. It shows the number of rows going into the nested loop join (outer input cardinality) and inner input cardinality (this is the number of rows that are expected to qualify in the inner table, as if it was the outer table). This is very valuable information, as it provides an easy way to spot the “odd man out”. In case one of the single table row estimates is wrong, this often leads to a poor join sequence and bad performance.

10.19.7 Sort information

Previously, estimating the size of a sort usually ended up in a lot of guesswork, unless you turned on some DB2 performance traces (class 9 in particular) to determine the size of the sorts that are executed as part of the query.

With V8's enhanced Visual Explain, the optimizer now tells you what your sorts are going to look like (Figure 10-7).

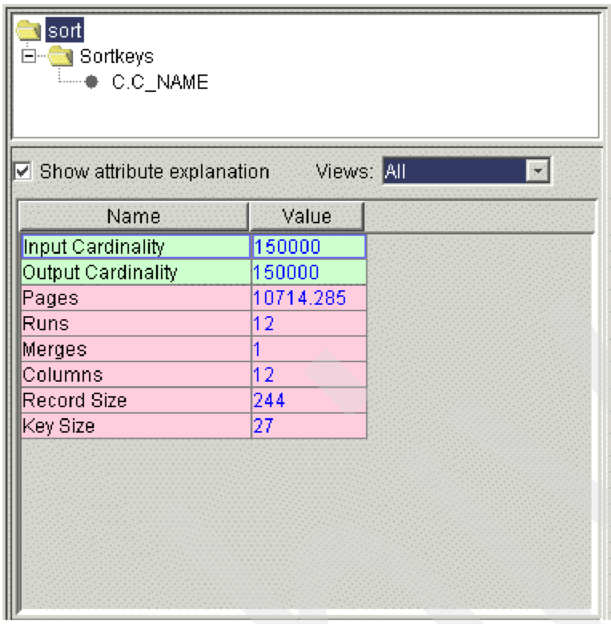


Figure 10-7 Sort node description

Visual Explains shows:

- The expected number of rows that go into the sort.
- The expected number of rows that come out of the sort. In this case, the number is the same because this is a simple ORDER BY. However, this does not have to be the case, for example during a GROUP BY sort.
- The number of pages scanned (from the input table / workfile) during the sort.
- The number of merges indicates how expensive the sort is. When the number of merges is more than one, DB2 cannot merge all the runs in a single operation. The result is that more resources will be required to perform the sort, more CPU, I/O, and sort workfiles.
- The record size is the total size of the sort record. It includes the sort columns (also called the key columns) and the data columns (non-key columns).
- The key size is the size of all the columns that we sort on, in our case the columns specified in the ORDER BY.

> **Note:** The sort key size has been increased from 4K to 16K in DB2 Version 8

10.19.8 Analyzing parallel queries with Visual Explain

When analyzing SQL queries that use DB2 query parallelism, Visual Explain also provides you with more information than you ever had before. Without going into too much detail, we use a simple parallel query as an example (Figure 10-8).

When looking at a single table access parallelism graph, there are two new node types that show up:

- Partition (of work) node (initiate a parallel group)
- Merge node (bring parallel streams back together, sometimes to continue with a different degree (not used in our example))

Both nodes indicate what the degree of parallelism is. For the partition of work node, that is the degree at which the index and/or table are accessed. For the merge node, the degrees that were merged (both three in our example).

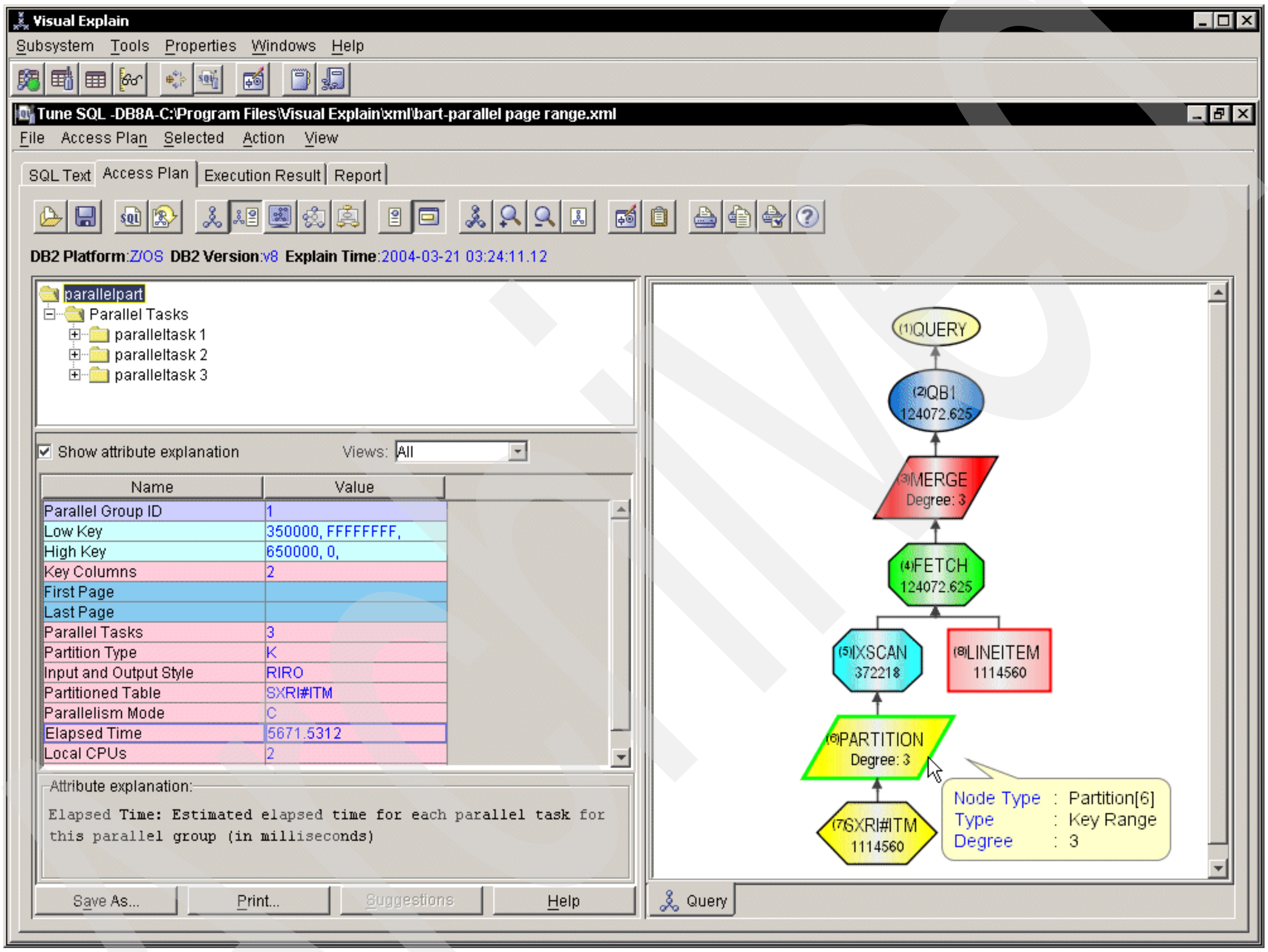


Figure 10-8 Parallel query in Visual Explain

When looking at the PARTITION node, you can see the parallelism details (on the left-hand side of Figure 10-8). They include:

- The parallelism mode: I/O, CPU, or sysplex parallelism)
- The partition type (type of parallelism): Page range or key range. Our example uses key range
- Other parallel task details, such as the number of CPUs, the expected elapsed time for each parallel task in this parallel group, and the number of parallel tasks

When you expand the tree in the top-left window, you can explore the actual key ranges (or page ranges, depending on the type of parallelism) as shown in Figure 10-9.

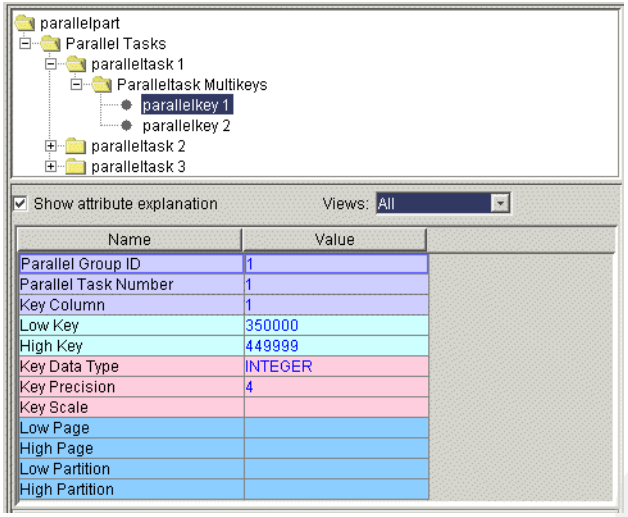


Figure 10-9 Parallel key range

For the first parallel task, the key range is from 350 000 until 449 999.

When you click on the fetch node (Figure 10-10), you can also find out which partitions will be accessed (qualified partition range 2-3).

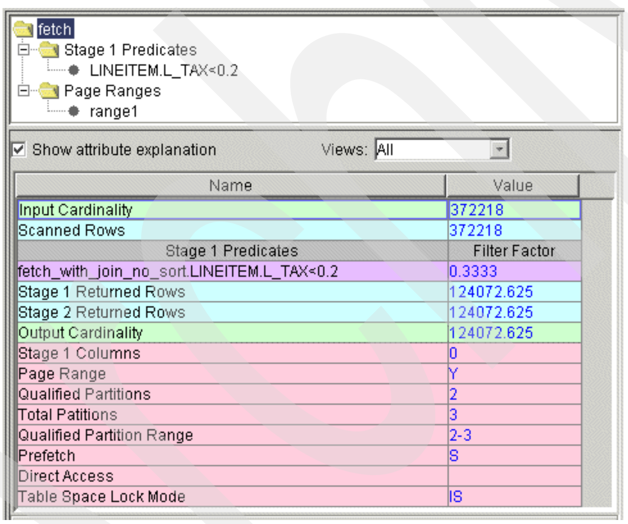
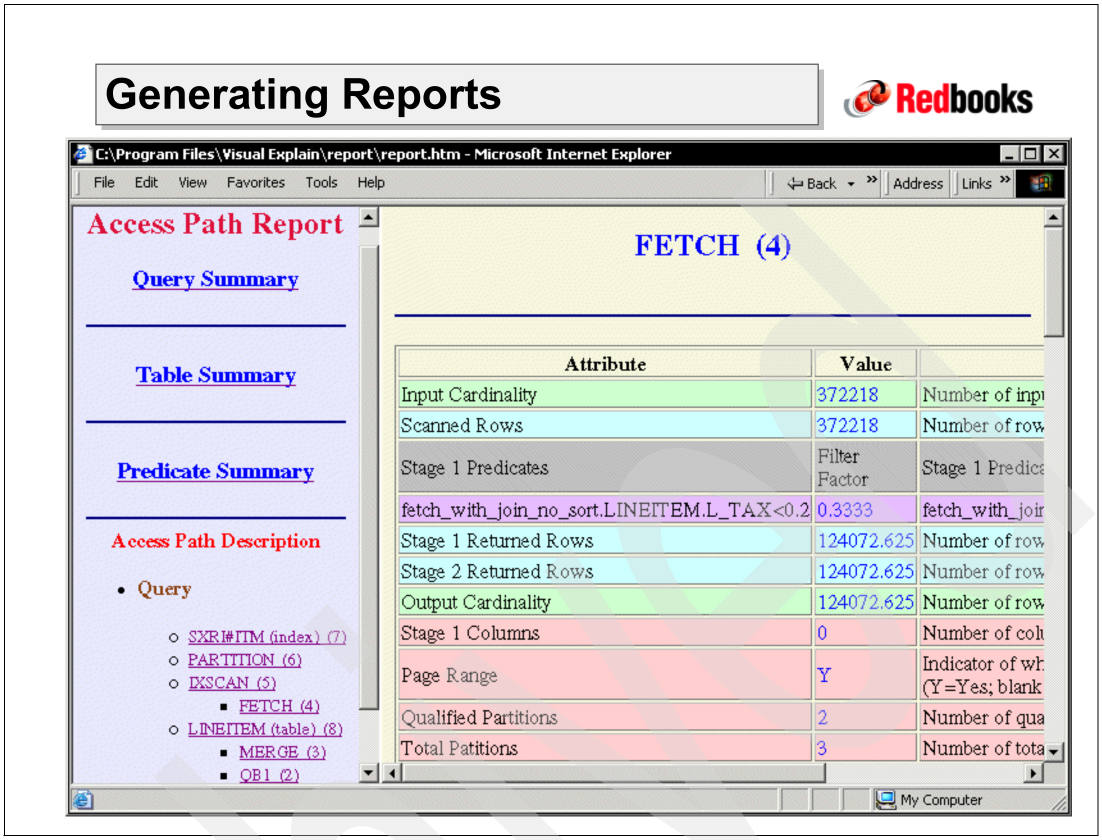


Figure 10-10 Parallelism fetch node



10.19.9 Generating reports

Other than printing individual screens, you can also generate a report detailing the explain and statistics information related to your query. To do so, click on the **Report** tab in the Tune SQL window. This brings up the generate report selection pane where you can check the items you want to include in the report (Figure 10-11).

After selecting the reporting options that you are interested in, you press the **Generate Report** button. The result is shown in the figure above.

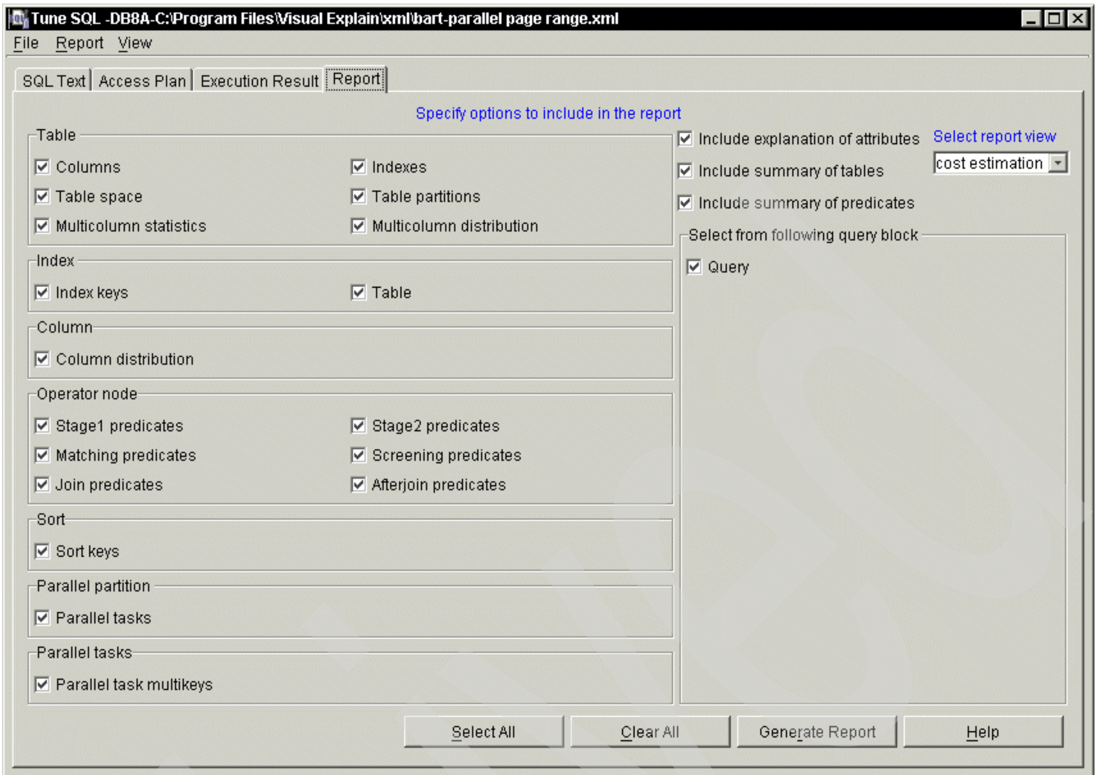


Figure 10-11 Generate report options

10.19.10 Statistics Advisor

Statistics Advisor is a new function available with Visual Explain. Its main purpose is to automate the analysis of the statistics required for an SQL statement. Sometimes it is not simple at all to understand which statistics should be provided to enable the optimizer to find the optimal access path for your queries. Often a query can have an inefficient access path or unstable performance due to a lack of statistics.

Statistics Advisor analyzes the predicates, the column (groups) used as predicates, and the type of predicates used. It also performs statistical analysis, checks missing statistics (default), conflicting statistics, missing appropriate correlations, and skew statistics. Based on all this information, Statistics Advisor gives you suggestions on the RUNSTATS statements to execute. Furthermore, it gives you an explanation and a conflict report.

You can even invoke RUNSTATS from within Statistics Advisor using the DSNUTILS stored procedure. You can continue to run the Statistics Advisor until it no longer provides suggestions.

For more information about Statistics Advisor, see also *DB2 UDB for z/OS Version 8 Performance Topics*, SG24-6465.

11

Data sharing enhancements

List of Topics

Redbooks

- CF lock propagation reduction
- CF request batching
- Improved LPL recovery
- Restart light enhancements
- Change to IMMEDWRITE bind option
- Change to -DISPLAY GROUPBUFFERPOOL command

In this chapter we introduce the following data sharing enhancements in DB2 Version 8:

- CF lock propagation reduction:

  This enhancement remaps IX parent L-locks from XES-X to XES-S. Data sharing locking performance will benefit because this allows IX and IS parent global L-locks to be granted without invoking global lock contention processing to determine that the new IX or IS lock is compatible with existing IX or IS locks.

- CF request batching:

  Current architecture allows multiple pages to be registered to the coupling facility with a single command. This enhancement utilizes new functionality in z/OS 1.4 and CF level 12 to enable DB2 to:

  – Register and write multiple pages to a group buffer pool.
  – Read multiple pages from a group buffer pool for castout processing.

  This reduces the amount of traffic to and from the coupling facility for writes to group buffer pools and for reads for castout processing, thus reducing the data sharing overhead for most workloads.

- Improved LPL recovery:

  Prior to Version 8, you had to manually recover pages that DB2 put into the logical page list (LPL). DB2 Version 8 automatically attempts to recover LPL pages as they go into LPL, when it determines the recovery will probably succeed.

  Currently LPL recovery requires exclusive access to the table space page set. Version 8 introduces a new serialization mechanism whereby LPL recovery is much less disruptive.

  In addition, instrumentation has improved whereby message DSNB250E is enhanced to indicate the reason the pages are added to the LPL.

- Restart light enhancements:

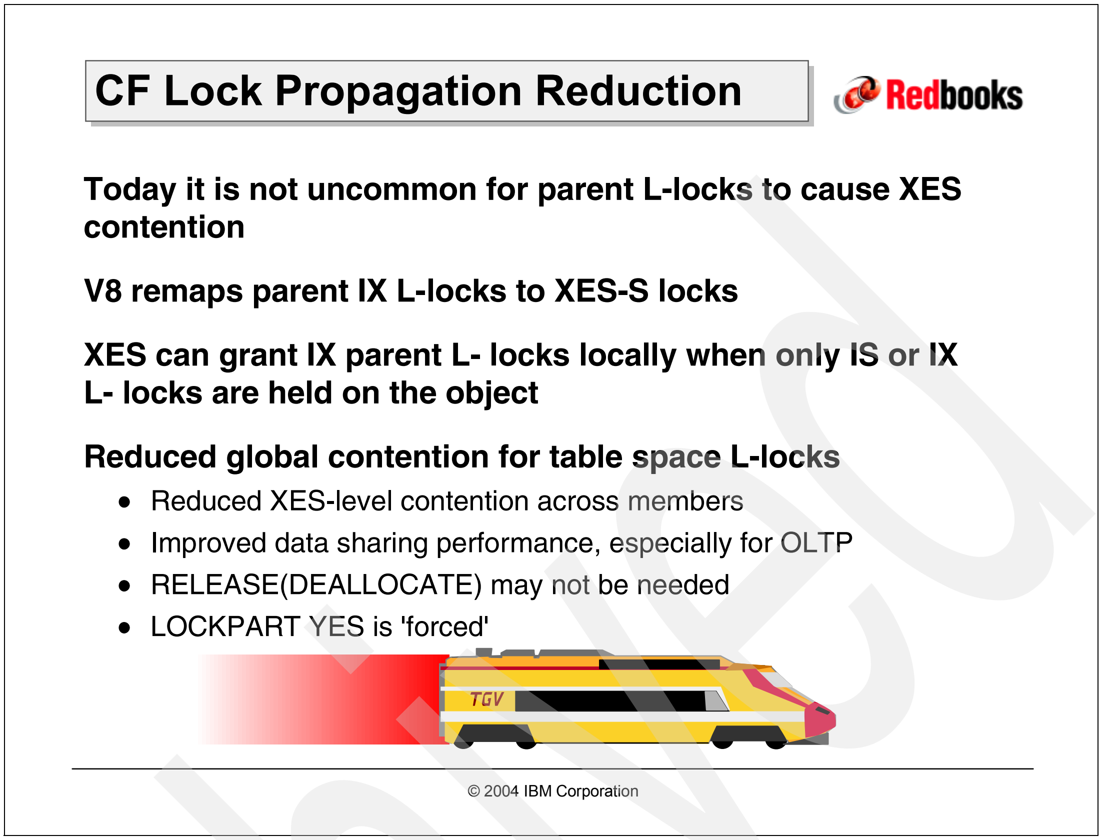
  DB2 V7 introduced the concept of DB2 restart light which is intended to remove retained locks with minimal disruption in the event of an MVS system failure. When a DB2 member is started in restart light mode (LIGHT(YES)), DB2 comes up with a small storage footprint, executes forward and backward restart log recovery, removes the retained locks, and then self-terminates without accepting any new work. Restart light is improved in DB2 Version 8. If indoubt units of recovery (UR) exist at the end of restart recovery, DB2 will now remain running so that the indoubt URs can be resolved. After all the indoubt URs have been resolved, the DB2 member that is running in LIGHT(YES) mode will shut down and can be restarted normally.

- Change the IMMEDWRITE default bind option:

  Currently, changed pages in a data sharing environment are written during phase 2 of commit, unless otherwise specified by the IMMEDWRITE BIND parameter or IMMEDWRI DSNZPARM parameter. This enhancement will change the default processing to write changed pages during commit phase 1. DB2 will no longer write changed pages during phase 2 of commit processing.

- Change to -DISPLAY GROUPBUFFERPOOL output:

  Currently, the CF level displayed by the -DISPLAY GROUPBUFFERPOOL command may be lower than the actual CF level as displayed by a D CF command. The -DISPLAY GROUPBUFFERPOOL is now enhanced to display both the operational CF level as before, and also the actual CF level. (The operational CF level indicates the capabilities of the CF from DB2's perspective. The actual CF level is the microcode level as displayed by the D CF command.)



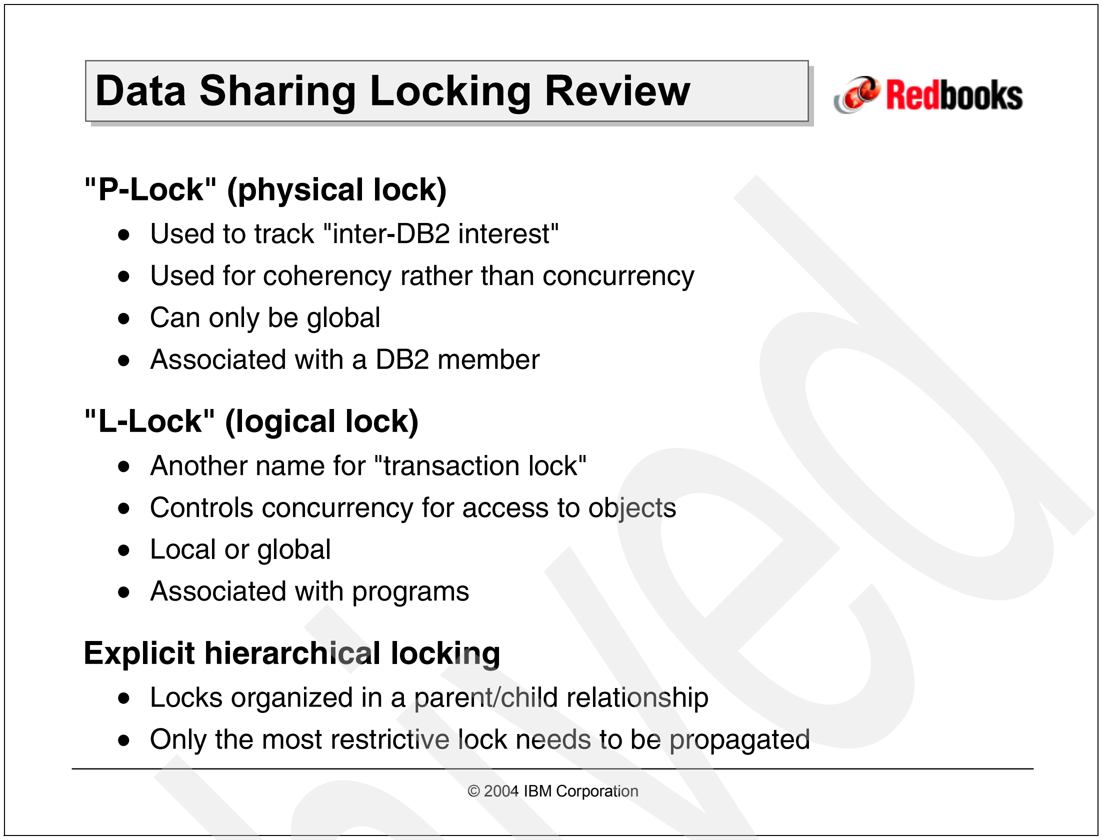
11.1 CF lock propagation reduction

DB2 Version 8 will remap parent IX L-locks from XES-X to XES-S locks. Data sharing locking performance will benefit because parent IX and IS L-locks are now both mapped to XES-S locks and are therefore compatible and can now be granted locally by XES. DB2 will no longer need to wait for global lock contention processing to determine that a new parent IX or IS lock is compatible with existing parent IX or IS locks.

This enhancement will reduce data sharing overhead by reducing global lock contention processing. It is not uncommon for parent L-Locks to cause global contention. On page set open (an initial open or open after a pseudo close) DB2 normally tries to open the page set in RW. To do this, DB2 must ask for an X or IX page set L-Lock. If any other DB2 member already has the data set open, global lock contention now occurs.

The purpose of this enhancement is to avoid the cost of global contention processing whenever possible. It will also improve availability due to a reduction in retained locks following a subsystem failure.

In the next few foils we explain this enhancement in more detail.



11.1.1 Data sharing locking review

Data sharing locking

DB2 data sharing uses two types of locks:

- Physical locks (P-locks):

  Physical locks are used to do many different things. Next we discuss the two most commonly used P-locks: page set P-locks and page P-locks. Other types of P-locks include DBD, castout, GBP structure, index tree, and repeatable read tracking P-locks.

  – Page set physical locks:

    Page set P-locks are used to track inter-DB2 read-write interest, thereby determining when a page set has to become GBP-dependent.

    When a DB2 member requires access to a page set or partition, a page set P-lock is taken. This lock is always propagated to the lock table in the coupling facility and is owned by the member. No matter how many times the resource is accessed through the member, there will always be only one page set P-lock for that resource for a particular member. This lock will have different modes depending on the level (read or write) of interest the member has in the resource.

    The first member to acquire a page set P-lock on a resource takes the most restrictive mode of lock possible, that is, an S page set P-lock for read or an X page set P-lock for write interest. An X page set P-lock indicates that the member is the only member with interest (read/write) in the resource. Once another member becomes interested in the resource, the page set P-lock mode can be negotiated, that is, it can be made less restrictive if the existing page set P-lock is incompatible with the new page set P-lock request.

The negotiation always allows the new page set P-lock request to be granted, except when there is a retained X page set P-lock. A retained P-lock cannot be negotiated. (Retained locks are locks that must be kept to protect possibly uncommitted data left by a failed DB2 member.) Page set P-lock negotiation signifies the start of GBP dependence for the resource.

Although it may seem strange that a lock mode can be negotiated, remember that page set P-locks do not serialize access to a resource; they are used to track which members have interest in a resource and for determining when a resource must become GBP-dependent.

Page set P-locks are released when a page set or partition data set is closed. The mode of page set P-locks is downgraded from R/W to R/O when the page set or partition is not updated within an installation-specified time period or a number of system checkpoints. When page set P-locks are released or downgraded, GBP dependency is re-evaluated.

– Page physical locks:

Page P-locks are used to ensure the physical consistency of a page across members of a data sharing group in much the same manner as latches do in a non-data sharing environment. A page P-lock protects the page while the structure is being modified. Page P-locks are used when row locking is in effect, or when changes are being made to GBP-dependent space map pages. Page physical locks are also used to read and update index pages.

- Logical locks (L-locks)

  Logical locks are also referred to as transaction locks. L-locks are used to serialize access to data to ensure data consistency.

  L-locks are owned by a transaction, and the lock duration is controlled by the transaction. For example, the lock is generally held from the time the application issues an update until the time it issues a commit. (Exceptions are share locks associated with cursors defined WITH HOLD, and table space and partition locks acquired by SQL, associated with plans and packages bound using RELEASE(DEALLOCATE).) The locks are controlled locally per member by each member's IRLM.

P-locks and L-locks work independently of each other, although the same processes are used to manage and maintain both. The lock information for all these locks are stored in the same places (the IRLM, XES and the coupling facility).

Explicit hierarchical locking

Conceptually all locks taken in a data sharing environment are global locks; that is, they are effective groupwide, even though all locks do not have to be propagated to the lock structure in the coupling facility.

DB2 data sharing has introduced the concept of *explicit hierarchical locking*, to reduce the number of locks that must be propagated to the coupling facility.

Within IRLM, a hierarchy exists between certain types of L-locks, where a parent L-lock is the lock on a page set (table space or partition) and a child L-lock is the lock held on either the table, data page, or row within that page set. For partitioned table spaces, if you use LOCKPART YES in DB2 V7, each locked partition is a parent of the child locks held for that partition. If you use LOCKPART NO (the default in V7), the last data partition is the parent lock for all child locks.

By using explicit hierarchical locking, DB2 is able to reduce the number of locks that must be propagated to the lock structure in the coupling facility. The number of locks propagated to the lock structure for a page set or partition is determined by the number of DB2 members interested in the page set and whether their interest is read or write. Wherever possible, locks are granted locally and not propagated to the coupling facility.

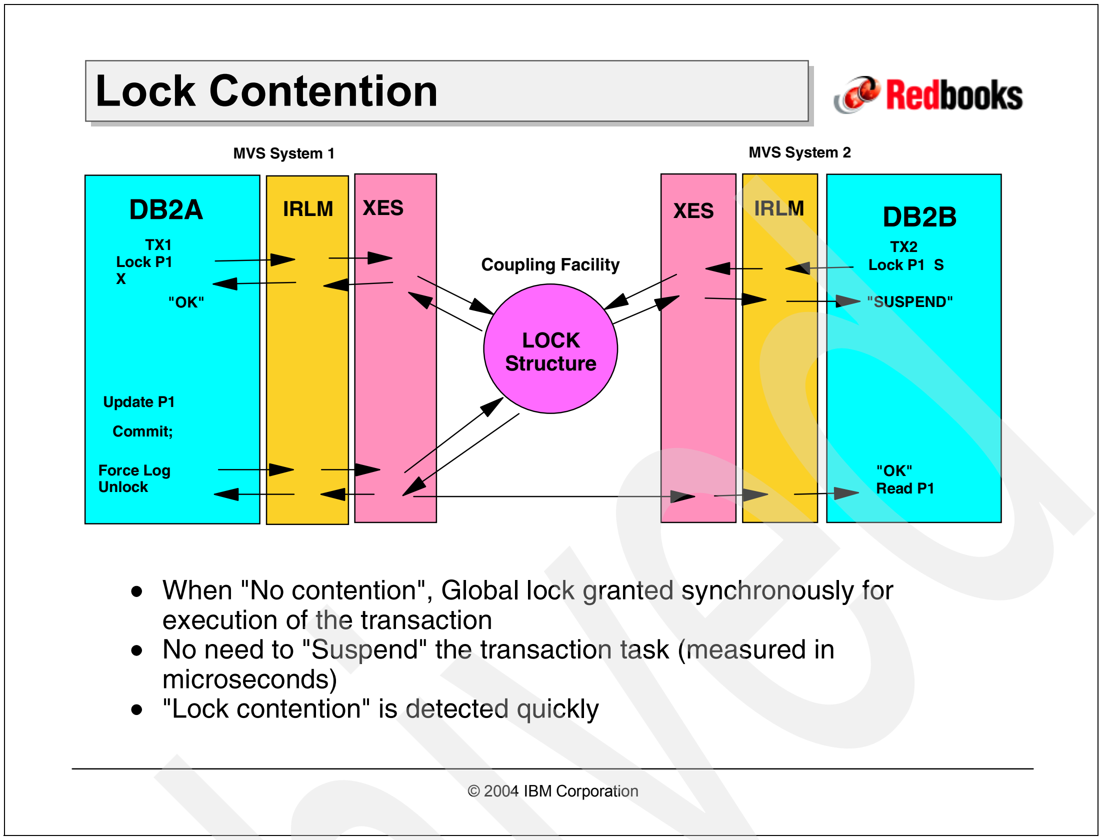
If an L-lock has already been propagated to XES protecting a particular resource for this member, subsequent lock requests for the same lock do not have to be sent to XES by the same member for the same resource. They can be serviced locally. In addition, a parent L-lock is propagated only if it is more restrictive than the current state that XES knows about for this resource from this member.

Parent L-locks are released either when the transaction commits, or when the thread terminates, depending on the value you have specified for the RELEASE parameter on the bind. Child L-locks are propagated to the lock table in the coupling facility only when there is inter-DB2 interest for the page set.

Child L-locks (page and row locks) are propagated to XES and the coupling facility based on inter-DB2 interest on the parent (table space or partition) lock. If all the table space locks are IS, then no child locks are propagated. However, if there is a parent IX lock on the table space or partition, then all the child locks must be propagated.

For example, assume that transactions A and B are running in the same member:

- If transaction A has a parent IS L-lock, the IS L-lock gets propagated to the lock structure on the coupling facility. If transaction B has a parent IX L-lock, the IX L-lock gets propagated to the lock structure on the coupling facility.
- If transaction A has a parent IX L-lock, the IX L-lock gets propagated to the lock structure on the coupling facility. If transaction B has a parent IS L-lock, the IS L-lock does not get propagated to the lock structure on the coupling facility; that is, the resultant state does not change.
- If transaction A has a parent S L-lock, the S L-lock gets propagated to the lock structure on the coupling facility. If transaction B has a parent IX L-lock, the SIX L-lock gets propagated to the lock structure on the coupling facility
- If transaction A has a parent IX L-lock, the IX L-lock gets propagated to the lock structure on the coupling facility. If transaction B has a parent S L-lock, the SIX L-lock gets propagated to the lock structure on the coupling facility.



11.1.2 Lock contention before Version 8

This foil presents a logical overview of how each IRLM works together to maintain data integrity for a page set where both DB2 members have interest in the page set.

Consider transaction TX1 on DB2A which needs a X lock on page P1. IRLM passes this lock request to XES and the lock is granted. Now, transaction TX2 on DB2B needs a S lock on page P1. IRLM passes this lock request through XES to the coupling facility. As transaction TX1 already has a X lock for page P1, transaction TX2 must be “suspended”.

Transaction TX1 now updates page P1 and commits. The IRLM releases the X lock and passes an unlock request through XES to the coupling facility. The S lock is now granted and transaction TX2 can be un-suspended to continue its work.

Now, let us have a closer look at the various reasons why a transaction may be suspended.

Lock information is held in three different components:

- IRLM
- SLM component of XES, which in many publications is simply referred to as XES.
- Lock structure on the coupling facility.

The types of lock granularity supported by each component differ. IRLM contains the most detailed lock information, whereas XES and the lock table on the coupling facility recognize only two types of locks — S and X. Each IRLM lock maps to a particular XES lock. IS and S map to XES-S locks, while U, IX, SIX and X map to XES-X locks.

Lock contention occurs when a task is requesting a lock for a resource and the lock may already be held by another task. In data sharing, the other task could be running in the same DB2 subsystem or running in another DB2 member. For this discussion we are only concerned about global contention, when the contention is across DB2 members.

Global lock contention

In data sharing, three types of global contention can occur. These are listed in order of increasing time needed for their resolution:

- False contention:

  False contention occurs when the hashing algorithm for the lock table provides the same hash value for two different resources. The different resources then share that one lock table entry.

  False contention can occur only when the lock table entry is managed by a global lock manager or when the lock request causes global management to be initiated, (we have inter-DB2 R/W interest in the page set.). The XES requesting the lock needs to know the owning resource name to resolve this apparent contention. That information already resides in the XES that is the global lock manager for the lock table entry. If the global lock manager is not the requesting XES, communication between XES components is needed to resolve the false contention.

  In our example, false contention would occur if transaction TX2 were to request a lock for a different resource, say page P2, and the lock request hashed to the same lock table entry in the coupling facility.

  Transaction TX2 must be suspended while the XES who is the global lock manager for the lock table entry, determines that the lock can be granted.

- XES contention:

  The MVS XES component is aware of only two lock modes, share and exclusive. IRLM locking supports many additional lock modes. When the MVS XES component detects a contention because of incompatible lock modes for the same resource, that contention is not necessarily a real contention by IRLM standards. For example, the IRLM finds the IX-mode to be compatible with the IS-mode. For the MVS XES component, however, these are not IX-mode and IS-mode, but X-mode and S-mode which are incompatible. To see if a real contention exists, MVS XES must give control to the IRLM contention exit associated with the global lock manager. The IRLM contention exit must determine if the contention is real or not, that is, if the locks are incompatible. If the contention is not real, it is called “XES contention” and the requested lock can be granted.

  In our example, XES contention would occur if transaction TX1 held an IX lock on the page set that contains page P1 and transaction TX2 was requesting an IX lock on the same page set that contains page P1. Both of these lock requests are passed to XES as X locks. XES sees these lock requests as not compatible, however IRLM knows they are compatible.

  Transaction TX2 must be suspended while the XES who is the global lock manager for the lock table entry, must defer to IRLM to decide if the lock request can be granted.

- Real contention:

  Real contention is caused by normal IRLM lock incompatibility between two members. For example, two transactions may try to update the same resource at the same time. DB2 PE reports real contentions as IRLM contentions.

  This is the example we have just explained. Transaction TX2 is requesting a lock which is not compatible with a lock already help by transaction TX1. Transaction TX2 must be suspended while XES defers to IRLM who cannot grant the lock.

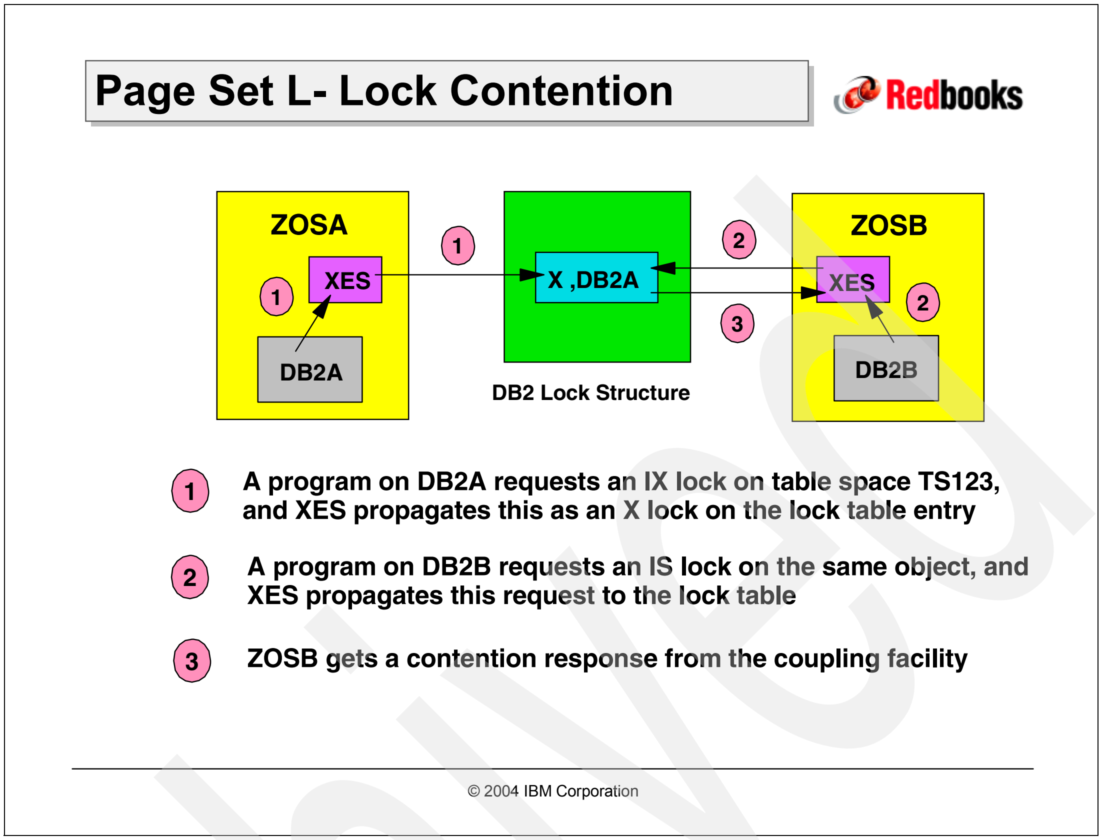
Resolving contention

Contentions require additional XES and XCF services if the requesting member is not global lock manager, that is, the owner of the lock registered in the lock table entry.

Information about locks that IRLM has passed to XES is stored in XES. When contention occurs (false, XES, or real contention), one of the XESs is assigned to be the global lock manager to resolve the contention. This resolution involves all of the other XESs in the group that have locks which have been assigned to the lock table entry, passing their lock information to the global lock manager XES. This global lock manager XES can then drive resolution of the contention.

When any contention occurs, execution of the requester's SQL statement is suspended until the contention is resolved. If the contention is real, the requester remains suspended until the incompatible lock is released.

Therefore any contention can adversely impact performance. The SQL is suspended while the contention is resolved and extra CPU is consumed resolving the contention.



11.1.3 Page set L-lock contention

The flow of a global logical lock with inter-DB2 read-write interest and with a XES contention is as follows:

1. An application on DB2A issues an SQL UPDATE:
   a. DB2A decides a lock is required and requests an IX page set L-lock to IRLMA.
   b. IRLMA registers the lock and passes a request for an X lock to XES on ZOSA.
   c. XES on ZOSA sends the X lock to the coupling facility.
2. An application on DB2B issues an SQL SELECT on the same page set:
   a. DB2B decides a lock is required and requests an IS page set L-lock to IRLMB.
   b. IRLMB registers the lock and passes a request for an S lock to XES on ZOSB.
   c. XES on ZOSB sends the S lock to the coupling facility.
3. The coupling facility finds a contention, in this case, XES contention.

XES contention on page set L-locks is reasonably common in data sharing environments. For example, when DB2 opens a page set (either as a real data set open or logical open after a pseudo close), DB2 will normally try to open the page set in RW. To do this DB2 must ask for an X or IX page set L-lock. If any other DB2 member already has the data set open, global lock contention now occurs. (It is much more common for DB2 to open a page set in R/W rather than R/O.)

XES Contention Before Version 8

Various IRLM lock levels can ONLY map to one of two XES lock levels

- IRLM IS and S locks map to XES -S lock
- IRLM U, IX, SIX and X locks map to XES -X lock

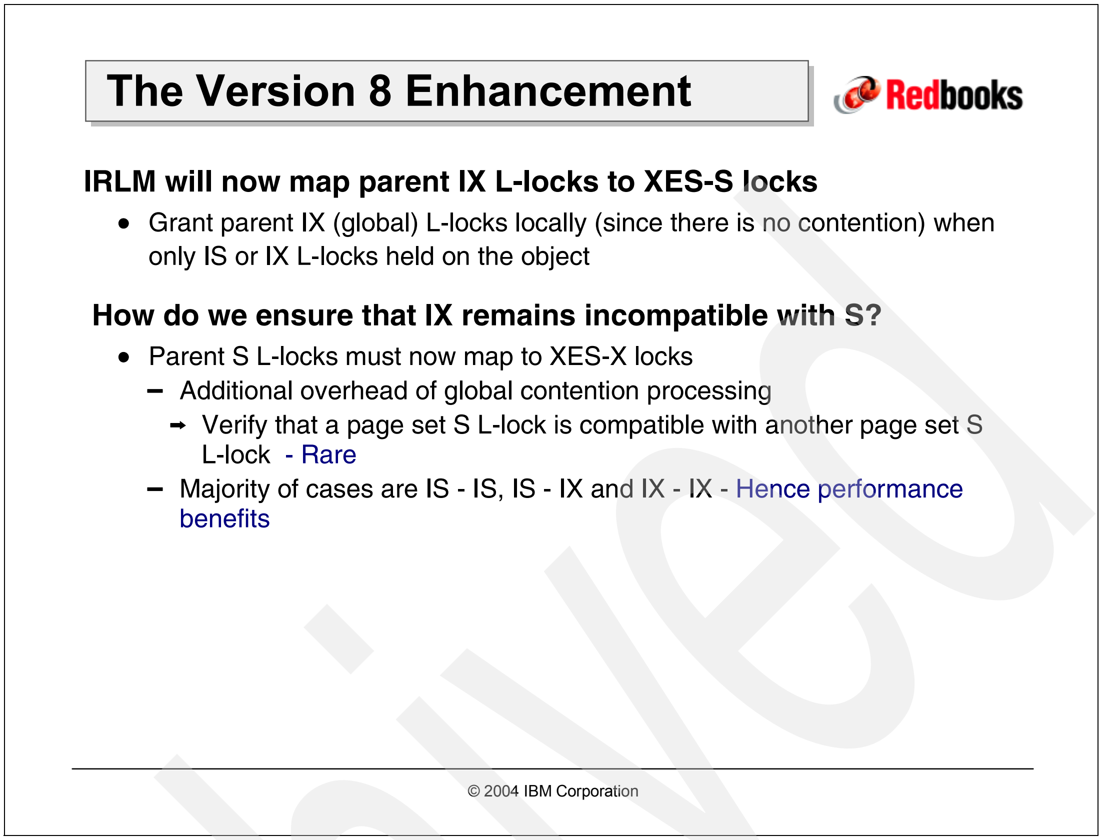
In previous example with two members holding an IS lock and an IX lock on same table space

- IRLM IX lock maps to XES X lock
- IRLM IS lock maps to XES S lock
- XES detects contention
- Global contention processing invoked by IRLM
  - Determine if IX is really compatible with IX
  - Grant lock request

11.1.4 XES contention before Version 8

To resolve the contention described on the previous foil:

1. The coupling facility replies to XES on ZOSB with a finding of contention and the global lock manager identifier.
2. XES on ZOSB asks IRLMB to suspend SQL statement.
3. XES on ZOSB use XCF to query the XES global lock manager, (XES on ZOSA).
4. The global lock manager finds that the lock is in contention.
5. XES on ZOSA drives the IRLMA contention exit identifying contention.
6. IRLMA determines this is not real contention and tells XES on ZOSA. The contention is XES contention.
7. XES on ZOSA replies to XES on ZOSB through XCF that no real contention exists.
8. XES om ZOSB replies to IRLMB: No contention.
9. IRLMB replies to DB2B.
10. DB2B resumes the SQL statement and passes the result to the application.



11.1.5 The Version 8 enhancement

In DB2 Version 8, parent IX L-locks will be remapped to XES-S locks, rather than XES-X locks. This will allow the parent global IX L-locks to be granted without having to invoke the contention exit (it can be granted by local system's XES) when only IS or IX L-locks are held on the object.

To ensure that parent IX L-locks remain incompatible with parent S L-locks, S table and table space locks are remapped to XES-X locks. This means that additional global contention processing will now be done to verify that a page set S L-lock is compatible with another page set S L-lock, but this is a relatively rare case (executing read-only SQL against a page set, and there is only one other member who currently has some read-only SQL active against the same page set).

The majority of cases are as follows:

- IS-IS: We want to execute some read-only SQL against a page set and there are some other members who currently have some read-only SQL active against the same page set.
- IS-IX: We want to execute some update SQL against a page set and there are some other members who currently have some read-only SQL active against the same page set.
- IX- IX: We want to execute some update SQL against a page set and any number of other members who currently have some update SQL active against the same page set.

Hence, global contention processing will be reduced. Parent lock contention with parent S L-locks is less frequent than checking for contention with parent IS and IX L-locks.

V8 Child L-Lock Propagation

Redbooks

Child L-lock propagation no longer based on the cached state of parent L-lock

Now based on cached (held) state of page set P-lock

- If page set P-lock negotiated from X to SIX or IX, then child L-locks propagated
- Reduced volatility
- If P-lock not held at time of child L-lock request, child lock will be propagated

Parent L-locks no longer need to be held in retained state after DB2 failure

- For example, a page set IX L-lock no longer held as a retained X-lock
  - Important availability benefit in data sharing

11.1.6 V8 child L-lock propagation

Another impact of this change is that child L-locks are no longer propagated based on the parent L-lock. Instead, child L-locks are propagated based on the held state of the page set P-lock. If the page set P-lock is negotiated from X to SIX or IX, then child L-locks will be propagated.

It may be that some child L-locks are acquired before the page set P-lock is obtained. In this case child L-locks will automatically be propagated. This situation occurs because DB2 always acquires locks before accessing the data. In this case, DB2 acquires the page set L-lock before opening the page set to read the data.It can also happen during DB2 restart.

An implication of this change is that child L-locks will be propagated for longer that they are needed, however this should not be a concern. There will be a short period from the time where there is no inter-system read/write interest until the page set becomes non-GBP-dependent, that is, before the page set P-lock reverts to X. During this time, child L-locks will be propagated unnecessarily.

Another consequence of this enhancement is that, since child L-lock propagation is no longer dependent upon the parent L-lock, parent L-locks will no longer be held in retained state following a system failure. This means, for example, that a page set IX L-lock will no longer be held as a retained X-lock after a system failure. This can provide an important availability benefit in a data sharing environment. Because there is no longer a retained X-lock on the page set, most of the data in the page set remains available to applications running on other members. Only the pages (assuming page locking is used) with a retained X-lock will be unavailable.

LOCKPART Considerations

It is now important that L-locks and P-locks are maintained at the same level of granularity, because

- P-Locks now determine child L-lock propagation

LOCKPART NO table spaces can no longer get a lock only on the last partition

- Changed to LOCKPART YES
- Data sharing and non-data-sharing
- DDL restriction on LOCKPART YES and LOCKSIZE TABLESPACE remains for now
  - We ONLY lock the partitions as we need them
  - May see additional locks acquired on individual partitions even though LOCKPART NO is specified

11.1.7 LOCKPART considerations

It is now important that L-locks and P-locks are maintained at the same level of granularity. Page set P-locks now determine when child L-locks must be propagated to XES.

For partitioned table spaces defined with LOCKPART NO, DB2 currently (V7) locks only the last partition to indicate we have a lock on the whole table space. There are no page set L-locks held on each of the partition page sets. So, when should we propagate the child L-locks for the various partitions that are being used? (We cannot tell by looking at the page set parent L-locks that we need to propagate the child locks since there no longer is a lock contention conflict that can trigger child lock propagation, and we cannot determine how each partition page set is being used by looking at the page set P-locks that are held at the partition level, while the parent L-lock is at the table space level with LOCKPART NO.)

To overcome this problem, LOCKPART NO table spaces will now obtain locks at the part level. LOCKPART NO will behave the same as LOCKPART YES.

In addition, LOCKPART YES is not compatible with LOCKSIZE TABLESPACE. However, if LOCKPART NO and LOCKSIZE TABLESPACE are specified then we will lock every partition, just as every partition is locked today when LOCKPART YES is used with ACQUIRE(ALLOCATE). With this change you may see additional locks being acquired on individual partitions even though LOCKPART(NO) is specified.

This change applies to both data sharing and non-data sharing environments.

With this change you may see additional locks acquired on individual partitions even though LOCKPART NO is specified.

Benefits of Less CF Lock Propagation

Redbooks

Faster lock processing for IX and IS parent L-locks

- IX-IX, IX-IS, IS-IS

Child locks propagated based on page set P-lock

- Less volatility

Less need for RELEASE(DEALLOCATE)

- Used in the past to reduce CF lock propagation

Avoids the cost of global contention whenever possible

- Reduce XES contention
- Improve availability by reducing "Retained Locks" following a subsystem failure

11.1.8 Benefits of less CF lock propagation

Data sharing locking performance will benefit because this allows IX and IS parent L-locks to be granted locally without invoking global lock contention processing to determine that the new IX or IS lock is compatible with existing IX or IS locks.

In data sharing, the recommendation for RELEASE(DEALLOCATE) (and thread reuse) to reduce XES messaging for page set L-locks is no longer required. This is good news because using RELEASE(DEALLOCATE):

- Can cause increased EDM pool consumption, because plans and packages stay allocated longer in the EDM pool.
- May also cause availability concerns due to parent L-locks being held for longer. This can potentially prevent DDL from running, or cause applications using the LOCK TABLE statement and some utilities to fail.

However, as in previous versions of DB2, to avoid locking overhead, you should use ISOLATION UR, or try to limit the table space locks to IS on all data sharing members to avoid child lock propagation.

Additionally, the ability to grant IX and IS locks locally implies less thrashing on changing inter-system interest levels for parent locks, requiring less IRLM SRB time and less XCF messaging. When DB2 decides it must propagate its locks to the coupling facility for a given page set, DB2 must collect and propagate all the locks it currently owns for that page set to the coupling facility. This can cause some overhead, particularly when a page set is not used often enough for lock propagation to occur all the time. Page set P-locks are long duration locks and tend to be more static than L-locks, so the chances are higher that lock propagation will continue for longer.

Fallback/Co-existence/Enablement

New locking protocol cannot co-exist with old

Enabling feature

- Migrate to new-function mode (NFM)
- Restart of first member after successful quiesce of all members in group

Disabling feature

- Returning to CM (full restore of environment is the only option)
- Restart of first member

11.1.9 Fallback/co-existence/enablement

Since the new locking protocol cannot co-exist with the old, the new protocol will only take effect after the first group-wide shutdown when the data sharing group is in new-function mode (NFM). No other changes are required to take advantage of this enhancement.

If you recover the catalog and directory to a point-in-time prior to the point where new-function mode was enabled, a group-wide shutdown is required. On the next restart, whether it be on Version 7 or Version 8, the new locking protocol will be disabled.

> **Note:** You have to be in new-function mode to be able to benefit from this new way of mapping IRLM lock states to XES lock states. The new mapping takes effect after the restart of first member, after successful quiesce of all members in the DB2 data sharing group. So, to enable this feature, a group-wide outage is required.

You can use the -DIS GROUP command to check whether the new locking protocol is used (mapping IX IRLM L-locks to an S XES lock), as shown in Example 11-1. Protocol level(2) indicates that the new protocol is active.The use of locking protocol level 2 requires that the PTFs for the following APARs are applied: PQ87756, PQ87168, and PQ86904 (IRLM).

Example 11-1 -DIS GROUP output

```
DSN7100I  -DT21 DSN7GCMD
 *** BEGIN DISPLAY OF GROUP(DSNT2   ) GROUP LEVEL(810) MODE(N)
                  PROTOCOL LEVEL(2)  GROUP ATTACH NAME(DT2G)
 --------------------------------------------------------------------
 DB2                                    DB2 SYSTEM    IRLM
 MEMBER   ID  SUBSYS CMDPREF   STATUS   LVL NAME      SUBSYS IRLMPROC
 -------- --- ----   --------  -------- --- --------  ----   --------
 DT21       1 DT21   -DT21     ACTIVE   810 STLABB9   IT21   DT21IRLM
 DT22       3 DT22   -DT22     FAILED   810 STLABB6   IT22   DT22IRLM
......
```

CF Request Batching

Batching of GBP writes and castout reads

Objectives

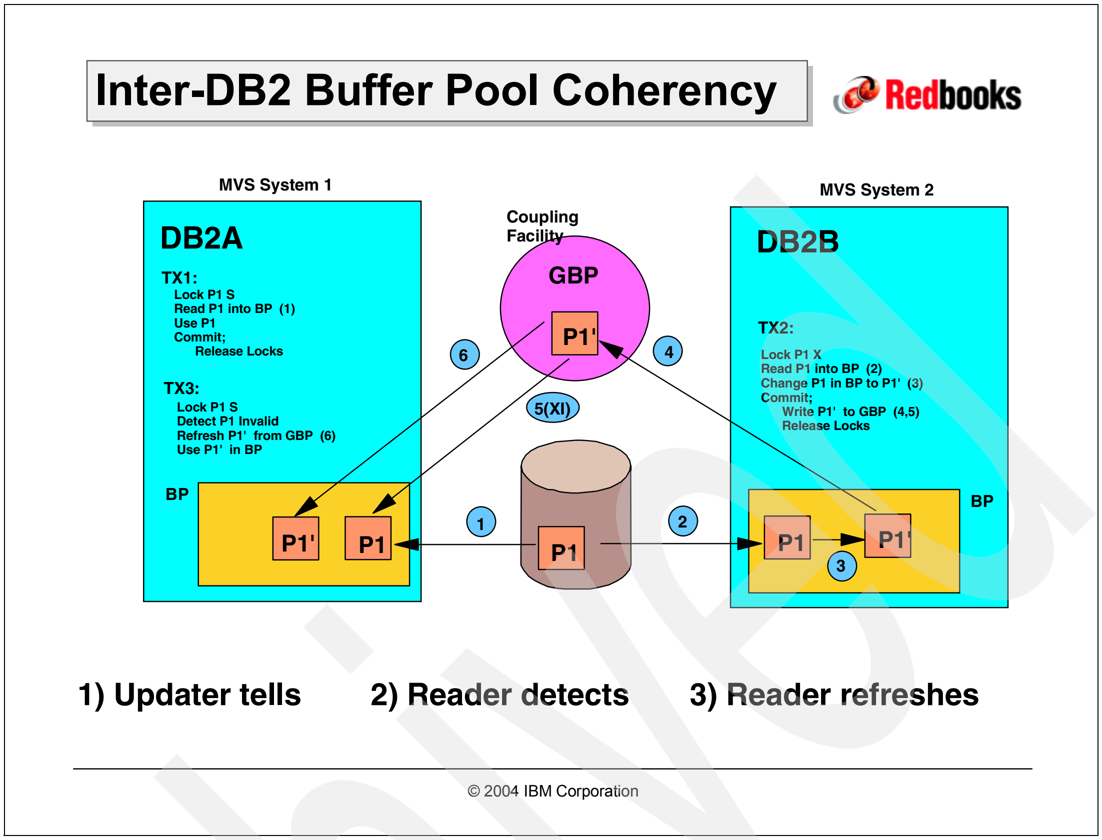
- Write/castout multiple pages to/from the CF in a single operation
- Reduce traffic to and from the CF
- Improve data sharing performance for most workloads, especially for batch
  - Workloads that update large numbers of pages for GBP-dependant objects

11.2 CF request batching

The current architecture allows multiple pages to be registered to the coupling facility with a single command. z/OS 1.4 and CF level 12 introduces two new “batch” processes to:

- Write And Register Multiple (WARM) pages of a group buffer pool with a single command.
- Read multiple pages from a group buffer pool for castout processing with a single CF read request. The actual command is called Read For Castout Multiple (RFCOM).

This enhancement reduces the data sharing overhead for most workloads. The most benefit is expected for workloads which update large numbers of pages belonging to GBP-dependent objects, for example, batch workloads.



11.2.1 Inter-DB2 buffer pool coherency

Applications can access data from any DB2 subsystem in the data sharing group. Many subsystems can potentially read and write the same data. DB2 uses special data sharing locking and caching mechanisms to ensure data consistency. This foil provides a brief overview of how shared data is updated and how DB2 protects the consistency of that data.

Suppose that an application issues an UPDATE statement from DB2A and that the data does not reside in the member's buffer pool or in the group buffer pool. In this instance, DB2A must retrieve the data from disk and get the appropriate locks to prevent another DB2 from updating the same record at the same time.

Because no other DB2 subsystem shares the table at this time, DB2 does not need to use data sharing integrity mechanisms to process for DB2A's update.

Next, suppose another application, running on DB2B, needs to update that same data page. Now inter-DB2 interest exists (both DB2A and DB2B are using this page set). After DB2B updates the data, it must move a copy of the data page into the group buffer pool (both primary and secondary) in the coupling facility, and the data page is invalidated in DB2A's buffer pool. Cross-invalidation occurs from the group buffer pool.

Now, when DB2A needs to read the data, the data page in its own buffer pool is invalid. Therefore, it reads the latest copy from the (primary) group buffer pool.

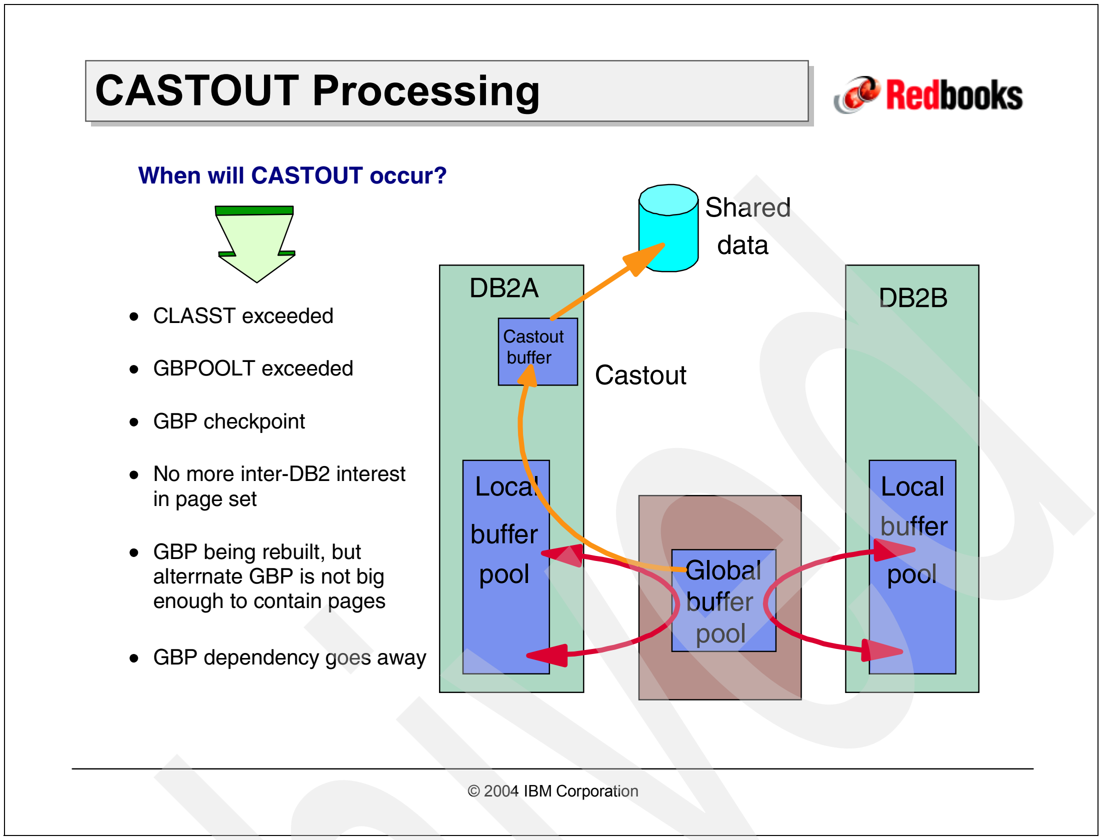
If the group buffer pool is allocated in a coupling facility with CFLEVEL= 0 or 1, then DB2 registers one page at a time in the group buffer pool.

When the group buffer pool is allocated in a coupling facility with CFLEVEL= 2 or higher, DB2 can register a list of pages that are being prefetched with one request to the coupling facility. This can be used for sequential prefetch (including sequential detection) and list prefetch.

DB2 does not include on the list any valid pages that are found in the local virtual buffer pool or hiperpool.

For those pages that are cached as “changed” in the group buffer pool, or those that are locked for castout, DB2 still retrieves the changed page from the group buffer pool one at a time. For large, sequential queries, there most likely won’t be any changed pages in the group buffer pool.

For pages that are cached as “clean” in the group buffer pool, DB2 can get the pages from the group buffer pool (one page at a time), or can include the pages in the DASD read I/O request, depending on which is most efficient.



11.2.2 CASTOUT processing

Periodically, DB2 must write changed pages from the group buffer pool to disk. This process is called *castout*.

There is no physical connection between the group buffer pool and DASD, so the castout process involves reading the pages from the group buffer pool into a group member's private buffer (not part of the member's buffer pool storage) and writing the page from the private buffer to DASD.

Castout is triggered when:

- A GBP checkpoint is taken.
- The GBP castout threshold is reached.
- The class castout threshold is reached.
- GBP dependency is removed for a page set.

Within a group buffer pool, there are a number of castout classes; the number of classes is an internal value set by DB2. Data sets (DB2 page sets or partitions) using the group buffer pool are mapped to a specific castout class. DB2 will preferably have only one data set assigned to a particular castout class, although it is possible to have more than one data set mapped into the same castout class, depending on how many data sets are using the group buffer pool concurrently.

Castout classes are used to limit the number of changed pages a data set can have in the group buffer pool at any one time, thereby limiting the amount of I/O to the data set at castout time. (Large amounts of I/O could cause DASD contention.) This limitation is achieved through the use of the castout class threshold.

The default value of the castout class threshold parameter is 5 (new default in V8 — the V7 default is 10), which means that castout is initiated for a particular class when 5% of the group buffer pool contains pages for that class or, if only one data set is assigned to that class, when 10 percent of the group buffer pool contains pages for that data set. The castout class threshold applies to all castout classes. You can change the castout class threshold by using the ALTER GROUPBUFFERPOOL command.

Data sets have a group buffer pool castout owner assigned to them. The group buffer pool castout owner is the first member to express write interest in the data set. After castout ownership is assigned, subsequent updating DB2 subsystems become backup owners. One of the backup owners becomes the castout owner when the original castout owner no longer has read-write interest in the page set or partition. At castout time, the castout owner is responsible for enforcing the actual castout process for all changed pages for the data set.

CF Request Batching

GBP Writes and CASTOUT processing

What are the exploitational prerequisites?

- New commands in z/OS 1.4
- CF control code shipped with CFLEVEL 12

z/OS 1.4 commands

- WARM - Write And Register Multiple command
  - Registers and writes multiple pages to a GBP
- RFCOM - Read For Castout Multiple
  - Read multiple pages from a GBP for CASTOUT processing

11.2.3 CF request batching

The current architecture allows multiple pages to be registered to the coupling facility with a single command.

z/OS 1.4 and CF level 12 introduces two new “batch” processes to:

- Write And Register Multiple (WARM) pages of a group buffer pool with a single command.
- Read multiple pages from a group buffer pool for castout processing with a single CF read request. The actual command is called Read For Castout Multiple (RFCOM).

When available, Version 8 data sharing will use these new CF commands to reduce the amount of traffic to and from the coupling facility for writes to group buffer pools and reads for castout processing, thus reducing the data sharing overhead for most workloads.

Benefits and Management

Reduced traffic to and from CF

Reducing data sharing overhead for most workloads

- Workloads updating large numbers of pages for GBP-dependent objects - BATCH

Currently uses the batched commands if >1 page

- Will change based on performance measurements

Statistics and accounting records updated for measurement

11.2.4 Benefits and management

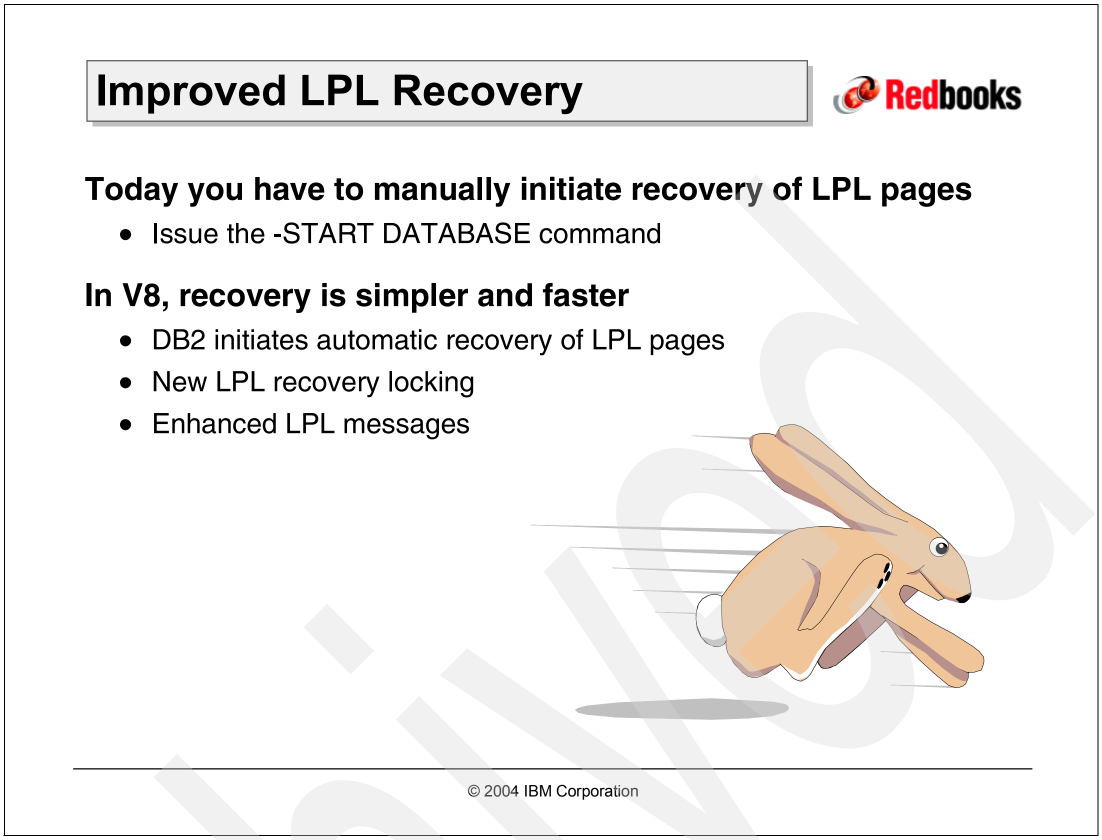
CF request batching allows Version 8 to reduce the amount of traffic to and from the coupling facility for both writes to group buffer pools and reads from the group buffer pool for castout processing, thus reducing the data sharing overhead for most workloads.

The most benefit is expected for workloads that update large numbers of pages belonging to GBP-dependent objects, for example, batch workloads.

CF request batching also benefits DB2 performance in other ways. DB2 commit processing performance is improved, where any remaining changed pages must be synchronously written to the group buffer pool during commit processing. For GPBCAHE(ALL) page sets, DB2 is able to more efficiently write prefetched pages into the group buffer pool as it reads them from DASD.

DB2 currently uses the CF batching commands to read and write pages to and from the group buffer pool if more than one page needs to be read or written. However, this behavior may change after further performance implications are known.

DB2 instrumentation (statistics IFCID 2 and accounting 3 and 148) records are enhanced to reflect the usage of these new commands. The DB2 PM and PE accounting and statistics reports are enhanced to externalize the new counters that indicate the number of WARM and RFCOM requests.



11.3 Improved LPL recovery

Prior to Version 8, you have to recover pages that DB2 put into the logical page list (LPL) manually. DB2 Version 8 automatically attempts to recover LPL pages as they go into LPL, when it determines that the recovery will probably succeed. (DB2 does not attempt automatic LPL recovery for pages put into LPL by disk I/O errors.)

Currently (V7 and before), LPL recovery requires exclusive access to the table space page set. Version 8 introduces a new serialization mechanism whereby LPL recovery is much less disruptive.

In addition, instrumentation has improved whereby message DSNB250E is enhanced to indicate the reason the pages are added to the LPL.

LPL Recovery Today

Redbooks

- Use DISPLAY DATABASE command to establish what page sets have pages on the LPL
- DB2 can automatically recover pages when group buffer pools are defined with AUTOREC(YES), the default
  - Only for GRECP, not LPL
- Issue the command START DATABASE
- Run the RECOVER utility on the object
- Run the LOAD utility with the REPLACE option on the object

None of the above work if retained locks held on the object.
You must restart any failed DB2 that is holding those locks.

11.3.1 LPL recovery today

The logical page list (LPL) contains a list of pages in logical error that could not be read or written for "must-complete" operations such as commit or a restart.

DB2 can put pages into logical error status and place them in LPL for a number of reasons:

- Transient disk read and write problems that can be fixed without redefining new disk tracks or volumes, (data sharing and non-data sharing)
- A problem with the coupling facility (CF)
- Channel failure to the CF
- Channel failure to DASD
- Locks being held by a failed subsystem, preventing access to the desired page

DB2 customers are demanding higher and higher levels of availability. However, once a page is entered into the LPL, that page is inaccessible until it is recovered.

The LPL is kept in the DBET and therefore in the SCA in data sharing environments. The information is therefore accessible to all members. Applications requiring access to data in LPL will receive the usual "resource unavailable" SQLCODE.

The -DISPLAY DATABASE command can be used to find out what page sets have pages in LPL which must be recovered:

```
-DB1G DIS DB(DSNDB01) SPACENAM(*) LIMIT(*) LPL ONLY
```

If LPL entries exist, you need to manually issue the START DATABASE command with the SPACENAM option, to initiate LPL recovery, for example:

```
-DB1G STA DB(DSNDB01) SPACENAM(*) ACCESS(RW)
```

DB2 will then read the DB2 log and apply any changes to the page set. The -START DATABASE command drains the entire page set or partition, therefore making the entire page set or partition unavailable for the duration of the LPL recovery process, even if only one page is in the LPL for that page set or partition.

The RECOVER and LOAD utilities can also be used to recover LPL pages. If the START DATABASE command fails to successfully recover the LPL pages, you are forced to recover the whole page set using the RECOVER utility.

Automatic LPL Recovery

Redbooks

Recovery is simplified and faster

- Requires less manual intervention

DB2 initiates automatic recovery of LPL pages

- At point when pages are added to the LPL
- Automatic recovery not initiated for ALL LPL situations

Enhanced messages:

- As pages are added to LPL, DSNB250E shows the reason
- During LPL recovery (automatic or not) DSNI0042I, DSNI0043I and DSNI0044I with log record ranges being applied
- (Automatic) LPL recovery has started, DSNI006I
- Automatic LPL recovery suppressed, DSNB357I
- (Automatic) LPL recovery progress message(s), DSNI022I
- (Automatic) LPL recovery successful, DSNI021I
- (Automatic) LPL recovery failed, DSNI005I
  - Recover as per Version 7

11.3.2 Automatic LPL recovery

DB2 Version 8 will automatically attempt to recover pages that are added to the LPL at the time that they are added to LPL. When the pages are added into LPL, DB2 issues message DSNB250E, to indicate the LPL page range and the names of the database, the page set or partition, and the reason for adding the page to the LPL.

Automatic LPL recovery is not initiated by DB2 in the following situations:

- DASD I/O error
- During DB2 restart or end_restart time
- GBP structure failure
- GBP 100% loss of connectivity.

DB2 issues the DSNI006I message to indicate the start of automatic LPL recovery, otherwise DB2 issues the message DSNB357I to indicate the reason why the automatic LPL recovery processor is suppressed.

If the automatic LPL recovery runs successfully, LPL pages are deleted from the LPL and DB2 issues the message DSNI021I to indicate the completion.

If the automatic LPL recovery does not run successfully, pages are kept in the LPL and DB2 issues the DSNI005I to indicate the failure of the automatic LPL recovery. You must now recover the pages in LPL manually, as in Version 7.

To recover the LPL pages manually, you will need to first check for the reason type and take action based on console messages for any system conditions. To recover pages from the LPL, perform one of the following actions:

- Issue the START DATABASE command with the SPACENAM option.
- Run the RECOVER utility or LOAD utility with REPLACE option.

Less Disruptive LPL Recovery

What happens today?

- Issuing the -START DATABASE command, drains the entire page set or partition
  - Drain forces requesting users to wait for completion of recovery
  - Very disruptive especially where only a few pages exist in the LPL and/or DB2 must go far back in the logs to apply all the changes

New support

- Less disruptive in the automatically initiated LPL recovery and -START DATABASE command processing
  - Avoids the DRAIN operation

11.3.3 Less disruptive LPL recovery

Today, when you issue the -START DATABASE command to recover LPL pages, the command must drain the entire page set or partition. The "drain" means that the command must wait until all current users of the page set or partition reach their next commit point.

All users requesting new access to the page set or partition are also suspended and must wait until the recovery completes (or until the user times out). Therefore, the drain operation can be very disruptive to other work that is running in the system, especially in the case where only one or a few pages are in LPL.

In Version 8, the -START DATABASE command, and also the automatic LPL recovery function we have just talked about, now help to avoid the drain operation so that the recovery of the LPL pages can be done much less disruptively.

New LPL Recovery Locking

Acquire a WRITE CLAIM instead of DRAIN ALL

- Other pages can still be accessed during LPL recovery
- Serializes with utility functions like RECOVER or LOAD, which can also resolve LPL recovery
- Serialize with DDL

A conditional X mode LPL recovery lock with COMMIT duration

- Acquired by the automatic LPL recovery process
- Released when LPL processing finishes
- Manual -START DATABASE method also acquires the conditional X mode lock
  - Serializes with the automatic LPL recovery processor
  - Serializes with other -START DATABASE commands

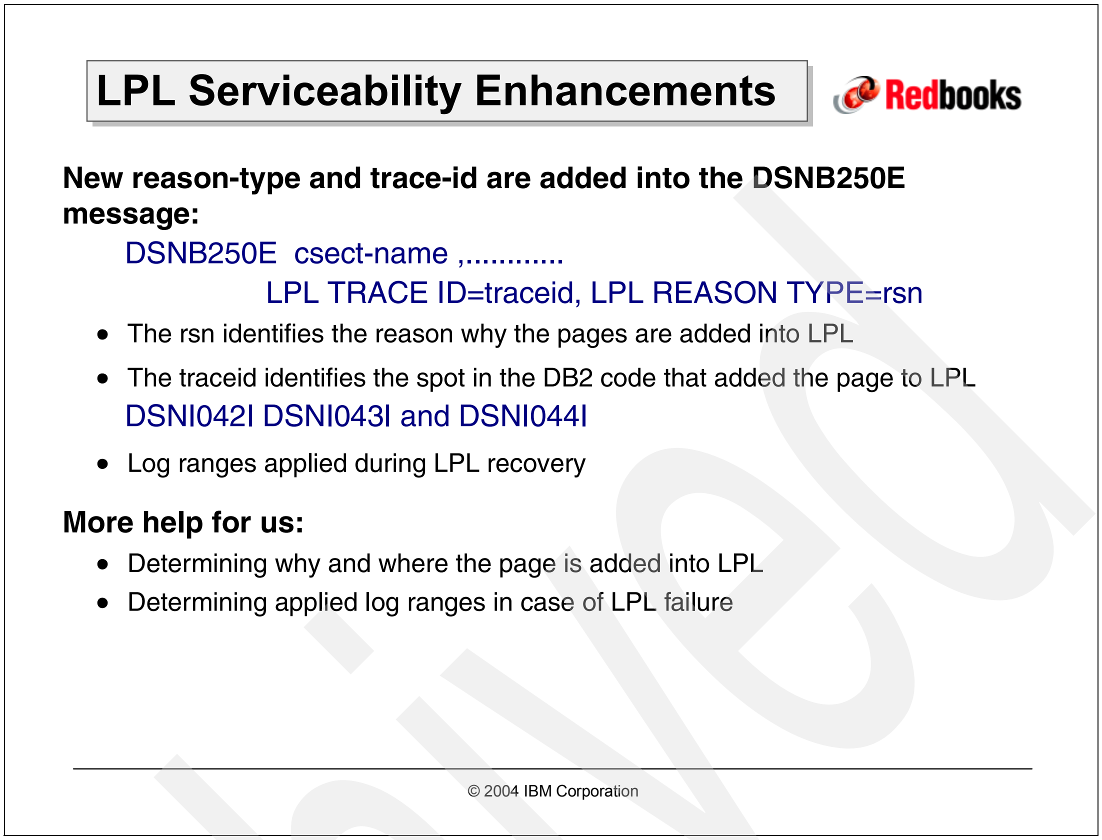
11.3.4 New LPL recovery locking

The locking and serialization schemes in the -START DATABASE command have changed when doing the LPL recovery. DB2 V8 makes a WRITE CLAIM on the page set or partition. In prior versions of DB2, the -START DATABASE command acquires a DRAIN ALL lock on the page set or partition when doing the LPL recovery. By acquiring a WRITE CLAIM instead of a DRAIN ALL, the “good” pages can still be accessed by SQL while the -START DATABASE is recovering the LPL pages.

This new locking strategy is implemented for both automatic LPL recovery and LPL recovery resulting from a -START DATABASE command.

The WRITE CLAIM serializes with the utility functions like RECOVER or LOAD, which can also recover objects from an LPL pending status. The WRITE CLAIM also lets automatic LPL recovery and the -START DATABASE command serialize with the -STOP DATABASE command and the DROP TABLESPACE statement which currently acquire DRAIN ALL lock.

A new "LPL recovery" lock type is also introduced to enforce that only one LPL recovery process is running at a time for a given page set or partition. The conditional X mode LPL recovery lock must be acquired by the LPL recovery process and released when it finishes the job. If an LPL recovery process is already in progress when a subsequent one is initiated (either automatically or manually), then the second recovery process is not scheduled. It is blocked by the conditional LPL recovery lock already held. When the first LPL recovery process completes, it will check for more work that is outstanding on the same page set or partition before it terminates.



11.3.5 LPL serviceability enhancements

Currently, message DSNB250E is issued whenever a page is added to LPL, but this message does not provide sufficient information to know exactly why DB2 decided that the page should be added to LPL. Many times customers report that they have LPL pages, but it is not apparent why they encountered LPL pages in the first place. Knowing why a page is added to LPL is the first step to avoiding pages being added to LPL in the future.

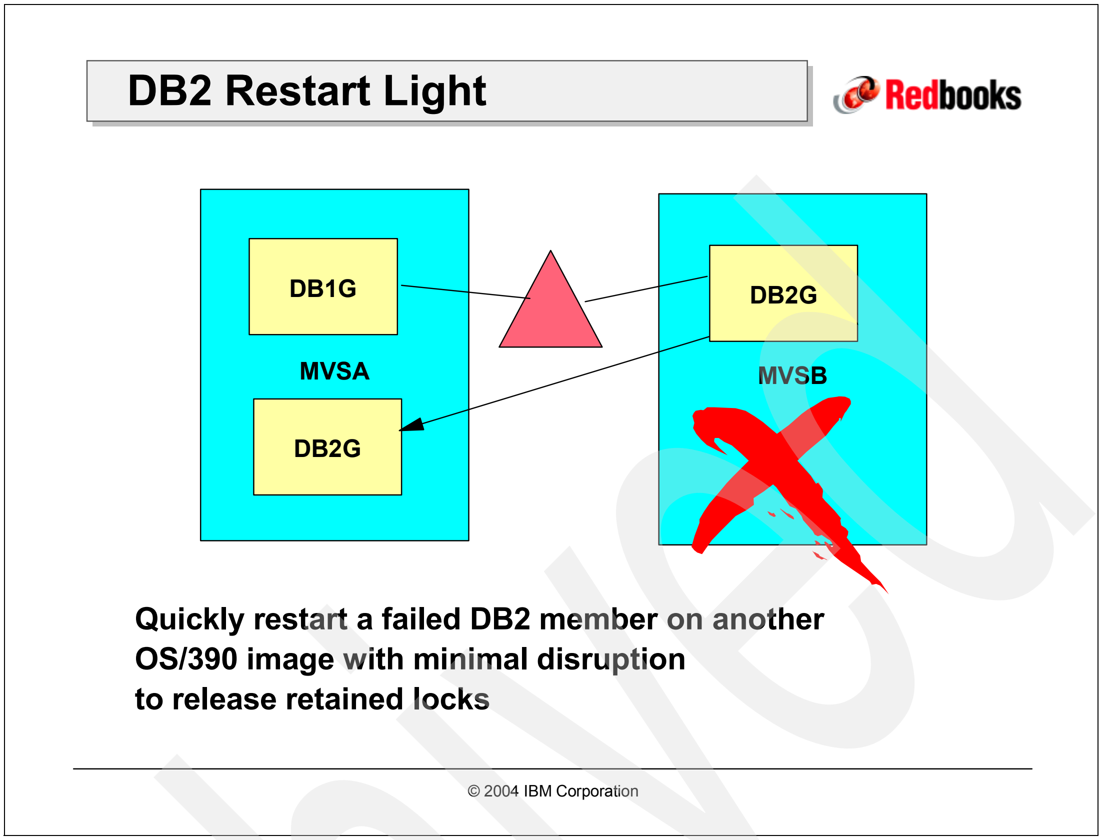
DB2 now provides more detailed information as to why a page has been added to the LPL. A new reason type and a new trace id are added to the message DSNB250E. The new reason type will explain why the pages are added into LPL. The reason types reported in message DSNB250E are as follows:

- **DASD:** DB2 encountered a DASD I/O error when trying to read or write pages on DASD.
- **LOGAPPLY:** DB2 cannot apply log records to the pages.
- **GBP:** DB2 cannot successfully read/externalize the pages from/to the group buffer pool due to link or structure failure, GBP in rebuild, or GBP was disconnected.
- **LOCK:** DB2 cannot get the required page latch or page P-lock on the pages.
- **CASTOUT:** The DB2 Castout processor cannot successfully cast out the pages.
- **MASSDEL:** The DB2 encountered an internal error at the mass delete processor during the phase 2 commit.

In addition, during LPL recovery (automatic or not), DB2 will produce additional messages indicating the log ranges that are being applied during LPL recovery. The following three new messages can be seen:

- **DSNI042I:** This message displays the header page RBA that is used to determine the LPL or GRECP recovery range for the specified page set. It is displayed once per LPL recover, per page set, per data sharing group.
- **DSNI043I:** This message displays the broad LRSN or RBA range, merged from all members of the data sharing group. The range is used to determine the LPL or GRECP recovery range for the specified page set. It is displayed once per LPL recovery, per member.
- **DSNI044I:** This message displays the LRSN or RBA range that is used to determine LPL or GRECP recovery range for the specified page set in the data sharing group member. This message is displayed once per LPL recovery, per page set, per member.

These messages should allow you to identify more easily which log records have been applied by LPL recovery, in case LPL recovery fails, and you need to determine the cause of the failure.



11.4 DB2 restart light

DB2 Version 7 introduced the concept of DB2 “restart light”, which is intended to remove retained locks with minimal disruption in the event of an MVS system failure. When a DB2 member is started in restart light mode (-START DB2 LIGHT(YES)), DB2 comes up with a small storage footprint, executes forward and backward restart log recovery, removes the retained locks, and then self-terminates without accepting any new work.

However, retained locks that pertain to any indoubt units of recovery (URs) will persist, and the indoubt URs remain in the failed member's log. The data protected by these retained locks is not available to any other DB2 member until the indoubt URs have been resolved.

Restart Light Enhancements

Now attempts to resolve in doubt URs

- Remains up after successful restart
- DDF will start if allowed in DSNZPARM
- Only in doubt UR commit coordinators allowed to connect
  - Otherwise RC00F300A2
- Can issue -RECOVER INDOUBT to manually resolve the URs

Once all in doubt URs are resolved

- DB2 will terminate normally

11.4.1 Restart light enhancements

Restart light is improved in DB2 Version 8 to handle indoubt units of recovery.

When DB2 is started with LIGHT(YES) and indoubt URs exist at the end of restart recovery, DB2 will now remain up and running so that the indoubt URs can be resolved, either automatically via resynch processing with the commit coordinators or manually via the -RECOVER INDOUBT operator command. DB2 will also issue a new message, DSNR052I, to indicate that a LIGHT(YES) DB2 is remaining up and running to resolve indoubt URs.

If DDF startup is allowed via DSNZPARM, Restart Light will also start DDF to facilitate the resolution of any distributed indoubt URs. However, no new DDF connections are allowed. Clients that attempt to connect to a restart light DB2 will be rejected with a return code indicating that MAXDBAT has been reached. Only resynch requests will be allowed from DB2 clients.

As with previous versions of DB2, when DB2 is started with LIGHT(YES), it starts with only a small storage footprint and cannot support SQL requests. For example, the EDM pool has not been initialized (since it is not needed for log recovery or removing retained locks). Therefore, a LIGHT(YES) DB2 member that remains up and running to resolve indoubt URs will not accept any new connection requests, except those that originate from connection names that have indoubt URs.

If an attempt is made to connect to a LIGHT(YES) DB2 from a connection name that does not have indoubt URs, then return code 8 with new reason code 00F300A2 is returned with SQLCODE -923 (similar to the 00F30056 that gets returned for ACCESS MAINT). If 00F300A2 is received from the DSN command processor, then new message DSNE136I is issued (similar to DSNE132I for ACCESS MAINT).

A connection name with indoubt URs is allowed to connect to the LIGHT(YES) DB2, but it is not allowed to create a thread. If create thread is attempted, then return code 8, reason code 00F300A2 is returned with SQLCODE -923.

Connection requests using the group attach name will not attempt to connect to a DB2 member that is started with LIGHT(YES). Connectors wanting to resynch with a LIGHT(YES) DB2 member must use that member's subsystem name to connect. (A DB2 member started in light mode will not post startup ECBs that are associated with the group attach name. It will only post startup ECBs that are associated with that member's subsystem name.)

For example, the RESYNCHMEMBER(YES) option added in CICS TS 2.2 causes CICS to force re-connection back to the original DB2 member (using that member's subsystem name instead of the group attach name) should CICS think that indoubt URs are outstanding for the last member connected to.

While DB2 remains up and running in light mode, the -DISPLAY THREAD command can be used to monitor the progress of the indoubt resolution and to display the detailed information about any indoubt URs that still exist. Also, the -RECOVER INDOUBT command can be used to manually resolve indoubt URs. However, the following commands are not allowed (new message DSN9038I):

- DISPLAY, START, STOP DATABASE
- DISPLAY, START, STOP RLIMIT
- SET SYSPARM

Once the final indoubt UR has been resolved, DB2 issues new message DSNR053I to indicate that there are no remaining incomplete URs, and that the DB2 member will self-terminate via the normal DB2 shutdown process. Alternatively, you can manually shut down DB2 running in light mode, with the -STOP DB2 command. If this is done and there still exist indoubt URs, then existing message DSNR046I is issued to inform you that incomplete URs still exist.

Change to IMMEDWRITE option

Default 'Immediate Write' process changed:

- Write changed pages during phase 1 of commit processing
  - Previously in data sharing default was to write during phase 2

ZPARM IMMEDWRI(PH1) option removed

BIND IMMEDWRITE(PH1) option kept for compatibility

CPU cost of writing changed pages to GBP charged to allied TCB

- Previously IMMEDWRITE NO option charged cost to MSTR SRB
  - Pages were written as part of phase 2 commit processing

Still can use IMMEDWRITE(YES)

- For improved concurrency for spawned applications
  - Can then see the update from the previous transaction
- Beware of performance impact

11.5 Change to IMMEDWRITE option

Consider the situation where one transaction, updates DB2 data using INSERT, UPDATE, or DELETE, and then, before completing (phase 2 of) commit, spawns a second transaction that is dependent on the updates that were made by the first transaction. This type of relationship is referred to as “ordered dependencies” between transactions.Consider the following scenario.

We have a two way data sharing group DB0G with members DB1G and DB2G. Transaction T1, running on member DB1G, makes an update to a page. Transaction T2, spawned by T1 and dependent on the updates made by T1, runs on member DB2G. If transaction T2 is not bound with isolation repeatable read (RR), and the updated page (on DB1G) has been used previously by DB2G and is still in its local buffer pool, there is a chance, due to lock avoidance, that T2 uses an old copy of the same page in the virtual buffer pool of DB2G if T1 still has not committed the update.

Here are some possible work-arounds for this problem:

- Execute the two transactions on the same member.
- Bind transaction T2 with ISOLATION(RR).
- Make T1 commit before spawning T2.

DB2 V5 APAR PQ22895 introduced a new bind/rebind option that can be considered when none of the above actions are desirable. IMMEDWRITE(YES) allows the user to specify that DB2 should immediately write updated GBP dependent buffers to the Coupling Facility instead of waiting until commit or rollback.

DB2 V6 APAR PQ25337 delivers the functionality introduced by APAR PQ22895 in DB2 V5 with the addition of a third value for IMMEDWRITE and a new DSNZPARM parameter. IMMEDWRITE(PH1) allows the user to specify that a given plan or package should write updated group buffer pool dependent pages to the Coupling Facility at or before Phase 1 of commit. If the transaction subsequently rolls back, the pages will be updated again during the rollback process and will be written again to the CF at the end of abort. This option is only useful if the dependent transaction is spawned during syncpoint processing of the originating transaction.

In prior versions of DB2, changed pages in a data sharing environment are written during phase 2 of the commit process, unless otherwise specified by the IMMEDWRITE BIND parameter, or IMMEDWRI DSNZPARM parameter.

This enhancement changes the default processing to write changed pages during phase 1 of commit processing. The options you can specify for the IMMEDWRITE BIND parameter remain unchanged. However, whether you specify “NO”' or “PH1”, the behavior will be identical, changed pages are written during phase 1 of commit processing. The “PH1” option remains for compatibility reasons, but its usage should be discouraged. The DSNZPARM IMMEDWRI parameter will no longer accept a value of “PH1”. With this change pages are either written at (the latest at) phase 1, never at phase 2 of the commit processing.

The impact of IMMEDWRITE YES remains unchanged. Changed pages are written to the group buffer pool as soon as the buffer updates are complete (so definitely before committing). Specifying this option may impact performance.

With IMMEDWRITE NO (or PH1) and YES options, the CPU cost of writing the changed pages to group buffer pool is charged to the allied TCB. Prior to Version 8, this was true only for PH1 and YES options. For the NO option, the CPU cost of writing the changed pages to group buffer pool was charged to MSTR SRB, since the pages were written as part of phase 2 commit processing under MSTR SRB.

This enhancement provides a more accurate accounting for all DB2 workloads. DB2 is now able to charge more work back to the user who initiated the work in the first place.

Attention: Customers who use the allied TCB time for end user charge back may see additional CPU cost with this change.

The IMMEDWRITE enhancements are immediately available when you migrate to DB2 Version 8. You do not have to wait until new-function mode.

-DIS GBPOOL Enhancement

Current CFLEVEL displayed is the 'operational' level

- Typically this is the level requested by DB2 during 'connect' processing
- May be lower than the **'actual'** CFLEVEL displayed by the D CF command

DB2 Version 8 -DIS GBPOOL output will now display both CFLEVELs:

- **'Operational'**
  - Indicating the capability of the CF from a DB2 functionality perspective
  - If GBP duplexed, CFLEVEL equates to primary GBP
- **'Actual'**
  - The CFCC level as displayed by the D CF command
  - If GBP duplexed, CFLEVEL equates to primary GBP

11.6 -DIS GBPOOL enhancement

Currently, the CF level displayed by the **-DISPLAY GROUPBUFFERPOOL** command may be lower than the actual CF level as displayed by a **D CF** command.

This enhancement changes **-DISPLAY GROUPBUFFERPOOL** command output. Instead of having the CFLEVEL field, the command now displays both, the OPERATIONAL CF LEVEL as before, and also the ACTUAL CF LEVEL.

The operational CF level indicates the capabilities of the CF from DB2's perspective. The actual CF level is the CF control code level as displayed by the **D CF** command.

When DB2 connects to the coupling facility, it requests a CF “function” level. DB2 Version 7 requests a CF level of 7 and DB2 Version 8 requests a CF Level of 12. The CF level determines the level of function DB2 is able to use when it interacts with the coupling facility.

Note: The first DB2 member to connect to the coupling facility determines the CF level that the data sharing group will use.

Note for mixed release groups:

- If the first member to connect to the coupling facility is Version 7, then the actual CF level will be 7. Subsequent DB2 members (Version 7 or Version 8) interact with the coupling facility using CF level 7. Implications here are that DB2 Version 8 cannot use CF request batching.
- If the first member to connect to the coupling facility is Version 8, then the requested CF level will be 12. Subsequent DB2 members (Version 7 and Version 8) can interact with the coupling facility using CF level 12. Implications are that the Version 8 members can use CF request batching. (DB2 Version 7 cannot because CF request batching is not implemented.)

In addition, a number of other messages in the -DIS GBPOOL output have been tidied up. For example, some counters related to secondary GBP have been removed (because page writes to the secondary group buffer pool are always the same as writes to the primary group buffer pool).

The messages that have changed include: DSNB758I, DSNB762I, DSNB764I, DSNB775I, DSNB776I, DSNB777I, DSNB777I, DSNB779I, DSNB786I, DSNB787I, DSNB789I, and DSNB799I.

Attention: If you have any automation of programs in place to interrogate the output of the -DISPLAY GROUPBUFFERPOOL command, we recommend you to review these facilities, as the output from the command has significantly changed.

-DIS GBPOOL Output

Redbooks

```
  -DIS GBPOOL(GBP0) MDETAIL(INTERVAL) GDETAIL(INTERVAL)
DSNB750I  =DBA1 DISPLAY FOR GROUP BUFFER POOL GBP0 FOLLOWS
DSNB755I  =DBA1 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  =DBA1   CLASS CASTOUT THRESHOLD                  = 10%
          GROUP BUFFER POOL CASTOUT THRESHOLD          = 50%
          GROUP BUFFER POOL CHECKPOINT INTERVAL        = 8 MINUTES
          RECOVERY STATUS                              = NORMAL
          AUTOMATIC RECOVERY                           = Y
DSNB757I  =DBA1 MVS CFRM POLICY STATUS FOR DSNDB2A_GBP0    = NORMAL
          . . . . . . .
DSNB758I  =DBA1     ALLOCATED SIZE                         = 8192 KB
           VOLATILITY STATUS                           = VOLATILE
           REBUILD STATUS                              = NONE
           CFNAME                                      = NSD1CF
           CFLEVEL - OPERATIONAL                       = 12
           CFLEVEL - ACTUAL                            = 12
DSNB759I  =DBA1     NUMBER OF DIRECTORY ENTRIES                = 6628
           . . . . . . . .
```

- **DISPLAY GBPOOL(GBP0) MDETAIL(INTERVAL) GDETAIL(INTERVAL)**

11.6.1 -DIS GBPOOL output

Here is a partial output from the -DISPLAY GROUPBUFFERPOOL command, showing the operational and actual CFLEVEL.

12

Installation and migration

List of Topics

- Planning for Version 8
- Installation
- Migration and fallback
- DB2 catalog changes
- msys for Setup DB2 Customization Center
- Samples
- DB2 Version 8 packaging

Version 8 of DB2 brings major changes to the installation and migration processes.

In this chapter we describe these changes. We assume that you are already familiar with the installation and migration procedures used by earlier versions of DB2. Please refer to the *DB2 UDB for z/OS Version 8 Installation Guide,* GC18-7418 and the *DB2 UDB for z/OS Version 8 Data Sharing Planning and Administration Guide* (SC18-7417) for more details.

We define *installation* as the process of installing a *new* DB2 subsystem. In this case there are no compatibility and regression issues. With a newly installed DB2 Version 8 subsystem, you can immediately take advantage of all the new functions in Version 8.

Migration is the process of converting an *existing* DB2 Version 7 subsystem, user data, and catalog data, to Version 8. This process is changed with Version 8 of DB2 in order to minimize the possible impact of regression and fallback incompatibilities.

The key changes to the installation and migration processes are as follows:

- Valid CCSIDs must be defined for ASCII, EBCDIC, and Unicode.
- You must supply your own tailored DSNHDECP module. You can no longer start DB2 with the DB2-supplied DSNHDECP.
- DSNHMCID is a new data-only module in V8. It is required by DB2 facilities that run in DB2 allied address spaces, such as attaches and utilities.
- Buffer pools of sizes 4 KB, 8 KB, 16 KB, and 32 KB must be defined.
- Only WLM-established stored procedures can be defined.
- The migration process now consists of three distinct phases:
  - **CM: compatibility mode:** During this phase, which can last as long as deemed necessary, you should execute all the tests needed to ensure that you happy with the new version and will not have to fallback to Version 7 later on. In CM, you are able to fallback to Version 7 in case of problems.
  - **ENFM: enabling-new-function mode:** During this phase you will convert the DB2 subsystem to the format ready to support the new function in Version 8, by using the on-line REORG Utility. No fallback to DB2 Version 7 is allowed once ENFM is entered.
  - **NFM: new-function mode:** This is the target phase, triggered when you execute a job confirming that all the previous conversion steps have completed successfully and update the DB2 subsystem or data sharing group as being in new-function mode.

These phases are discussed in more detail under the following topics:

- Planning for version 8
- Installation
- Migration and fallback
- DB2 catalog changes
- msys for Setup DB2 Customization Center
- Samples
- DB2 packaging

Planning for Version 8

Review the available documentation

- Preventive service planning for latest maintenance
- DB2 Version 8 Installation Guide
- DB2 Version 8 Program Directory

Review planning considerations

- What has changed since DB2 Version 7 ?

To migrate - must have the SPE applied - PQ48486

- DB2 enforces for both data sharing and non-data sharing
  - New for non-data sharing
- PQ48486 -> PTF UQ81009 (11/03)
- See also info APAR II13695

12.1 Planning for Version 8

> **Attention:** Version 8 is the first release of DB2 to fully exploit the new 64-bit hardware and 64 bit operating system. It therefore comes with some firm hardware and software prerequisite requirements that did not exist in previous versions of DB2. Planning for this new version is more important than ever.

In the following visuals we introduce the major z/OS prerequisites for DB2 Version 8. We will also introduce the significant DB2 prerequisites and release incompatibilities.

Before you migrate to any new version of DB2, it is very important that you fully understand what has changed from the previous version. This is no less important for DB2 Version 8. Please refer to the DB2 UDB for z/OS Version 8 Program Directory for an up-to-date list of prerequisites for Version 8, and to the *DB2 UDB for z/OS Version 8 Installation Guide,* SC18-7418 for a complete list of release incompatibilities.

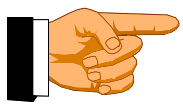
It is also important to keep up-to-date with current software maintenance levels. This is even more important with DB2 Version 8. DB2 now enforces that you have the correct prerequisite maintenance implemented before it will allow you to migrate to Version 8. DB2 now enforces this requirement for both data sharing and non-data sharing environments.

The V8 fallback SPE for V7 is PQ48486. Its PTF, UQ81009 has been available since early November 2003. You must start DB2 V7 at least once with the fallback SPE applied before you migrate to Version 8. There is also an Informational APAR II13695 that is used to document any important migration and fallback topics.

z/OS Prerequisites

Important prerequisites exist for DB2 for z/OS Version 8

- zSeries box (z800 or z890 or z900 or z990)
- z/OS 1.4 or above (z/OS 1.3 - out of service since March 31 2005)
  - z/OS 1.3 requires WLM Goal Mode
- z/OS running in 64 bit mode



WARNING!
Disaster Recovery

Some functions will require z/OS 1.4 or later

- System level point in time functionality requires z/OS 1.5 plus data storage control units with Flashcopy support
- Multi-level (row level) security requires z/OS 1.5 SecureWay Server (RACF) or equivalent
- Up to 100,000 open data sets requires z/OS 1.5
- CF request batching in data sharing required z/OS 1.4 and CF Level 12

12.1.1 z/OS prerequisites

Version 8 can only run on z/Architecture machines and requires that those machines are running in 64 bit addressing mode. If an attempt is made to start DB2 Version 8 on a non-64 bit machine, DB2 issues an error message during startup and self terminates.

For z900 systems, microcode level 3G or later is required. For emulation packages or other comparable processors, the Long Displacement Facility of the z/Architecture must be available. For further information about determining or updating your microcode level, search for the word “microcode” from the Web page at:

http://www.ibm.com/software/db2zos/support.html

DB2 version 8 also requires z/OS V1R3 or above, or more precisely z/OS V1.3 Base Services (5694-A01) or z/OS.e (5655-G52) plus APARs OW56073, OW56074, OA03519, OA03095, OA04043, and OA04069 with DFSMS V1.3, Language Environment and z/OS V1.3 Security Server (RACF). If an attempt is made to start DB2 Version 8 on an OS/390 or a z/OS R1 or R2 system, then once again DB2 issues an error message during startup and self terminates.

> **Note:** When DB2 V8 became generally available in March 2004, z/OS V1.3 was the minimum z/OS level required for DB2 V8. Since March 31 2005, z/OS V1.3 is out of service. Therefore the actual minimum z/OS level to run DB2 V8 is now z/OS V1.4.

Attention: Please note that these prerequisites have important implications for disaster recovery and sysplex cross-system restart scenarios. All systems that DB2 Version 8 runs on or may run on in an emergency or test scenario must meet these hardware and software prerequisites. So you really need to plan to have these important prerequisites in place in all the environments where you may need to start DB2 Version 8, before you install Version 8 in any environment.

In addition, some new functions in Version 8 have further prerequisites:

- The new DB2 System level backup and recovery solution requires DFSMS shipped with z/OS 1.5 plus data storage control units which support Flashcopy.
- The multi-level security functionality requires the z/OS 1.5 Secureway Server (RACF) or equivalent.
- When you are using z/OS 1.5 you have the possibility to increase the maximum number of open data sets (DSMAX) to 100 000. When running on z/OS 1.3 or 1.4, DB2 will enforce limit of 32 767 open data sets. For more information, see 2.28.3, "Up to 65 000 open data sets" on page 51.
- The data sharing enhancement where DB2 can batch page requests to the group buffer pool, require z/OS 1.4 and CF Level 12.
- The lock holder priority increase enhancement requires z/OS 1.4 WLM functionality.

Once again, please refer to the DB2 UDB for z/OS Version 8 Program Directory for an up-to-date list of these functional prerequisites.

Despite the work DB2 Version 8 has done in removing many restrictions which contribute to memory constraints and the new 64 bit architecture, not all of the restrictions have been removed, and some memory constraint problems may still occur for very large systems. DB2 V8 has gone a long way; however, there is still more work to be done.

DB2 Version 8 will require some more memory than Version 7, typically around 10%. This is largely because many memory structures have changed to support 64 bit addressing. Although Version 8 introduces many changes designed to provide memory relief, with many memory structures moving above the bar (for example; bufferpools, RID pool, sort pool, EDM pool, and compression dictionaries), you are not isolated from some memory constraints in very large systems as many thread related structures still remain below the bar. In addition, DB2 is not the only user of the real memory on the machine. It has to be shared by all other subsystems and applications on the LPAR.

We therefore suggest that after you migrate to Version 8, you continue to monitor the paging rates on your system. This will give an indication of your memory usage and show if it is over-committed.

Other Prerequisites

Redbooks

Customize the z/OS Conversion Services

- Used to convert to and from Unicode for example

COBOL

- Migrate to IBM Enterprise COBOL V3.2 or above
  - Older COBOL compilers cannot be used with DB2 V8
    - No OS/VS COBOL or VS COBOL II
  - Can run older COBOL load modules under LE

PL/I

- Migrate to Enterprise PL/I for z/OS and OS/390 V3R2 (or above) or PL/I for MVS & VM V1R1
  - All versions of OS PL/I (including 2.3) and VisualAge PL/I no longer supported

Obtain latest version of other products

- IMS V7 or above
- CICS TS 1.3, 2.2 or 2.3

12.1.2 Other prerequisites

In order to be able to run DB2 Version 8, a number of other prerequisites must also be in place, such as Unicode conversion services and programming languages have to be at a certain level as well to be able to work with the V8 precompiler.

Unicode Conversion Services

You need to define and customize the z/OS Conversion Services as described in the *z/OS V1R3.0 Support for Unicode Using Conversion Services*, SA22-7649-01. DB2 uses this z/OS service to convert to and from Unicode.

The z/OS Conversion Services must be configured and active *before* you migrate DB2 to Version 8 compatibility mode. Even in compatibility mode, when no new function is enabled, DB2 needs to convert all the SQL to Unicode in order to process it. All SQL statements are parsed in Unicode. In fact, DB2 Version 8 will not start if there is no conversion available to and from the EBCDIC and ASCII CCSIDs defined in DSNHDECP and UTF-8 (1208).

We have suggested a sample definition for customizing the z/OS Conversion Services in Appendix B, “Unicode setup” on page 1039. Further information can be found in Appendix A of the *DB2 UDB for z/OS Version 8 Installation Guide,* SC18-7418 and the Information APARs II13048 and II13049.

COBOL support

DB2 has a commitment to support currently supported releases of other IBM software. However, OS/VS COBOL and COBOL II are no longer supported by IBM, and therefore DB2 Version 8 is also removing support for these products. Only the following COBOL compilers are supported by the DB2 precompiler; however, older COBOL load libraries can still run with DB2 and LE:

- Enterprise COBOL for z/OS and OS/390 Version 3 (5655-G53).

When using the integrated SQL coprocessor, you should use:

- Enterprise COBOL for z/OS and OS/390 V3.2 or V3.3 (5655-G53) with APAR PQ83744

See also the following Techdocs, available at:

  – `http://www.ibm.com/support/docview.wss?rs=64&context=SSEPEK&q1=v8&uid=swg21164264&loc=en_US&cs=utf-8&lang=en`

  – `http://www.ibm.com/support/docview.wss?rs=64&context=SSEPEK&q1=v8&uid=swg21166881&loc=en_US&cs=utf-8&lang=en`

In case you are not on any of these releases, you need to migrate. The key resource is the *Enterprise COBOL for z/OS Compiler and runtime migration guide*, GC27-1409. It can be foung on the Web at:

`http://www.ibm.com/software/awdtools/cobol/zos/library/`

> **Attention:** IBM COBOL for OS/390 Version 2 Release 2 (5648-A25) is out of service since December 2004.
>
> Note also that Enterprise COBOL for z/OS and OS/390 V3.2 is going out of servive in October 2005.

PL/I

To use PL/I with DB2 Version 8, you should use any of the following products:

- IBM Enterprise PL/I for z/OS and OS/390 V3.2 or above (5655-H31)
- IBM PL/I for MVS & VM V1.1 (5688-235)

Using the DB2 precompiler services requires the DB2 coprocessor provided with:

- IBM Enterprise PL/I for z/OS and OS/390 V3.2 and APAR PQ84513 or later releases.

See also the following Techdoc, available at:

  – http://www.ibm.com/support/docview.wss?rs=64&context=SSEPEK&q1=v8&uid=swg21166873&loc=en_US&cs=utf-8&lang=en

> **Attention:** Enterprise PL/I for z/OS and OS/390 V3.2 is going out of servive in October 2005.

C/C++

When coding in C or C++, make sure to use any of the following products:

- C/C ++ (with or without Debug Tool), optional feature of z/OS
- SAA AD/Cycle C/370 Compiler V1.2 (5688-216)

DB2 UDB for z/OS V8 does not yet support the coprocessor or precompiler services with C or C++. Use of DB2 precompiler services with C requires, the DB2 coprocessor provided with z/OS V1.5 and the DB2 UDB for z/OS and OS/390 V7 libraries. For use of DB2 UDB for z/OS V8 function, use the precompiler as an alternative.

DB2 also supports programming languages such as Java, FORTRAN, Assembler, and REXX. To find out what products are required, refer to the IBM software announcement letter for DB2 Version 8, 204-029 for the US, or equivalent for other countries.

For additional information about supported coprocessor options, see also informational APAR II13782.

IMS

The following versions of IMS (Information Management System) can work with DB2:

- IMS V9 (5655-J38)
- IMS V8 (5655-C56)
- IMS V7 (5655-B01)

CICS

CICS (Customer Information Control System) is required to be at any of the following versions to support DB2 V8:

- CICS Transaction Server for z/OS V2.2 or V2.3 (5697-E93)
- CICS Transaction Server for OS/390 V1.3 (5655-147)

Although CICS TS V1.3 is supported, we strongly suggest that you plan to use the latest versions of these products, to help you to maximize the use of the new functions in DB2.

IBM DB2 Tools

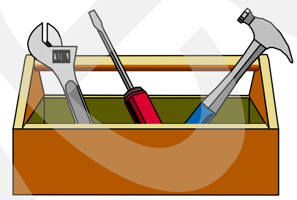
Continue to support all IBM supported releases of DB2

Support DB2 Version 8 at GA

- Some tools will need a version/release upgrade
- Some tools will need a PTF upgrade
  - Available through normal service

All tools must be at the right level BEFORE upgrading DB2 to Version 8 CM

- Otherwise the tool and/or some tool functions may no longer work



12.2 IBM DB2 Tools

Version 8 brings many changes to DB2 that impact almost every IBM DB2 tool:

- Unicode catalog tables
- Long names in the catalog
- Online schema evolution
- New log records
- And so on...

Many of these enhancements require substantial changes to some IBM DB2 Tools, which require a new release/version to be shipped. While other tools may be less impacted and support for DB2 Version 8 can be delivered through the normal service stream (via PTF).

Irrespective of the tool, we highly recommend that you plan to implement the correct level and version of the tool before you migrate from DB2 Version 7 to Version 8. If you have not installed a version of the tool that supports DB2 Version 8, unpredictable results may occur. Some tools will no longer continue to work, while others may suffer from loss of function.

To assist you in planning for DB2 Version 8, in Appendix A., “IBM DB2 DM Tools support” on page 1035, we provide a list of the IBM DB2 Tools and what versions and PTFs are required to support DB2 Version 8.

DB2 Early Code Considerations

Early code MUST be installed very early on in your plans

- Version 8 early code works fine with V7 system
- Version 7 early code requires:
  - APAR PQ59805 / PTF UQ67466
- IPL is required for early code changes / installation

12.3 DB2 early code considerations

As in previous releases and versions, the DB2 V8 early code, living in SDSNLINK, is downward compatible with version 7. Likewise, if your V7 is at the prerequisite maintenance level (with APAR PQ59805, PTF UQ67466 applied), your V7 early code is upward compatible with V8. Therefore you can run both V7 and V8 systems on the same LPAR. However, it is probably a good idea to be current on maintenance for your V7 systems, and have the fallback SPE (PQ48486) and its prerequisites applied.

In case you also have V6 systems running on the same LPAR, you should use the V7 early code, as that is downward compatible with V6 and upward compatible with V8.

Note that activating changes in the early code require an IPL.

Other Recommendations / Information

Ensure a fully tested "Stand Alone Dump" is available

- Install the z/OS version of SAD with the "High Virtual Option"

Watch and grow CSA when increasing thread limits

Start by running a DB2 for OS/390 V7 subsystem under a 64-bit operating system

- Don't go 64-bit and V8 on the same day

Set low values for everything in the beginning

- Buffer pools, EDM, Thread limits

MVS runs out of auxiliary storage slots at 4TB

SDSNLOAD library is a PDSE by default

- Can still be a normal PDS

12.4 Other recommendations / information

In order to minimize the impact of changing z/OS and DB2, it is appropriate to experiment Global Trace and diagnostics — in general, in a pilot system with minimal users. A fully tested Stand Alone Dump procedure should be available with the High Virtual Option.

The growth of threads and corresponding ECSA should be kept under control. The ECSA previously used by IRLM is now freed up if you were using PC=NO.

Start by running a DB2 UDB for OS/390 Version 7 subsystem under a 64-bit virtual Operating system and set low values for everything in the beginning, then gradually increase the values based on resources consumption due to the increase of buffer pools, EDM pool, number of threads.

The DB2 SDSNLOAD data set, that contains most of the DB2 executable code, is now by default allocated (via the DSNALLOC job) as a PDSE data set. However, you can still allocate it as a PDS if you desire to do so. When you allocate the DB2 SDSNLOAD dat set as a PDSE, and use it in combinattion with the IMS Attach, you must apply the IMS toleration PTF for APAR PQ79118. Please also be aware of the operational differences between PDS and PDS/E data sets. These are explained in the DB2 V8 program directory, section 5.2.2.2. Note however that SMP/E must know which type of data set it is managing; it will compress a PDS but not compress a PDSE. If you change the data set organization, you must also update the SMP/E definition.

Release Incompatibilities

TYPE 2 keyword has been deprecated on CREATE INDEX

- If specified, it will be ignored

DB2 now requires BP8K0 and BP16K0 buffer pools

- Some catalog tables now use these buffer pools
- Declared temporary tables need at least one table space with a page size of 8KB or greater
- Global temporary tables need a 16 KB buffer pool

Change to data types and lengths for some special registers

- May need to review your applications

You must now have a customized DSNHDECP

You must now specify valid CCSIDs in DSNHDECP

New data-only load module DSNHMCID

12.4.1 Release incompatibilities

Here we discuss the major incompatibilities that you may encounter when migrating to DB2 Version 8.

Type 1 indexes

Hopefully, now we have seen the end of Type 1 indexes.

Support for Type 1 indexes was planned to be dropped in DB2 Version 6. Before migrating from Version 5 to Version 6 or Version 7, you were asked to migrate all of your indexes from Type 1 to Type 2. The migration job DSNTIJTC would abend if it found any unsupported objects, including Type 1 indexes. Some customers found this too restrictive.

After APAR PQ38035, the migration job DSNTIJTC completes successfully even if it finds any unsupported objects, including Type 1 indexes. However, when you try to use these unsupported objects on Version 6 or Version 7, DB2 returns a resource unavailable error with SQLCODE -904 and reason code 00C900CF. Although these indexes were unusable in Version 6 and Version 7, you could still DROP them or convert them to Type 2 indexes.

The DB2 Version 8 catalog migration job, DSNTIJTC, will now once again abend if it encounters any Type 1 indexes. In addition, DB2 does not pay any attention to the Type 2 keyword if it is specified on the CREATE INDEX or ALTER INDEX statement.

DB2 now requires BP8K0 and BP16K0 buffer pools

Support for longer names has caused some DB2 catalog table rows to grow beyond the current catalog page size of 4K. So, DB2 Version 8 moves some catalog table spaces from BP0 to BP8K0 and BP16K0.

TEMP database needs at least one 8K table space

In the database that is defined “AS TEMP” (TYPE='T' in SYSDATABASE), you need to define at least one table space with an 8K page size or more, if you want to use declared global temporary tables in DB2 Version 8. Otherwise you receive:

```
DSNT408I SQLCODE = -904, ERROR:  UNSUCCESSFUL EXECUTION CAUSED BY AN UNAVAILABLE
RESOURCE. REASON 00E7009A, TYPE OF RESOURCE 200, AND RESOURCE NAME TABLESPACE IN tempdb
```

This must be available before you move to Version 8 compatibility mode. DB2 needs to create a copy of some catalog tables when a declared temporary table is created and used in Version 8 and these tables have page sizes larger than 4K.

Changes to some special registers

DB2 Version 8 changes the data type and length of some special registers. These are the changes:

- CURRENT OPTIMIZATION HINT is now VARCHAR(128)
- CURRENT PACKAGESET is now VARCHAR(128)
- CURRENT SQLID is now VARCHAR(8)
- CURRENT USER is now VARCHAR(8)
- CURRENT PATH is now VARCHAR(2048)

If your application program uses these registers in comparison statements such as a LIKE predicate, you may need to adjust your application program for the new lengths.

DB2 start-up and precompile require a user-supplied DSNHDECP

This change only impacts you if you have not previously generated your own DSNHDECP module and have relied on the IBM supplied default. You will now need to maintain your own DSNHDECP with your own tailored defaults.

DSNHDECP is a data only module that supplies various application programming defaults, like default date and time formats and default code pages. This module is normally tailored by the DSNTINST CLIST and generated by the installation job DSNTIJUZ.

DB2 ships an IBM supplied default version of DSNHDECP in SDSNLOAD, and continues to do so in DB2 V8 from compatibility with older applications.

However, Version 8 is more reliant than previous versions on DSNHDECP. Therefore, DB2 checks to see whether the DSNHDECP is the default DSNHDECP that ships with DB2, or whether it loaded a customized version of DSNHDECP. When a default DSNHDECP is loaded during DB2 start-up, or during the invocation of the DB2 precompiler, start-up, and precompilation fail. A customized DSNHDECP must exist in a library that is before the default DSNHDECP in the STEPLIB concatenation or link list.

If you normally link-edit your customized DSNHDECP into SDSNLOAD (and override the default), you can continue to do so; if not, you will need to link-edit your DSNHDECP into SDSNEXIT and concatenate that data set ahead of SDSNLOAD.

You must now specify valid CCSIDs

Recent versions of DB2 permit specifying a so-called undefined CCSID of 0 when creating the DSNHDECP module. (In DSNHDECP, the single-byte EBCDIC CCSID is specified as the argument to the SCCSID parameter. The single-byte ASCII CCSID is specified as the argument to the ASCCSID parameter.)

Over time, with the evolution of new DB2 functions and features, string data has become increasingly dependent on CCSIDs. CCSIDs can be specified explicitly when binding an object but in most cases the CCSID is determined from the DSNHDECP module. The original requirements for setting SCCSID in DSNHDECP were for the use of distributed data, or the use of mixed and graphic (DBCS) data.

In Version 5, support for optionally storing data in ASCII format was added. With this support DB2 needed a way to distinguish between the ASCII and EBCDIC data and this was done through CCSID tagging. Creation of an ASCII table required specification of ASCII CCSIDs in DSNHDECP in conjunction with EBCDIC CCSIDs.

In Version 6, new object types were added. When a large-object (LOB) data type, distinct type, user-defined function, or stored procedure that references EBCDIC “string” data types is created via an SQL CREATE statement, there is a requirement that defined (non-zero) EBCDIC CCSIDs be provided in DSNHDECP. Furthermore, if ASCII was used for these object types, then ASCII CCSIDs are also required in DSNHDECP.

In Version 7, support was added for storing data encoded in Unicode. Although DSNHDECP has fields for Unicode, creation of Unicode objects also requires defined CCSIDs for EBCDIC.

Any specification of a string data type in the CREATE statement for a LOB or distinct type, user-defined function, or stored procedure requires an implicit or explicit specification of an encoding scheme — EBCDIC, ASCII, or (V7 only) UNICODE — via the CCSID clause for any specification of a string data type.

If no CCSID parameter is specified in the CREATE statement, the encoding scheme is the value of the DEFAULT ENCODING SCHEME on installation panel DSNTIPF. DB2 then determines the actual, “numeric” CCSID from the encoding scheme in place (specified implicitly/explicitly as part of the CREATE statement). If the CCSID (such as in DSNHDECP) determined for a column/parameter is invalid or undefined (0), then an error such as SQLCODE -879 or -189 is returned.

In recognition of the increasingly essential role of CCSIDs in storing and manipulating string-type data, the “undefined” EBCDIC CCSID of 0 is being discontinued in DB2 Version 6 and subsequent versions. Also, the notion of a DB2-provided “default” EBCDIC CCSID is eliminated because it is not possible or appropriate for DB2 to provide a “correct” default CCSID. Therefore, use of the DB2-supplied DSNHDECP module in SDSNLOAD is no longer recommended.

PQ56697 adds provisions to alert you if you are using an undefined CCSIDs or the DB2-supplied DSNHDECP in SDSNLOAD that these practices are unsupported in DB2 Versions 6 and 7.

PQ71079 adds restrictions to prevent you from creating a DSNHDECP module that specifies an undefined CCSID for single-byte ECBCIC, that is SCCSID=0.

In V8, DB2 startup processing checks the CCSIDs that are specified in DSNHDECP. If the values are not valid, then DB2 issues message DSNT109I with reason code 00E3009B, and DB2 startup processing terminates. This is to avoid any potential data corruption issues.

Appendix A of the *DB2 UDB for z/OS Version 8 Installation Guide,* SC18-7418 has a list of the valid CCSIDs for ASCII and EBCDIC. Note that there are two tables; one for MIXED=NO, another one for MIXED=YES.

Attention: The CCSID of character strings at your site is determined by the CCSID that you specify on installation panel DSNTIPF. **If this CCSID is not correct, character conversion produces incorrect results.** The correct CCSID for your installation is determined by the coded character set that is supported by your site's I/O devices (for example, 3270 terminals or terminal emulation programs such as IBM's PCOMM), local applications such as IMS and QMF, and remote applications such as CICS Transaction Server. If you find a mismatch between what your applications use and what DB2 uses, do not change your DSNHDECP without evaluating all of the implications, and certainly not without talking to IBM Service personnel before doing so.

SPUFI expects to receive EBCDIC data in the same CCSID as the one that is used to bind the SPUFI packages and plans. When SPUFI is initiated by a TSO terminal that uses a different CCSID, data corruption can occur for both input (UPDATE and INSERT) and output (SELECT), although the data corruption on output usually is not persistent — it only occurs on the users screen and the actual data in the table may be fine. Any characters that do not map to the same codepoint in the two CCSIDs may be affected.

The PTF for APAR PQ89018 enhances SPUFI to compare the CCSID of the TSO (ISPF) terminal to the CCSID used to bind SPUFI. When they are not the same, SPUFI alerts the user by displaying a new message:

```
DSNE345I WARNING: DB2 DATA CORRUPTION CAN RESULT FROM THIS SPUFI SESSION BECAUSE THE
                  CCSID USED BY THE TERMINAL IS NOT THE SAME AS THE CCSID USED BY SPUFI.
                  TERMINAL CCSID: <terminal-ccsid>. SPUFI CCSID: <spufi-ccsid>.
                  NOTIFY THE DB2 SYSTEM ADMINISTRATOR.
```

A PTF is available for both DB2 V7 and DB2 V8

New data-only load module DSNHMCID

The new data-only load module DSNHMCID contains EBCDIC CCSIDs (single byte, double byte, mixed) for offline message conversion. Version 8 utilities and applications must have access to this module. DSNHMCID is generated by DSNTIJUZ and is link-edited into both SDSNLOAD and SDSNEXIT.

Release Incompatibilities - 2

Redbooks

Support for DB2 established stored procedures is deprecated

- Can no longer create DB2 established stored procedures
  - NO WLM ENVIRONMENT clause is now invalid
- Existing stored procedures can still run in a DB2-established stored procedure address space
  - Move to WLM as soon as possible

COMPJAVA stored procedures no longer supported

- Use JIT

Multiple calls to the same stored procedure at same level

- 2nd call closes previous open cursor in V7
- Multiple identical open cursors now allowed in SP

Support for DB2-established stored procedures is being deprecated

In Version 8, you can no longer specify the NO WLM ENVIRONMENT option when you CREATE or ALTER stored procedure definitions. Although existing stored procedures can still run in a DB2-established stored procedure address space, you should plan to move your stored procedures to WLM environments as soon as possible. DB2-established stored procedures will probably be no longer available in future releases of DB2.

In earlier versions of DB2 the supplied stored procedure DSNWZP had to run in a DB2-established stored procedure address space. DB2 Version 8 defines DSNWZP to run in a WLM-established stored procedure address space.

So, when you fallback from Version 8 to Version 7, or re-migrate from Version 7 to Version 8, the DSNWZP stored procedure will not work. You will need to manually issue the appropriate ALTER commands to change the external name for the stored procedure.

We describe this in a little more detail in later visuals.

COMPJAVA stored procedures no longer supported

DB2 Version 8 will no longer support LANGUAGE COMPJAVA stored procedures, since Visual Age Java will no longer support compiled Java link library files, known as High Performance Java (HPJ).

After migrating to Version 8 compatibility mode, you can no longer define or run COMPJAVA stored procedures. DB2 will return a SLQCODE -471 with reason code 00E79000 and message DSNX900E will be written to the system console, if you try to execute a stored procedure with LANGUAGE COMPJAVA. You need to convert all LANGUAGE COMPJAVA stored procedures to LANGUAGE JAVA before migrating to Version 8 compatibility mode, by following these steps:

1. Use ALTER PROCEDURE to change the LANGUAGE and the WLM ENVIRONMENT parameters. (The recommendation is that COMPJAVA stored procedures do not run in the same WLM environment as JAVA stored procedures, for performance reasons. So, you will probably need to change the WLM ENVIRONMENT parameter as well.) The EXTERNAL NAME clause must also be specified even if it has not changed, as DB2 needs to verify it.

   Use the following example as a model:

   ```
   ALTER PROCEDURE SYSPROC.JAVADVR
   LANGUAGE JAVA
   EXTERNAL NAME 'display.display.main'
   WLM ENVIRONMENT WLMENVJ;
   ```

2. Ensure that the WLM environment is configured and that the required JVM is installed.
3. Ensure that the .class file that is identified in the EXTERNAL NAME clause of the ALTER PROCEDURE is present in one of the following places:
   – In a JAR that was installed to DB2 by an invocation of the INSTALL_JAR stored procedure
   – In a directory in the CLASSPATH ENVAR of the data set that is named on the JAVAENV DD statement of the WLM stored procedures address space JCL

Multiple calls to the same stored procedure at the same nesting level

In previous DB2 versions, if a stored procedure was called twice from the same program and at the same nesting level, DB2 closes the result set cursors and releases storage for the first instance of the stored procedure before making the second call.

In DB2 Version 8, if the requester and the server are both DB2 Version 8 subsystems in new-function mode, when the second call is made, both instances of the stored procedure can run at the same time. DB2 does not close the result sets from the first call or release storage for the first instance of the stored procedure. This is an “incompatible” change.

Release Incompatibilities - 3

NO outstanding Version 7 utilities

- Cannot restart or terminate any outstanding V7 utilities in V8

And many more.....

- Job DSNTIJPM (pre-migration checking for potential problems)
- Auto rebind all plans bound prior to V2.3
- Preventing CCSID changes
- Location name must be specified (after PQ91009/UQ90701)
- Review the migration considerations chapter in the Installation Guide

No outstanding Version 7 utilities

DB2 Version 8 enforces a restriction that you can restart or terminate a utility only on the same release on which it was started. So, any outstanding utilities prior to Version 8 cannot be restarted or terminated after you have migrated from Version 7 to Version 8 compatibility mode. To ensure that you do not have outstanding utility jobs before you migrate the Version 8, issue the following command:

```
-DISPLAY UTILITY(*)
```

If you find that you have an outstanding Version 7 Utility after you have migrated to Version 8 compatibility mode, you must first fallback to Version 7 to restart or terminate the utility.

And many more.....

We have only identified the key differences between DB2 Version 8 and earlier releases. In your planning for DB2 Version 8, we urge you to review the section on "Migration considerations" in the *DB2 UDB for z/OS Version 8 Installation Guide,* SC18-7418.

DB2 pre-migration checks - DSNTIJPM

DB2 provides a tailored job which we strongly recommend that you run prior to migrating your DB2 subsystem from Version 7 to Version 8. Job DSNTIJPM, which is shipped in the Version 8 SDSNSAMP data set, searches for any release incompatibilities and unsupported objects, which will prevent a successful migration. The jobs checks the following things:

- Existence of Type 1 indexes
- DB2 catalog tables on which DATACAPTURE is enabled
- Partitioned table spaces that use selective partition locking (SPL)
- Partitioned table spaces that have a truncated limit key
- Stored procedures that use LANGUAGE COMPJAVA

- Stored procedures that use the DB2 stored procedures address space
- Use of the DSNWZPR module by DSNWZP
- Existence of the V7 sample database
- Evidence of multiple CCSIDs in the same encoding scheme
- Packages for routines and plans for callers of routines that need to be rebound because of a incompatible change in the DBINFO control block

In order to provide customers plenty of time to prepare for V8 migration before they actually buy DB2 V8, a job similar to V8’s DSNTIJPM will be shipped in V7. The new job is called *DSNTIJP8*. It is shipped with the PTF for APAR PQ84421.

> **Attention:** As a part of your migration planning for DB2 Version 8, we strongly recommend that you review your environment and the code pages you have been using. Any discrepancies may cause problems after you migrate to Version 8 when DB2 parses in Unicode and starts to perform code page conversions more often. We recommend that you check the following items in all of your DB2 environments before migration:
>
> - Do your DB2 environment’s CCSIDs match your terminal emulators and local applications?
> - Identify if there are any objects in DB2 with different CCSIDs within the same encoding scheme. The SQL shipped in the V8 job DSNTIJPM or the V7 equivalent called DSNTIJP8, may be able to help to a certain extent, as it searches for cases where more than one CCSID exists. For example, all table spaces defined as EBCDIC should have CCSID 37 or all should have CCSID 500 in a single system, not a mixture.
>
> If either of these checks raise any issue with your DB2 environments, please contact IBM for advice on how to resolve these issues before migrating to Version 8. We highly recommend that you do not change any DB2 CCSID values without first consulting IBM support.
>
> These problems can be much harder to identify and fix once you have moved to Version 8 and are converting your data to Unicode. They are also better resolved in Version 7 before you start seeing incorrect results as soon as you migrate to Version 8.
>
> We recommend that you perform this code page health check as soon as you begin planning for Version 8, in order to have enough time for any remedial action, if required.

Automatic rebind of plans bound prior to Version 2.3

DB2 Version 8 will autobind plans that were bound prior to DB2 Version 2.3. Many changes have been made over the last 10 years to DB2. There are many old code paths in DB2 that are in place to deal with special cases and issues with old plans. The change sets a precedence to retire old plans and packages (execution and runtime structures) on an ongoing basis as new releases of DB2 are introduced.

Preventing CCSID changes

In Version 8, the CCSID information that is specified at installation (on panel DSNTIPF) is stored in the BSDS. Currently this information is stored in the DSNHDECP module. This change is introduced to ensure that you do not change your CCSIDs, either accidentally or by intention (something that is not supported by DB2).

At startup, DB2 checks the BSDS to see if the CCSIDs are recorded in the BSDS. If the CCSIDs are not recorded, DB2 will place them in the BSDS. If the CCSIDs are recorded in the BSDS, we will check to make sure they match the values in the DSNHDECP. If the values do not match, message DSNT108I will be issued and DB2 startup processing will terminate.

In addition, the Change Log Inventory Utility (DSNJU003) is also changed to delete the CCSIDs in the BSDS. A new clause, DELETE CCSIDS, deletes the CCSID information in the BSDS.

Location name required

After applying PQ91009/UQ90701, you must have a location name defined in the BSDS for your DB2 system. The location name is required, even if you are NOT using the Distributed Data Facility (DDF). After applying this PTF, DB2 start-up processing will verify whether a location name is specified in the DDF communication record of the BSDS. If a location name is not specified, DB2 start-up processing will terminate with a new reason code 00E80059. You can check to see whether you have a location name define by using the Print Log Map utility (DSNJU004). If a location name is not defined, you must do so using the Change Log Inventory utility (DSNJU003) using the "DDF LOCATION=location-name" keyword.

A complete list of things to watch out for can be found in the “Migration considerations” section of the *DB2 UDB for z/OS Version 8 Installation Guide,* GC18-7418.

DB2 Universal Driver for SQLJ/JDBC

Both the Universal Driver and JDBC/SQLJ Driver for OS/390 are shipped with DB2 Version 8

- Many enhancements in the Universal Driver
- Base for all future Driver development
- Strongly encouraged to migrate to the Universal Driver

Some differences exist that may impact existing applications

- URL syntax
- Security design
- SQLJ default connection
- SQLJ program preparation process
  - New utilities, e.g.. db2sqljcustomize instead of db2profc
  - No more DBRMs
  - New profile customization with new customized profile layout
  - Can upgrade existing .ser files (on z/OS) with db2sqljupgrade utility

12.4.2 DB2 Universal Driver for SQLJ and JDBC

Both the DB2 Universal JDBC Driver (new) and the JDBC/SQLJ Driver for OS/390 (now also called the legacy Driver) are shipped with DB2 Version 8.

The legacy Driver is put into the HFS in the same directory path structure as it is in Version 7:

```
/usr/lpp/db2/db2810/
```

The Universal driver, by default, is stored in the directory:

```
/usr/lpp/db2/db2810/jcc/
```

As a lot of the classes are identical between both drivers, you must change your CLASSPATH in order to be able to use the DB2 Universal Driver.

The DB2 Universal JDBC Driver differs from the JDBC/SQLJ Driver for OS/390 in many ways. Listing all the differences between them is an almost impossible job. In this section we briefly introduce some of the differences that can impact your existing applications. A list of the supported JDBC APIs can be found in the “Comparison of driver support for JDBC APIs” section of the manual, *DB2 Application Programming Guide and Reference for Java,* SC18-7414.

Difference in URL syntax

The syntax of the url parameter in the DriverManager.getConnection method is different for each driver.

Using the JDBC/SQLJ Driver for OS/390, you use:

```
jdbc:db2os390:location-name
jdbc:db2os390sqlj:location-name
```

With the Universal JDBC Driver, for Type 2 connectivity, you use:

```
jdbc:db2:database
```

However, for downward compatibility, you can still use `jdbc:db2os390:location-name` and `jdbc:db2os390sqlj:location-name`. Our recommendation is to make the change, and start using the new syntax. In case you are using the datasource interface, your programs are not affected, and you can make the change in the datasource definition.

Note that the Universal Driver also supports Type 4 connectivity. In that case the syntax is:

```
jdbc:db2://server:portnumber/database
```

Note also that database in the context of the Universal Driver, is the DB2 location name of the system you want to connect to.

Difference in error codes and SQLSTATEs returned for driver errors

The DB2 Universal JDBC Driver does not use existing SQLCODEs or SQLSTATEs when an error occurs inside the driver itself, as the other drivers do. You can look up the error codes and SQLSTATEs issued by the Universal Driver in the "Error codes issued by the DB2 Universal JDBC Driver" and "SQLSTATEs issued by the DB2 Universal JDBC Driver" sections of *DB2 Application Programming Guide and Reference for Java,* SC18-7414. The JDBC/SQLJ driver for z/OS returns SQLSTATE FFFFF when such an error occurs.

Security mechanisms

The JDBC drivers have different security mechanisms. Therefore it is important to understand their differences. For information on DB2 Universal JDBC Driver, and JDBC/SQLJ Driver for OS/390 security mechanisms, see the "Security under the DB2 Universal JDBC Driver" and "Security under the JDBC/SQLJ Driver for OS/390" of *DB2 Application Programming Guide and Reference for Java,* SC18-7414.

How connection properties are set

With Universal Driver Type 4 connectivity, you set properties for a connection by setting the properties for the associated DataSource or Connection object.

With *Universal Driver* type 2 connectivity, you set properties for a connection in one of these ways:

- You can set properties only for a connection by setting the properties for the associated DataSource or Connection object.
- You can set driver-wide properties through an optional run-time properties file.

For the *JDBC/SQLJ driver for z/OS* driver, you set properties through the JDBC/SQLJ run-time properties file.

Results returned from ResultSet.getString for a BIT DATA column

The DB2 Universal JDBC Driver returns data from a ResultSet.getString call for a CHAR FOR BIT DATA or VARCHAR FOR BIT DATA column as a lowercase hexadecimal string.

The JDBC/SQLJ Driver for OS/390 returns the data in the encoding scheme of the caller.

Exceptions for PreparedStatement.setXXXStream with length mismatch

Another difference between the two drivers is the point in them when an exception is thrown for PreparedStatement.setXXXStream with a length mismatch. When you use the PreparedStatement.setBinaryStream, PreparedStatement.setCharacterStream, or PreparedStatement.setUnicodeStream method, the length parameter value must match the number of bytes in the input stream. If the number of bytes for these does not match:

- The DB2 Universal JDBC Driver does not throw an exception until the subsequent PreparedStatement.executeUpdate method executes. Therefore, for the DB2 Universal JDBC Driver, some data might be sent to the server when the lengths do not match. That data is truncated or padded by the server. The calling application needs to issue a rollback request to undo the database updates that include the truncated or padded data.
- The JDBC/SQLJ 2.0 Driver for OS/390 throws an exception after the PreparedStatement.setBinaryStream, PreparedStatement.setCharacterStream, or PreparedStatement.setUnicodeStream method executes.

Default mappings for PreparedStatement.setXXXStream

With the DB2 Universal JDBC Driver, when you use the PreparedStatement.setBinaryStream PreparedStatement.setCharacterStream, or PreparedStatement.setUnicodeStream method, and no information about the data type of the target column is available, the input data is mapped to a BLOB or CLOB data type.

For the JDBC/SQLJ driver for z/OS, the input data is mapped to a VARCHAR FOR BIT DATA or VARCHAR data type.

How character conversion is done

When character data is transferred between a client and a server, the data must be converted to a form that the receiver can process:

- For the DB2 Universal JDBC Driver, character data that is sent from the database server to the client is converted using Java's built-in character converters. The conversions that the DB2 Universal JDBC Driver supports are limited to those that are supported by the underlying JRE implementation. A DB2 Universal JDBC Driver client sends data to the database server as Unicode.
- For the JDBC/SQLJ driver for z/OS, character conversions can be performed if the conversions are supported by the DB2 server.

Implicit or explicit data type conversion for input parameters

If you execute a PreparedStatement.setXXX method, and the resulting data type from the setXXX method does not match the data type of the table column to which the parameter value is assigned, the driver returns an error unless data type conversion occurs:

- With the DB2 Universal JDBC Driver, conversion to the correct SQL data type occurs implicitly if the target data type is known and if the deferPrepares connection property is set to false. In this case, the implicit values override any explicit values in the setXXX call. If the deferPrepares connection property is set to true, you must use the PreparedStatement.setObject method to convert the parameter to the correct SQL data type.
- For the JDBC/SQLJ driver for z/OS, if the data type of a parameter does not match its default SQL data type, you must use the PreparedStatement.setObject method to convert the parameter to the correct SQL data type.

Data returned from ResultSet.getBinaryStream against a binary column

With the DB2 Universal JDBC Driver, when you execute ResultSet.getBinaryStream against a binary column, the returned data is in the form of lowercase, hexadecimal digit pairs.

With the JDBC/SQLJ driver for z/OS, when you execute ResultSet.getBinaryStream against a binary column, a string value is returned. The driver uses the Java client's default local encoding to construct the string from bytes.

Result of using getBoolean to retrieve a value from a CHAR column

With the DB2 Universal JDBC Driver, when you execute ResultSet.getBoolean or CallableStatement.getBoolean to retrieve a Boolean value from a CHAR column, and the column contains the value "false" or "0", the value false is returned. If the column contains any other value, the value "true" is returned.

With the JDBC/SQLJ driver for z/OS, when you execute ResultSet.getBoolean or CallableStatement.getBoolean to retrieve a Boolean value from a CHAR column, and the column contains the value "0", the value "false" is returned. If the column contains any other value, the value "true" is returned.

Differences for SQLJ

SQLJ support in the DB2 Universal JDBC Driver differs from SQLJ support in the other DB2 JDBC drivers in the following areas.

Connection associated with the default connection context

If you are using the DataSource interface to connect to a data source, before you can use a default connection context, the logical name jdbc/defaultDataSource must be registered with JNDI. The JDBC/SQLJ Driver for OS/390 creates a connection to the local data source for the default connection context.

To create a default connection context, the SQLJ runtime now does a JNDI lookup for jdbc/defaultDataSource. If nothing is registered, a null context exception will be thrown when the driver attempts to access the context. The recommended solution is to use an explicit connection context on the sqlj clause. However, registering a jdbc/defaultDataSource with JNDI will also suffice.

Production of DBRMs during SQLJ program preparation

The SQLJ program preparation process for the DB2 Universal JDBC Driver does not produce DBRMs. Therefore, with the DB2 Universal JDBC Driver, you can produce DB2 packages only by using the DB2 Universal JDBC Driver utilities.

Difference in connection techniques

As mentioned earlier, the connection techniques that are available, and the driver names and URLs that are used for those connection techniques, vary from driver to driver.

New layout for customized profiles

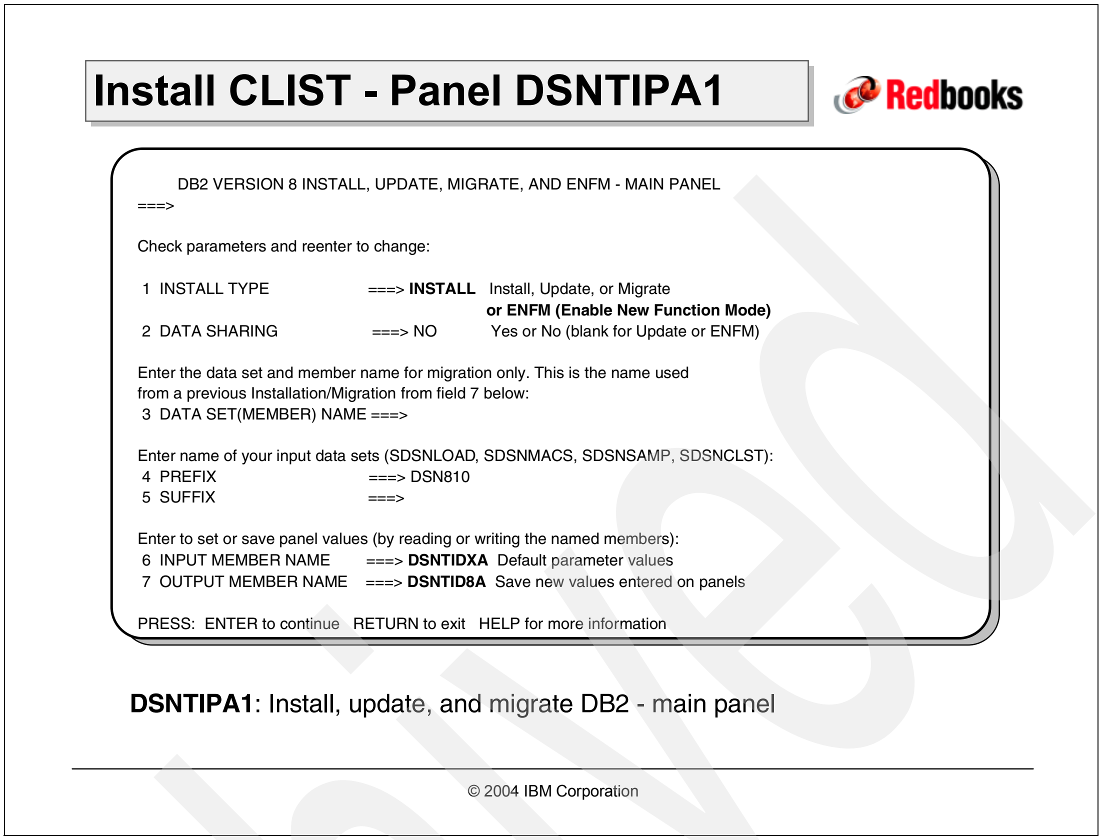
The new db2sqljcustomize program that is used to customize your serialized profile, and by default also, to bind your packages on the DB2 for z/OS, generates a serialized profile with a new format. The new format is not compatible with the old format. To be able to use the existing, installed SQLJ programs customized with the legacy Driver, they must be upgraded first.

To convert serialized profiles that you customized under JDBC/SQLJ Driver for OS/390 to a format that is compatible with the DB2 Universal JDBC Driver, run the db2sqljupgrade utility. After you run the db2sqljupgrade utility, you do not need to bind new packages for the associated SQLJ applications.

Before you can run the db2sqljupgrade utility, your CLASSPATH must contain the full path names for the db2j2classes.zip file for the JDBC/SQLJ Driver for OS/390, and the db2jcc.jar and sqlj.zip files for the DB2 Universal JDBC Driver. For example:

```
db2sqljupgrade -collection new_collection_name MyinputFileName.ser
```

The upgrade utility will save the existing profile as .ser_old. Customers can revert back to the old .ser in case upgrade is not successful. It is strongly recommended that customers back up original files, including but not limited to .ser, .class, .java, and .sqlj files, to another directory before attempting to upgrade the profile.



12.5 Installation CLIST processing

You can install DB2 Version 8 either as a host based installation or via the “msys for Setup Facility”. We will briefly explore the msys for Setup DB2 Customization Center in later visuals in this chapter. Briefly, the msys Setup DB2 Customization Center is a workstation based facility that replaces the DB2 Installer workstation tool. The remainder of this discussion will concentrate on the TSO, or host based, installation and migration processes.

12.5.1 Introduction

The process to install a new DB2 Version 8 subsystem is the same as installing a new DB2 Version 7 subsystem. However, there are a few small differences from previous versions of DB2, which we highlight in the next few visuals.

Migration (from V7) to Version 8 is quite different from previous migrations. The migration process now consists of three distinct phases:

- **CM: compatibility mode:** During this phase, which can last as long as deemed necessary, you will make all the tests needed to ensure that you are happy with the new version and will not have to fallback to Version 7 later on. In CM, you are able to fallback to Version 7 in case of problems.
- **ENFM: enabling-new-function mode:** During this phase you will convert the DB2 subsystem to the format ready to support the new function in Version 8, by using the on-line REORG Utility. No fallback to DB2 Version 7 is allowed once the ENFM is entered.
- **NFM: new-function mode:** This is the target phase, triggered when you execute a job confirming that all the previous conversion steps have completed successfully and update the DB2 subsystem or data sharing group as in new-function mode.

12.5.2 Install CLIST - Panel DSNTIPA1

After completing the SMP/E work to create and populate the DB2 libraries, you are now ready to invoke the DB2 Installation CLIST to generate the jobs required for installation. Nothing has changed here for Version 8.

First, make the DB2 ISPF libraries available to TSO. This can be done by concatenating the DB2 ISPF libraries to your normal allocations. (Refer to the *DB2 UDB for z/OS Version 8 Installation Guide,* SC18-7418 for more details.). You can now invoke the DB2 Installation CLIST in either of two ways:

1. To use DB2 Online help:

   EXEC ’prefix.SDSNCLST(DSNTINS0)’

2. To bypass the DB2 Online help:

   EXEC ’prefix.SDSNCLST(DSNTINST)’

The panel DSNTIPA1 is the first panel displayed. From here, you tell the installation process what you want to do. In addition, the DB2 Installation CLIST needs a set of default values and uses them on the subsequent panels.

You will notice a new option on the panel, ENFM, which is highlighted. This is the option you use after you have successfully migrated to Version 8 and now want to generate the jobs to enable the new-function mode in DB2. We will talk more about this in the migration visuals.

To install DB2 for the first time, use the IBM-supplied defaults in member DSNTIDXA for the INPUT MEMBER NAME. To install DB2 by using parameters from a previous run as defaults, you must supply the member that contains the output from the previous run. It was the OUTPUT MEMBER NAME during the last run.

Specify the member name of the output data set in which to save the values that you enter on the panels. This member is stored in *prefix*.SDSNSAMP (not the data set created by the DSNTINST CLIST). To avoid replacing any members of *prefix*.SDSNSAMP that are shipped with the product, specify DSNTID*xx* as the value of OUTPUT MEMBER NAME, where *xx* is any alphanumeric value except XA or VB. Always give a new value in the OUTPUT MEMBER NAME field for a new panel session. You supply the name from your current session in the INPUT MEMBER NAME field for your next session. You should not use the same member name for output as for input.

The outputs from the DB2 Installation CLIST session are:

- A new data set, prefix.NEW.SDSNSAMP, that contains the edited JCL
- A new data set, prefix.NEW.SDSNTEMP, that contains tailored CLISTs for input to job DSNTIJVC
- A new member in prefix.SDSNSAMP, containing the resulting parameter values

DSNTIPA1 Processing Options

Specify whether you are installing, updating, migrating, or converting to NFM

INSTALL to install DB2 for the first time

- Default for the first run of the CLIST
- Creating a new Version 8 DB2 subsystem
  - In "New Function Mode"

UPDATE to update parameters for an existing DB2 subsystem

MIGRATE to migrate from Version 7 to Version 8 (CM)

- Compatibility Mode (CM) - no new function available

ENFM to convert the DB2 catalog from Version 8 (CM) to NFM

- You must run the CLIST in MIGRATE mode before you can choose this option

12.5.3 DSNTIPA1 processing options

As you can see from the previous visual, the DB2 Installation CLIST has a new option, ENFM, to support conversion from DB2 for z/OS, Version 8 compatibility mode (CM) to DB2 for z/OS, Version 8 new-function mode (NFM). We shall discuss this in more detail when we talk about the DB2 migration process in later visuals.

However, in summary:

- INSTALL is used to generate the jobs to create a new DB2 subsystem.
- UPDATE is used to update and maintain the DSNZPARM and DSNHDECP parameters.
- MIGRATE is used to generate the jobs used to migrate a DB2 version 7 subsystem to Version 8, running in compatibility mode (CM). In this mode no new Version 8 functions are available.
- Once the CLIST has successfully completed while specifying MIGRATE, it can be executed specifying ENFM to generate the jobs used to migrate the DB2 subsystem to enabling-new-function mode (ENFM). It is recommend to run the CLIST in ENFM mode only after you have successfully migrated to V8 (including all members if data sharing) and are committed to commencing enabling-new-function mode.

Installation CLIST Changes

Materialized query tables
- CURRENT REFRESH AGE
- CURRENT MAINT TYPES

Remove hiperpool definitions and VPTYPE settings
- No longer supported in V8

DDF panels have new terminology
- Use “Inactive DBAT" instead of "Type 1 Inactive Thread"
- Use "Inactive Connection" instead of "Type 2 Inactive Thread"

Panel DSNTIP7
- VARY DS CONTROL INTERVAL
- OPTIMIZE EXTENT SIZING

and many more .....

12.5.4 Installation CLIST changes

As you move through the DB2 Installation CLIST panels, you will notice a number of changes from the previous versions of the panels. There have been some new fields added to support new Version 8 function, as well as some panels have been removed as they are no longer required. Please refer to the *DB2 UDB for z/OS Version 8 Installation Guide,* SC18-7418, for a description of all of the Installation panels and their contents.

The major changes are as follows:

- The Performance and Optimization panel, DSNTIP8, provides the defaults for two new special registers which have been created to support Materialized Query Tables:
  - CURRENT REFRESH AGE:

    Specifies the default value to be used for the CURRENT REFRESH AGE special register when no value is explicitly set using the SQL statement SET CURRENT REFRESH AGE. The values can be 0 or ANY). The default of 0 disables query rewrite using deferred materialized query tables.
  - CURRENT MAINT TYPES:

    Specifies the default value to be used for the CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION special register when no value is explicitly set using the SQL statement SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION.

    Acceptable values are NONE, SYSTEM, USER, ALL. The default (SYSTEM) allows query rewrite using system-maintained materialized query tables when the CURRENT REFRESH AGE is set to ANY. Alternatively, specifying USER allows query rewrite using user-maintained materialized query tables when CURRENT REFRESH AGE is set to ANY, and specifying ALL means query rewrite using both system-maintained and user maintained materialized query tables.

- The buffer pool panels have been re-designed to remove hiperpool definitions and rename virtual buffer pools to buffer pools.

  In DB2 Version 8, there are no longer any concept of hiperpools, buffer pools in data spaces, or virtual buffer pools. They are all just buffer pools.

  In addition, the default size for BP0 has been raised from 2000 to 20000 and the default size for BP32K has been raised from 24 to 250. As BP8K0 and BP16K0 are now required buffer pools (used by the DB2 catalog), their default value is increased from 0 to 1000, and 500 respectively.

- The Data Set Names panels have been re-designed.

  The Data Set Names panels have been re-designed to remove old data sets which are no longer required. For example, the old COBOL compiler libraries have been removed.

- The Distributed Data Facility panel, DSNTIPR, has renamed MAX TYPE 1 INACTIVE to MAX INACTIVE DBAT.

  DB2 Version 8 uses the term “Inactive DBAT” instead of “Type 1 Inactive Thread” and uses the term “Inactive Connection” instead of “Type 2 Inactive Thread”. These terms are much more descriptive of the actual status of the threads and brings the terminology more in line with DB2 UDB on other platforms.

- The Data Set Size panel, DSNTIP7, has four new fields that change how DB2 manages VSAM data sets:

  – TABLE SPACE ALLOCATION and INDEX SPACE ALLOCATION:

    Specify the amount of space in KB for primary and secondary space allocation for DB2-defined data sets for table spaces and index spaces that are being created without the USING clause. A value of 0 for a non-LOB table space or index indicates that DB2 is to use a default value of one cylinder, or ten cylinders for a LOB table space. These parameters were introduced via APAR PQ53067 in DB2 V6 and V7, but did not appear on the install panels until V8.

  – VARY DS CONTROL INTERVAL:

    This field specifies whether DB2-managed data sets created by CREATE TABLESPACE will have variable VSAM control intervals. If you specify YES, when DB2 creates a DB2-managed data set for a table space. It will have a VSAM control interval that corresponds to the buffer pool used for the table space. A value of NO indicates that DB2-managed data sets are to be created with a fixed control interval of 4-KB, regardless of the buffer pool size (as in Version 7). Also, if you specify YES in this field then the following installation jobs will take this into account:

    • DSNTIJIN is configured to use variable CI sizes when defining the data sets for the DB2 catalog and directory (in INSTALL and MIGRATE mode)

    • DSNTIJNE is configured to use variable CI sizes for DB2 catalog and directory data sets when enabling new-function mode.

  – OPTIMIZE EXTENT SIZING:

    This field specifies whether secondary extent allocations for DB2-managed data sets are to be sized according to a sliding scale that optimizes the likelihood of reaching the maximum data set size before secondary extents are exhausted. If you select NO, the default value, you will manage secondary extent allocations manually. If you select YES, DB2 will automatically optimize the secondary extent allocations.

When the sliding scale is used, secondary extent allocations that are allocated earlier are smaller than those allocated later, until a maximum allocation is reached. The maximum secondary extent allocation is 127 cylinders for data sets with a maximum size of 16 GB or less, and 559 cylinders for data sets with a maximum size of 32 GB or 64 GB. For more information in this enhancement see 3.18.9, "SMART DB2 extent sizes for DB2 managed objects" on page 211.

- MAX OPEN CURSORS on panel DDNTIPX (MAX_NUM_CUR DSNZPARM) allows you to specify maximum number of cursors, including allocated cursors, that are open at a given DB2 site per thread. The default is 500.
- MAX STORED PROCS (MAX_ST_PROC) allows you to specify a maximum number of stored procedures per thread. The default is 2000.

The previous two new DSNZPARMs are introduced as safety valves. Now that V8 allows you to call the same stored procedure multiple times at the same nesting level, as well as allowing the same SQLJ iterator being instantiated multiple times in the same program, this increases the number of open cursors and active stored procedures for a single thread.

In case of an application error that causes some sort of loop and continues to open cursors or keeps calling the same stored procedure over and over, DB2 needs to make sure that such errors cannot bring down the system. To prevent this from happening, MAX_NUM_CUR and MAX_ST_PROC are activated and will return an SQLCODE -904 to the application before things get out of control.

- LONG-RUNNING READER is a new field on panel DSNTIPE:

  Specify the number of minutes that a read claim can be held by an agent before DB2 issues a warning message to report it as a long-running reader. If you specify a value of 0, DB2 will not report long-running readers. For more information, see 3.18.4, "Detecting long readers" on page 208.

- PAD INDEXES BY DEFAULT is a new field on panel DSNTIPE:

  This field specifies whether new indexes are padded by default. YES indicates that a new index will be padded unless the NOT PADDED option is specified on the CREATE INDEX statement. The default value, NO, indicates that a new index will not be padded unless the PADDED option is specified on the CREATE INDEX statement.

- The Tracing panel, DSNTIPN, introduces two new fields to enhance DB2 accounting reporting:
  - DDF/RRSAF ACCUM:

    Specify whether DB2 accounting data should be accumulated by the user for DDF and RRSAF threads. If NO is specified, DB2 writes an accounting record when a DDF thread is made inactive or when signon occurs for an RRSAF thread. If a value between 2 and 65535 is specified, DB2 writes an accounting record every n occurrences of the user on the thread, where n is the number that is specified for this parameter.

  - AGGREGATION FIELDS:

    Specify the aggregation fields to be used for DDF and RRSAF accounting rollup. The choices are to rollup accounting data by:

    0. End user ID, transaction name, and workstation name
    1. End user ID
    2. End user transaction name
    3. End user workstation name
    4. End user ID and transaction name
    5. End user ID and workstation name
    6. End user transaction name and workstation name

- The Storage Calculation panel, DSNTIPC, allows you to change some new values:
  - EDM STATEMENT CACHE:

    Specify the size (in KB) of the statement cache that can be used by the EDM. This is a new storage pool which is located above the 2-GB bar. The CLIST calculates a default statement cache size; however, you can change it here.
  - EDM DBD CACHE:

    Specify the minimum size (in KB) of the DBD cache that can be used by the environmental descriptor manager (EDM). This is another new storage pool which is located above the 2-GB bar. The CLIST calculates the DBD cache size however you can change it here.
  - STG AVAILABLE ABOVE 2GB:

    This is the default amount of storage available above 2 GB that is available to the DBM1 address space. This value is used to initialize the MEMLIMIT parameter in the DSNDBM1 address space.

  In addition, the fields EDMPOOL DATA SPACE SIZE and EDMPOOL DATA SPACE MAX are removed from the panel (as the EDMPOOL no longer resides in a data space).
- The IRLM parameters PC and MAXCSA parameters are no longer used.

  DB2 Version 8 required IRLM 2.2. In Version 2.2, IRLM ships both 32 bit code and 64-bit code (so, you can expect your SDXRRESL to almost double in size). IRLM 2.2 no longer supports placing locks in ECSA. All IRLM locks are now placed in the IRLM private address space.

  The PC and MAXCSA parameters are no longer used, but you must maintain them for compatibility reasons. In the IRLMPROC JCL, you must specify the parameters and values, but their values are not used. The amount of available storage for IRLM private control blocks, including locks, is now determined by the operating system and site-specific IPL parameters.

  IRLM reserves approximately 10% of the available private storage to be used for must-complete lock requests. Instead you can now use the MLMT parameter on the IRLM startup procedure to control the maximum amount of private storage available for IRLM above the 2 GB bar for storing locks (active and retained locks). MLMT is used to set the z/OS MEMLIMIT value for the IRLM address space.
- “TEMPORARY UNIT NAME” on the DSNTIPA2 panel

  As before, this field is used to specify the device type or unit name for allocating temporary data sets. It is the direct access or disk unit name that is used for the precompiler, compiler, assembler, sort, linkage editor, and utility work files in the tailored jobs and CLISTs. However in V8, DB2 utilities that dynamically allocate temporary data sets also use this parameter. Therefore, this field is now also a DSNZPARM named VOLTDEVT.

Changed Defaults

ALTER BUFFERPOOL

- Deferred write threshold (DWQT) decreased from 50% to 30%
- Vertical deferred write threshold (VDWQT) decreased from 10% to 5%

ALTER GROUPBUFFERPOOL

- Class castout threshold (CLASST) decreased from 10% to 5%
- Group buffer pool castout threshold (GBPOOLT) decreased from 50% to 30%
- Group buffer pool checkpoint (GBPCHKPT). The default is lowered from 8 minutes to 4 minutes.

Installation CLIST Changes

- Cache dynamic SQL now enabled by default
- Fast log apply enabled (100MB) by default
- Checkpoint frequency increased from 50000 to 500000 log records
- Archive log blocksize reduced from 28672 to 24576

12.5.5 Changed defaults

As a result of work done by the DB2 Performance team, a number of DB2 Installation CLIST default settings for *new* systems change in Version 8. This visual shows the major changes.

The initial defaults for the ALTER BUFFERPOOL command change:

- **DWQT**: Specifies the buffer pools deferred write threshold as a default percentage of the total virtual buffer pool size. The initial default is decreased from 50% to 30%.
- **VDWQT**: Specifies the virtual deferred write threshold for the virtual buffer pool. This parameter accepts two arguments. The first argument is a percentage of the total virtual buffer pool size. The default is decreased from 10% to 5%.

The initial defaults for the ALTER GROUPBUFFERPOOL command change:

- **CLASST:** Specifies the threshold at which class castout is started. It is expressed as a percentage of the number of data entries. The default is decreased from 10% to 5%.
- **GBPOOLT:** Specifies the threshold at which the data in the group buffer pool is cast out to DASD. It is expressed as a percentage of the number of data entries in the group buffer pool. The default is decreased from 50% to 30%.
- **GBPCHKPT:** Specifies the time interval in minutes between group buffer pool checkpoints. The default is lowered from 8 minutes to 4 minutes.

In addition, a number of Installation CLIST defaults change:

- The dynamic SQL cache is now enabled by default.
- Fast log apply is now enabled by default.

- The DB2 checkpoint frequency is increased from 50,000 to 500,000 log records written.
- The default block size of the archive log data set is reduced from 28672 to 24576.

  The block size must be compatible with the device type you use for the archive log data sets. If the archive log is written to tape, using the largest possible block size improves the speed of reading the archive logs, while specifying a lower block size for DASD devices reduces the amount of DASD required. A block size of 28672 is the best block size to use for TAPE devices while 24576 is the best block size to use for 3390 type devices.
- The default for "Describe for static" (DESCSTAT) is changed from NO to YES.

  As stored procedures have become increasingly prevalent, the number of folks that turn on this DSNZPARM parameter has correspondingly grown. In addition, all users of the new Universal JDBC driver and new CLI drivers (on DB2 for Linux, UNIX, and Windows) will be dependent on this column information for retrieving meta data from the catalog correctly. Note that the native ODBC driver on DB2 for z/OS does not use the common meta data stored procedures, so does not have the same dependency.

> **Important:** If you are migrating, and DESCSTAT=NO in V7, make sure to change it to YES, when migrating to V8. It can save you a lot of headaches when starting to use the new CLI driver on the DB2 distributed platforms, or the new Universal JDBC driver later on.

The table below details all of the changes that are made to the DB2 installation CLIST default values.

Table 12-1 DB Installation CLIST default changes

Panel Id	Panel Parameter	Current Default	Version 8 Default
DSNTIP7	User LOB value storage	2048(kb)	10240(kb)
DSNTPIE	Max users	70	200
DSNTIPE	Max remote active	64	200
DSNTIPE	Max remote connected	64	10000
DSNTIPE	Max TSO connect	40	50
DSNTIPE	Max batch connect	20	50
DSNTIPF	Describe for static	NO	YES
DSNTIPN	SMF statistics	YES (1,3,4,5)	YES (1,3,4,5,6)
DSNTIP8	Cache dynamic SQL	NO	YES
DSNTIPP	Plan auth cache	1024 bytes	3072 bytes
DSNTIPP	Package auth cache	32 KB	100 KB
DSNTIPP	Routine auth cache	32 KB	100 KB
DSNTIPL	Log apply storage	0	100 (MB)
DSNTIPL	Checkpoint frequency	50 000 records	500 000 records
DSNTIPA	Block size (archive log)	28672	24576
DSNTIPR	DDF threads	ACTIVE	INACTIVE
DSBTIPR	IDLE tread time-out	0	120

Panel Id	Panel Parameter	Current Default	Version 8 Default
DSNTIPR	Extended security	NO	YES
DSBTIP5	TCP/IP KEEPALIVE	ENABLE	120 (seconds)
DSNTPIC	Max open data sets	3000	10000
DSNTIPC	EDMPOOL Storage Size	7312 (KB)	32768 (KB)
DSNTIPC	EDM Statement Cache	n/a	102400 (KB)
DSNTIPC	EDM DBD Cache	n/a	102400 (KB)

Installation Considerations

Same as prior releases

MUST run job DSNTIJTC

- A new installation step

NEW!

Must run DSNTIJUZ (DSNZPARM, DSNHDECP, DSNHMCID) before DSNTIJID (initialization C+D data sets)

- DSNTIJID runs DSNJU003 and uses info fromDSNHMCID

All new functions are immediately available

- No need to enable new functions (see later)

The Version 8 IVP jobs are created by the installation CLIST as part of the Installation customization process

- Migration is different (see later)

12.5.6 Installation considerations when installing a brand new V8 DB2 system

The process to install a new DB2 Version 8 subsystem is the same as installing a new DB2 Version 7 subsystem. However, there is one extra job that must be run.

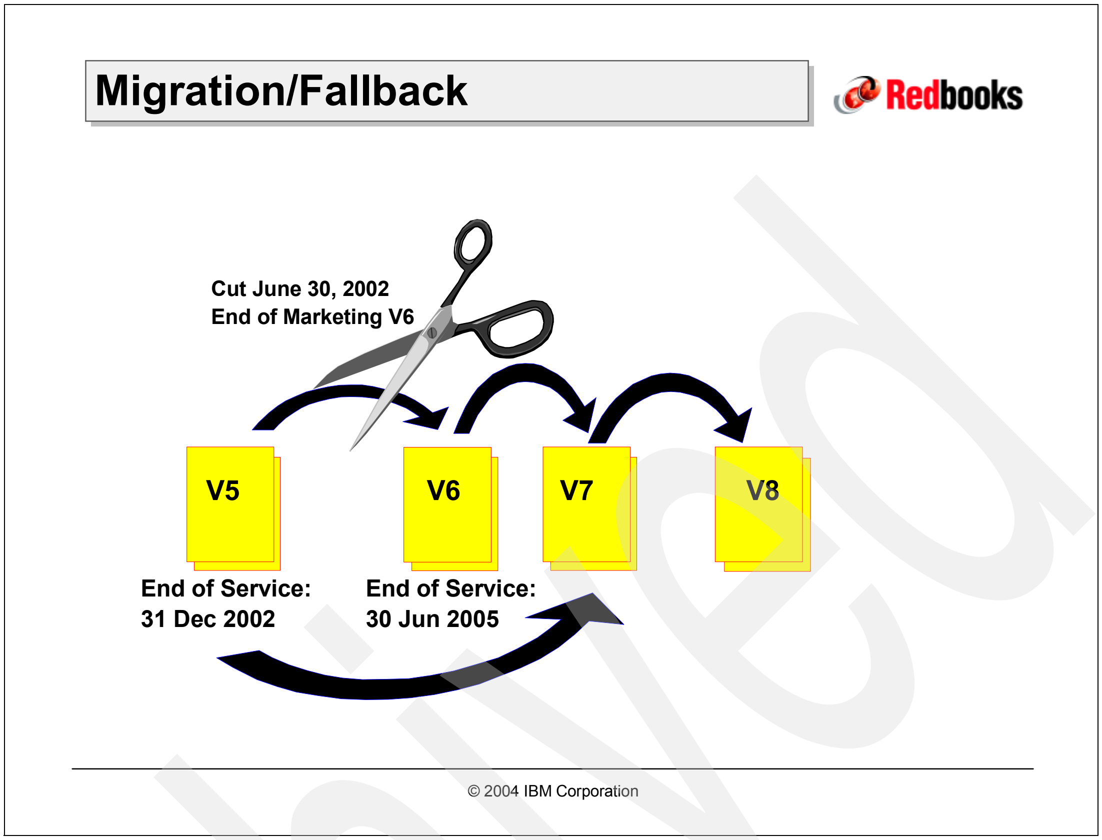
The DB2 Version 8 Installation CLIST generates the job DSNTIJTC, which must also be run. The job DSNTIJTC executes the CATMAINT utility. In previous versions, you run the CATMAINT utility only during migration, to update the DB2 catalog for the new version.

With DB2 V8, job DSNTIJTC must be run for new installations, to update the DB2 catalog for the table spaces which are not moving to Unicode, with the default EBCDIC encoding schemes from DSNHDECP. The CATMAINT UPDATE utility should be very quick.

There is another little difference from a V7 installation. It is related to the sequence in which you execute the installation jobs. In V7, the DB2 installation guide directed you to run job DSNTIJID (initialization of the DB2 system data sets, including catalog and directory) prior to DSNTIJUZ (creating the system parameters). In V8, you need to run DSNTIJUZ before you can successfully run DSNTIJID, because DSNTIJID executes the DSNJU003 (change log inventory) utility which is dependent on the DSNHMCID module created by DSNTIJUZ.

The procedure to migrate a DB2 subsystem to Version 8 has changed from previous releases (more on this later). After the DB2 catalog has been migrated to Version 8, no new DB2 Version 8 functions are available to you until you first “enable” the DB2 subsystem for the new functions. This restriction does not apply when you install a brand new DB2 subsystem. All the new Version 8 functions are immediately available to you.

When you use the DB2 Installation CLIST to install a new DB2 subsystem, the DB2 Version 8 IVP jobs are also generated. You can run these jobs to verify the new DB2 Version 8 subsystem. When you migrate, the IVP jobs are only generated when using the ENFM migration option. In order to be able to do some basic testing in V8 compatibility mode, you should install the V7 samples (through phase 3) before migrating.



12.6 Migration/Fallback

As you can see by this visual, you can only migrate to DB2 Version 8 from DB2 Version 7.

The “skip” migration from DB2 Version 5 to DB2 Version 7 was a “once off” migration strategy, to allow customers to move forward quickly and take advantage of the rich new function provided in later releases of DB2 after getting over the Y2K hurdle. This strategy will not be continued in future migrations.

You will need to first migrate your DB2 subsystems to Version 8 and have moved to new-function mode before you will be allowed to migrate to the next version of DB2 after Version 8.

Please also note that migrating from DB2 Version 5 to DB2 Version 6 is no longer supported. This is because DB2 Version 6 was withdrawn from marketing in June 2002.

Also note that DB2 Version 6 will be going out of service on 30 June 2005; again another reason to move to Version 7 and beyond.

Typical Migration Process Today

Redbooks

Process built up over years of DB2 migration experience

1. Test install and migration/fallback on test system
2. Rollout across strategic development environments
   - Verify compatibility of old functions on new release
3. Finally migrate production to the new release
4. Begin to use new functions
   - When satisfied with new release

Use no new function during migration timeframe

12.6.1 Typical migration process today

Here is the typical migration process we have today. It is built up over many successful DB2 migrations; from DB2 Version 1.1 through until Version 7:

1. Test the install and migration/fallback processes on a software testing environment.

   This is to make sure the migration and fallback processes works, you are able to successfully migrate your DB2 environment to the new version, you have tested fallback, and you are familiar and comfortable with what is involved in migration and fallback.

2. Roll out the new version into development environments.

   This provides more extensive testing of the new version and is done to ensure that your current applications execute correctly with the new software.

3. Migrate the new release into production.

   Once you are satisfied that your existing applications function correctly under the new version of DB2, you are ready to migrate your production environment to the new version of DB2.

4. Begin to exploit new features.

   We recommend that you begin to exploit the new features, beginning in the development DB2 environments, only once you are satisfied with the new level of software and have successfully migrated all of your production DB2 environments.

This migration process is designed to avoid objects becoming unavailable and “frozen” should you have to fallback to the previous release.

Possible Fallback Issues

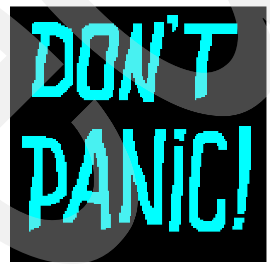
No control over use of new functions when verifying compatibility of old functions on new release

Multiple user groups eager to test / validate DB2 software

- New function dabbling

Use of "new functions" sometimes precludes fallback

- Firefight through the problems
- Use of "new function" usually not discovered until fallback
  - A time for panic!



Fallback requires application of fallback SPE

- Sometimes overlooked during migration

Continuous migration and fallback scenario

- Data sharing coexistence

12.6.2 Possible fallback issues

While this typical migration strategy has served well in the past, it is not without its faults. We have no control of when new features can be used and by whom. As soon as a DB2 environment is migrated to the new version, anyone can potentially begin to explore the new function. Some features may require extra DB2 objects or extra security to be defined. The use of these new features can of course be controlled, but other features cannot be restricted.

Many developers are keen to dabble in the new features offered by the new release and sometimes these new features find their way into new application releases before you would like them to be. This brings about its own set of problems.

Application developers may be putting pressure on you to implement the new version of DB2 into production sooner than you would like, as the new application release is now dependent on the new DB2 release.

After the production DB2 environments have been migrated to the new version, new features can find their way into production without you knowing. This may complicate any fallback that you may have to perform, because after fallback, some applications (or parts of applications) may no longer work. In more extreme cases, you may not be able to fallback to the previous version because you cannot afford to lose the application.

All supported fallback scenarios require the fallback SPE to be applied to the previous version of DB2. However, sometimes the fallback SPE is overlooked! This may place your fallback plans in jeopardy, and there is no worse time to find out than during a fallback.

In addition, DB2 Version 8 has so many enhancements over Version 7, that the Version 7 fallback SPE would be huge to develop and get Version 7 code to run on a Version 8 catalog. The one-step migration strategy that has been used in past releases has therefore become very expensive and error prone (maintaining the fallback SPE).

Why a "New" Migration Process

Redbooks

Significant enhancements to the product - Largest release ever!

- Support for long names
- Unicode catalog tables

More control over the migration process

- In line with system availability / maintenance window
- Timing the introduction and use of the new functions

A tighter, more robust migration process

- Fewer migration & fallback errors - Forced SPE Installation

12.6.3 Why a “new” migration process

Migrating to Version 8 is by far the largest migration of any release of DB2. For example, the DB2 catalog will be converted from EBCDIC to Unicode and many object names will be increased to 128 characters long. During the migration process you are essentially replacing the whole DB2 catalog with a new one. Almost nothing remains unchanged.

DB2 Development has designed the new migration process to be a number of distinct phases, to break the whole migration task into a number of smaller steps which you can better manage. Now, you do not have to plan to perform all of the migration tasks at the one time, thereby increasing availability and allowing you to better plan your DB2 upgrade activities. The migration process to Version 8 also becomes less complicated and therefore less prone to failure because the one big task is broken down into a number of smaller less complicated tasks.

The introduction of “phases” has some benefits for you as well, as it addresses the problems we have described on the previous visual.

It gives you control over when you want to release all the new function in DB2. Application developers can no longer implement the new features offered by DB2 into their applications in development and subsequently move them to production, without you having to first ‘enable’ those DB2 environments for the new features. You can now control what environments are enabled for the new functions and when. A suggested strategy is to not enable any DB2 environment for the new function until all of your DB2 environments have been migrated to Version 8, they are stable, you are happy with them and you no longer need to fallback to Version 7. Once you have decided you are not going to fallback to Version 7, you can first enable the development DB2 environment(s) for the new function, then the production DB2 environment(s) for the new function.

Planning for DB2 migration and fallback should now be much easier.

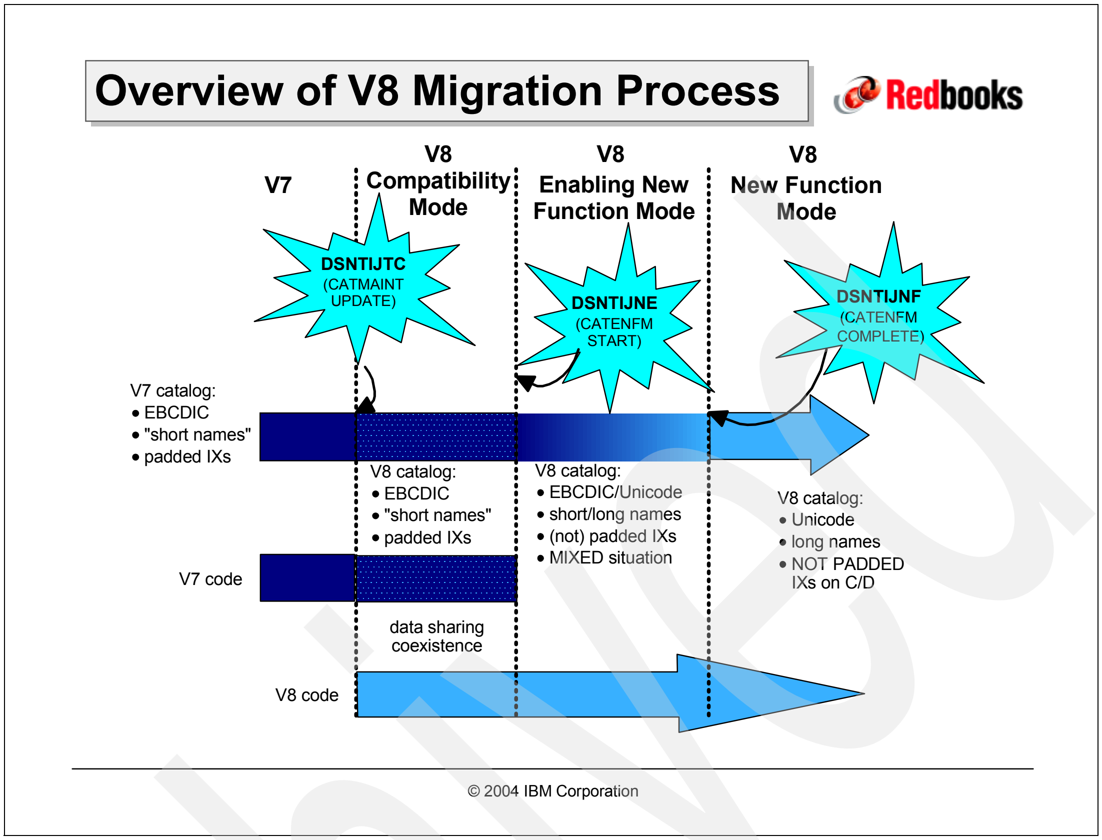
DB2 now also checks to see if the fallback SPE(PQ48486) has been applied to your DB2 Version 7 before it allows you to migrate to Version 8. You must apply the fallback SPE and restart your V7 with the fallback SPE at least once.This restriction is now enforced for both data sharing and non-data sharing. Previously the restriction was only enforced for data sharing environments. This enhancement should also reduce the number of fallback errors.

In case the fallback has not been applied when you want to migrate to V8, the following message appears (Example 12-1) and DB2 will terminate.

Example 12-1 Starting V8 without the fallback SPE applied

```
DSNR045E  -D38C DSNRRPRC DB2 SUBSYSTEM IS STARTING AND
          IT WAS NOT STARTED IN A
          PREVIOUS RELEASE WITH THE FALLBACK SPE APPLIED.
          FALLBACK SPE APAR: PQ48486
          NEW RELEASE LEVEL: 0000D301
          KNOWN LEVEL(S): 0000D20200000000000000000000000000
```

In addition, the fallback SPE is smaller, easier to manage, and less error prone.



12.6.4 Overview of the V8 migration process

DB2 Version 8 moves through three distinct phases during migration from Version 7 to Version 8:

- Version 8 compatibility mode (CM):

  This is only a transitional phase where no new Version 8 external functions are enabled (for example, no new long names). Compatibility mode allows you to test and ensure that the new Version 8 code is functioning correctly and your existing applications are working correctly under the new code.

- Enabling-new-function mode (ENFM):

  This is the phase where the catalog and directory are converted to a Version 8 new-function mode catalog. Once again, most of the new V8 functions are not available.

- New-function mode (NFM):

  DB2 enters new-function mode after all the migration tasks are complete. All the new function in DB2 Version 8 is now available.

Migration now consists of two distinct migration processes to advance DB2 from one phase to the next:

1. New release migration processing:

   This migration consists of “normal” release migration processing as we have had in previous releases. It uses the CATMAINT utility to convert the DB2 catalog and directly to Version 8 format. The completion of this migration phase places the DB2 subsystem into Version 8 compatibility mode.

2. Enabling-new-function mode processing:

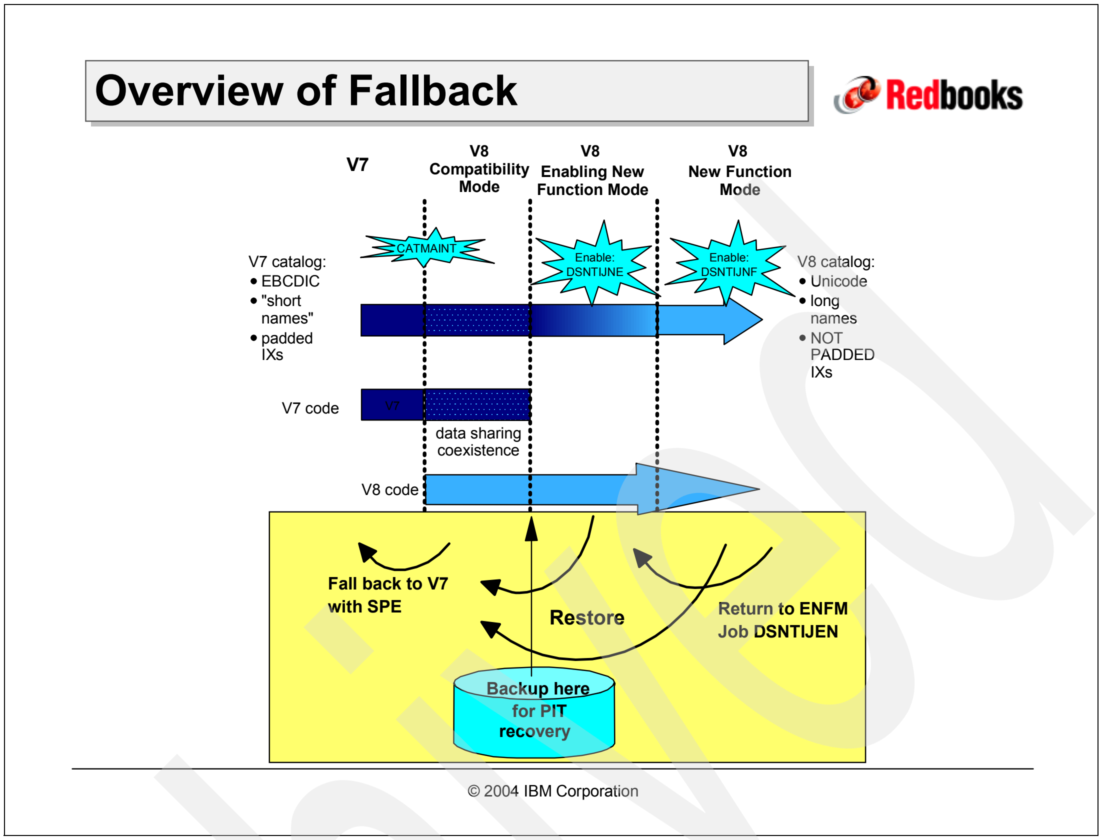
   This process consists of additional Version 8 conversion processing. Before you consider moving to new-function mode, you must first ensure that you are happy with the Version 8 code that is running, that it is in production, and it is stable.

   During the enabling-new-function mode phase, the DB2 catalog and directory are converted to Unicode. This phase can be scheduled over a period of time. After the completion of this phase, you can move the DB2 into new-function mode. This is the end of the DB2 Version 8 migration process and all new DB2 Version 8 function is now available.

Although you can return from “new-function mode” to “enabling-new-function mode” (the transition phase from “compatibility mode” to “new-function mode”), there is no path to fallback to DB2 Version 7, once you are either in “enabling-new-function mode” or “new-function mode”. More information on this will be given later.

This new migration process merely enforces and formalizes what many people already do today when they plan their migration to a new version of DB2.

We explain the migration phases in more detail in the next few visuals.



12.6.5 Overview of fallback

What do we mean by fallback?

"Returning to a stable DB2 code base after successfully migrating catalog and directory to Version 8 compatibility mode"

Fallback from V8 compatibility mode to V7

The fallback procedure is the same as for previous versions:

1. Stop Version 8.
2. Restore the Version 7 libraries.
3. Start DB2 under Version 7.
4. Rebind DSNTIAD and SPUFI if required.
5. Run the Version 7 IVP.

However, before a successful fallback can occur, the fallback SPE must have been applied to your DB2 Version 7 system and it has at least been started once under Version 7 before attempting to start DB2 in a fallback release. This should be OK, as DB2 now checks to see if the fallback SPE has been applied before starting CATMAINT processing during the migration to compatibility mode. (See the discussion on "Catalog and DB2 Code Level" in the previous visuals.)

Important: Fallback to Version 7 is ONLY permitted from Version 8 running in compatibility mode. The structure of the Version 8 catalog is used in Version 7 after fallback. Once DB2 has entered enabling-new-function mode, or new-function mode, fallback to Version 7 is not supported.

Falling back to Version 7 from Version 8 compatibility mode does not undo any changes made to the catalog and directory during a migration to Version 8. The migrated catalog is used after fallback. Some objects in this catalog that have been affected by new function in this release might become frozen objects after fallback. Frozen objects are unavailable, and they are marked with a release dependency indicator. The release dependency indicator is recorded in the DB2 catalog against each object using the IBMREQD column. DB2 Version 8 records a release dependency indicator of 'L'.

Frozen objects should be rare in Version 7 after a successful fallback from Version 8. This is because much of the new function is not available in Version 8 until new-function mode. Remember that DB2 does not support fallback from Version 8 new-function mode to Version 7.

Frozen objects can include plans, packages, views, indexes, tables, table spaces, and routines. In Version 8, the following objects can become frozen after fallback to Version 7:

- Plans, packages, or views that use any new syntax, objects, or bind options.
- DBRMs that are produced by a precompile in Version 8 with a value of YES for the NEWFUN option. These DBRMs are in Unicode.
- User-defined functions created in Version 8 with the PARAMETER CCSID option.
- User-defined SQL procedures and functions created in Version 8 with the PARAMETER CCSID option.

After fallback, all plans or packages that are bound in Version 8 are automatically rebound on their first execution in Version 7. If you try to use plans or packages that are frozen, the automatic rebind in Version 7 that takes place the first time that you try to run the plan in Version 7 fails.

Returning from V8 new-function to V8 enabling-new-function mode

Once you are in new-function mode, you can only return to enabling-new-function mode by running the job DSNTIJNE and re-assembling DSNHDECP with NEWFUN=NO. No catalog updates that were made in new-function mode will be undone. No objects including plans and packages are frozen. Plans and packages are not subject to automatic rebinds and they will continue to work, even if they use any new function or reference any new function dependent objects. However, after returning to enabling-new-function mode, you cannot use the new functions any more. For example, you can still use tables that are defined using table based partitioning; however, you cannot create any new tables using table based partitioning.

Important: We therefore suggest that not many would use this option. It merely prevents any new Version 8 function from being used, but does not prevent any existing Version 8 new function objects from being used.

There is no support for returning from enabling-new-function mode to compatibility mode. If you need to do this, then the only way is by performing a point-in-time restore of the whole DB2 environment.

When you are executing under Version 8 compatibility mode, there are no more hiperpools or buffer pools in data spaces. On fallback to Version 7, DB2 remembers the buffer pool allocations used in Version 7 and reinstates those values.

Compatibility Mode

Redbooks

No new release function can be used that may preclude fallback or coexistence

- Some new function available like online REORG of entire DB2 catalog and 64 bit benefits also available now

Migrate to V8 compatibility mode requires normal CATMAINT

- Time will mostly likely be determined by the time it takes to scan SYSDBASE

DSNTIJTC -- CATMAINT is a single step

- Single commit scope -- all or nothing
  - Authorization check
  - Ensures catalog is at correct level
  - DDL processing
  - Additional processing and tailoring
  - Directory header page and BSDS / SCA updates
- No need to have additional steps to look for unsupported objects

12.7 Compatibility mode

The DSNTIJTC job invokes the CATMAINT UPDATE utility in order to migrate the catalog and directory to the current release, in this case Version 8. This is the same as all prior releases of DB2. The end result of this migration process is a current Version 8 catalog.

A successful migration to DB2 Version 8 places the DB2 subsystem into Version 8 compatibility mode. In compatibility mode, no new Version 8 function is available for use that may compromise a successful fallback to Version 7, or coexistence in a data sharing environment. To be able to use new Version 8 function, a DB2 subsystem or data sharing group must first convert their catalog to a new-function mode catalog via the enabling-new-function mode process (this is described later).

While in compatibility mode, you can already take advantage of some of the new functions in DB2 V8. For example, as DB2 V8 is always running in 64 bit mode, irrespective of whether you are in compatibility mode, enabling-new-function mode, or new-function mode, you can immediately take advantage of the larger buffer pools (provided that you have enough real storage) and memory relief due to the new 64 bit architecture.

You may choose to rebind your applications while in compatibility mode, to get the benefits of some of the Version 8 optimizer enhancements (and hopefully better access paths). This does not pose a problem if you need to fallback to Version 8, as an automatic rebind is forced for all your Version 8 bound plans and packages anyway. However, while it can be said that there is no new function available in compatibility mode that would preclude a successful fallback, it cannot be said that all new function that falls into this category is available in compatibility mode. So, while rebinding in compatibility mode might be a good thing, rebinding in new-function mode is a much better strategy.

Attention: In Version 8, DB2 now enforces the restriction that there must be no outstanding utilities started from prior releases when running the CATMAINT UPDATE utility on a non-data sharing system, otherwise message DSNU790I will be issued. The reason being is that all utilities must be terminated in the same release they were started.

The CATMAINT utility does not enforce this restriction for data sharing environments because it is valid to have Version 7 members active with utilities while CATMAINT is being run on a Version 8 member to upgrade the catalog.

Migration to DB2 Version 8 is only supported from Version 7. Before attempting to migrate to Version 8, make sure that Version 7 is at the proper service level. Refer to 12.1, “Planning for Version 8” on page 881 for a discussion of what the proper service level is for both data sharing and non-data sharing systems.

The end result of the migration from Version 7 to Version 8 is a new, Version 8 catalog. However, the catalog is still in EBCDIC and is not equipped for using long names at this point. The processing needed for each catalog migration changes from release to release. The migration process continues to change and evolve as we continually try to improve the CATMAINT utility update process. Continuous availability is one of the main driving forces of these process improvements.

The catalog migration (from V7 to V8 compatibility mode) in Version 8 is a single step job, with an all or nothing commit scope. Job DSNTIJTC now only needs to be a single step job. There is no need to have an extra step to migrate stored procedure definitions from SYSPROCEDURES to SYSROUTINES that was performed in previous versions. In addition, the CATMAINT utility is now designed to fail if it finds any unsupported functions. Therefore, there is no need to have an extra step to report on these unsupported functions. The job DSDNTIJPM is provided to report on any unsupported functions, which you can run prior to running the CATMAINT utility.

The CATMAINT UPDATE utility performs the following tasks:

- It adds entries in the catalog and directory for new catalog objects:

  Briefly, two new table spaces are created (SYSALTER and SYSEBCDC) and three new tables are created (SYSIBM.IPLIST, SYSIBM.SYSSEQUENCEAUTH, and SYSIBM.SYSOBDS). In addition, new columns are added to existing tables and other columns have new values. Some new indexes are defined and some new constraints are created. Refer to “DB2 Catalog Changes” in this chapter for a little more detail on what will change, or the *DB2 SQL Reference,* SC18-7426 for a more complete description of the DB2 catalog changes.

  When changing the tables in the catalog, DB2 also regenerates the views that may exist on these tables. When that process fails, these views are marked with view regeneration errors. To determine which views were marked with view regeneration errors during migration, issue the following query:

  ```
  SELECT CREATOR,NAME
    FROM SYSIBM.SYSTABLES
    WHERE TYPE = 'V'
      AND STATUS = 'R'
      AND TABLESTATUS = 'V'
  ```

  If any views have view regeneration errors, you can issue the following query:

  ```
  ALTER VIEW view REGENERATE
  ```

  In this query, `view` is the name of the view with regeneration errors.

- The CATMAINT UPDATE utility makes additional updates to the migrated catalog and directory, to indicate the new “level” of the catalog and directory.
- This utility looks for any unsupported objects, and data in the catalog. Before going ahead with the conversion to a Unicode catalog (during enabling-new-function mode processing), DB2 wants to make sure that this conversion has a maximum chance of success. Therefore, during CATMAINT processing, DB2 will check for unsupported characters in the catalog, by scanning a number of DB2 catalog table spaces. Of those catalog table spaces, SYSDBASE is usually the largest. Therefore, you can expect the elapsed time of the CATMAINT utility run to be close to the time it takes to scan SYSDBASE.

Version 8 CATMAINT processing is terminated if any Type 1 indexes are found in the DB2 catalog. All CATMAINT processing will be rolled back if any Type 1 indexes are found. It should be noted that CATMAINT processing is terminated when the first Type 1 index is discovered. There could be others. The queries provided in the DSNTESQ and DSNTIJPM jobs should be used to identify all the unsupported objects in a catalog.

Compatibility Mode Considerations

Unicode parser

- DB2 now parses all SQL in Unicode
- SQL written to IFCIDs can be in Unicode (UIFCIDS DSNZPARM)
- Message DSNH330I can occur in some applications now which must be converted to Unicode
- String Constants may now exceed the maximum length
  - DB2 must convert string constants to Unicode before using them
  - Some characters are represented longer in Unicode
    - EG: “¬”
  - Message DSNH102I or SQLCODE -102

Buffer pools may require more memory

12.7.1 Compatibility mode considerations

Here we discuss some considerations regarding compatibility mode.

Unicode parser

- DB2 V8 always parses SQL statements in Unicode UTF-8 (CCSID 1208). Therefore, if an application program uses a different CCSID (such as an EBCDIC CCSID), DB2 converts the application program to Unicode for processing by the precompiler. This conversion occurs for the entire application program (to allow the precompiler to check the host language definitions), not just for SQL statements.

  If certain errors occur in host language statements (not just in SQL statements), the message DSNH330I can appear. Examples of such errors are an invalid code point, a mismatch between shift-in and shift-out, and absence of half of a DBCS character. These may have existed already anywhere in the program source code, even in comment lines, but were not detected until now, when the entire source code must be converted to Unicode.

  Programs that execute dynamic SQL are no different. DB2 must convert the program to Unicode to parse the code to discover PREPARE, DESCRIBE, EXECUTE, OPEN, FETCH, and CLOSE statements, etc. During the processing of the PREPARE statements, DB2 must convert the contents of the host variables containing the SQL statements to Unicode before they can be processed.

- Various contexts of string constants have various maximum lengths. To check the length of a string constant, DB2 uses the Unicode UTF-8 representation of the string constant, even if the eventual destination of the constant (for example, a column of a table) does not use UTF-8. This length might differ from the length that you entered, if you entered the string constant in a CCSID other than UTF-8.

For example, the character “¬” requires one byte in EBCDIC, but two bytes in UTF-8. Therefore, in some contexts, a string that was valid in Version 7 might be flagged as too long in Version 8, and you will receive the message DSNH102I (in a precompilation) or SQLCODE -102 (otherwise).

This incompatibility can occur only if both of the following conditions exist:

a. The string contains one or more characters whose UTF-8 representations require more bytes than their original representations did.

b. This expansion causes the string to grow beyond the maximum allowed length.

There are two groups of contexts in which this incompatibility arises. In the first group of contexts, such a string is flagged as too long in any mode of DB2 Version 8, because the maximum permitted lengths did not grow in Version 8:

- In ALTER INDEX or CREATE INDEX, VALUES (constant)
- In ALTER TABLE or CREATE TABLE, FIELDPROC program-name (constant)
- In ALTER TABLE or CREATE TABLE, CHECK (check-condition)
- In ALTER TABLE, CREATE TABLE, and DECLARE GLOBAL TEMPORARY TABLE, under DEFAULT, in a constant
- In COMMENT ON, IS string
- In LABEL ON, IS string
- SET CURRENT LOCALE LC_CTYPE
- SET CURRENT OPTIMIZATION HINT
- SET CURRENT SQLID
- In SIGNAL SQLSTATE, diagnostic-string-constant.

In the second group of contexts, such a string is flagged as too long in compatibility mode and enabling-new-function mode, but the string is not flagged as too long in new-function mode, because new-function mode allows longer strings:

- CASE expression
- CALL procedure-name (expression)
- Expression in a predicate
- Expression in a WHERE clause
- Expression in a HAVING clause
- Expression in a SELECT clause in a subselect
- Expression specified in a built-in function.Join expression

► While all trace records remain in EBCDIC, you have a DSNZPARM option to write all SQL statements into IFCID records in Unicode UTF-8, not in EBCDIC. The DSNZPARM parameter is UIFCIDS. When using the default value (NO), DB2 will continue to write the SQL statements into IFCID records as EBCDIC.

Only a subset of the character fields (identified in the IFCID record definition by a “%U” in the comment area to the right of the field declaration in the DSNDQWxx copy files) are encoded in Unicode. The remaining fields maintain the same encoding of previous releases.

Buffer pool considerations

When you are executing under Version 8 compatibility mode, there are no more hiperpools or buffer pools in data spaces. On migration to Version 8 compatibility mode, DB2 allocates buffer pools that are equal in size to the equivalent virtual buffer pools plus any equivalent hiperpool size. (Version 8 also imposes a restriction that no one buffer pool can be larger than 1 TB, nor can the sum of all buffer pools exceed 1 TB.) On fallback to Version 7, DB2 remembers the buffer pool allocations used in Version 7 and reinstates those values.

Important: If you currently use hiperpools and/or buffer pools in data spaces, we recommend that you review your buffer pool allocations before migration to Version 8 compatibility mode. (Refer to the discussion above regarding how buffer pools sizes are migrated to Version 8.)

This recommendation is to check that you will have enough real memory to back the memory that DB2 Version 8 will use for buffer pools. If you do not perform this "sanity check" before migrating to Version 8, you may see an unexpected leap in the amount of memory DB2 version 8 demands. This may adversely impact performance.

Coexistence in Data Sharing

Coexistence only between V7 and V8 CM

- Must have fallback SPE (**PQ48486**) on all V7 members
- **Enforced** for both, data sharing and non-data sharing
  - Information in the BSDS / SCA is checked at DB2 startup time to ensure that all group members have the SPE on
  - New starting member's code level checked to ensure coexistance with the catalog

NO coexistence between an ENFM and CM

- Once ENFM begins, all active members of group are considered to be in ENFM status
  - Cannot start a V7 subsystem
  - Could start a V8 member. It would join in ENFM status
  - ENFM (and NFM) mode are group wide events

12.7.2 Coexistence in data sharing

DB2 data sharing in coexistence mode has its own complexities. (Please refer to the manual, *DB2 Data Sharing: Planning and Administration,* SC18-7417 for more details.)

Coexistence of multiple releases of DB2 is particularly of interest in a data sharing environment. The objective is that a data sharing group can be continuously available for any planned reconfiguration, including release migrations. So, if you have a data sharing group consisting of all Version 7 DB2 members, you can migrate your group to Version 8 by “rolling-in” the release migration one member at a time (similar to the way you would roll in a PTF for an APAR fix), thus keeping at least a subset of the DB2 members available at all times. During the “rolling-in” process, you will have a period of time where there are Version 7 and Version 8 members coexisting within the same data sharing group.

DB2 will support the coexistence of only two releases at a time. DB2 does not support the coexistence with a Version 7 DB2 member if any Version 8 DB2 member is in new-function mode. Coexistence is only supported in compatibility mode. During this period of coexistence, any new function available in Version 8 CM is not available to the downlevel members.

All DB2 members MUST be in Version 8 compatibility mode before you move to enabling-new-function mode.

During a migration to Version 8, the other group members may be active. CATMAINT processing will take whatever locks are necessary for the processing it needs to do. The other active group members may experience delays and/or time-outs if they try to access the catalog objects that are being updated or locked by migration processing. We therefore recommend that migration to compatibility mode be scheduled during a period of low activity, or preferably, only have one DB2 subsystem active for the migration. Once the migration to Version 8 completes, the other group members can be brought up to Version 8 at any time.

Catalog and DB2 code level

DB2 Version 8 more strictly enforces the concept of "Catalog and DB2 Code Level". The level of DB2 code (PTF level) must match the level that is recorded in the DB2 catalog. This check ensures that you are starting DB2 with the right SDSNLOAD library, to avoid any catalog corruption caused by executing the wrong code base against the DB2 catalog.

Non-data sharing: At DB2 startup time, the code level of the starting DB2 is checked against the code level required by the current DB2 catalog. If the starting DB2 has a code level mismatch with the catalog then the message DSNX208E is issued and DB2 will not start. A code level mismatch indicates that the starting DB2 is at a level of code that is down level from what it needs to be for the current catalog.

If the catalog has been migrated to Version 8, then the starting DB2 must be at Version 8 or Version 7 with the fallback SPE (PQ48486) on.

Before attempting to migrate to Version 8, it is necessary to start DB2 at least once as Version 7 with the fallback SPE on, prior to attempting to migrate to Version 8. If this is not done, then the Version 8 migration will be terminated. This is a change from the way the fallback SPE was handled for non-data sharing in previous releases.

Data sharing: At DB2 startup time, the code level of the starting DB2 is checked against the code level required by the current DB2 catalog and against the code level of the other DB2s that are active. If the starting DB2 has a code level mismatch with the catalog or any of the other DB2s that are running then either the DSNX208E or DSNX209E message will be issued and DB2 will not start.

A code level mismatch indicates that the starting DB2 is at a level of code that is down level from what it needs to be for the current catalog, or that one or more of the already running DB2s are down-level from where they need to be. Before attempting to migrate to Version 8, all started DB2 subsystems must have maintenance through the Version 8 fallback SPE on before any attempt is made to migrate any member to Version 8. If the fallback SPE is not on all active group members, then DB2 Version 8 will not start and you will not be able to attempt the Version 8 migration. One of the messages, DSNX208E or DSNX209E, will be issued in these cases.

Quiesced DB2 members are not a concern because it may be valid to have a quiesced member that does not have the right code level. The quiesced member may be a DB2 member that is no longer used or is rarely started. The quiesced member will fail to start anyway if it is not at the right code level.

Call attachment and TSO attachment coexistence

While you are in a coexistence environment, you can attach to either release of DB2 with your existing TSO logon procedures or with JCL. After you migrate all members of the group to the latest level of DB2, Version 8, update those procedures and jobs to point to the latest level of DB2 load libraries.

Avoiding automatic rebinds

When planning for migration, new functions introduced in the new release are not available to members of the group who have not yet migrated.

Plans and packages that have the new release dependency indicator set (column IBMREQD = “L” in the tables SYSIBM.SYSPLAN and SYSIBM.SYSPACKAGE), cannot be executed on members who have not been migrated. An automatic rebind must first occur on a DB2 member running the old release. When the plan/package is re-executed on the newly migrated member again, another automatic rebind must occur. This can lead to a “thrashing” situation, where plans and packages are continually being rebound on Version 7 and Version 8 as they are being executed.

A number of strategies exist to avoid this thrashing. For example; do not allow packages and plans bound on Version 8 to execute on members that have not yet been migrated, or, do not allow plans or packages to be bound on Version 8 until all members are migrated.

This serves two purposes. First, if those Version 8 bound plans or packages are using new functions, you can avoid the application errors that occur if the plan or package tries to execute an SQL statement that is not allowed in the release from which you are migrating (Version 7). Second, it avoids the automatic rebind that occurs when any plan or package that is bound on Version 8 is run on the previous release. It also avoids the automatic rebind that occurs when a Version 8 bound plan or package that was automatically rebound on the previous release is later run on Version 8.

If it is not possible to enforce on which member a plan or package runs, consider how you want to handle binds and automatic rebinds while two releases are coexisting. One approach is to disallow all binds and disable all automatic rebinds on the Version 8 subsystem. The other approach is to disable only those automatic rebinds that occur on Version 8 in step 3 of the following scenario:

1. A plan or package is bound on Version 8.
2. The plan or package is run on a non-Version 8 (automatic rebind occurs on non-Version 8)
3. The plan or package is run on Version 8 (automatic rebind occurs on Version 8).

Because DB2 perceives this scenario as a fallback and remigration scenario, the autobind that occurs in step 3 is called a remigration rebind. By disallowing the automatic rebind in step 3, you are avoiding the thrashing that can occur by having the plan or package rebind every time it runs on a member of a different level.

Disallowing all binds: You can specify NO on the AUTO BIND field of installation panel DSNTIPO for all Version 8 members (ABIND parameter in DSNZPARM). This disables all automatic rebinds on the Version 8 member for any reason. You need to also use the resource limit facility to disallow BIND operations. Do this by inserting rows in the resource limit specification table (RLST) to set RLFFUNC to “1” and RLFBIND to “N”. This ensures that nobody binds plans or packages on Version 8 (or V7).

Disallowing only the automatic remigration rebind: To avoid the automatic remigration rebind, specify COEXIST for the AUTO BIND field on installation panel DSNTIPO of the Version 8 members. This means that automatic rebind occurs on Version 8 only in the following circumstances:

- The plan or package is marked invalid.
- You migrate to a future release, bind a plan or package on that release, and then run the plan or package on Version 8.

Recommendations for BIND

If the DSN TSO command is at Version 8 and the DB2 member that is named in the DSN command is at Version 7, using certain bind options causes a BIND or REBIND subcommand to be rejected. (The list of options differs, depending on the release from which you are migrating.). If you are migrating from Version 7, the ENCODING option on BIND and REBIND PLAN or PACKAGE will cause the BIND and REBIND subcommand to fail.

To avoid problems, make sure the DB2 subsystem named in the DSN subcommand matches the load libraries that are used for the DSN command.

Recommendations for utilities

Until all members of the data sharing group are running at the new release, avoid using any of the new utility functions available in Version 8 compatibility mode. However, as long as you use utility options that are supported in Version 7, utilities can attach to a member at either a Version 7 or Version 8 subsystem.

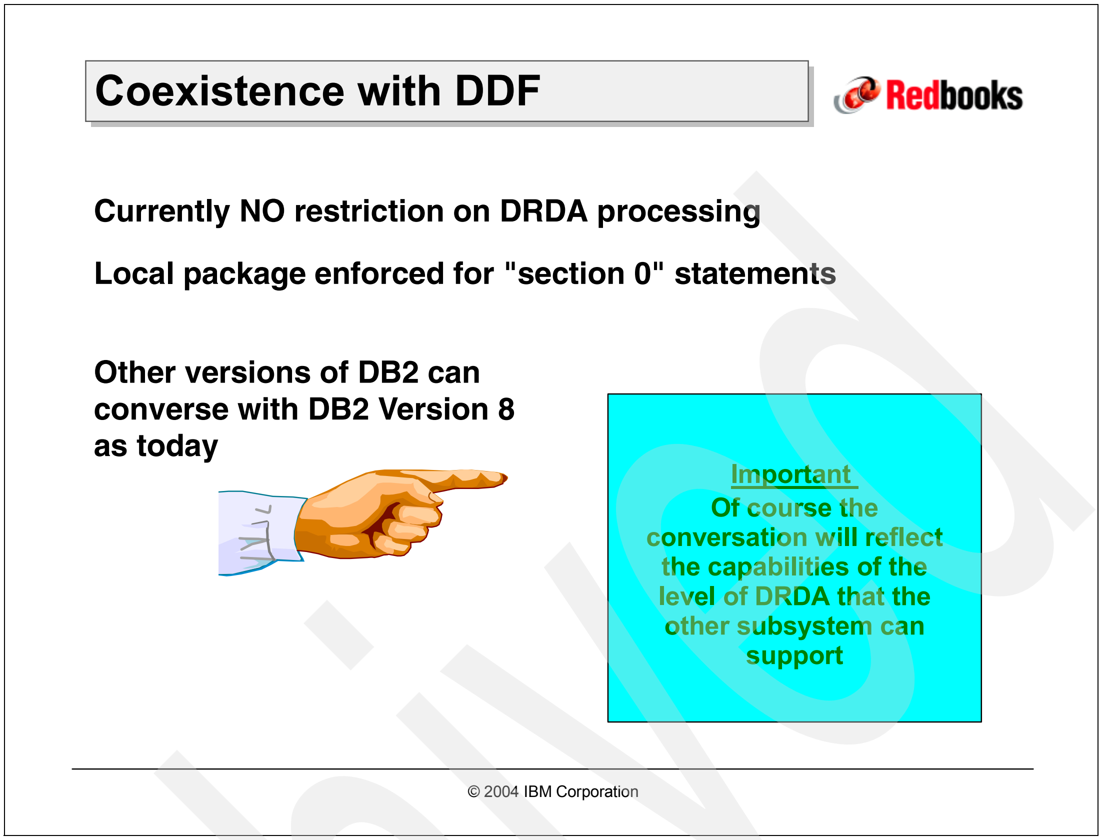
Recommendation for group restart

If a group restart is necessary while the data sharing group is running with mixed releases, issue the START command only for Version 8 members. Do not start the Version 7 members until the Version 8 members have completed forward log recovery. If a Version 7 member performs the group restart for a Version 8 member, Version 7 adds pages to the logical page list during the peer-forward recovery phase when it tries to apply redo log records against a release-dependent object.

Recommendation for SPUFI

When you migrate the first member of the data sharing group to Version 8, you run DSNTIJSG which rebinds SPUFI in Version 8. Binding SPUFI in Version 8 causes SPUFI to be unavailable to the Version 7 members. If you attempt to run an SQL statement in a data sharing member that is yet to be migrated to Version 8, expect messages that indicate an unavailable resource.

An alternative is to modify job DSNTIJSG and defer the SPUFI BIND until all member have successfully ben migrated to Version 8.



12.7.3 Coexistence with DDF

DB2 Version 8 communicates in a distributed data environment with DB2 UDB for z/OS Version 6 and later, using either DB2 private protocol access or DRDA access. However, the distributed functions introduced in Version 8 can be used only when using DRDA access.

Other DRDA partners at DRDA Version 3 can also take advantage of the functions that are introduced in this release of DB2.

Package required for “section 0” statements

The PTF for APAR PQ59207 (V7) made some changes to the packages you need to have at the local site. When you use the following, so-called “section 0 SQL statements”, and you do not have a package or DBRM bound into the local plan, this PTF requires you to have a package or DBRM for these statements. The affected “section 0 SQL statements” are: CONNECT, COMMIT, ROLLBACK, DESCRIBE TABLE, RELEASE, SET CONNECTION, SET :HV = CURRENT SERVER, and VALUES CURRENT SERVER INTO :HV.

The SET CURRENT PACKAGSET, SET :HV = CURRENT PACKAGESET and VALUES CURRENT PACKAGESET INTO :HV statements are not affected. By default, DB2 will enforce this new rule and requires you to have a package at the local site. However, specifying YES for the PKGLDTOL DSNZPARM (the default is NO) allows you to continue to have the same behavior as before the introduction of this PTF. This restriction is enforced in the DB2 V8 base. The PKGLDTOL DSNZPARM has been removed.

Also note that two-phase commit is no longer supported when you use DB2 Connect Version 8 to access a DB2 UDB for z/OS (any supported version) using SNA. When coming from DB2 Connect V8, in order to be able to use two-phase commit, you must use TCP/IP.

Enabling-New-Function Mode

Redbooks

The process by which the catalog is converted from V8 Compatibility Mode to V8 New Function Mode

- This is **NOT** optional. Everyone must run the Enabling New Function Mode before migrating to a later release

During ENFM processing

- Change types and lengths of existing catalog columns
- Converts the catalog data from EBCDIC to Unicode
  - Using online REORG
- Change buffer pools for some catalog table spaces
- Change data capture (CDC) is disabled on every catalog table that has it enabled
  - Must manually re-enable CDC
    - ALTER TABLE SYSIBM.SYSTABLES DATA CAPTURE CHANGES

12.8 Enabling-new-function mode

Enabling-new-function mode (ENFM) converts a Version 8 compatibility mode (CM) catalog to a Version 8 new-function mode (NFM) catalog.

Conversion to a DB2 Version 8 new-function mode catalog is only allowed after a successful migration has been completed to DB2 Version 8 compatibility mode. (DB2 checks the level of catalog and DB2 code before it will successfully enter enabling-new-function mode.)

The following tasks are performed in the enabling-new-function mode:

1. The catalog is flagged as being in enabling-new-function mode.
2. The types and/or lengths of existing catalog columns are changed.

   Many columns in many tables will change data type and/or length to support long names. Typically, character identifier columns change to varchar columns 3 times the size (for example, CHAR(8) columns are converted to VARCHAR(24), and other columns will change to VARCHAR(128).

   When these columns are converted to overcharge, they will be converted with their maximum previous length. For example: CHAR(8) columns with “A” (containing 7 blanks) will be converted to VARCHAR(24) with a length of 8 containing “A “(with 7 blanks). DB2 does not know how many of these trailing spaces are valid. However, when these objects are created without trailing spaces under Version 8, DB2 will store them with their true length.
3. Columns that contain data that should not be converted are marked FOR BIT DATA. For example, this is the case with the CONTOKEN and STMT column in SYSPACKSTMT, and the TEXT column in SYSSTMT.

Tip: Remember that in V8, all statements are parsed in Unicode, and are normally stored in the catalog in Unicode (once you are in NFM). With the TEXT and STMT column marked “FOR BIT DATA”, they do not get translated to EBCDIC, when you are using SPUFI, for example. To be able to read the statement text information stored in the DB2 catalog, you can use Visual Explain. The tool has been enhanced (as all regular DB2 DM Tools) to support Version 8.

4. The catalog is converted to Unicode.

   The following catalog and directory table spaces are converted to Unicode via special Online Reorg processing:

   – Directory (DSNDB01):
     - SPT01
   – Catalog (DSNDB06):
     - SYSDBASE
     - SYSDBAUT
     - SYSDDF
     - SYSGPAUT
     - SYSGROUP
     - SYSGRTNS
     - SYSHIST
     - SYSJAVA
     - SYSOBJ
     - SYSPKAGE
     - SYSPLAN
     - SYSSEQ
     - SYSSEQ2
     - SYSSTATS
     - SYSSTR
     - SYSUSER
     - SYSVIEWS

   The following catalog and directory table spaces are not converted to Unicode and remain as EBCDIC. Many of these tables already contain binary data so there is no need to convert them to Unicode. They also contain many logical names that must interface with external MVS names (for example data set names), which remain EBCDIC.

   – Directory:
     - DBD01
     - SCT02
     - SYSLGRNX
     - SYSUTILX
   – Catalog:
     - SYSCOPY
     - SYSEBCDC

   During the online REORG process, the system-defined indexes on the catalog are converted to NOT PADDED.

   Please note that despite having Unicode and the z/OS Conversion Services active, DB2 still uses the table SYSIBM.SYSSTRINGS for character conversion support.

For character conversion, DB2 performs these steps:

a. First it looks in SYSIBM.SYSSTRINGS for the combination of source and target CCSIDs it needs for conversion.

b. Then it turns to the z/OS Conversion Services.

If nothing is found, then an error is returned.

5. Buffer pools for some catalog table space are changed.

   Increasing the length of some of the catalog table columns causes many catalog table rows to exceed the current BP0 4K page size maximum. In these cases, the table spaces that contain these tables are moved to an appropriately sized buffer pool. DB2 creates a BP8K0 buffer pool and a BP16K0 buffer pool during migration from Version 7 to Version 8 compatibility mode, if they don't already exist. Once in Version 8 compatibility mode, you cannot use the -ALTER BUFFERPOOL command to delete these buffer pools because they must exist before you use the REORG utility to convert the catalog tables to Unicode, in enabling-new-function mode.

   Table 12-2 shows the table spaces which move to new buffer pools.

Table 12-2 New catalog table space buffer pools

Table space name	Buffer pool	Page size
SPT01	BP8K0	8 K
SYSDBASE	BP8K0	8 K
SYSGRNTS	BP8K0	8 K
SYSHIST	BP8K0	8 K
SYSOBJ	BP8K0	8 K
SYSSTR	BP8K0	8 K
SYSSTATS	BP16K0	16 K
SYSVIEWS	BP8K0	8 K

   If you are running a data sharing system, it is your responsibility to define these new group buffer pools before you move into enabling new-function mode.

   In addition, if you specified "YES" in field 6 "VARY DS CONTROL INTERVAL"on install panel DSNTIP7 (during MIGRATE mode), the installation CLIST configures DSNTIJNE to use variable CI sizes for DB2 catalog and directory data sets that have a page size greater than 4 K.

6. Old catalog tables are dropped.

   The SYSIBM.SYSLINKS and SYSIBM.SYSPROCEDURES catalog tables are dropped from the catalog, as they are no longer used.

   The contents of SYSIBM.SYSPROCEDURE was moved to SYSIBM.SYSROUTINES when DB2 was migrated from Version 5 to Version 6 or 7. The table SYSIBM.SYSLINKS is no longer used in DB2 Version 8.

7. Catalog tables are moved.

   The SYSIBM.SYSDUMMY1 catalog table is being moved from the SYSSTR catalog table space to the new Version 8 SYSEBCDC catalog table space (which remains encoded in EBCDIC). This table space is created during migration by the CATMAINT utility.

SYSIBM.SYSDUMMY1 is probably already intrenched into many applications today. If it were to be converted to Unicode, all SQL statements accessing SYSIBM.SYDUMMY1 would become multiple CCSID statements. Results of multiple CCSID SQL statements can be different from single CCSID statements; for example, the ordering of result sets, or the use of range predicates, can affect the result set. This could cause unnecessary complications for existing applications.

During the enabling-new-function mode, views on the catalog tables in the table space being processed are regenerated. However, plans and packages dependent on the catalog tables in the table space being processed will be invalidated. DB2 will attempt to auto rebind these invalidated plans and/or packages the next time they are needed.

Once a table space is converted to Unicode, there is no way to go back for that table space. All converting table spaces must be converted to Unicode before you can change from enabling-new-function mode to new-function mode.

This enabling-new-function mode processing phase can last for quite some time as the table space changes may be staged in over a number of “conversion” windows. However, we do not recommend staying in this mode for a long period of time.

ENFM - When Should I Start?

Redbooks

When you are sure you will not want to fallback to compatibility mode

- Your applications are stable in CM
- Version 8 is stable in CM

Enabling process CAN span maintenance windows

- You have the flexibility to plan the schedule
- Can be stopped after the REORG of any table space
- Can be restarted without modification
  - Skips already processed table spaces
  - Resumes processing at first table space not successfully converted
- Recommendation is **NOT** to stay in ENFM phase any longer than necessary

12.8.1 ENFM - When should I start?

We recommend that you plan to enable new-function mode in DB2 Version 8 only after you have had time to verify that your applications are stable running with the Version 8 code in compatibility mode and that the DB2 Version 8 code itself is stable in compatibility mode.

Allow yourself adequate time to complete this testing. Typically, you may choose to run in Version 8 compatibility mode for a month or two before you plan to move into ENFM.

Once you have decided to enter ENFM, there are a few things for you to consider when you schedule the work.

- The enabling process can span a number of windows. It all does not have to be performed at the same time. The enabling job which performs the conversion can be stopped and restarted at any time (more details will be provided later). It will skip table spaces it has already processed and resume processing from where it left off.
- Although DB2 can process other work while the conversion process takes place, we recommend that you schedule the conversion work at a quiet time. DB2 uses the ONLINE REORG Utility with SHRLEVEL REFERENCE to transform each table space. This means that DB2 only needs exclusive access to the catalog table spaces for only a very short period of time. However, applications may still be impacted by resource unavailable conditions while the conversion process is active.
- While the ENFM conversion process can take as long as you need, we recommend that you plan to complete ENFM sooner rather than later. Do not plan to stay in ENFM any longer than necessary. The sooner you move to new-function mode, the sooner you will be able to use the new function provided by DB2 Version 8.

A Checklist Before ENFM

Redbooks

Run online REORGs (in V8 CM) against the catalog to check:

- Timings
  - Elapsed times
  - Plan staged execution around outage windows

Review catalog data set sizings

- Eliminate space failures during ENFM
- Consider increasing the size of the catalog table spaces and indexspace to accommodate longer names
  - Review space for any user defined catalog indexes

For declared temporary tables

- Ensure at least one table space with page size of 8K or greater

For data sharing

- Ensure GBP8K0, GBP16K0, GBP32K is defined

12.8.2 A checklist before enabling-new-function mode

This visual highlights a few activities you should consider performing before you begin the enabling-new-function mode processing. Although they are not mandatory, we highly recommend that you perform them:

- Run online REORG against the DB2 Version 8 catalog in compatibility mode.

  This will provide you with some timings which should help you plan and schedule the ENFM jobs around available windows of low activity. It will help to verify the integrity of your DB2 catalog tables prior to conversion and also provide an opportunity to resize any table space, if you need to, before conversion.

- Review the size of your existing DB2 catalog data sets.

  During the ENFM, many catalog tables will increase in size to accommodate new column definitions for long names. We strongly advise you to have adequate space available for the expected growth in the catalog tables before you begin to convert them. This will help to avoid conversion failures due to space problems.

  The size of these catalog and directory table spaces will change as a result of the Unicode conversion. Many columns are increasing in length to support long names. Unicode is a variable length encoding scheme. Anywhere from 1 to 4 bytes are required to store some characters (for example; the character “¬” requires 2 bytes in UTF-8).

  During lab measurements on real customer catalogs, catalog tables grew by no more than 10%, but more typically between 1% and 5%. In fact, some catalog table spaces have decreased in size (merely due to the fact that they were reorganized, and dead space was reclaimed).

The size of any user-defined index on the DB2 catalog is likely to grow substantially during the conversion process. This is because many column lengths increase to 128 bytes, to support long names. Unlike the system-defined DB2 catalog indexes, which are converted from PADDED to NOT PADDED during the conversion process, any user-defined indexes on the catalog will remain PADDED, and therefore require more space.

Once you are in new-function mode, you can ALTER these user-defined indexes to NOT PADDED and rebuild them to reclaim the space. Alternatively, you may decide to DROP these user-defined indexes before you proceed with the ENFM process.

- Ensure that there is at least one table space with page size of 8K or greater in the TEMP database.

  DB2 Version 8 now requires at least one table space to be available in the TEMP database which has a page size of 8K or greater. This is to support declared temporary tables.

- Ensure that GBP8K0, GBP16K0, and GBP32K are defined.

  For data sharing, you will need to define a group buffer pool for GBP8K0, GBP16K0 and GBP32K. The DB2 catalog now uses BP0, BP8K0, BP16K0, for DB2 catalog objects.

  When using the installation CLIST to generate the ENFM jobs, the panels will force you to put in a value for a virtual 8K and 16K buffer pool.

ENFM in Data Sharing

Cannot start a V7 member once ENFM begins

- Regardless of whether or not ENFM process is running
- Now in ENFM phase

Cannot begin ENFM process if any active V7 subsystems in group

ENFM is group wide

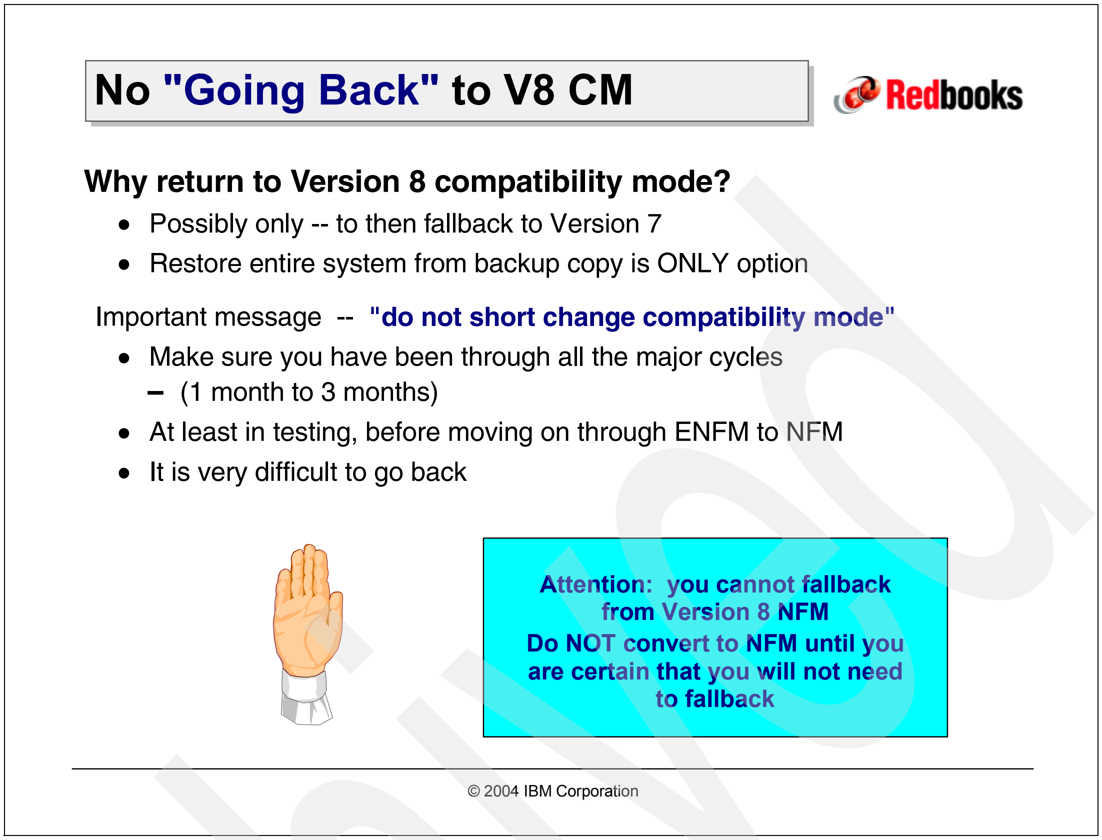
- Once ENFM begins, the **entire group** is in ENFM
- Only done once for the data sharing group

12.8.3 ENFM in data sharing

The important point to note here is that enabling-new-function mode is a data sharing group wide activity. You perform it only once for the whole data sharing group.

Enabling-new-function mode cannot begin until all data sharing members are at a Version 8 level of code, running in compatibility mode.

Once you have entered enabling-new-function mode, you cannot start a DB2 Version 7 member.



12.8.4 No “going back” to V8 CM

Fallback is defined as going back to a stable code base. By returning to compatibility mode we are not returning to a stable code base as we are running with the same code case (the same Version 8 code we are executing in enabling-new-function mode.). So, fallback to compatibility mode is rather meaningless. However, there may be some reasons why you may want to “return” to compatibility mode.

Remember that once the first step of job DSNTIJNE completes successfully, your DB2 subsystem is deemed to be in enabling-new-function mode and there is no returning to Version 8 compatibility mode. This is true for both data sharing and non-data sharing environments.

For this reason we recommend that you spend an appropriate amount of time in compatibility mode before moving to the next phase. Make sure you have performed adequate testing, the environment is stable and you are comfortable with the code before you plan to move to the next phase.

The restriction that you cannot go back to compatibility mode should not cause any problems because the need to return from ENFM for CM should be very rare. The only reason to return may be to subsequent fallback to Version 7.

If you absolutely need to return to Version 8 compatibility mode, then the only option available to you is to perform a point-in-time recovery of both the catalog and directory to a point before ENFM was entered. This implies a point-in-time recover of the entire DB2 subsystem or data sharing group! You can use whatever means you have available to perform this PIT recovery, online via DB2 utilities of offline via pack restores. However, remember that all updates that were done while in ENFM or NFM will be lost!

Enabling New Function Mode

Use DB2 installation CLIST to generate conversion jobs

Conversion done via the DSNTIJNE job

- First step enters enabling-new-function mode
- Several steps for each table space
- Uses **CATENFM** utility (not CATMAINT)

Conversion complete signaled by job DSNTIJNF

- To enter new-function-mode

Reassemble DSNHDECP

- Job DSNTIJNG

12.8.5 Enabling-new-function mode

To begin the conversion process into enabling-new-function mode, you first need to invoke the DB2 installation CLIST with the new ENFM option. The CLIST generates a number of new jobs which you need to run.

Job DSNTIJNE initiates and completes all of the necessary ENFM processing. The first step in this job executes the new CATENFM utility to bring DB2 from compatibility mode into enabling-new-function mode. Subsequent steps convert the DB2 catalog tables.

When this process is complete, the subsystem is ready to formally enable NFM via the DSNTIJNF job.This job once again invokes the CATENFM utility to bring DB2 into new-function mode.

The data only module DSNHDECP must then be re-assembled with the parameter NEWFUN=YES. This is required to tell the DB2 precompiler that DB2 is now in new-function mode and to accept the new SQL syntax. DB2 does not have to be cycled to bring this change into effect. A sample job DSNTIJNG is provided which can be used, or you can choose to assemble DSNHDECP using the existing job DSNTIJUZ.

Panel DSNTIPA1 - ENFM

```
      DB2 VERSION 8 INSTALL, UPDATE, MIGRATE, AND ENFM - MAIN PANEL
===>

Check parameters and reenter to change:

 1  INSTALL TYPE              ===> ENFM      Install, Update, or Migrate
                                             or ENFM (Enable New Function Mode)
 2  DATA SHARING              ===>           Yes or No (blank for Update or ENFM)

Enter the data set and member name for migration only. This is the name used
from a previous Installation/Migration from field 7 below:
 3  DATA SET(MEMBER) NAME ===>

Enter name of your input data sets (SDSNLOAD, SDSNMACS, SDSNSAMP, SDSNCLST):
 4  PREFIX                    ===> DSN810
 5  SUFFIX                    ===>

Enter to set or save panel values (by reading or writing the named members):
 6  INPUT MEMBER NAME         ===> DSNTID8A  Default parameter values
 7  OUTPUT MEMBER NAME        ===> DSNTID8N  Save new values entered on panels

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

DSNTIPA1: Install, update, and migrate DB2 - main panel

12.8.6 Panel DSNTIPA1 - ENFM

To begin preparation for enabling-new-function mode, you invoke the DB2 installation CLIST in the same way as you would for a new DB2 install, migrating a DB2 subsystem to a new version or enabling data sharing. The primary panel DSNTIPA1 is first displayed.

You specify ENFM as the primary option on this panel.

Leave the DATA SHARING and DATA SET NAME(MEMBER) fields blank. This is true for both data sharing and non-data sharing.

The INPUT MEMBER NAME specified for conversion should always be the same as the OUTPUT MEMBER NAME that you specified during migration to compatibility mode.

In a data sharing environment, the INPUT MEMBER NAME specified for conversion must be the same as the OUTPUT MEMBER NAME used when migrating the first member of the data sharing group to Version 8.

The DB2 installation CLIST displays a smaller set of panels for the primary option of" ENFM" than the "install" and "migrate" primary options. In fact, there are only a few panels in common. We will walk through some of the major panels in the next few visuals.

Briefly, the DSNTINST CLIST performs the following tasks:

- Displays the 18 catalog and directory table spaces which are transformed during ENFM on panel DSNPIT00, together with an estimate for the space allocations for the shadow data sets.
- Customizes a number of new installation jobs, together with the DB2 Version 8 sample jobs.

Panel DSNTIPT - Data set Names

```
              ENFM DB2 - DATA SET NAMES PANEL 1
===>

Data sets allocated by the installation CLIST for edited output:
  1  TEMP CLIST LIBRARY        ===> DSN810.NEW.SDSNTEMP
* 2  SAMPLE LIBRARY            ===> DSN810.NEW.SDSNSAMP
Data sets allocated by the installation jobs:
  3  CLIST LIBRARY             ===> DSN810.NEW.SDSNCLST
  4  APPLICATION DBRM          ===> DSN810.DBRMLIB.DATA
  5  APPLICATION LOAD          ===> DSN810.RUNLIB.LOAD
  6  DECLARATION LIBRARY       ===> DSN810.SRCLIB.DATA
Data sets allocated by SMP/E and other methods:
  7  LINK LIST LIBRARY         ===> DSN810.SDSNLINK
  8  LOAD LIBRARY              ===> DSN810.SDSNLOAD
  9  MACRO LIBRARY             ===> DSN810.SDSNMACS
 10  LOAD DISTRIBUTION         ===> DSN810.ADSNLOAD
 11  EXIT LIBRARY              ===> DSN810.SDSNEXIT
 12  DBRM LIBRARY              ===> DSN810.SDSNDBRM
 13  IRLM LOAD LIBRARY         ===> DSN810.SDXRRESL
 14  IVP DATA LIBRARY          ===> DSN810.SDSNIVPD
 15  INCLUDE LIBRARY           ===> DSN810.SDSNC.H

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

DSNTIPT : Data set names panel 1

12.8.7 Panel DSNTIPT - data set names

Panel DSNTIPT is displayed after the primary panel DSNTIPA1.

The SAMPLE LIBRARY field (option 2), is the only updateable field on this panel. Here is where you specify a destination data set for the ENFM customized jobs.

You can choose to use the same data set name as you used during migration. In this case, the data set is not deleted or re-allocated. The CLIST merely compresses the data set, then updates the data set with the new jobs required for enabling-new-function processing, together with the Version 8 IVP suite of jobs. Alternatively, you can specify a new data set. In this case, the CLIST creates the new data set, then generates the jobs into that data set.

Panel DSNTIP00 - Shadow Data Sets

ENABLE NEW FUNCTION MODE FOR DB2 - SHADOW DATA SET ALLOCATION

===>

	OBJECT	DASD DEVICE	VOL/SERIAL	PRIMARY RECS	SECONDARY RECS
1	SPT01	==> 3390	==> SBOXCC	==> 636	==> 636
2	SYSDBASE	==> 3390	==> SBOXCC	==> 7049	==> 7049
3	SYSDBAUT	==> 3390	==> SBOXCC	==> 478	==> 478
4	SYSDDF	==> 3390	==> SBOXCC	==> 144	==> 144
5	SYSGPAUT	==> 3390	==> SBOXCC	==> 3060	==> 3060
6	SYSGROUP	==> 3390	==> SBOXCC	==> 48	==> 48
7	SYSGRTNS	==> 3390	==> SBOXCC	==> 144	==> 144
8	SYSHIST	==> 3390	==> SBOXCC	==> 144	==> 144
9	SYSJAVA	==> 3390	==> SBOXCC	==> 144	==> 144
10	SYSOBJ	==> 3390	==> SBOXCC	==> 616	==> 616
11	SYSPKAGE	==> 3390	==> SBOXCC	==> 9673	==> 9673
12	SYSPLAN	==> 3390	==> SBOXCC	==> 13373	==> 13373
13	SYSSEQ	==> 3390	==> SBOXCC	==> 144	==> 144
14	SYSSEQ2	==> 3390	==> SBOXCC	==> 144	==> 144
15	SYSSTATS	==> 3390	==> SBOXCC	==> 53355	==> 53355
16	SYSSTR	==> 3390	==> SBOXCC	==> 661	==> 661
17	SYSUSER	==> 3390	==> SBOXCC	==> 1675	==> 1675
18	SYSVIEWS	==> 3390	==> SBOXCC	==> 7093	==> 7093
19	INDEXES	==> 3390	==> SBOXCC Catalog and directory index shadows		

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

DSNTIP00: Enable new function mode for DB2 - shadow data set allocations

12.8.8 Panel DSNTIP00 - shadow data sets

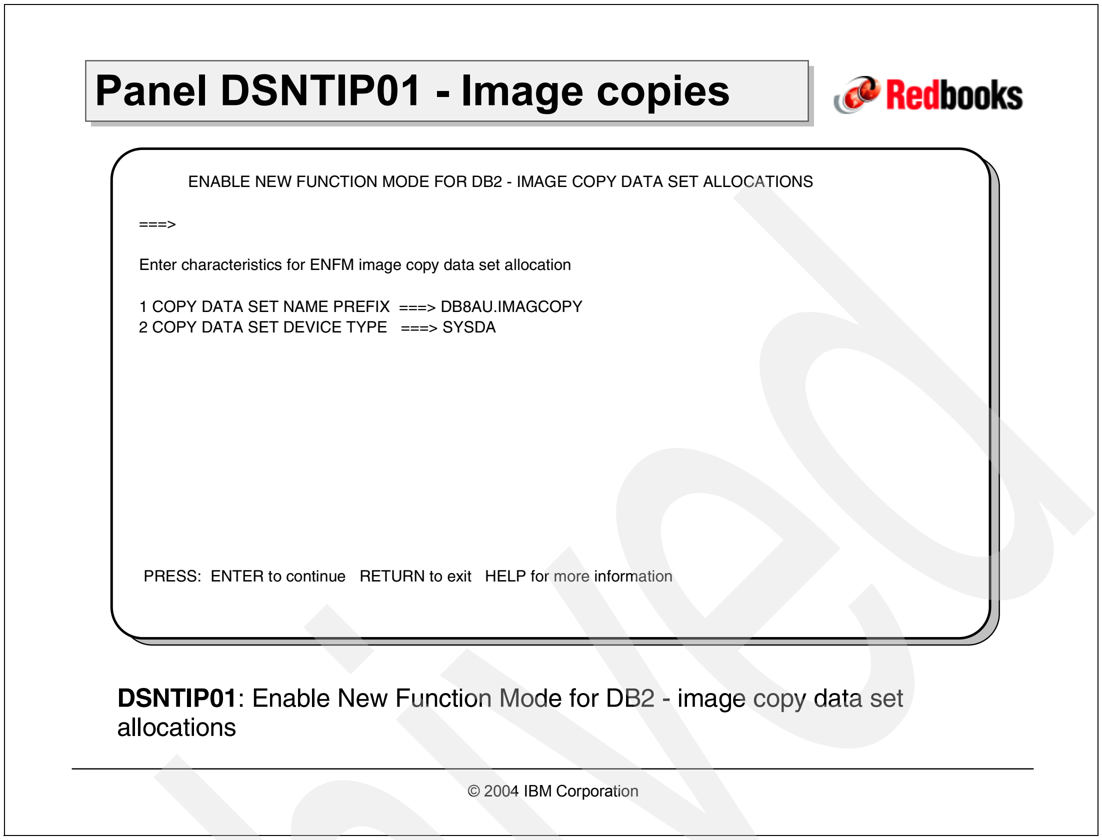
Panel DSNTIP00 is displayed after panel DSNTIPT.

The shadow data sets are used by the online REORG utility, to convert the table spaces to the ENFM format. Since the table spaces of the DB2 catalog are user-defined table spaces, these data sets must be defined before the REORG utility is run.

The conversion job DSNTIJNE will allocate these shadow data sets based on the parameters provided by this panel. The values are based on the fields PERMANENT UNIT NAME and VOLUME SERIAL from panel DSNTIPA2, and the fields PRIMARY RECS and SECONDARY RECS calculated by the DSNTCALC REXX, based on information entered during migration to compatibility mode. The PRIMARY RECS and SECONDARY RECS fields show the number of records (VSAM CIs), that will be allocated. These values, along with the device type and volser(s) can be overridden in this panel.

When you run DSNTIJNE, the shadow data sets replace the current data sets for the table and index spaces being converted.

We recommend that you review these data set allocations and space estimates carefully before moving on.



12.8.9 Panel DSNTIP01 - image copies

Panel DSNTIP01 is displayed after panel DSNTIP00

When you reorganize a DB2 table space using either SHRLEVEL REFERENCE or CHANGE, it is mandatory that you take an image copy as part of the reorganization. This is the same as when you use the REORG utility in enabling-new-function mode to convert the catalog. An inline image copy must be taken.

This panel allows you to enter the image copy data set names prefix hat you want to use for the inline image copies, as well as which output device you like for the data sets.

If TAPE, 3480, or 3490 is specified for DEVICE TYPE, then no SPACE parameter is included in the generated job for each image copy data set. Otherwise, the same space settings are used as specified for the table space's shadow data set list.

Even though tape devices are supported, the stacking of image copy data sets on the same tape is not supported.

Panel DSNTIP02 - Storage

Redbooks

```
DSNTIP02      ENABLE NEW FUNCTION MODE FOR DB2 - STORAGE REQUIREMENTS
===>
 1  DSNT488I  VOLUME SBOXCC WILL REQUIRE AT LEAST 182192 4K BLOCKS



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

DSNTIP02: Enable New Function Mode for DB2 - storage requirements

12.8.10 Panel DSNTIP02 - storage

Panel DSNTIP02 is displayed after panel DSNTIP01. It displays a summary of the storage requirements based on the data entered on the previous panels.

To accept the space requirements that are displayed for each volume, press the ENTER key. Alternatively you can press PF3 to return to previous panels and review your values.

Pressing the ENTER key also signals the CLIST to generate the jobs required for the enabling-new-function mode. The following jobs are generated:

- **DSNTIJNE**: Enabling-new-function mode processing:

  This new job uses the online REORG utility to convert the catalog and directory table spaces to the new Unicode format. It can be stopped with job DSNTIJNH and it will restart from the next table space to be converted.

- **DSNTIJNH**: Halt DSNTIJNE:

  This new job stops the execution of DSNTIJNE at the end of the active group.

- **DSNTIJNF**: Turn new-function mode on:

  This new job flags DB2 as in new-function mode.

- **DSNTIJNG**: Update DSNHDECP for new-function mode:

  This new job updates DSNHDECP with NEWFUN=YES in SDSNEXIT.

- **DSNTIJEN**: Return to enabling-new-function mode status:

  This new job returns from new-function mode to enabling-new-function mode status.

- **DSNTIJNR**: Convert the DSNRLST table for long name support:

  This new job ALTERs the columns of the DSNRLST table to support long names.

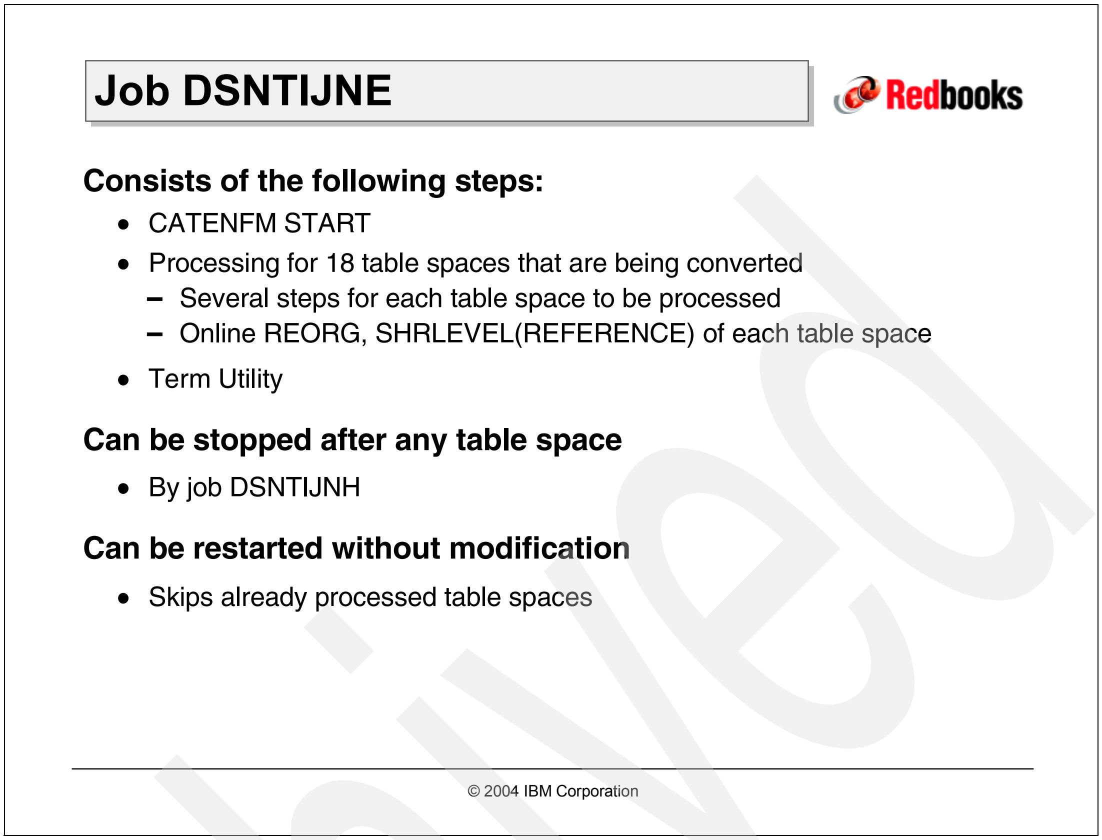
- **DSNTIJMC**: Needed if you use the ODBC/JDBC metadata methods:

  This new job switches the stored procedures used by these methods from compatibility mode to new-function mode.

- **Version 8 IVP:** The new suite of Version 8 IVP jobs:

  This is the new suite of Version 8 IVP jobs.

We shall now take a closer look at these jobs in the next few visuals.



12.8.11 Job DSNTIJNE

Job DSNTIJNE is new for DB2 Version 8. It is generated by the DB2 installation CLIST when you specify ENFM on the primary panel.

The job can only be executed by a user with install SYSADM authority. It has three main phases:

1. The first step executes the CATENFM utility (CATENFM START) to move the DB2 subsystem or data sharing group into enabling-new-function mode. CATENFM is a new utility in DB2 Version 8.
2. Then, 18 catalog and directory table spaces are converted to Unicode via the online REORG Utility using SHRLEVEL(REFERENCE). Each table space is processed in turn and a number of steps are required to process each table space (details provided later).
3. Any outstanding utilities are terminated. Actually, the first step of the DSNTIJNE job also terminates all outstanding utilities related to ENFM processing (DSNENFM.* utility IDs) before starting.

Job DSNTIJNE can be stopped at any time using the job DSNTIJNH. More on this in later visuals. Job DSNTIJNE can be restarted afterwards, without any modification to the JCL and it will skip any already processed table spaces.

What happens if you have a space problem during the ENFM process?

If you have a space problem, the following actions are taken:

- All the succeeding steps are skipped.
- A -TERM UTIL command is issued to make the table space available.
- You can then change the space parameters and re-submit the job.
- The next job execution will skip any already processed table spaces and resume processing at the first table space that has not been successfully converted.

DSNTIJNE - CATENFM START

Only done once

- Makes sure no V7 subsystems are up in a data sharing group
- Makes sure catalog is at correct level
- Makes sure user is authorized to convert
- Marks ENFM mode in BSDS / SCA and directory header page
- SYSLINKS and SYSPROCEDURES catalog tables are dropped
- SYSDUMMY1 table moved from SYSSTR table space to SYSEBCDC table space which remains EBCDIC

12.8.12 DSNTIJNE - CATENFM START

CATENFM is a new utility in DB2 Version 8. It is used to control and manage the conversion processes into new-function mode.

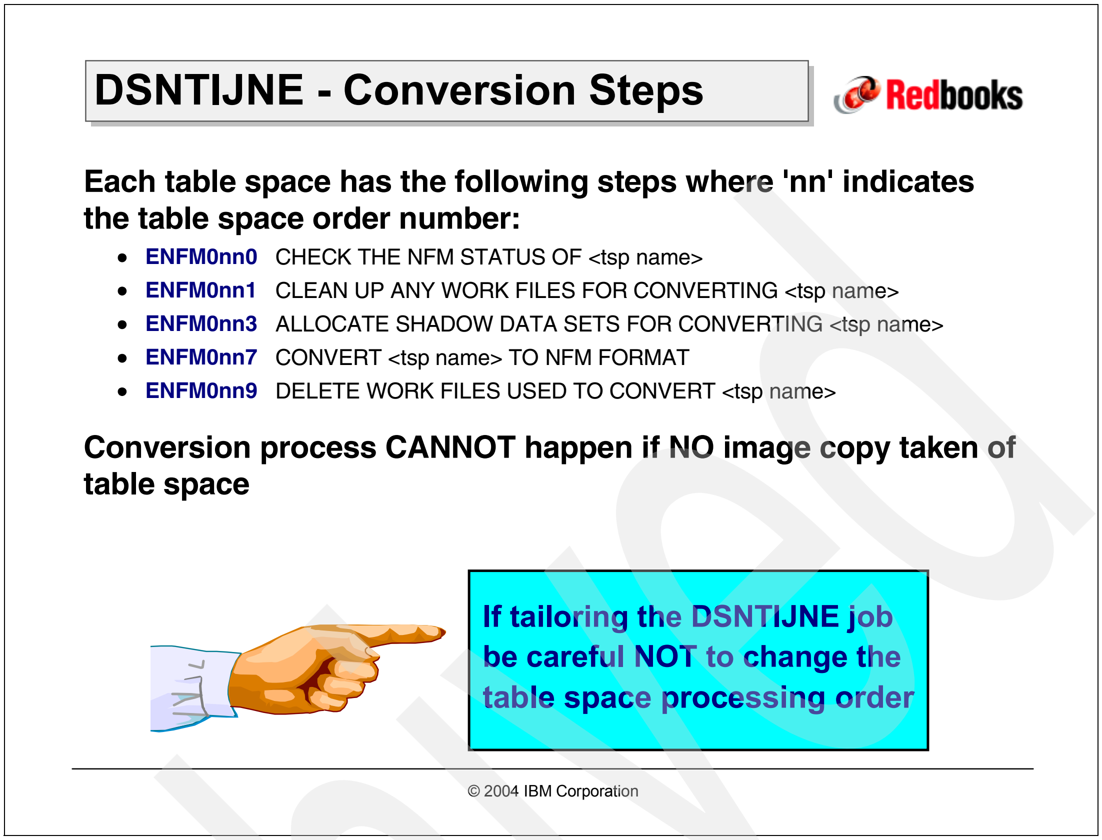
CATENFM START is the first “real” step in job DSNTIJNE. (The actual first step terminates outstanding utilities related to ENFM processing from previous runs).It first preforms a number of consistency checks:

- The utility first checks it you are authorized to run the utility. Only install SYSADMs can execute the CATENFM Utility.
- The utility checks the DB2 code level to ensure DB2 is running in compatibility mode.
- If data sharing, the utility makes sure there are no Version 7 members active in the data sharing group.

If everything is OK, the CATENFM utility will then update the DB2 BSDS/SCA and directory header page, to indicate that DB2 has moved from compatibility mode into enabling-new-function mode. Once in ENFM, you cannot start any DB2 Version 7 system either in data sharing or non-data sharing.

The CATENFM utility also performs these tasks:

- It DROPs the old catalog tables SYSIBM.SYSLINKS and SYSIBM.SYSPROCEDURE which are no longer used.
- It moves the SYSIBM.SYSDUMMY1 table from DSNDB06.SYSSTR to DSNDB06.SYSEBCDC, which is a new catalog table space created during the migration of DB2 from Version 7 to Version 8. SYSEBCDC remains an EBCDIC table space.



12.8.13 DSNTIJNE - conversion steps

Job DSNTIJNE converts 17 catalog and one directory table space from EBCDIC to Unicode.

The table spaces DSNDB01.DBD01, DSNDB01.SCT02, DSNDB01.SYSLGRNX, DSNDB01.SYSUTILX and DSNDB06.SYSCOPY remain in EBCDIC. Table space DSNDB06.SYSEBCDC is a new table space in Version 8 which also remains in EBCDIC.

Conversion cannot happen unless the table space to be converted has been successfully image copied. This is a normal requirement on the REORG utility when reorganizing catalog table spaces.

Job DSNTIJNE performs five steps to convert each table space:

1. The CATENFM Utility is first used to check if the table space has already successfully been converted.
2. The next step cleans up any work files left after a previous execution.
3. Next, shadow data sets are allocated for the REORG utility.
4. Here is where the conversion is performed.
5. Finally, the work files are cleaned up.

You need to update the job to include the IDCAMS allocation for any user-defined indexes on the catalog table space. You only need to include data sets for user-managed data sets, however, we recommend that you review the PRIQTY and SECQTY parameters in the catalog for any DB2-managed indexes, as they will probably need to be increased.

The order in which the table spaces are converted is very important. If you are tailoring job DSNTIJNE be very careful not to change the order of the table spaces. RI relationships exist between catalog tables that need to be maintained during the conversion process. Changing the order may cause some table spaces to fall into CHECK PENDING and therefore impact availability.

Table 12-3 Table space conversion order

Order	Table space	Order	Table space
1	SYSVIEWS	10	SYSOBJ
2	SYSDBASE	11	SYSPKAGE
3	SYSDBAUT	12	SYSPLAN
4	SYSDDF	13	SYSSEQ
5	SYSGPAUT	14	SYSSEQ2
6	SYSGROUP	15	SYSSTATS
7	SYSGRTNS	16	SYSSTR
8	SYSHIST	17	SYSUSER
9	SYSJAVA (1)	18	SPT01
(1) This includes the LOB table spaces associated with the LOB columns related to the tables in this table space.			

If the job fails in any of the five steps required to convert a table space, you do not need to do anything to restore availability to the catalog table space. The last step of the job will terminate any outstanding conversion REORG utility, restoring availability to the catalog table space. All you need to do is fix the error and re-submit the job again when you are ready. Typically the job fails because of space problems with the IDCAMS DELETE/DEFINE statements.

DSNTIJNE - ENFM0nn0 Steps

ENFM0nn0 CHECK THE NFM STATUS OF <tsp name>

Can only be done if all the previous table spaces were successfully converted

Execute SQL to add long name support to columns in tables

- Change data type and / or maximum length of these columns

System-defined indexes with keys with VARCHAR columns (includes new ENFM generated indexes)

- Changed to NOT PADDED indexes
- Placed in advisory REORG-pending state (AREO*) if not already from other process

12.8.14 DSNTIJNE - ENFM0nn0 steps

For DSNTIJNE, these are the steps:

ENFM0nn0 CHECK THE NFM STATUS OF <tsp name>

This is the first step of each JCL block in DSNTIJNE, which will convert an individual catalog and directory table space to the DB2 Version 8 new-function mode format. This step will only execute if all previous table spaces were successfully converted.

This step uses the new Version 8 CATENFM utility to:

- Check the status and availability of the table space to be converted.
- Update the catalog to add long names support to the columns in the catalog tables. This involves changing the data type and/or column lengths for these catalog columns.
- All system-defined indexes for tables in this table space which contain VARCHAR columns, including the new VARCHAR columns, will be changed to Advisory Reorg Pending (AREO*) status, in preparation for being converted from PADDED to NOT PADDED during the REORG step. While the indexes are in this pending state, access to the catalog is not impacted and you can still execute DDL. Any user-defined indexes on the catalog tables are not affected and they remain PADDED indexes after the conversion is complete.

Advisory Reorg Pending (AREO*) is a new database exception status introduced in DB2 Version 8. It indicates that the table space, index, or partition identified should be reorganized for optimal performance. Access to the data is not restricted.

DSNTIJNE - ENFM0nn7 Steps

Only attempted if:

- Previous table spaces have been successfully processed
- Long name DDL for this table space completed successfully

Converted to Unicode using online REORG (SHRLEVEL REFERENCE)

- No special locking considerations
- 'FOR BIT DATA' columns NOT converted to Unicode
- If this breaks the old catalog data is still there, so no outage
- Once a table space is Unicode, there is NO going back -- Only **RESTORE** option
- Buffer pools / page sizes changed during the process
  - Need to allocate BP8K0 and BP16K0 !!

12.8.15 DSNTIJNE - ENFM0nn7 steps

For DSNTIJNE, these are the steps:

ENFM0nn7 CONVERT <tsp name> TO NFM FORMAT

This step uses the online REORG utility with SHRLEVEL(REFERENCE) to convert the individual catalog and directory table space to Unicode. This step will only execute if all previous table spaces were successfully converted and the CATENFM utility has successfully updated the catalog with the new long name definitions for table columns in this table space.

There is no special locking that is used by the REORG utility. So, even though online REORG is being used, there still remains the possibility that other DB2 work may be impacted by delays and “resource unavailable” conditions while this step is running. We therefore suggest you schedule these conversions when there is little or no activity on the system.

As online REORG Utility is loading the data into a shadow copy of the table space and indexes, there should be no significant outage in the event of a failure. However, once the table space has been successfully converted to Unicode, there is no easy fallback. The only was it to restore the entire DB2 subsystem to a prior point-in-time.

Job DSNTIJNE can be used to convert user-defined indexes on the catalog. However, the AMS DELETE/DEFINE statements have to be manually added to the job. The job also does not alter the user-defined indexes to NOT PADDED.

Attention: If any user-defined indexes are defined on the DB2 catalog and the index columns that increase in size to support long names, which is likely to be the case, these indexes may grow dramatically in size (because user-defined indexes are not changed to NOT PADDED by the ENFM processing.

We therefore recommend that you review the space used by these indexes carefully and substantially increase their space allocation before you run jon DSNTIJNE, or drop the indexes before ENFM processing, and recreate them (as NOT PADDED) afterwards.

Once you are in new-function mode, these user-defined indexes can be altered from PADDED to NOT PADDED, which will place them in rebuild pending. They can then be rebuilt to get them back to a reasonable size.

Alternatively, you may decide to drop any user-defined indexes on the catalog tables before job DSNTIJNE is run, then re-evaluate if you need to recreate them again after you are in new-function mode.

For those table spaces that require larger than 4K buffers after conversion to Unicode, DB2 must load the data into the shadow tables using these new buffer pools. This is why BP8K0 and BP16K0 are allocated during the migration process by DB2. You may need to adjust the size.

Important: The REORG utility unloads the index data in PADDED format. So make sure the SYSUT1, SORTOUT and SORTWORK DD cards have plenty of space in job DSNTIJNE.

DSNTIJNE - ENFM0nn7 Switch

Switch phase processing -- has a special interface for CATENFM processing:

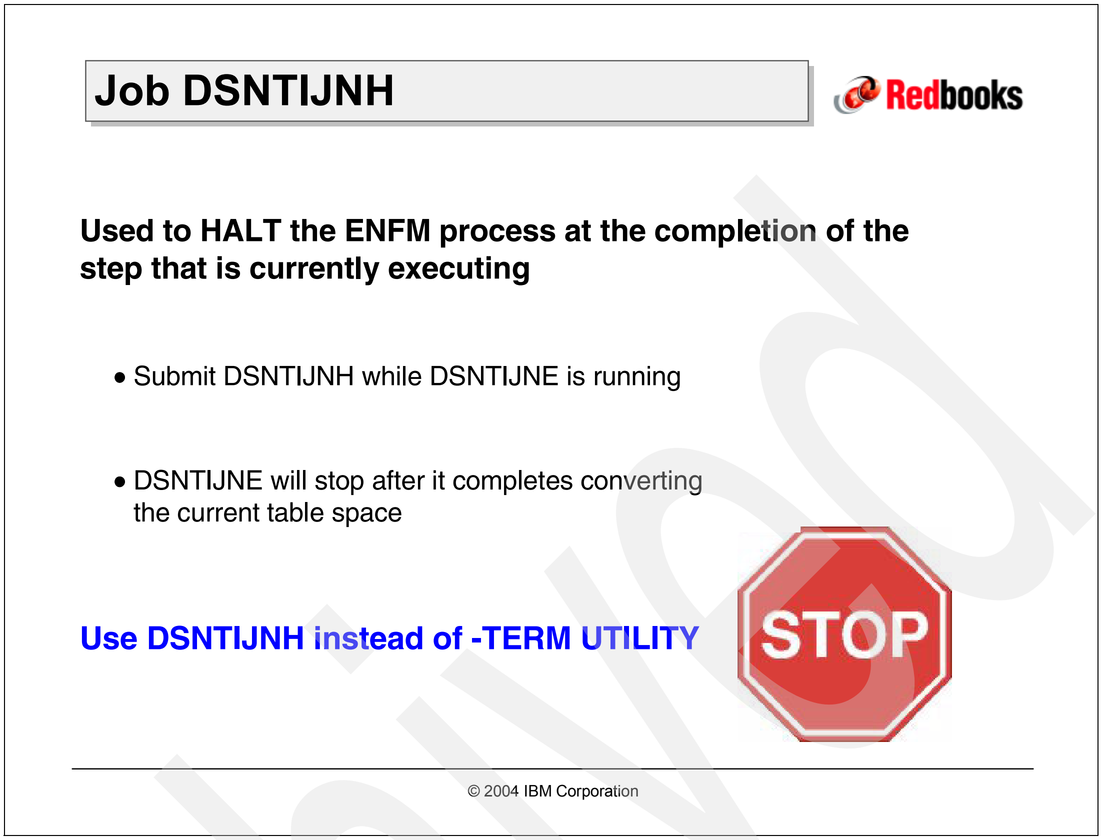
- Update catalog and directory:
  - New page size values
  - New encoding scheme and CCSID information
- View regeneration
- Table check constraint regeneration
- Plan and package invalidation
- Dynamic cached statement invalidation

12.8.16 DSNTIJNE - ENFM0nn7 switch

During conversion of the catalog and directory table spaces to NFM, online REORG needs to perform some “special” processing during the “switch” phase:

- The DB2 catalog and directory needs to be updated to reflect the table space is now Unicode. The DB2 catalog and directory also needs to be updated to reflect the new buffer pool assignments for those table spaces which will no longer be using BP0.
- All the views on the catalog tables that have been converted are re-generated.

  If a dependent view cannot be successfully regenerated, it is marked in the catalog to indicate the error (the STATUS column in SYSIBM.SYSTABLES has the value “R” and the TABLESTATUS column is “V”). Please note that this behavior is different from situations in which dependent views are regenerated as a result of an ALTER TABLE statement, when data types have changed and the dependent views cannot successfully be regenerated. In that case, the ALTER TABLE statement fails. (So “flagging” a view during view re-generations normally only occurs during the migration process.)
- All the table check constraints on the catalog tables that have been converted are also re-generated.
- All the plan and packages that are dependent upon the catalog tables that have been converted are marked as invalid. They will be rebound on the next allocation.
- All the statements in the dynamic statement cache that are dependent on the catalog tables that have been converted are also marked as invalid.



12.8.17 Job DSNTIJNH

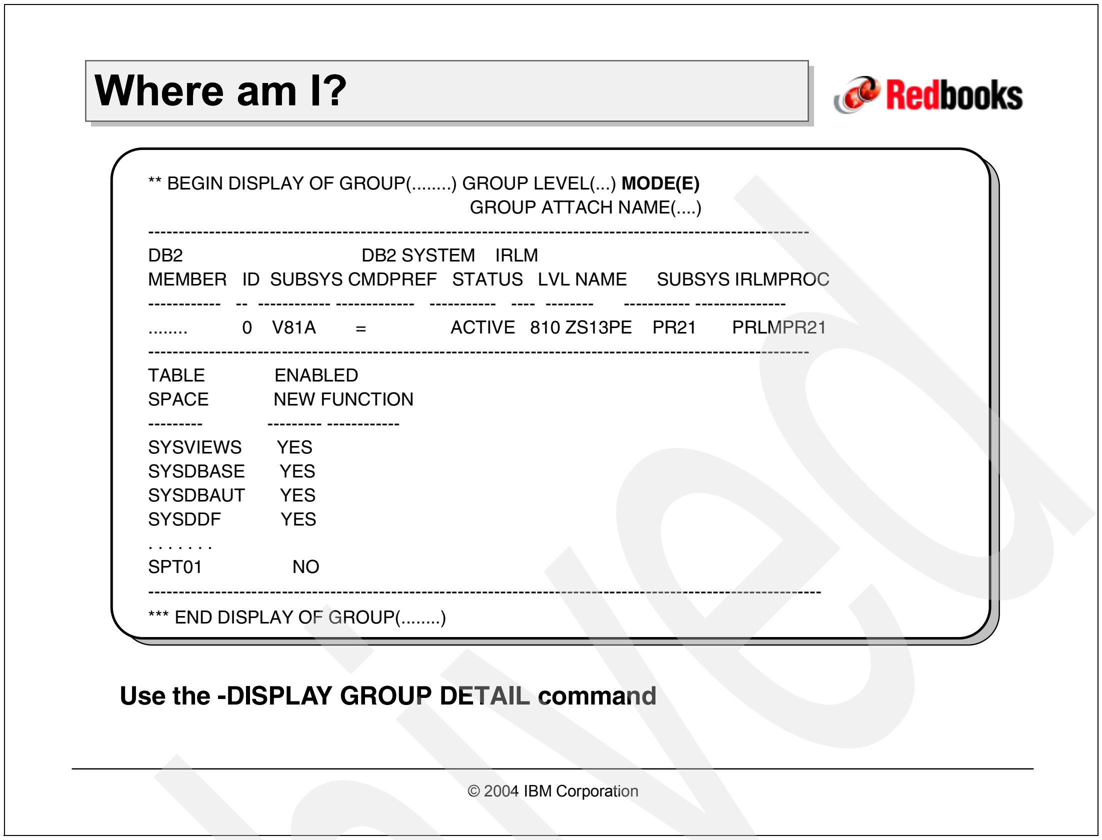
Job DSNTIJNH is new for DB2 Version 8. It is generated by the DB2 installation CLIST when you specify ENFM on the primary panel. The job is used to halt the ENFM process at the completion of the step that is currently executing. It can only be executed by a user with install SYSADM authority.

You submit job DSNTIJNH while job DSNTIJNE is currently executing. Job DSNTIJNH uses the new CATENFM utility (CATENFM HALTENFM) to signal job DSNTIJNE to terminate when it completes all the steps associated with converting the current table space to the new format. Job DSNTIJNE will then terminate and not move on to the next table space. Job DSNTIJNE can be resubmitted at a later time and it will move on to convert the next table space.

In this way, you can terminate the ENFM conversion process if it is taking too long and you have exhausted your batch window. The ENFM conversion process can easily be resumed during another window.

We strongly recommend that you use job DSNTIJNH to stop an active DSNTIJNE job, rather than using any other process such as the -TERM UTILITY command.

All you need to do to restart job DSNTIJNE, after you have used job DSNTIJNH to halt its processing, is to simply re-submit the DSNTIJNE job. There is nothing else you need to do. Job DSNTIJNE will continue from where it left off, and will start converting the next table space in sequence.



12.8.18 Where am I?

The output from the DISPLAY GROUP DETAIL command is enhanced to show what mode the DB2 subsystem is in (top right of the output).

- **MODE(C):** DB2 is in Version 8 compatibility mode.
- **MODE(E):** DB2 is in enabling-new-function mode.
- **MODE(N):** DB2 is in new-function mode.

In addition, the DISPLAY GROUP DETAIL command also lists the conversion state of each catalog and directory table space while DB2 is in enabling-new-function mode.

Please note that the DISPLAY GROUP command can be used on non-data sharing systems, as well as data sharing systems. The sample on this visual is a non-data sharing system. Data sharing specific fields are not filled in.

New-Function Mode

Job DSNTIJNF

Ensures all 18 table spaces have been converted to Unicode and all the long name DDL updates were made plus DB2 defined catalog indexes not padded

Marks NFM mode in BSDS / SCA and directory header page

Your DB2 system is now in NFM!

12.8.19 New-function mode

Job DSNTIJNF is new for DB2 Version 8. It is generated by the DB2 installation CLIST when you specify ENFM on the primary panel. By providing the job DSNTIJNF to move into new-function mode, you are forced to make a conscious decision, by submitting the job, when you are ready to enter into new-function mode.

This can be useful when you want to move a number of DB2 subsystems into new-function mode at the same time, for example, all systems that are linked via DDF, and you want to use some of the new distributed functions in Version 8.

Job DSNTIJNF is a single step job which invoked the new CATENFM utility (CATENFM COMPLETE). It can only be executed by a user with install SYSADM authority.

This job must be run after a successful completion of job DSNTIJNE to formally bring the DB2 subsystem or data sharing group into Version 8 new-function mode. DB2 will only be brought into new-function mode if all of the enabling-new-function mode processing has completed successfully.

Before flagging DB2 as in new-function mode, the CATENFM Utility will:

- Check the catalog for long names support.
- Check that all the catalog indexes are available and defined as NOT PADDED.
- Check that all of the 18 table spaces have successfully been converted to Unicode.

After successful completion of job DSNTIJNF, this is what you can expect:

- You can now exploit the new DB2 Version 8 functionality.
- The 18 DB2 catalog and directory table spaces are now in Unicode.
- There is NO fallback or coexistence with DB2 Version 7.
- There is NO going back to Version 8 compatibility mode.
- YES — you can return to enabling-new-function mode. This is done by the new job DSNTIJEN, which is described in 12.9, “Returning to enabling-new-function mode” on page 979.

NFM Considerations

Redbooks

Schema evolution

- Varying length index keys may no longer be padded
- When using index based partitioning (as in V7)
  - Trying to create an index with the PARTITIONED keyword will now convert the table to table-based partitioning

Multiple CCSIDs per SQL statement

- SQL that accesses the DB2 catalog will be evaluated and sorted in Unicode
- The SQL results and order may differ from CM

RUNSTATS on the catalog

REBIND plans/packages and update DBDs

12.8.20 NFM considerations

Here are some considerations regarding new-function mode.

Schema evolution

After enabling-new-function mode, the following items may behave slightly differently than when DB2 was running in compatibility mode:

- Varying length index keys may no longer be padded.

  If the PADDED/NOT PADDED keywords are not specified on a CREATE INDEX statement, the default padding type used will depend on whether the Version 8 subsystem was migrated from Version 7 or if it was a new Version 8 install.

  For a new Version 8 Install, the default is NOT PADDED. DB2 always generates PADDED indexes while it is in compatibility mode or enabling-new-function mode. When DB2 is in new-function mode, it generates either PADDED or NOT PADDED indexes, depending on the PADIX system parameter in DSNZPARM.

  The PADDED/NOT PADDED keywords only apply to indexes on variable length column(s). Indexes with all fixed length column(s) are defined as PADDED by default.

- When you try to create an index with the new PARTITIONED keyword in NFM, the table will be converted from index-based to table-based partitioning. For example:
  - When using index-based partitioning, this will happen:
    - Create the table space specifying NUMPARTS.
    - Create the table without the PARTITIONING KEY keyword.
    - Create the partitioning index with the VALUES keyword. Trying to create an index with the PARTITIONED keyword will convert the table to table-based partitioning.

– When using table-based partitioning, this will happen:

- Create the table space specifying NUMPARTS.
- Create the table with the PARTITIONING KEY keyword.
- Trying to create an index with the VALUES keyword will fail.
- Any partitioned indexes are created using the PARTITIONED keyword.

Multiple CCSIDs per SQL statement

As each DB2 catalog table space is converted to Unicode, SQL that references tables in these converted table spaces may be impacted.

SQL that returns, compares, or orders by a DB2 catalog column that has been converted to Unicode will be evaluated in Unicode instead of EBCDIC. Depending on the specific SQL statement the result set and result set sequence may differ from compatibility mode when the DB2 catalog was EBCDIC. In addition, any SQL statement which joins a DB2 catalog table that has been converted to Unicode and any EBCDIC table (including a DB2 catalog table that has not yet been converted to Unicode) might also return different result sets.

For example, if an EBCDIC column is compared with a DB2 Unicode catalog column, the comparison will be done in Unicode. Since EBCDIC collating sequence is different from Unicode, the result set may be different then it was during compatibility mode when the DB2 catalog was EBCDIC.

Consider the following query:

```
SELECT NAME FROM SYSIBM.SYSTABLES
WHERE NAME < 'TO' AND NAME > 'TA';
```

In compatibility mode and DB2 Version 7, the predicate uses the EBCDIC collating sequence and returns table names that begin with 'T' and are followed by any letter. In Enabling new-function mode, after DB2 has converted the table space that contains SYSTABLES from EBCDIC to Unicode, both range predicates will be evaluated in Unicode. Since the UTF-8 collating sequence differs from EBCDIC, the query won't return any rows.(In EBCDIC, numbers are collated after letters. UTF-8 is collated like ASCII, where numbers are collated before letters.)

This incompatibility only potentially impacts SQL with range predicates (for example, >. >=. <. <=, between), and SQL which contains ORDER BY clauses. Equals type predicates are not affected.

Unicode parser

Various types of identifiers have various maximum lengths. Much of DB2's own processing of SQL statements uses Unicode UTF-8. So, the length of the Unicode representation of an identifier might differ from the length that you entered, if you entered the identifier in a CCSID other than Unicode. For example, the character “¬” requires one byte in EBCDIC but two bytes in UTF-8.

Therefore, an identifier that was valid in Version 7 might be flagged as too long in Version 8 new-function mode, and you will receive message DSNH107I (in a precompilation) or SQLCODE -107 (otherwise). This incompatibility can occur only if both of the following situations exist:

1. The identifier contains one or more characters whose Unicode representations require more bytes than their EBCDIC representations did.
2. This expansion causes the identifier to grow beyond the maximum allowed length.

The length of the EBCDIC representation of a column-name in an object that was created in Version 8 (before new-function mode) or before Version 8 must have been 18 bytes or less. Such a pre-existing object is allowed to exist in Version 8 new-function mode, even if the Unicode representation exceeds 30 bytes.

SQL statements can reference a column-name whose Unicode representation exceeds 30 bytes if they do not create new objects containing that column-name. However, if you drop the object that contains that column-name, you cannot recreate the object containing that column-name (in Version 8 new-function mode) if the Unicode representation exceeds 30 bytes.

Run RUNSTATS to re-create catalog statistics

Enabling-new-function mode invalidates catalog statistics. DB2 uses default statistics when it calculate access paths. Run RUNSTATS to gather the statistics again.

REBIND plans/packages and update DBDs

DB2 Version 8 in new-function mode, uses a different format for its DBDs, packages and plans. So, before DB2 can use a DBD, plan or package from an earlier release of DB2, it must first be expanded it the new Version 8 format. This is an overhead you can easily do away with.

This is also true for DB2 Version 8 running in compatibility mode and enabling-new-function mode. DB2 must first expand the DBDs, plans and packages before it can use them. DB2 must also convert the DBDs, plans and packages to the old format before it can store them in the catalog. This is an extra overhead that exists while running in compatibility mode and enabling-new-function mode.

> **Attention:** After you have entered new-function mode, we recommend that you plan to rebind all of your plans and packages. DB2 will then store the plans and packages in the DB2 catalog in the new format. DB2 will no longer need to expand the plans/packages each time it needs to use them.
>
> We also recommend that you plan to make some small change to every database. After the update, the DBD is written back to disk using the Version 8 format. DB2 will no longer need to expand the DBD the next time it needs to use it.

NFM and the Precompiler

Precompiler option NEWFUN defaults to NO -- will require changing

- NEWFUN(YES)
  - ➡ Accepts new V8 syntax
  - ➡ Unicode DBRM and Unicode bound statements in catalog
- NEWFUN(NO)
  - ➡ Rejects new V8 syntax
  - ➡ EBCDIC DBRM and bound statements in catalog

Run DSNTIJNG or DSNTIJUZ to assemble Version 8 DSNHDECP with new parameter

- NEWFUN(YES)

12.8.21 NFM and the precompiler

The data only module DSNHDECP provides, among other things, the default parameter settings for the DB2 precompiler. Version 8 brings a new parameter for DSNHDECP. The NEWFUN parameter tells the precompiler it can accept new Version 8 functionality during program preparation.

- **NEWFUN=YES**

  Allows programs to use the new functions provided by DB2 Version 8. In addition, it will result in DBRMs being generated in Unicode and the statements are stored in the DB2 catalog in Unicode.

  Unicode DBRMs will not be legible under a standard ISPF browser unless you use some tool to convert the Unicode to EBCDIC. Tools such as SPUFI and QMF will also not be able to display the SQL statements directly from the DB2 catalog correctly as well. You can use a tool like the DB2 Path Checker, DB2 Administration Tool, or Visual Explain, which will convert the SQL statements from Unicode to EBCDIC.

  Columns in the DB2 catalog that are defined as character are converted from Unicode to EBCDIC, before they are displayed. However, a number of columns in the DB2 catalog, including column CONTOKEN in SYSIBM.SYSPLAN and SYSIBM.SYSPACKAGE and column TEXT in SYSIBM.SYSSTMT and STMT in SYSIBM.SYSPACKSTMT, are now defined as FOR BIT DATA. These columns are not converted before they are displayed.

- **NEWFUN=NO**

  This is the default when you migrate using the installation CLIST. It prevents programs from using the new Version 8 functionality, by instructing the precompiler to reject Version 8 SQL. DBRMs will be generated in EBCDIC, and SQL will be stored in the DB2 catalog as EBCDIC, the same as prior versions of DB2.

Irrespective of whether NEWFUN is set to YES or NO, the Version 8 precompiler will parse SQL in Unicode.

Job DSNTIJNG is new for DB2 Version 8. It is generated by the DB2 installation CLIST when you specify ENFM on the primary panel. This job can be used to generate a new version of DSNHDECP into SDSNEXIT. Alternatively you can change DSNHDECP in job DSNTIJUZ to re-assemble a new version of DSNHDECP into SDSNEXIT.

Once you are in new-function mode, all the plans and packages you bind will be marked with the Version 8 dependency indicator, even though they may not contain any new Version 8 syntax. Similarly, any DBDs that are written to DSNDB01 will be marked as Version 8 dependant. This is because plans, packages and DBDs have a different format in Version 8 than prior releases of DB2. This will not cause any problems should you choose to return to enabling-new-function mode, since these objects are not frozen.

You can use the precompiler with NEWFUN=YES while in compatibility mode or in enabling-new-function mode, to test the stability of your applications with the Version 8 precompiler. The resulting DBRM's will be in Unicode. You can then bind the DBRMs and run the programs. However the bind will fail if the DBRMs contain any new-function SQL that is not supported in compatibility mode.

You can also execute the precompiler with NEWFUN=NO while running in new-function mode. In this case any Version 8 syntax will be rejected. This could be useful when your production system is still in V7 or V8 compatibility mode, and you want to make sure that the applications you create in your development system that is in V8 NFM, will be able to run on your production system that is still at V7 or V8 CM.

Finally, when you migrate from Version 7, the default for NEWFUN is NO, while any new install will use a default of YES.

Application Programming

Value of NEWFUN Are V8 functions used?	NO NO	YES NO	YES YES
Is DBRM a V8 new object and is DBRM V8 dependent?	NO	YES	YES
Can V7 or earlier bind?	YES	NO	NO
Can V8 bind in CM/ENFM?	YES	YES (Note 1)	NO
Can V8 bind in NFM?	YES	YES	YES
Can run in V7?	YES	n/a	n/a
Can run in V7 after fallback from V8 CM?	YES (auto rebind)	frozen	n/a
Can run in V8 CM?	YES	YES (Note 1)	NO
Can run in V8 CM after fallback from V8 ENFM/NFM? (Note 2)	n/a	n/a	n/a
Can run in V8 ENFM?	YES	YES (Note 1)	NO
Can run in V8 ENFM after returning from NFM?	YES	YES	YES
Can run in V8 NFM?	YES	YES	YES

12.8.22 Application programming

For any release of DB2, if a DBRM was built from a source program that uses syntax that was introduced in that release, the precompiler marks the DBRM as dependent on that release, and a BIND on an earlier release of DB2 will fail.

However, in DB2 Version 8 and the new format of DBRMs, this behavior is not as clear. An application program that does not use new syntax does not appear to use any Version 8 new features or to produce a Version 8 new object, but it actually does produce a Version 8 new object (if the precompilation uses NEWFUN YES).

This visual explains the relationship between characteristics of precompilation of application programs and the ability to bind and run in various DB2 releases and modes:

- The first column lists certain questions about this behavior.
- The second column answers the questions for a DBRM that is produced from a precompilation that uses the NEWFUN NO option for SQL processing. This option prevents the use of Version 8 new functions.
- The third column applies to a DBRM that is produced from a precompilation that uses the NEWFUN YES option, where the application program does not use any Version 8 new functions.
- The fourth column applies to a DBRM that is produced from a precompilation that uses the NEWFUN YES option, where the application program uses Version 8 new functions.

Why is this table being used?

- With NEWFUN YES, the SQL statements in the DBRM use Unicode, so the DBRM is a Version 8 new object, even if the application program does not use any Version 8 new functions. Therefore, the DBRM is Version 8 dependent, and Version 7 and earlier releases of DB2 cannot bind the DBRM. Version 8 can bind it, even before new-function mode. With NEWFUN NO, the SQL statements use EBCDIC, the DBRM is not a new object, and Version 7 and earlier releases can bind it.
- If the application program uses Version 8 new functions, DB2 Version 8 can bind the DBRM only in new-function mode. If the program does not use any new function, DB2 Version 8 can bind the DBRM even before new-function mode.

The current release marker (DBRMMRIC) in the header of a DBRM is marked according to the release of the precompiler, regardless of the value of NEWFUN. When the DBRM is bound, this value is stored in the column RELBOUND of catalog tables SYSIBM.SYSPLAN or SYSIBM.SYSPACKAGE. This triggers autobind when the plan/package is scheduled to execute on a lower DB2 version.

In a Version 8 precompilation, the DBRM dependency marker (DBRMPDRM) in the header of a DBRM is marked for Version 8 if the value of NEWFUN is YES, otherwise it is not marked for Version 8. This value is stored in column IBMREQD of catalog tables SYSIBM.SYSPLAN or SYSIBM.SYSPACKAGE and is used to determine if the plan or package is to be frozen on fallback.

Table notes:

Here are some notes on the table presented in this visual:

- **Note 1:** In general, plans and packages can be bound and executed, provided that they do not use any new function. However, there are a few exceptions. For example, these plans and packages can use Multi CCSID SQL while in ENFM.
- **Note 2:** This row is marked not applicable (N/A) because DB2 does not support returning from ENFM/NFM to CM. However, you can return DB2 to CM by performing a point-in-time recovery of the whole DB2 subsystem to a time when DB2 was in CM. In this case, DB2 will behave exactly the same as if it was first in CM.

RLST Migration

Resource Limit Facility

- The DB2 'governor' for dynamic SQL

Convert Resource Limit Facility for Long Names

- DSNRLST
- After you are in new function mode
- Sample job DSNTIJNR
  - RLF must be stopped

12.8.23 RLST migration

The Resource Limit Facility is the “governor” facility used by DB2 to control DB2 resources used by dynamic SQL. It can also be used to govern binding, and to disable different flavors of parallelism. It relies on the table DSNRLSTxx.

As many of the columns have changed in the DB2 catalog to support long names, the equivalent columns in DSNRLST need to also change.

DB2 Version 8 provides a sample job DSNTIJNR with the required ALTER statements to convert the DSNRLST table to support long names. This job needs to be run after DB2 is in new-function mode and while the RLF is stopped.

BSDS Migration

BSDS conversion is optional in Version 8

Increasing the maximum number of active log data sets and archive log volumes requires conversion of the BSDS

- Avoid the B37 ABEND

Run the new utility program DSNJCNVB to convert the BSDS(s) to larger size

- Conversion job must be run in New Function Mode
- Running DSNJCNVB in Compatibility mode or Enabling New Function mode
  - Results in program termination with message DSNJ439I and return code 777

DSNTIJIN RECORDS value increased

12.8.24 BSDS migration

Some customers are finding that the current maximum of 1,000 archive log volumes (per log copy) recorded in the BSDS is no longer sufficient to remain recoverable without having to take frequent image copies. In addition, customers have requested the ability to have more than the current maximum of 31 active log data sets per log copy. Active log read is much faster than archive log read, and there would be no queuing for archive log tape volumes during recovery or backout.

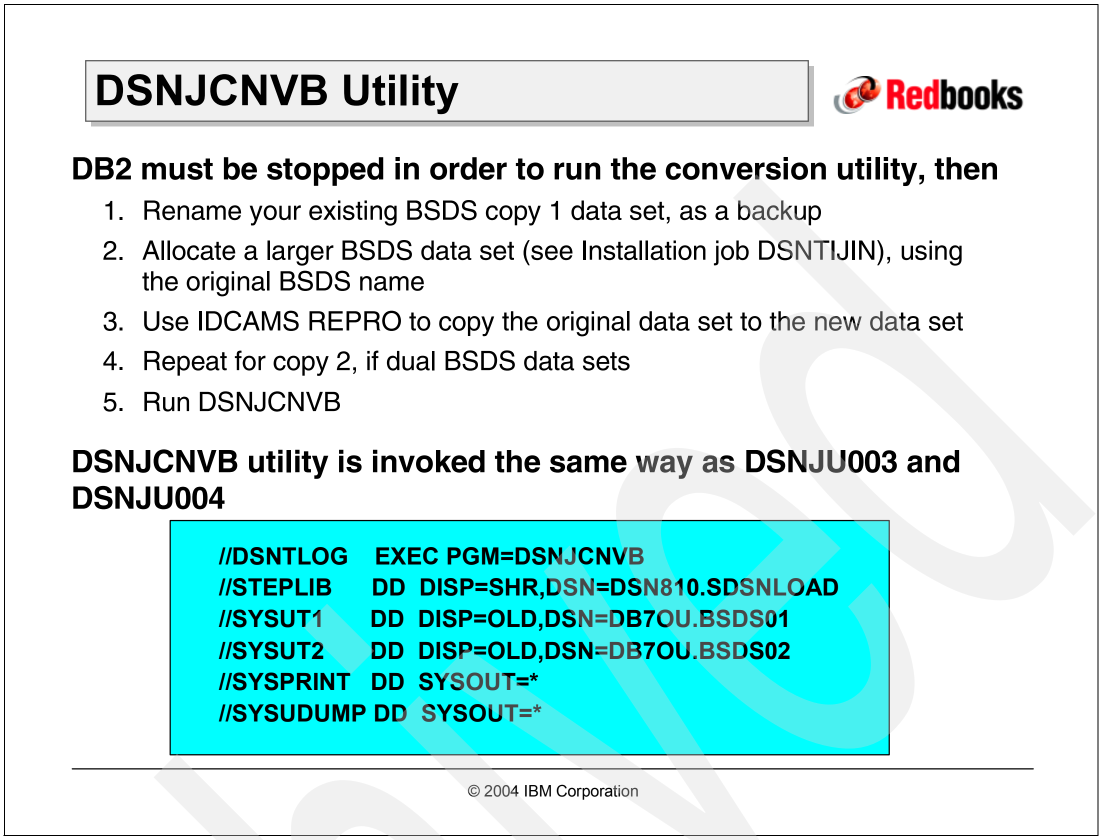
DB2 Version 8 increases the maximum number of archive log volumes recorded in the BSDS from 1,000 volumes per log copy to 10,000 volumes. Also, it increases the maximum number of active log data sets from 31 pairs of log data sets to 93 pairs of log data sets.

Increasing these limits requires a conversion of the BSDS data sets, to contain more formatted records. DB2 Version 8 provides a new utility, DSNJCNVB, to convert the BSDS data sets to the larger size.

The conversion can only be done while DB2 is in new-function mode (NFM). This is to minimize the fallback and data sharing coexistence impact. Running DSNJCNVB in compatibility mode or enabling-new-function mode results in program termination with message DSNJ439I and return code 777.

The BSDS conversion is optional. However, these enhancements are not available until the BSDS is converted to a new format.

For new DB2 Version 8 systems, DB2 provides a larger BSDS definition (space allocation) during installation. However, you must still manually convert the BSDS to the new format, by running the conversion utility, if you want to take advantage of these larger limits. The converted BSDS can be used to store more active and archive log data sets in the BSDS. For more information, see also 2.34, “More active log data sets” on page 68 and 2.35, “Increased maximum number of archive log data sets” on page 69.



12.8.25 DSNJCNVB utility

Prior to running the BSDS conversion utility, you should perform the following tasks:

1. Rename your existing BSDS copy 1 data set. You should retain the original copy of the BSDS in order to restore it in case of a failure during conversion.
2. Allocate a larger BSDS data set using the VSAM DEFINE statements in installation job DSNTIJIN, using the original BSDS name.
3. Use IDCAMS REPRO to copy the original data set to the new, larger data set.
4. Repeat for copy 2 if you are using dual BSDS data sets.
5. Run the DSNJCNVB utility.

   DSNJCNVB, the BSDS conversion utility, is invoked the same way as the DSNJU003 and DSNJU004 utilities:

```
//DSNTLOG     EXEC PGM=DSNJCNVB
//STEPLIB       DD   DISP=SHR,DSN=DSN810.SDSNLOAD
//SYSUT1        DD   DISP=OLD,DSN=DB70U.BSDS01
//SYSUT2        DD   DISP=OLD,DSN=DB70U.BSDS02
//SYSPRINT    DD   SYSOUT=*
```

Once the BSDSs have successfully been converted to the new format, you can re-assemble the DSNZPARM module to take advantage of the higher limits. However, if DB2 finds a value greater than 1000 and determines that conversion has not occurred, it issues the warning message DSNJ155I at startup, resets the maximum number of archive log to 1000 (MAXARCH in DSNZPARM), and continues to start.

Has BSDS Conversion Run?

```
DSNJCNVB CONVERSION PROGRAM HAS RUN   DDNAME=SYSUT1
 LOG MAP OF BSDS DATA SET COPY 2, DSN=DB7OU.BSDS01
 LTIME INDICATES LOCAL TIME, ALL OTHER TIMES ARE GMT.
    DATA SHARING MODE IS OFF
    SYSTEM TIMESTAMP     - DATE=2002.252  LTIME=22:15:43.72
    UTILITY TIMESTAMP    - DATE=2002.249  LTIME=18:23:22.80
    VSAM CATALOG NAME=DB7OU
    HIGHEST RBA WRITTEN      00000428E1F4  0000.000  00:00:00.0
    HIGHEST RBA OFFLOADED  0000041F3FFF
    RBA WHEN CONVERTED TO V4  000000000000
.........
```

Execute the DSNJU004 Utility

12.8.26 Has BSDS conversion run?

To determine if your BSDS data sets have been converted to support the larger number of active and archive log data sets, you can execute the print log map utility (DSNJU004). This will report if the DSNJCNVB utility has run, as shown in the visual above.

DSNWZP Considerations

Redbooks

A stored procedure to view DB2 subsystem parameter settings

- V7 defaulted to DB2-established SP
- V8 only WLM-established SP

On fallback to Version 7

- DSNWZP will not work!
- Need to alter the definition
  - ALTER PROCEDURE SYSPROC.DSNWZP EXTERNAL NAME DSNWZPR;

12.8.27 DSNWZP considerations

The stored procedure DSNWZP is used to view the DB2 subsystem parameter settings. It is used by a number of facilities including the DB2 Control Center and Visual Explain.

In prior releases of DB2, DSNWZP executed as a DB2 established stored procedure.

In Version 8, DSNWZP is defined to run in a WLM established stored procedure address space. The migration job DSNTIJSG will redefine DSNWZP as a WLM established stored procedure, and changes the definition to use the external module DSNWZP.

In Version 7, the DSNWZP stored procedure can execute, as either a DB2 established stored procedure using the external name of DSNWZP, or as a WLM established stored procedure using the name DSNWZPR. The default under Version 7 was to run DSNWZP as a DB2 established stored procedure.

So, provided that you were already running the DSNWZP stored procedure in a WLM managed environment in V7, when you fallback from Version 8 to Version 7, or re-migrate from Version 7 to Version 8, the DSNWZP stored procedure will fail to work. You will need to issue the appropriate ALTER commands to change the external name for the stored procedure.

When falling back to V7, issue the following SQL ALTER statement:

```
ALTER PROCEDURE SYSPROC.DSNWZP EXTERNAL NAME DSNWZPR;
```

When re-migrating to V8, issue:

```
ALTER PROCEDURE SYSPROC.DSNWZP EXTERNAL NAME DSNWZP;
```

> **Important:** You must set NUMTCB=1 in your WLM environment for the DSNWZP stored procedure.

Running the IVP Jobs

Redbooks

Important - You cannot run the Version 8 IVP jobs until DB2 is running in Version 8 new-function mode (NFM)

So, when migrating for Version 7 to Version 8 CM

- Use the Version 7 IVP jobs to
  - Verify the migration
  - Ensures that the old jobs work with Version 8 in NFM

After entering new-function mode

- Run the Version 8 IVP
- The Version 8 IVP jobs are created by the installation CLIST as part of the ENFM option

12.8.28 Running the IVP Jobs

The DB2 Installation Verification Procedure (IVP) jobs are designed to test that DB2 is functioning correctly after a new subsystem has been installed, or an existing subsystem has been migrated to a new version. The IVP jobs are designed to test both existing functionality as well as new functionality in the new release.

How can we use the Version 8 IVP suite to test a successful migration from Version 7 to Version 8 compatibility mode?

We cannot! DB2 will not allow any new function to be executed while running in compatibility mode.

We therefore recommend that you use the DB2 Version 7 IVP suite to test a successful migration from Version 7 to Version 8 compatibility mode.

Once DB2 has successfully moved from compatibility mode, through enabling-new-function mode and into new-function mode, you should then use the Version 8 IVP suite to test the migration. This is why the Version 8 IVP suite of jobs are only generated by the DB2 installation CLIST when the ENFM option is specified, and not during MIGRATE.

Returning to ENFM

You can only return from NFM to ENFM

- Job DSNTIJEN
- Does not undo ENFM changes
- Prevents users from exploiting V8 new functions
- No Version 8 dependent objects frozen
  - Dependency indicator "L"

Steps

- Job DSNTIJEN
- Reassemble DSNHDECP
  - Change for NEWFUN=NO

To return again to new-function mode

- Re-run job DSNTIJNF and reassemble DSNHDECP with NEWFUN=YES

12.9 Returning to enabling-new-function mode

Although there is no supported way to return from new-function mode to compatibility mode or fallback to Version 7, there is a path to return from new-function mode to enabling-new-function mode.

The new installation job DSNTIJEN invokes the new CATENFM utility (CATENFM ENFMON) to mark the DB2 subsystem or data sharing group in enabling-new-function mode. No ENFM processing will be undone. You will also need to re-assemble the data only module, DSNHDECP, with NEWFUN=NO to turn off new-function mode processing in the DB2 precompiler.

This is the only means by which NFM can be turned off so that a subsystem can prevent new Version 8 function from continuing to be used (create new objects/applications that use new functions). All Version 8 dependent objects will continue to be available, as they will not be frozen.

For example, table-based partitioned table space will continue to be available however you cannot create any new ones. All the plans and packages bound in Version 8 NFM are also available and they do not need to be rebound before use. Version 8 format DBDs, plans and packages are also available.

Re-migration to new-function mode is achieved by re-running the installation job DSNTIJNF and re-assembling DSNHDECP with NEWFUN=YES.

This job has merely been provided to “complete logical the set of migration jobs”. We do not expect this path to be used very often.

Migration and the Future

Redbooks

No skip release migration
- Can only get to V8 from V7

Will need to be in NFM before migrating to future releases

Multiple mode concept will be used in future releases
- There will not be a long ENFM conversion process
- Ability to prevent new function from being used until shop is ready
- No accidental use of new function that would make fallback difficult / impossible

12.10 Migration and the future

It is always difficult to look into the crystal ball. However, the new migration strategy introduced in DB2 Version 8 will probably survive well into the future.

You will need to be running DB2 Version 8 new-function mode before you will be allowed to migrate to any future release of DB2. We therefore recommend that you plan to move to NFM as soon as you are ready and your business allows, thereby satisfying this key prerequisite for any future migration of DB2.

There will be no more “skip” releases, as we have seen from Version 5 to Version 7. This was a “once off” strategy to move customers forward through and after the turn of the century “bug” called Y2K!

We can see the multiple migration strategy continuing in future releases of DB2; however, the enabling-new-function phase will probably not be as long. This strategy solves a number of problems. For example:

- Customers are able to restrict the use of new functions until the new version is stable, their applications are stable running under the new version, and they are now ready to exploit the new functions.
- Restricting the use of new functions makes the process of migration and fallback simpler and much less prone to error.

DB2 Catalog Changes

DB2 catalog continues to grow with every DB2 release

DB2 Version	Table Spaces	Tables	Indexes	Columns	Table Check Constraints
V1	11	25	27	269	N/A
V3	11	43	44	584	N/A
V4	11	46	54	628	0
V5	12	54	62	731	46
V6	15	65	93	987	59
V7	20	82	119	1206	105
V8	22	83	132	1265	105

12.11 DB2 catalog changes

This visual shows how the DB2 catalog continues to grow with every release of DB2. In addition to the new catalog objects required to support the new function in DB2 (tables, columns etc.), Version 8 introduces a number of other significant changes to the catalog:

- Adding a number of columns, indexes, and new tables to the DB2 catalog.
- Changing the definition of many existing columns to support long names.
- Conversion of the character columns in the catalog from EBCDIC to Unicode.

  In the next few visuals we introduce the major changes that are made to the DB2 catalog during migration to compatibility mode, and in enabling-new-function mode. Please refer to the *DB2 Release Planning Guide,* SC18-7425, and *DB2 SQL Reference,* SC18-7426, for a complete list of changes to the DB2 catalog.

Migration - New Table Spaces

SYSEBCDC

- EBCDIC table space
- Not used until NFM
  - SYSIBM.SYSDUMMY1 moved from SYSSTR to SYSEBCDC

SYSALTER

- New catalog table SYSIBM.SYSOBDS
  - To store version 0 information of ALTERed tables

12.11.1 Migration - new table spaces

SYSEBCDC is a new table space, defined in DSNDB06. Its encoding scheme is EBCDIC. This table space is used to store the catalog table SYSIBM.SYSDUMMY1, which needs to remain in EBCDIC. Its current (V7) table space, SYSSTR, will be converted to Unicode during enabling-new-function mode.

SYSIBM.SYSDUMMY1 is a “dummy” table which is used by many applications as a “default” or “dummy” table to use in SQL statements. For example:

```
SELECT CURRENT TIMESTAMP
  FROM SYSIBM. SYSDUMMY1
```

Therefore SYSIBM.SYSDUMMY1 can be thought of more as an application table, rather than a catalog table, and it may be imbedded into many applications today. So, if it were to be converted to Unicode, all SQL statements accessing SYSIBM.SYDUMMY1 would become multiple CCSID set statements. Results of multiple CCSID set SQL statements can be different from single CCSID statements; for example, the ordering of result sets, or the use of range predicates can affect the result set. This could cause unnecessary complications for existing applications.

Please note that SYSIBM.SYSDUMMY1 will not be moved until enabling-new-function mode. The table space SYSEBCDC will remain empty until that time. It is interesting to note that the table space will contain only one table, which has one row with one column defined as CHAR(1). We therefore recommend that you do not commit too much space to this new table space.

SYSALTER is another new table space in DSNDB06. It is used to host the new catalog table SYSIBM.SYSOBDS, and stores “version 0” information after a table has been altered.

Migration - New Tables

SYSIBM.IPLIST

- Created in DSNDB06.SYSDDF
- Allows multiple IP addresses to be specified for a given LOCATION

SYSIBM.SYSSEQUENCEAUTH

- Created in DSNDB06.SYSSEQ2
- Records the privileges that are held by users over sequences

SYSIBM.SYSOBDS

- Created in DSNDB06.SYSALTER
- Contains the initial version of a table space or an index space that is still needed for recovery

12.11.2 Migration - new tables

Version 8 introduces three new tables into the DB2 catalog:

- **SYSIBM.IPLIST** is a new table that is defined in the existing SYSDDF catalog table space. It allows multiple IP addresses to be specified for a given LOCATION. The same value for the IPADDR column cannot appear in both the SYSIBM.IPNAMES table and the SYSIBM.IPLIST table.

  The SYSIBM.IPLIST table is summarized in Table 12-4. For a more detailed explanation of this table and how it is used, refer to 7.2, “Member routing in TCP/IP” on page 577

Table 12-4 SYSIBM.IPLIST

Column Name	Data Type	Description
LINKNAME	VARCHAR(24) NOT NULL	This column is associated with the value specified in the LINKNAME column in the SYSIBM.LOCATIONS table and the SYSIBM.IPNAMES table. The values of the other columns in the SYSIBM.IPNAMES table apply to the server identified by the LINKNAME column in this row.
IPADDR	VARCHAR(254) NOT NULL	This column contains the IP address or domain name of a remote TCP/IP host of the server. If WLM Domain Name Server workload balancing is used, this column must contain the member specific domain name. If Dynamic VIPA workload balancing is used, this column must contain the member specific Dynamic VIPA address. The IPADDR column must be specified as follows: If IPADDR contains a left justified character string containing four numeric values delimited by decimal points, DB2 assumes the value is an IP address in dotted decimal format. For example,'123.456.78.91' would be interpreted as a dotted decimal IP address. All other values are interpreted as a TCP/IP gethostbyname socket call. TCP/IP domain names are not case sensitive.
IBMREQD	CHAR(1) NOT NULL WITH DEFAULT 'N	A value of Y indicates that the row came from the basic machine-readable material (MRM) tape.

The index listed in Table 12-5 is defined for SYSIBM.IPLIST

Table 12-5 SYSIBM.IPLIST Indexes

Index Name	Index Description	Index Columns
DSNDUX01	Unique Clustering	LINKNAME

- **SYSIBM.SYSSEQUENCAUTH** is a new catalog table defined in the existing table space DSNDB06.SYSSEQ2. It records the privileges that are held by users over sequences. These are listed in Table 12-6.

Table 12-6 SYSIBM.SYSSEQUENCAUTH

Column Name	Data Type	Description
GRANTOR	VARCHAR(128) NOT NULL	Authorization ID of the user who granted the privileges.
GRANTEE	VARCHAR(128) NOT NULL	Authorization ID of the user or group that holds the privileges or the name of an application plan or package that uses the privileges. PUBLIC for a grant to PUBLIC.
SCHEMA	VARCHAR(128) NOT NULL	Schema of the sequence.

Column Name	Data Type	Description
NAME	VARCHAR(1280 NOT NULL	Name of the sequence.
GRANTEETYPE	CHAR(1) NOT NULL	Type of grantee: **blank** An authorization ID. **P** An application plan or package. The grantee is a package if COLLID is not blank. **R** Internal use only.
AUTHHOWGOT	CHAR(1) NOT NULL	Authorization level of the user from whom the privileges were received. This authorization level is not necessarily the highest authorization level of the grantor. **L** SYSCTRL **S** SYSADM **blank** Not applicable.
ALTERAUTH	CHAR(1) NOT NULL	Indicates whether grantee holds ALTER privilege on the sequence: **blank** Privilege is not held. **G** Privilege is held with the GRANT option. **Y** Privilege is held without the GRANT option.
USEAUTH	CHAR(1) NOT NULLS	Indicates whether grantee holds ALTER privilege on the sequence: **blank** Privilege is not held. **G** Privilege is held with the GRANT option. **Y** Privilege is held without the GRANT option.
COLLID	VARCHAR(128) NOT NULL	If the GRANTEE is a package, its collection name. Otherwise, a string of length zero.
CONTOKEN	CHAR(8) NOT NULL FOR BIT DATA	If the GRANTEE is a package, the consistency token of the DBRM from which the package was derived. Otherwise, blank.
GRANTEDTS	TIMESTAMP NOT NULL	Time when the GRANT statement was executed.
IBMREQD	CHAR(1) NOT NULL	A value of Y indicates that the row came from the basic machine-readable material (MRM) tape.

The indexes listed in Table 12-7 are defined for SYSIBM.SYSSEQUENCEAUTH.

Table 12-7 SYSIBM>SYSSEQUENCAUTH Indexes

Index Name	Index Description	Index Columns
DSNWCX01	Non-Unique Clustering	SCHEMA NAME
DSNWCX02	Non-Unique	GRANTOR SCHEMA NAME
DSNWCX03	Non-Unique	GRANTEE SCHEMA NAME

- **SYSIBM.SYSOBDS** (shown in Table 12-8) is a new catalog table that resides in the new table space DSNDB06.SYSALTER. The table contains copies of old versions of DBDs that may be needed for point-in-time recoveries. It contains one row for each table space or index that can be recovered to an image copy that was made before the first version was generated.

Table 12-8 SYSIBM.SYSOBDS

Column Name	Data Type	Description
CREATOR	VARCHAR(128) NOT NULL	Authorization ID under which the table space or index was ALTERed.
NAME	VARCHAR(128) NOT NULL	Name of the object ALTERed.
DBID	SMALLINT NOT NULL	Identifier of the database to which the ALTERed object belongs.
PSID	SMALLINT NOT NULL	Identifier of the table space or index space descriptor.
OBID	SMALLINT NOT NULL	Identifier of the table or index fan set descriptor.
OBDTYPE	CHAR(1) NOT NULL	Type of object (ODBREC or ODBFS).
VERSION	SMALLINT NOT NULL	Version of original object when ALTERed.
CREATETS	TIMESTAMP NOT NULL	Timestamp when the first new version was created.
DBD	VARCHAR (30000) NOT NULL	OBDREC or OBDFS image.
IBMREQD	CHAR(1) NOT NULL	A value of Y indicates that the row came from the basic machine readable material (MRM) tape.

The indexes listed in Table 12-9 are defined for the table SYSIBM.SYSOBDS

Table 12-9 SYSIBM.SYSOBDS Indexes

Index Name	**Index Description**	**Index Columns**
DSNDOB01	Unique Clustering	CREATOR NAME
DSNDOB02	Non-Unique	DBID PSID

Migration - Other Changes

Redbooks

Column changes

- Existing binary columns will be ALTERed to be FOR BIT DATA columns to ensure they are handled properly during ENFM processing
- Some existing columns converted to FOR BIT DATA

Approx 60 columns added to existing tables

Approx 15 changes to values of existing columns

Approx 45 column definitions have been changed

Approx 10 changes to RI and table check constraints

Column length change to support long names are only done during ENFM processing

12.11.3 Migration - other changes

A number of columns within the DB2 catalog are used to store binary data rather than character data. However these columns are not defined as FOR BIT DATA in V7.

During catalog migration these columns are altered to be defined as FOR BIT DATA. This is to ensure they are converted correctly to Unicode during enabling-new-function mode. In addition, a few other columns which store character data are converted to FOR BIT DATA.

This visual summarizes the number of other changes that are made to existing catalog tables. Once again, please refer to Appendix A of the *DB2 SQL Reference,* SC18-7426 for a complete list of new and changed catalog tables.

During the migration to DB2 Version 8 compatibility mode, no column lengths are being changed to support long names. This occurs during enabling-new-function mode.

CM - Changed Indexes

Table Name	Index Name	Index Description	Index Columns
SYSCOLAUTH	**DSNACX01**	**Non-unique**	**CREATOR, TNAME, COLNAME**
SYSFOREIGNKEYS	DSNDRH01	Non-unique	CREATOR, TBNAME, RELNAME
SYSINDEXES	DSNDXX04	Non-unique	INDEXTYPE
SYSRELS	DSNDLX02	Non-unique	CREATOR, TBNAME
SYSSEQUENCESDEP	DSNSRX02	Non-unique	BSCHEMA, BNAME, DTYPE
SYSTABLEPART	DSNDPX03	Non-unique	DBNAME, TSNAME, LOGICAL_PART
SYSTABAUTH	DSNATX04	Non-unique	TCREATOR, TTNAME
SYSTABLES	DSNDTX03	Non-unique	TBCREATOR, TBNAME
SYSVIEWDEP	DSNGGX04	Non-unique	BCREATOR, BNAME, BTYPE, DTYPE

New indexes on existing catalog tables

12.11.4 Compatibility mode - changed indexes

This visual outlines the changes that are made to the catalog indexes on catalog tables that exist in V7. You might want to consider reviewing the indexes you have created on the catalog, if any. Some of these indexes may be no longer required.

ENFM - Catalog Changes

17 catalog and 1 directory table space converted to Unicode

2 DROPed tables

- SYSLINKS
  - No index data sets to be deleted
- SYSPROCEDURES
  - VSAM data sets for Index DSNKCX01 can be deleted after ENFM

SYSIBM.SYSDUMMY1

- Moved from DSNDB06.SYSSTR to DSNDB06.SYSEDCBC

Many columns changed to VARCHAR

- To support long names

System-defined catalog indexes changed to NOT PADDED

7 catalog and 1 directory table space moved from BP0

12.11.5 ENFM - catalog changes

This visual summarizes the changes made to the DB2 catalog during enabling-new-function mode processing.

- 18 table spaces are converted from EBCDIC to Unicode. The others remain as EBCDIC. This process is defined at length in the discussion on enabling-new-function mode earlier in this chapter. These table spaces are:
  - SPT01 (directory)
  - SYSDBASE
  - SYSDBAUT
  - SYSDDF
  - SYSGPAUT
  - SYSGROUP
  - SYSGRTNS
  - SYSHIST
  - SYSJAVA
  - SYSOBJ
  - SYSPKAGE
  - SYSPLAN
  - SYSSEQ
  - SYSSEQ2
  - SYSSTATS
  - SYSSTR
  - SYSUSER
  - SYSVIEWS

Please note that after the table spaces have been converted to Unicode, columns defined as FOR BIT DATA will not automatically be converted to EBCDIC for display in tools such as SPUFI in ISPF. For example, the SQL TEXT column in SYSIBM.SYSPACKSTMT will be displayed in Unicode rather that EBCDIC. Consequently they will be much harder to read and understand.

Please check your DB2 tools and facilities carefully. You may not be able to execute the same simple queries against the catalog tables to get the information you are used to.

> **Important:** We therefore recommend that you check the tools you regularly use against the DB2 catalog to see how they will handle catalog columns defined as FOR BIT DATA and the new long varchar columns.
>
> You may need to develop new ways to access the data you require. For example, Visual Explain will convert the TEXT column in SYSIBM.SYSPACKSTMT back to something you can read.

The following catalog and directory table spaces are not converted to Unicode and remain as EBCDIC. Many of these tables already contain binary data, so there is no need to convert them to Unicode. They also contain many logical names which must interface with external MVS names (for example, data set names), which remain EBCDIC.

- DBD01 (directory)
- SCT02 (directory)
- SYSLGRNX (directory)
- SYSUTIL (directory)
- SYSCOPY

► Two tables are dropped:

a. SYSIBM.SYSLINKS

This table is no longer used in Version 8. Because the table resides in the DSNDB06.SYSDBASE table space and has no indexes defined, there are no VSAM data sets to be cleaned up.

b. SYSIBM.SYSPROCEDURES

This table is no longer used in V7, since the DB2 stored procedure definitions have been moved to SYSIBM.SYSROUTINES when DB2 was migrated from Version 5 to Version 6 or Version 7. As the table resides in the DSNDB06.SYSPKAGE table space, there is no need to delete the underlying table space VSAM data set. However, the VSAM data set for its only index, DSNKCX01, needs to be deleted after you are in new-function mode.

► The table SYSIBM.SYSDUMMY1 is moved for the table space DSNDB06.SYSSTR to its own table space, DSNDB06.SYSEBCDC.

► Columns in almost every catalog and directory table have changed column type and length to support long names. Many columns are typically changing from char to varchar 3 times the size (for example; CHAR(8) will go to VARCHAR(24)), and other columns will grow to VARCHAR(128). The list of actual columns that will change is extensive. Once again, please refer to Appendix A of the *DB2 SQL Reference,* SC18-7426.

► When you migrate DB2 from Version 7 to Version 8 compatibility mode, the default behavior of the index will be PADDED, as in previous versions of DB2. However during enabling-new-function mode, all the DB2 defined catalog indexes will be changed to NOT PADDED.

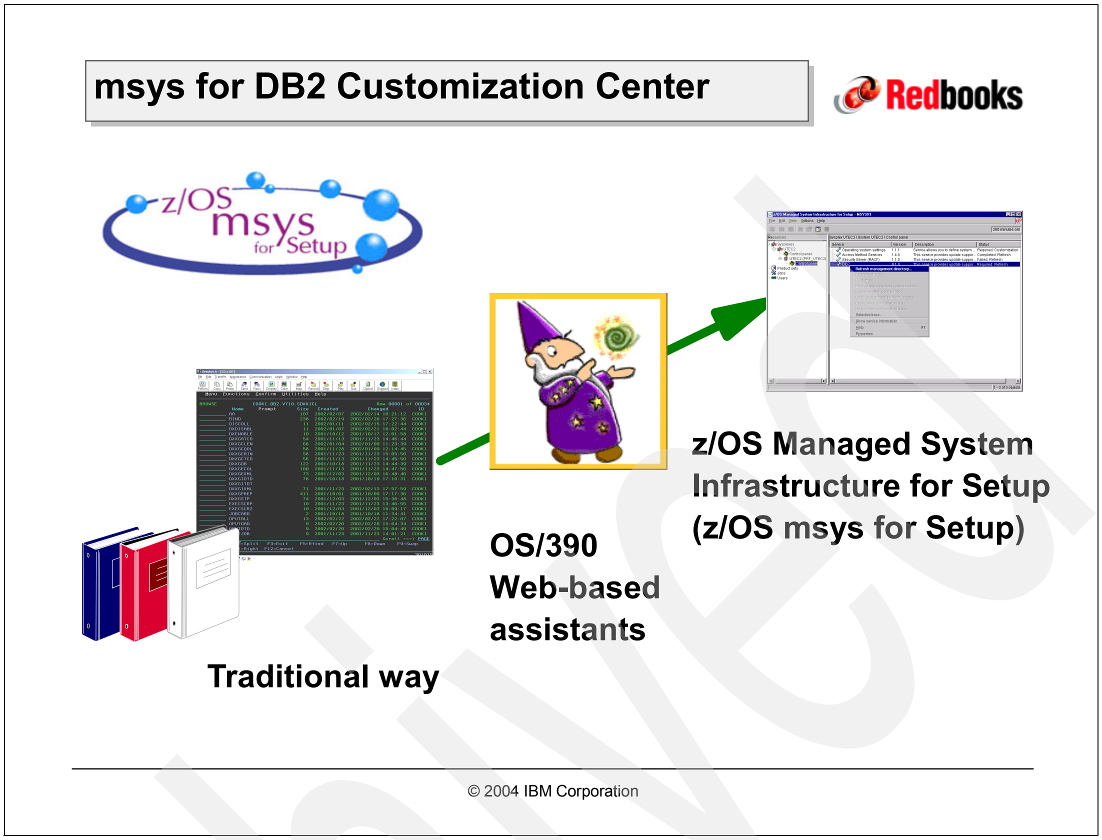
- Increasing the lengths of some of the catalog table columns to support long names causes some catalog table rows to exceed the current 4K page size maximum. In these cases, the table spaces that contain these tables are moved to an appropriately sized buffer pool during enabling-new-function mode processing.

  A total of 7 catalog table spaces and 1 directory table space will be moved from BP0 to either BP8K0 or BP16K0.

Table 12-10 New catalog table space buffer pool assignments

Table space name	Buffer pool	Page Size
SPT01	BP8K0	8K
SYSDBASE	BP8K0	8K
SYSGRNTS	BP8K0	8K
SYSHIST	BP8K0	8K
SYSOBJ	BP8K0	8K
SYSSTR	BP8K0	8K
SYSSTATS	BP16K0	16K
SYSVIEWS	BP8K0	8K

Note that the catalog and directory table spaces are always using a BPxx**K0** buffer pool. In case these buffer pools are currently used by other DB2 objects, you may want to reassign those to other buffer pools, and give the catalog objects their dedicated buffer pool (as you normally do with BP0).



12.12 msys for DB2 Customization Center

You can install your DB2 Version 8 subsystem either as a host based installation using the DB2 installation CLIST, or via the “msys for Setup Facility”. The msys Setup DB2 Customization Center is a workstation based facility which replaces the DB2 Installer workstation tool. In fact, msys for Setup DB2 Customization Center does a lot more for you than the DB2 Installer it replaces, as you will see in the next few visuals:

- Overview and benefits
- Hardware and software prerequisites
- The msys workplace
- How is msys for Setup used to customize a product?

In the Beginning

For a number of years, OS/390 has offered Web-based assistants - "WIZARDS"

- Convenient way of going through planning and ordering tasks
- Defining configuration parameters for some OS/390 components

Why did we introduce them?

- Configuring OS/390 products and components is a manual and document driven process
  - Program directories
  - Installation / Configuration guides
- Setting hundreds of parameters by hand
- Detailed descriptions in one manual and usage information in another

12.12.1 In the beginning

The work associated with installing and maintaining software products has become increasingly complex and costly over time. The drive to reduce costs has conversely increased over time. This has brought many customers today to question the way they install and manage software:

- It is too complex.
- It takes up too much time.
- It requires skills I either do not have available, or I would prefer to use these resources for other tasks.

To address these growing concerns, IBM invested in an initiative to make software more self managing. Such systems can reduce downtime, operating costs and administrative requirements. The goals of this initiative are for the systems to become:

- **Self configuring:** System designed to define itself “on the fly”
- **Self healing:** System capable of autonomic problem determination and resolution
- **Self protecting:** System designed to protect itself from unauthorized access anywhere
- **Self optimizing:** System designed to automatically manage resources in order to meet enterprise needs in the most efficient fashion

OS/390 Web-based Wizards are the first steps in this strategy. They offer these advantages:

- Reduced configuration steps
- No need for the user to understand the syntax in detail
- Reduction in choices and dependencies; do not overwhelm the user
- Automatic generation of jobs and PARMLIB statements

Web-based interactive dialogs for configuration guide the user through a set of high level questions, and use defaults and best practices where possible to reduce decision making. Inputs are checked immediately for syntactical and semantic correctness. The right level of help is available without having to trawl through manuals. The wizards deliver job skeletons that apply the defined values to the system.

However, wizards did not address issues like these:

- Implementation is still to be done by the customer.
- Configuration data is not stored in a central place.
- No discovery of the current system configuration is possible.

What is msys for Setup?

z/OS Managed System Infrastructure for Setup (msys for Setup)

- It is a base element of z/OS - NO extra charge

Part of a major ease-of-use initiative within IBM

- Strategic solution for product installation, configuration, and function enablement

Reduce the complexity associated with:

- Installation
- Function enablement
- Setup and configuration
- Customization

Currently, other z/OS products enabled for msys include:

- TCP/IP, UNIX System Services, RACF, AMS, SMS, ISPF, LE, BCP, Parallel Sysplex and DB2

12.12.2 What is msys for Setup?

Managed System Infrastructure for Setup (msys for Setup) is a z/OS initiative to simplify the customization and installation of all z/OS products. msys for Setup is a base element of z/OS, which currently supports TCP/IP, UNIX System Services, RACF, AMS, SMS, BCP, Parallel Sysplex, ISPF, LE, and now also DB2.

msys for Setup participates in the *IBM autonomic computing initiative* and is an approach to reduce the complexity associated with function enablement, setup, and configuration. msys for Setup builds on Web-based wizard technologies for installing and maintaining software.

msys for Setup has been developed to automate all setup processes that do not require decisions by a system programmer, and assists by deriving values when decisions by a human operator are required. msys for Setup uses wizard-like configuration dialogs that guide the user through a set of high-level questions. These configuration dialogs are part of the graphical user interface called the msys for Setup workplace. The configuration dialogs use defaults and best practices values wherever this is possible to cut down on the number of decisions that you have to make. Because the customization process is now handled by a program instead of being a manual process, input is immediately checked for syntactical and semantic correctness.

A second component of msys for Setup is the z/OS management directory, which is msys for Setup's central repository for configuration data of msys for setup-enabled systems. It is based on the Lightweight Directory Access Protocol (LDAP) directory support that is available as part of z/OS and on the Common Information Model (CIM) data schema. The management directory provides a single interface to all management-related system data.

More information about msys for Setup is available from the Web site:

http://www.ibm.com/servers/eserver/zseries/msys/moreset.html

msys for Setup - Benefits

Redbooks

In the past you needed to:

- Manually update a number of system resources
- Worry about multiple different interfaces and their syntax
- Decide how to set each value
- Check if the configuration is valid
- Read several feet of documentation

With msys:

- Updates to the system are done automatically
- The system takes care of the syntax and interfaces
- The system does the calculations
- Only a small number of values are shown when a decision is made
- The validity of a configuration is checked by the system
- Online help is just a mouse click away

12.12.3 msys for Setup benefits

msys for Setup is the IBM strategy for installing and maintaining z/OS products. It builds upon the Wizard technology employing the same easy "interview style" for defining configuration values. As with Wizards, msys for Setup does not ask you to specify hundreds of parameters. It uses defaults and best practices values and derives low-level answers from high-level questions, and it downloads generated skeletons to the host for execution. msys for Setup caters to both experienced and novice users:

- It provides a consistent look-and-feel for all z/OS products.
- Plug-ins use wizards that reduce the amount of information that the user has to enter to get up and running.
- Context-sensitive and menu-driven help is available at the user's fingertips.
- It eliminates JCL install jobs and automates many install steps that are typically performed manually.

In addition, msys for Setup:

- Allows a user to preview changes that will be made to the system.
- Logs updates made to a system and allows users to browse these updates easily.
- Can retrieve the current parameter settings of a product during refresh (for example; parameter settings for a DB2 subsystem that has to be migrated).
- Allows customization changes to be saved intermediately, so that the user does not have to customize in one sitting.
- Provides cloning support.
- Provides automatic access to the configuration data of other products, reducing the amount of information that is needed from the user (for example; LE library names).

msys for Setup - Overview

Product specific customization dialogs store configuration data in the z/OS Management Directory

- Will become the central repository for all configuration data

Update requests are also stored

- Describe HOW the configuration is to be applied to system
  - Set a field in a PARMLIB member
  - Create a new userid with a set of defined attributes

After the configuration parameters have been specified, msys for Setup can automatically update the system configuration directly

Details the changes before they are made

- Relieves the user of the intricacies of z/OS configuration interfaces
  - Parmlib members

12.12.4 msys for Setup - overview

Managed System Infrastructure for Setup (msys for Setup) establishes a central repository for product configuration data. It also provides a single interface to the repository. It automates all processes that do not require decisions by the system administrator and defines defaults to minimize the situations in which decisions are necessary.

z/OS products that provide an msys for Setup plug-in can be managed using msys for Setup. Parameter values for a particular product that is enabled for use with msys for Setup can be specified using the graphical user interface of the product plug-in. These values are stored in the msys for Setup management directory and eventually used by the host code of the product plug-in to customize and install the product.

For information about using msys for Setup with DB2 UDB for z/OS and other products in your z/OS environment, see *Managed System Infrastructure for Setup User's Guide*, SC33-7985.

msys for Setup - Framework

Redbooks

The msys framework consists of three components:

- msys host program
  - Resides and runs on a z/OS system and manages all installation/customization tasks
- msys management directory
  - Uses an LDAP Server
  - Stores configuration data for all msys-enabled products
- msys workplace
  - Runs on the workstation and provides the user with a *Windows Explorer*-style GUI to manage z/OS products

12.12.5 msys for Setup - framework

The msys for Setup framework consists of three components:

- **The msys for Setup workplace.** It runs on a Windows workstation and provides a graphical user interface similar to the Windows Explorer. The msys for Setup workplace is used to manage z/OS products. You simply interact with the workplace. It can be downloaded to the workstation from any z/OS host.
- **The msys for Setup host program**. The host program usually resides on the z/OS system. It manages all installation and customization tasks.
- **The msys for Setup management directory**. This is based on the Lightweight Directory Access Protocol (LDAP) directory support. This stores the configuration data for all systems on which msys for Setup is enabled. It resides on the z/OS system.

Workstation Requirements

Software requirements:

- Operating system - either Windows NT 4.0 with fixpack 4 (or higher), Windows 2000 or Windows XP

Communication:

- TCP/IP must be configured and active with connectivity to the host

Administrator rights:

- Must logon as Administrator to perform the installation

Hardware requirements:

- Processor Pentium or equivalent
  - Speed 266 MHz or faster
- Memory 128 MB, 192 MB is recommended.
- Available disk capacity 100 MB or more
- Screen resolution 800x600 (1024x768 recommended)

12.12.6 Workstation requirements

The msys for Setup Workplace can be installed onto any machine with either Windows NT4.0 with FixPak 4 (or higher) applied, Windows 2000, or Windows XP. TCP/IP must be configured and running, with connectivity to the host.

When installing the workplace code, the user ID used for performing the installation must have administrator rights on the workstation.

Driving System Requirements

Operating system:

- OS/390 R8 or higher / z/OS R1 or higher

z/OS LDAP Server

Non-OS/390 software:

- Developer kit for OS/390, Java 2 Technology Edition(Java 1.3.0) or
- Java for OS/390 (Java 1.1.8)
  - (Out of support)

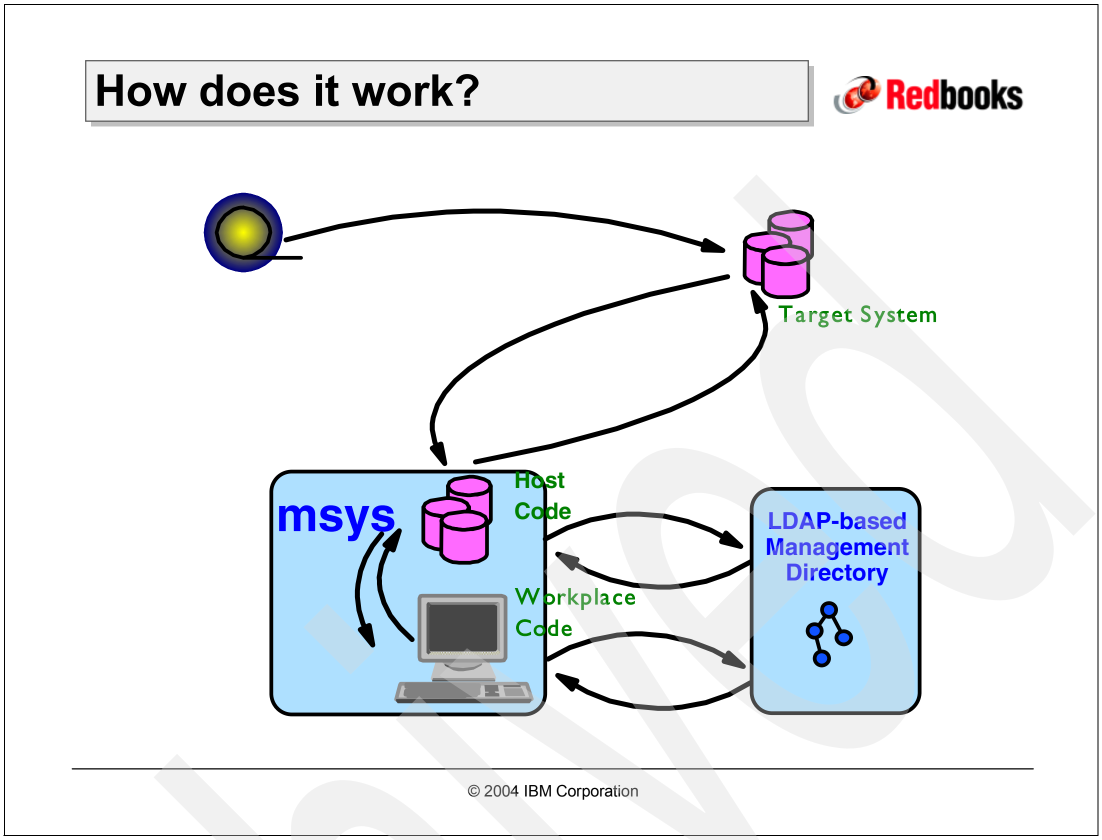
FTP connection from msys workplace

12.12.7 Driving system requirements

The msys for Setup Host program requires z/OS release 1 or higher, or OS/390 release 2.8 or higher, with TCP/IP connectivity to the msys for Setup workstation. So when you are ready to go to DB2 for z/OS V8, the z/OS version you are on should not pose a problem, as DB2 V8 requires z/OS 1.3 or above.

The msys for Setup Host program is a Java based application which runs in UNIX Systems Services. It therefore requires the Java development kit for z/OS to be installed and available.

msys for setup uses the z/OS LDAP Server as its management directory, or repository. This, in turn, requires DB2 for z/OS to store the data.



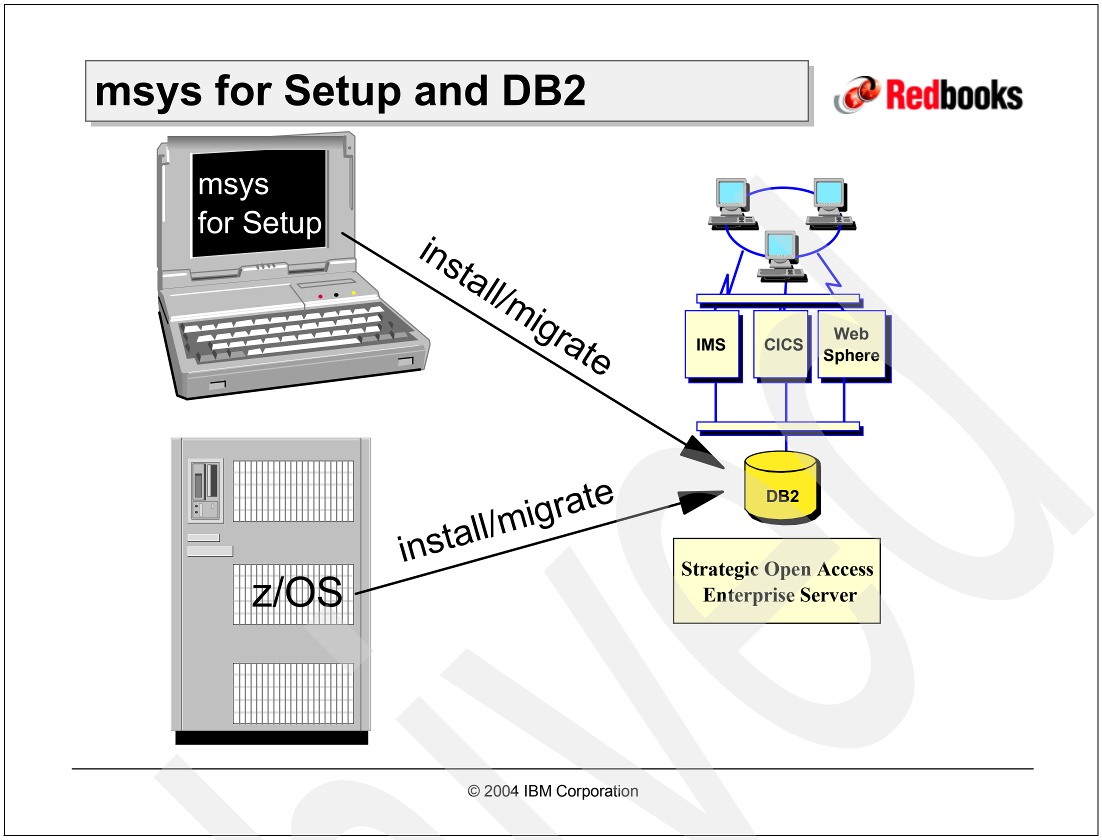
12.12.8 How does it work?

A z/OS product must provide plug-in code to the msys for Setup framework in order to be managed through msys for Setup.

A user interacts with the workstation component of msys for Setup, to enter parameter values for a specific product through the product's plug-in GUI.

These values are used by the product's host plug-in code to customize and install the product.

msys for Setup stores these values in the msys for Setup management directory.



12.12.9 msys for Setup and DB2

DB2 Version 8 ships a plug-in for msys for Setup, known as the DB2 Customization Center. It is an XML document in the new data set hlq.SDSNXML. In order to be able to run the DB2 Customization Center in the msys for Setup framework, PTF OA4581 must be installed.

After you have prepared your z/OS subsystem and workstation for use with msys for Setup, you can add the DB2 Customization Center to the msys for Setup workplace. After doing this, you are ready to use the DB2 Customization Center.

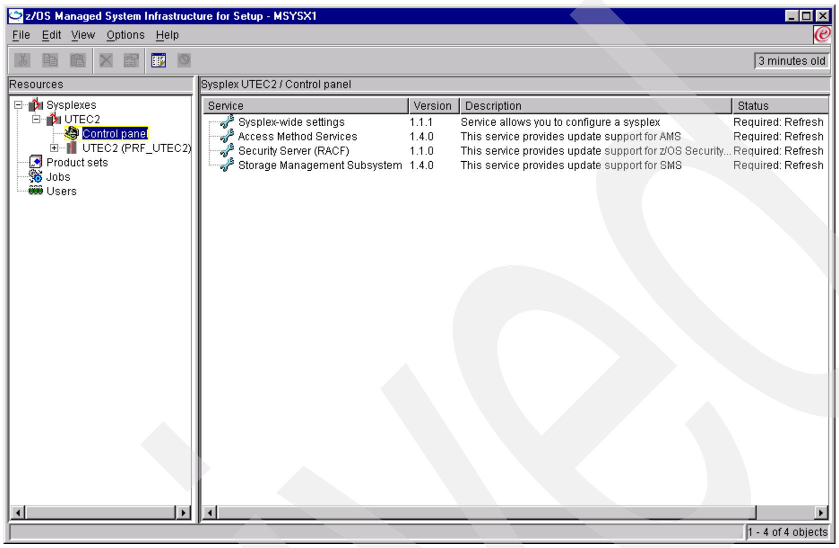
During *refresh*, the DB2 Customization Center retrieves current DB2 and z/OS settings. These values are stored in the msys for Setup management directory. Then, you provide information to the DB2 Customization Center that is used to set up your DB2 subsystem.

During *customization*, you review and, if necessary, modify the values of DB2 system parameters.

Then, during *update*, the DB2 Customization Center applies the changes that you made to the DB2 subsystem.

We introduce these processes in the next few visuals.

msys Workplace



12.12.10 msys for Setup Workplace

This visual provides an example of the msys for Setup workspace.

From here, you can use the navigation tree on the left to install a new product set into msys for Setup and use msys for Setup to install and customize the product in z/OS.

For more information on using the msys for Setup workplace and the various functions you need to perform, see *Managed System Infrastructure for Setup User's Guide*, SC33-7985.

How is msys for Setup Used?

A user must perform the following steps for any z/OS product enabled for msys

1. **Install Product Set** - Registers the product with msys
2. **Refresh Management Directory** - Retrieves current product settings and stores them in the management directory
3. **Customize** - Allows the user to set product specific settings
4. **Update** - Performs actual changes on z/OS resources based on customization

12.12.11 How is msys for Setup used?

You must perform the following steps for any z/OS product enabled for msys for Setup:

1. Install product set:

   You must first register a product, such as DB2, with msys for Setup, which will ask for the name of a product definition file. This is a simple XML file that provides msys for Setup with the names and locations of the entry points into the product plug-in code.

   DB2 Version 8 ships its product definition file in a new SMP/E managed data set called:

   ```
   hlq.SDSNXML(DSNMXML)
   ```

2. Refresh management directory:

   Refresh retrieves the current parameter settings for the product that is being customized. It also retrieves information from other product plug-ins that are also needed by the product that is being customized.

   The “retrieve” process attempts to minimizes the amount of information required from the user. For example, for a DB2 migration, the retrieve process can collect the current subsystem parameters, and the LE library names can be obtained from the LE plug-in.

3. Customize:

   The “customize” process presents the user with a series of customization dialogs to enter product specific settings. Wizards ask the user a series of high level questions that are used to set various customization parameters. Typical defaults are used for 'advanced' parameters. Wizard summary parameter panels, or “Property sheets”, allow the user to modify parameter settings explicitly.

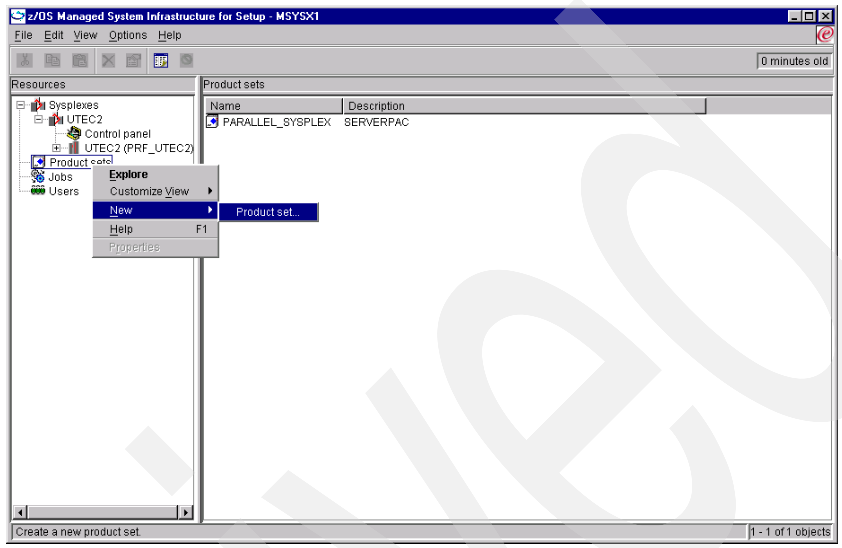
4. Update:

   The “update” process performs the actual customization of the product on the z/OS host using the parameter settings entered by the user in the previous step.

   During this step, the msys for Setup framework runs a batch job on the z/OS host which initiates the msys for Setup (Java) host code that customizes the product. For DB2, this step performs the same tasks as the DB2 install JCL jobs.

   msys for Setup also has an update log which lists all the of the steps that have been performed on the z/OS to customize and install a product. The update log can also list user actions that may have to be performed outside msys for Setup.

Install Product Set



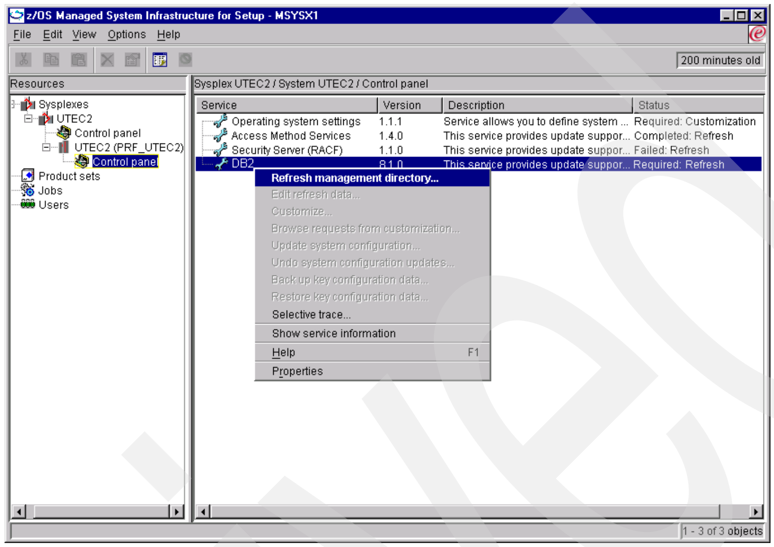
12.12.12 Install Product Set

Before you can use the DB2 Customization Center to install DB2, you must add the DB2 Customization Center to msys for Setup. This tells msys for Setup that the DB2 libraries have been loaded on the z/OS system via SMP/E.

DB2 UDB for z/OS includes an XML document that provides msys for Setup with necessary information. You can use the “Add a product set” wizard from the msys for Setup workplace to add the DB2 product set to the workplace. When using this wizard, specify that you already have an up-to-date XML document for DB2. Enter the data set and member name of the XML document in the following format:

```
prefix.SDSNXML(DSNMXML)
```

Refresh Management Directory



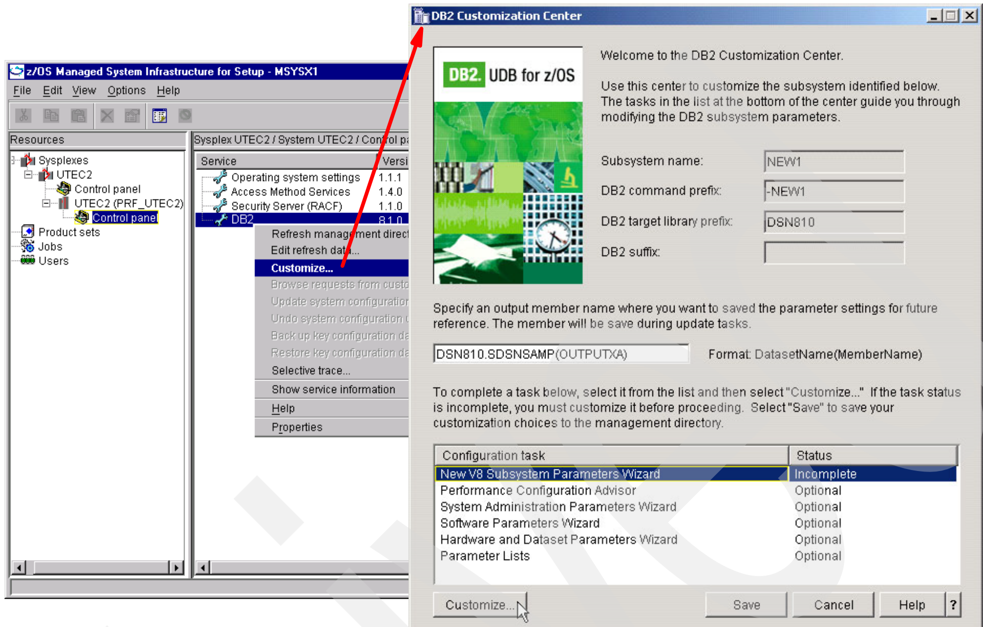
12.12.13 Refresh Management Directory

After you have added the DB2 Customization Center to the msys for Setup framework, you must perform a refresh. During this step, the msys for Setup workplace retrieves any configuration information about your DB2 subsystem that exists on the z/OS host.

The configuration information is stored in the msys for Setup management directory. If you are migrating DB2, customizing an existing DB2 subsystem, or enabling DB2 for data sharing, the refresh step obtains current parameter information from your DB2 subsystem on the z/OS host.

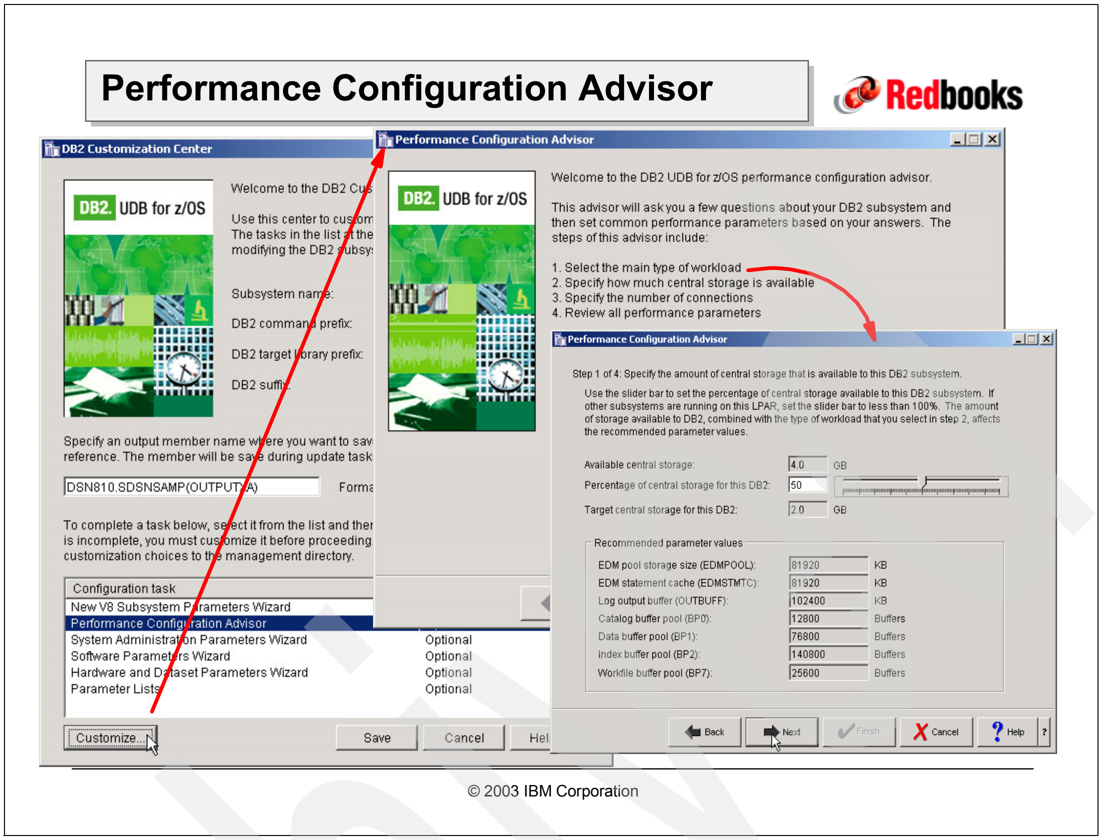
You enter information such as the DB2 subsystem name, command prefix, and target library prefix. If you have previously used the DB2 installation CLIST to customize a DB2 subsystem, you can clone this DB2 by specifying the name of the output member generated by the CLIST. After you have provided this information, you will be able to perform the refresh.

Customizing DB2



12.12.14 Customizing DB2

After you have refreshed the msys for Setup workplace, you can customize DB2. In this step, the DB2 Customization Center contains several wizards that ask you a series of questions. These questions are used to set values for various DB2 parameters. At the end of each wizard, you will be shown a list of DB2 parameters and their values. You can browse this list and modify any parameter value if necessary.



12.12.15 Performance configuration advisor

One of the wizards that are part of the DB2 Customization Center is the “Performance Configuration Advisor”. It is an optional task in the Customize step that will recommend settings for common performance parameters based on your answers to a few simple questions. It dynamically updates the recommended settings as you adjust your answers to the questions.

The DB2 for z/OS Performance Configuration Advisor (PCA) recommends a set of DB2 system parameter (DSNZPARM) values and buffer pool sizes. It is intended to provide good DB2 system level performance for the DB2 system installed through the DB2 Customization Center.

The following information is gathered from the user:

- Type of workload (transaction/decision support /mixed)
- Amount of central storage available to this DB2 subsystem on the LPAR
- Number of local concurrent applications and remote connected users.

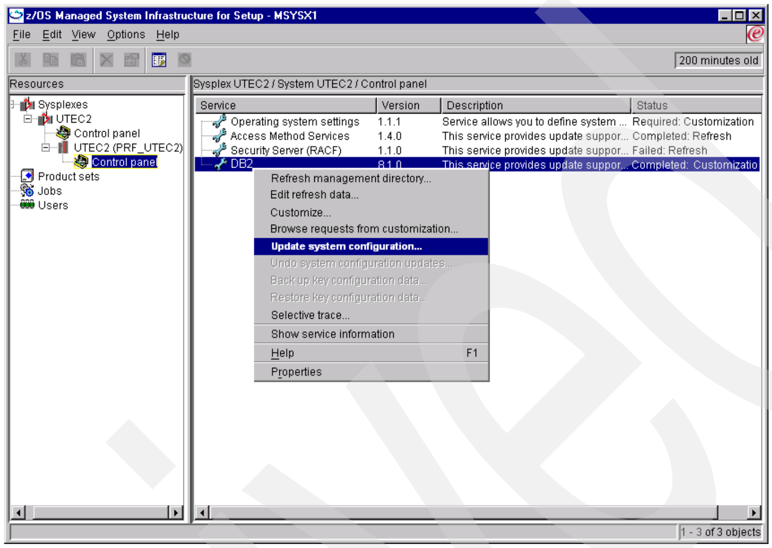
Based on the above data, PCA recommends a set of system parameter values and buffer pool sizes. As the user, you have the option to modify the suggested values.

After you have completed the wizards to customize DB2, an “Update tasks” window appears. This window shows a list of all the tasks that need to be performed on the z/OS host before DB2 can be used with these new values. These update tasks will vary depending on whether you have customized DB2 for installation, migration, or data sharing. Some of the update tasks can be performed by msys for Setup, but some will need to be performed by you or an authorized user outside of the msys for Setup framework. A few tasks may be performed by either msys for Setup or an authorized user.

You can choose to complete these tasks over time, but you cannot use DB2 until all of these tasks have been completed.

If you have already used the DB2 Customization Center to customize a DB2 subsystem, you can copy the parameter values to another DB2 subsystem that is using msys for Setup.

Update

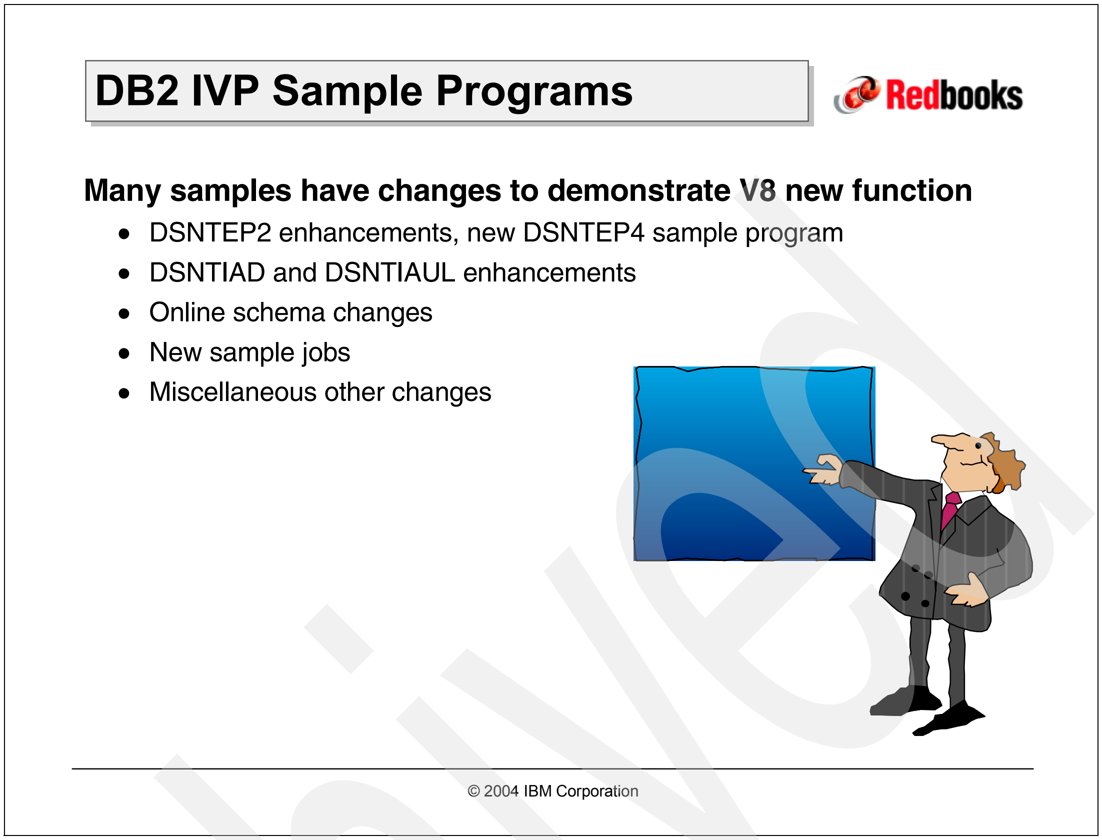


12.12.16 Update

After you have chosen the parameter values you want to customize, you must update the DB2 subsystem with the values that you have chosen. The DB2 Customization Center performs only those tasks that you specified in the “Update tasks” window in the previous step.

Unlike the DB2 installation CLIST, the DB2 Customization Center does not generate JCL jobs. The tasks executed during the update are equivalent to those that are performed by the DB2 installation JCL jobs. If you want to use JCL jobs to configure DB2 on the host, you can use the DB2 Customization Center to generate an output member that can be used as input to the DB2 installation CLIST.

The msys for Setup framework may provide the option to enable the batch job that initiates the “update” step to run independently of the msys for Setup workplace, in a future release.



12.13 DB2 IVP sample programs

DB2 continues to enhance the IVP samples that are shipped, to demonstrate new function in DB2, to provide more features and enhance the usability of the samples themselves. In the next few visuals we will review the major changes to the samples that are shipped with DB2 Version 8:

- DSNSTEP2 enhancements, and a new DSNTEP4 sample program
- DSNTIAUL enhancements
- Online schema changes
- New sample jobs
- Other miscellaneous changes

DSNTEP2 Enhancements - New DSNTEP4

Changed to incorporate new DB2 functionality

- Support for GET DIAGNOSTICS
- Can now handle SQL statements larger than 32k
- Handles longer than 18 character table or column names

Can now modify the MAXERRORS value

- Currently DSNTEP2 tolerates only 10 errors then it stops
- During runtime by including the -- #SET MAXERRORS functional comment in the SQL input statements

Performance enhancements

- Current performance impact when processing large result sets
- SYSPRINT blocking size was very small
- Speeded up the rate in which DSNTEP2 produces outputs results

New DSNTEP4

- Equivalent to DSNTEP2 but uses multi-row fetch

12.13.1 DSNTEP2 enhancements - new DSNTEP4

DSNTEP2 is a PLI program shipped with DB2 to demonstrate the support for dynamic SQL. DSNTEP2 has been enhanced for:

- GET DIAGNOSTICS

  DSNTEP2 now uses GET DIAGNOSTICS to retrieve error information.

- Large SQL statement:

  The sample program DSNTEP2 can now handle SQL statements larger than 32k in size.

- Greater than 18 character table/column names:

  The sample program DSNTEP2 has been modified to handle the longer table and column names.

- New MAXERRORS value:

  A new MAXERRORS parameter has been added in DSNTEP2. It allows you to dynamically set the number of errors that DSNTEP2 will tolerate. In previous versions of DB2, DSNSTEP2 stopped processing after it encountered 10 SQL errors. The MAXERRORS value can be modified during runtime via coding the following functional comment in the SQL statements.

  ```
  --#SET MAXERRORS
  ```

- SYSPRINT blocking in DSNTEP2:

  A change to SYSPRINT blocking in DSNTEP2 will speed up the rate in which DSNTEP2 outputs its results. The blocking size was very small before, thus impacting the performance when processing large result sets. DSNSTEP2 can now also use the system default or user assigned JCL block size.

DB2 V8 also ships a new flavor of DSNTEP2, called DSNTEP4. Its functionality is identical to DSNTEP2, but it allows you to use multi-row fetch when retrieving data from the result set of a query.

- Multi-row fetch:

  The DSNTEP4 sample program uses multi-row FETCH. You can specify a new parameter MULT_FETCH. This option is valid only for DSNTEP4. Use MULT_FETCH to specify the number of rows that are to be fetched at one time from the result table. The default fetch amount for DSNTEP4 is 100 rows, but you can specify from 1 to 32676 rows. It can be coded as a functional comment statement as follows:

```
//SYSIN DD *
   --#SET MULT_FETCH 250
   SELECT * FROM DSN8810.EMP;
```

DSNTIAD, DSNTIAUL

DSNTIAD is a sample dynamic SQL program written in assembler

- Does NOT allow SQL statements larger than 32k

DSNTIAUL is a sample assembler table unload program

- Enhanced to use multi-row FETCH
- Modified to handle SQL statements up to 2MB in size

12.13.2 DSNTIAD and DSNTIAUL

Here we discuss two popular programs that can be used to execute DDL (DSNTIAD) or unload data from DB2 tables (DSNTIAUL).

DSNTIAD

DSNTIAD is a sample dynamic SQL program written in assembler. It has NOT been enhanced to support SQL statements greater than 32k. This is because DSNTIAD is used as a part of the DB2 migration process to Version 8 compatibility mode when new function is not supported.

DSNTIAUL

DSNTIAUL is a sample assembler table unload program. It has been enhanced to:

- Handle SQL statements up to 2 MB in size.
- Use multi-row FETCH.

  You can specify an additional invocation parameter called "number of rows per fetch". It indicates the number of rows per fetch that DSNTIAUL retrieves. You can specify a number from 1 to 32767. If you do not specify this number, DSNTIAUL retrieves 100 rows per fetch. This parameter can be specified together with the SQL parameter, as shown in Example 12-2.

Example 12-2 Invoking DSNTIAUL with multi-row fetch

```
//SYSTSIN DD *
   DSN SYSTEM(DSN)
   RUN PROGRAM(DSNTIAUL) PLAN(DSNTIB81) PARMS('SQL,250') -
       LIB('DSN810.RUNLIB.LOAD')
...
//SYSIN DD *
   LOCK TABLE DSN8810.PROJ IN SHARE MODE;
   SELECT * FROM DSN8810.PROJ;
```

In this case, 250 rows are retrieved in a single fetch operation.

Online Schema Change

Four new steps in DSNTEJ1

- The first step changes the partitioning key on partition 4 of table space DSN8S81E
- The second step adds a fifth partition to the DSN8S81E table space for the EMP table
- The third step reorganizes table space DSN8D81A.DSN8S81E
- The fourth step extends the length of a fixed char column in the PARTS table

Running the DSNTEJ1 job with DSN8S81E table space started demonstrates that these steps can be done without stopping the table space

- An online schema change

12.13.3 Online schema change

The sample job DSNTEJ1 has been enhanced to demonstrate the use of online schema changes: Four new steps have been added to the job:

- The first step reduces the partitioning key on partition of table space DSN8S81E.
- The second step adds a fifth partition to the DSN8S81E table space for the EMP table.
- The third step reorganizes table space DSN8D81A.DSN8S81E.
- The fourth step extends the length of a fixed char column in the PARTS table. It also converts a small integer field to a decimal type field in the EEMP table.

Running the DSNTEJ1 job with DSN8S81E table space started demonstrates that these steps can be done without stopping the table space.

New Sample Jobs

Job DSNTEJ3M uses MQTs containing materialized data derived from existing sample tables

- Creates and populates base and materialized query tables
- Issues EXPLAIN statements that demonstrate the use of MQT by the optimizer for queries against the base tables

Job DSNTEJ6R uses the z/OS Conversion Services

- Prepares and executes program DSN8ED8
  - A C language sample caller of the utilities Unicode parser (DSNUTILU) stored procedure
  - DSNUTILU is the same as DSNUTILS, but accepts Unicode utility statements
- Since DB2 IVPs are provided in EBCDIC, DSN8ED8 uses z/OS Conversion Services to:
  - Convert arguments for DSNUTILU statements from EBCDIC to Unicode
  - Convert DSNUTILU results from Unicode to EBCDIC

12.13.4 New sample jobs

The following sample jobs are provided for your convenience.

Materialized Query Tables (MQTs)

A new sample job is delivered with DB2 Version 8, to demonstrate the use of Materialized Query Tables.

The job DSNTEJ3M creates and populates base and materialized query tables. It also issues EXPLAIN statements that demonstrate the use of MQTs by the optimizer for queries against the base tables.

Utilities Unicode parser:

DSNUTILU is a new stored procedure shipped with Version 8. DSNUTILU is a Unicode version of DSNUTILS. It allows DB2 utilities to be executed as stored procedures; however, it accepts its input as Unicode.

The new job DSNTEJ6R prepares and executes program DSN8ED8, a C language sample caller of the utilities Unicode parser (DSNUTILU) stored procedure. Since DB2's IVPs are provided in EBCDIC, DSN8ED8 uses z/OS Conversion Services to:

- Convert arguments for DSNUTILU statements from EBCDIC to Unicode
- Convert DSNUTILU results from Unicode to EBCDIC

Attention: When you run this job, you must take care when working with the mixed character set in the macro CUNHC. If you change anything here, the job fails.

New Sample Jobs

Redbooks

Three new COBOL samples demonstrate the usage of LOBs

- Job **DSNTEJ76** prepares and runs the programs DSN8CLPL and DSN8CLTC
  - DSN8CLPL is a COBOL version of DSN8DLPL
    - Uses a LOB locator data type to populate BLOB columns greater than 32 KB in length
  - DSN8CLTC is a COBOL version of DSN8DLTC
    - Fetches the data back from DSN8CLPL and verifies that this was the same as the source data
- Job **DSNTEJ77** prepares and runs the program DSN8CLRV
  - DSN8CLRV is a COBOL version of DSN8DLRV
    - Demonstrates LOB locator functions, parsing a CLOB column to pull out resume data to be displayed on an ISPF panel
- Job **DSNTEJ78** prepares and runs the program DSN8CLPV
  - DSN8CLPV is a COBOL version of DSN8DLPV
  - Extracts the BLOB data that contains a photo image and displaying it using GDDM

12.13.5 New sample jobs

DB2 V8 ships a number of new and enhanced sample jobs.

LOBs

Version 8 provides four new sample programs which demonstrate the usage of LOBs with COBOL. These sample programs are prepared and executed using three new sample jobs:

- Job DSNTEJ76 prepares and runs the programs DSN8CLPL and DSN8CLTC. DSN8CLPL is a COBOL version of DSN8DLPL (that is written in C). It uses a LOB locator data type to populate BLOB columns greater than 32 KB in length. DSN8CLTC is a COBOL version of DSN8DLTC (written in C). It fetches the data back from DSN8CLPL and verifies that this was the same as the source data.
- Job DSNTEJ77 prepares and runs the program DSN8CLRV. DSN8CLRV is a COBOL version of DSN8DLRV (that is written in C). It demonstrates LOB locator functions, parsing a CLOB column to pull out resume data to be displayed on an ISPF panel.
- Job DSNTEJ78 prepares and runs the program DSN8CLPV. DSN8CLPV is a COBOL version of DSN8DLPV (that is written in C). It extracts the BLOB data that contains a photo image and displays it using GDDM.

Miscellaneous Other Changes

Redbooks

Changes made to all C and C++ IVP jobs

- All C and C++ language IVP jobs now specify the new precompiler option CCSID(1047) because the IVP C and C++ source code is CCSID 1047

Java stored procedure

- New sample program MRSPcli.java and MRSPsrv.java
- This stored procedure sample returns multiple result sets back to the caller

WLM stored procedures

- All sample stored procedures are converted to WLM-established stored procedures
- Ensure all WLM-established stored procedures have the same environment attributes
- Run in same WLM address space

12.13.6 Other miscellaneous changes

Here we discuss several changes made regarding IVP jobs and stored procedures.

Changes made to all C and C++ IVP jobs:
All C and C++ language IVP jobs now specify the new precompiler option CCSID(1047) because the IVP C and C++ source code is CCSID 1047.

New sample Java stored procedures:
New sample stored procedures, MRSPcli.java and MRSPsrv.java, demonstrate how to return multiple result sets back to the caller. The current plan is to ship them as part of the JDBC FMID after DB2 V8 becomes generally available.

WLM stored procedures
All the DB2 established stored procedure samples have been converted to WLM established stored procedures. Most of the sample stored procedures in Version 7 were DB2 established which are no longer supported in Version 8.

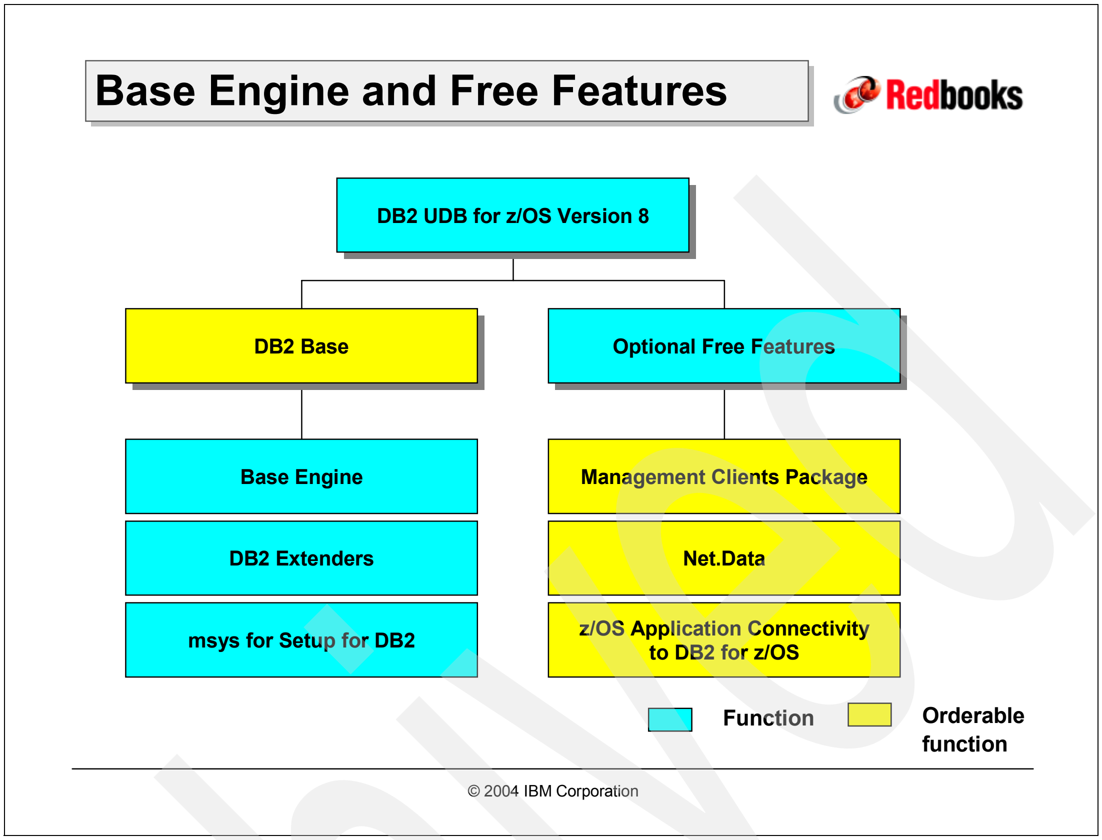
In addition, all the sample stored procedures have the same environment attributes and can run in the same WLM application environment and WLM established stored procedure address space.

12.14 DB2 Version 8 packaging

DB2 Version 8 incorporates several features which include tools for data warehouse management, Internet data connectivity, database management and tuning, installation, and capacity planning.

These features and tools work directly with DB2 applications to help you use the full potential of your DB2 system. When ordering the DB2 base product, you can select the free and chargeable features to be included in the package.

You must check the product announcement and the program directories for current and correct information on the contents of DB2 Version 8 package.



12.14.1 Base engine and no-charge features

In this section we list the DB2 Version 8 base features and optional no-charge features.

DB2 Version 8 base

The DB2 Version 8 Base Engine is program Number 5625-DB2 and currently consists of the following:

- DB2 object code
- Externalized parameter macros
- JCL procedures
- TSO CLISTs
- Link-edit control statements, JCLIN
- Install verification
  - Sample program source statements (sample problem)
  - Sample database data
  - Sample JCL
- DB2 Directory/catalog database
- ISPF components (Installation panels, messages, skeleton library, command table)
- DBRM library
- Online help reader and associated books (it can used be instead of BookManager READ/MVS)
- IRLM Version 2.2

- Call Level Interface feature (which includes JDBC and SQLJ)
- DB2I ISPF panels (in the ordered language)
- REXX language support:

  REXX language and REXX language stored procedure support are now shipped as a part of the DB2 Version 8 base code.(It is no longer a separate FMID as was the case with V7). As before, the DB2 installation job DSNTIJRX binds the REXX language support to DB2 and makes it available for use.

- Utilities:

  With DB2 Version 7 most utilities were grouped in three separate independent products, Operational utilities, Diagnostic and Recovery utilities and the Utilities Suite (which includes all the utilities of the above two products).

  In DB2 Version 8, the Operational utilities and the Diagnostic and Recovery utilities are no longer offered. The Utilities Suite remains and contains all of the IBM DB2 Utilities.

  All the utilities are shipped deactivated with the Base Engine. A valid product licence must be obtained to activate the utilities function. However all utilities are always available for execution on the DB2 catalog and directory and the DB2 IVP objects.

- DB2 Extenders:

  – Audio, Image and Video Extenders

    Audio, Image, and Video Extenders have been stabilized at the V7 level and are provided in DB2 for z/OS, V8, to ensure continuity for current customers. Audio, Image, and Video Extenders do not support the DB2 for z/OS data sharing function in a parallel sysplex environment. For the most current information, visit this site:

    `http://www.ibm.com/software/data/db2/extenders/aiv/aiv390`

  – XML Extender

  – Text Extender

    DB2 Text Extender has been stabilized at the V7 level and is provided in DB2 for z/OS, V8, to ensure continuity for current customers. DB2 Text Extender does not support the DB2 for z/OS data sharing function in a parallel sysplex environment. For the most current information, visit:

    `http://www.ibm.com/software/data/db2/extenders/text/te390`

  The Extenders discussed so far come with the base product (you do not have to order them separate), and they are free of charge.

  – Net Search Extender:

    Although this is a DB2 Extender product, it does *not* come as part of the “base” extenders. It is a separately orderable charge feature.

    Net Search Extender V7 is compatible and delivered with DB2 for z/OS, V8. Net Search Extender does not support the DB2 for z/OS data sharing function in a parallel sysplex environment. For the most current information, visit:

    `http://www.ibm.com/software/data/db2/extenders/netsearch/`

- msys for Setup DB2 Customization Center:

  msys for Setup DB2 Customization Center is enabled for the IBM Managed System Infrastructure (msys). The msys for Setup DB2 Customization Center provides installation customization for DB2 z/OS. DB2 Customization Center is available as a plug-in for msys for Setup and therefore requires msys for Setup, which is included with z/OS Version 1.3.

In addition, msys for Setup has three components:

– msys for Setup workplace, which runs on the workstation
– msys for Setup host code, which runs on a driving system
– msys for Setup management directory, which uses an LDAP server

msys for Setup DB2 Customization Center replaces the DB2 Installer, which was shipped with DB2 Version 7.

Optional no-charge features

The optional no-charge features are the same as with DB2 Version 7.

- DB2 Management Clients Package:

  We explore this package in more detail in the next visual.

- Net.Data:

  Net.Data, a no-charge feature of DB2 Version 8, takes advantage of the z/OS capabilities as a premier platform for electronic commerce and Internet technology. Net.Data is a full-featured and easy to learn scripting language allowing you to create powerful Web applications. Net.Data can access data from the most prevalent databases in the industry: DB2, Oracle, DRDA-enabled data sources, ODBC data sources, as well as flat file and Web registry data. Net.Data Web applications provide continuous application availability, scalability, security, and high performance.

  Net.Data is functionally stable at the level it was when it was shipped with DB2 Version 7.

- z/OS Application Connectivity to DB2 for z/OS:

  This is a no-charge, optional feature of DB2 Universal Database Server for z/OS V8. This feature consists of a component known as the DB2 Universal Database Driver for z/OS, Java Edition, a pure Java, type 4 JDBC driver designed to deliver high performance and scalable remote connectivity for Java-based enterprise applications on z/OS to a remote DB2 for z/OS database server. The driver:

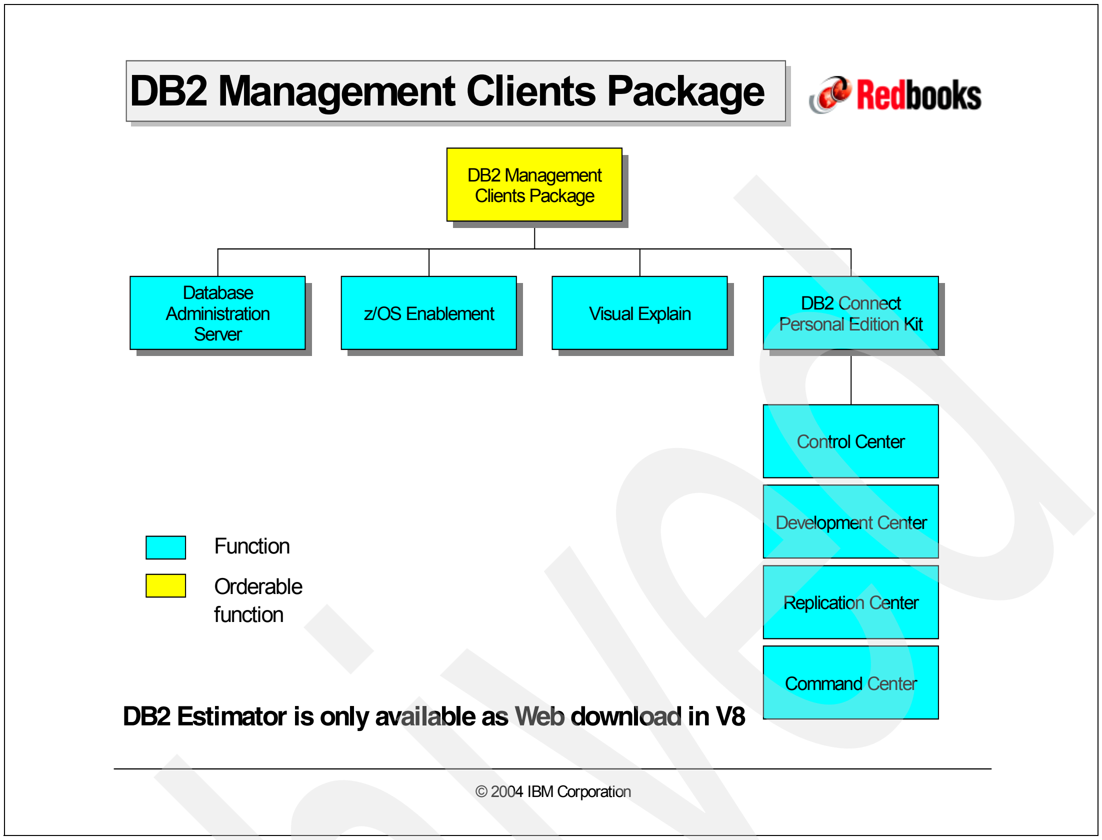
  – Supports JDBC 2.0 and 3.0 specification and JDK V1.4 to deliver the maximum flexibility and performance required for enterprise applications

  – Delivers robust connectivity to the latest DB2 for z/OS and WebSphere Application Server for z/OS

  – Provides support for distributed transaction support (2-phase commit)

  – Allows custom Java applications that do not require an application server to run in a remote partition and connect to DB2 z/OS.

  This feature is ideal for z/OS customers who require the ultimate scalable and reliable DB2 connectivity solution anchored on a WebSphere Application Server framework. DB2 Universal Driver, Java Edition is an integral part of an e-business solution stack that can help achieve the highest level of Web application availability by leveraging the most durable OLTP platform.



12.14.2 DB2 Management Clients Package

The DB2 Management Clients Package is enhanced in Version 8. It is a collection of workstation-based client tools that you can use to work with and manage your DB2 for z/OS environments.

The DB2 Management Clients Package is a separately orderable no-charge feature of DB2 Version 8 (program number 5625-DB2 and feature number is 6001) and it currently consists of the following:

- DB2 Administration Tools (including Control Center, Replication Center, Command Center and other tools that support DB2 for z/OS).
  - Database Administration Server (DAS):

    DAS provides a general mechanism for running z/OS level functions to support the IBM Universal Database GUI Tools such as Control Center, Command Center and Replication Center. DAS provides the following functions:

    - Building and creating JCL jobs (Control Center Version 8 supports creating and storing JCL jobs for most functions including executing DB2 utilities or cloning a subsystem).
    - Reading and writing data sets (supports PS, PDS, PDSE data sets with RECFM=FB).
    - Querying operating system catalog information.
    - Executing shell scripts in z/OS UNIX.
    - Issuing MVS system commands through an extended console.

DAS provides these functions as an element of the DB2 Management Clients Package. DAS is required by, and supports administrative tasks by the DB2 UDB Control Center and Replication Center.

It can be installed in the form of an SMP/E installable package (with FMID HDAS810).The DB2 Administration Server for z/OS can be ordered as a part of the DB2 Management Clients Package (with the 390 Enablement feature), or on its own (without the 390 Enablement feature). This is to allow you to use the DB2 Administration Server for z/OS with your existing DB2 Version 7 systems (where the Center Support is already installed).

– z/OS Enablement:

IBM DB2 Control Center provides support to help you manage DB2 databases on an array of operating systems in your work place. A set of stored procedures, a user-defined function and a set of batch programs must be installed at each DB2 UDB for z/OS subsystem that you want to work with using Control Center and other tools including Replication Center and Information Catalog Center.

The z/OS Enablement provides these stored procedures, user-defined functions and batch programs in the form of an SMP/E installable package (with FMID JDB881D).

- DB2 for z/OS Visual Explain:

  Visual Explain is a workstation based feature of DB2 for z/OS that displays:

  – An easy-to-understand graph of the access paths of SQL statements.

  – Catalog statistics for referenced objects from the access path graph.

  – A list of explainable statements from plans and packages, optionally filtered by costs or access path criteria.

  The graphical representation of the access path allows you to instantly distinguish operations such as a sort, parallel access or the use of one or more indexes. You can view suggestions from the graph that describe how you might improve the performance of your SQL statement.

  Visual Explain allows you filter capabilities by access path of explainable SQL statements. For example, you can choose to only display statements that contain a sort or have an estimated cost greater than 500 milliseconds.

  The report feature of Visual Explain allows you to generate an HTML report regarding the access path descriptions, statistics, SQL text and cost of current explained SQL statement.

  You can also EXPLAIN SQL statements dynamically and immediately, and graph their access path. You can enter that statement, have Visual Explain read it from a file, or extract it from a bound plan or package.

  Also available through Visual Explain is the capability for you to browse the real time settings of DSNZPARMs (subsystem parameters) and DSNHDECP. This function requires an activated WLM address space where the DSNZWP stored procedure is installed.

  DB2 Visual Explain requires Windows NT Version 4.0/2000/XP, DB2 Connect Version 7 or higher, and one of the following communications software: TCP/IP or Communications Server 5.0, or SNA Version 3 integrated SNA support in DB2 Universal Database.

  The latest version of Visual Explain is available on the Web site:

  `http://www.ibm.com/software/db2os39_/db2ve/`

- DB2 Connect Personal Edition Kit:

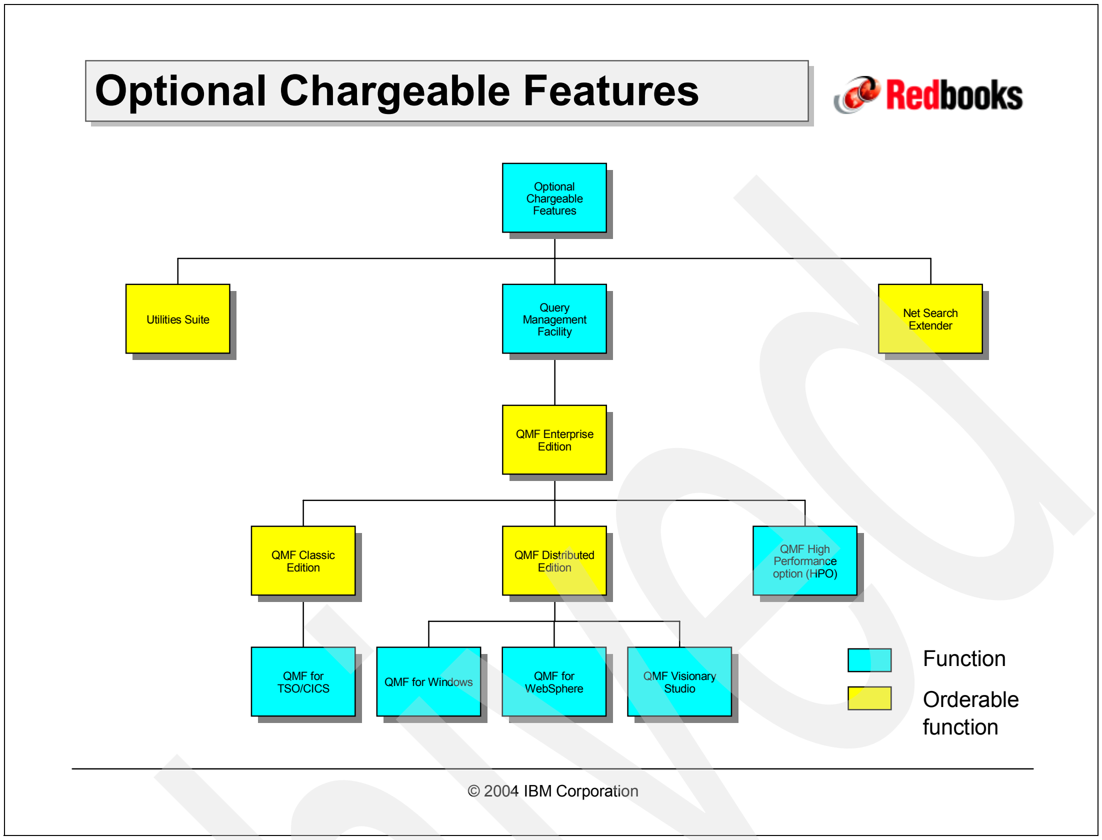
DB2 Connect provides connectivity to the mainframe and midrange databases from Windows, Linux, and UNIX-based platforms. You can connect to DB2 database on AS/400, VSE, VM, MVS, and OS/390. You can also connect to non-IBM databases that comply with the Distributed Relational Database Architecture (DRDA). DB2 Connect Personal Edition is designed for a two-tier environment, where each client connects directly to the host. DB2 Connect Personal Edition does not accept inbound client requests for data.

The DB2 Administration Tools and DB2 Development Center are delivered with all editions of DB2 Universal Database and DB2 Connect products. A restricted-use copy of DB2 Connect Personal Edition Version 8.1 (5724-B56) for Windows is provided in the DB2 Management Clients Package feature of DB2 for z/OS, Version 8 to satisfy this functional dependency.

- DB2 Estimator:

  IBM DB2 Estimator for Windows works with DB2 data to estimate application feasibility, to model application cost and performance, and to estimate required CPU and I/O capacity.DB2 Estimator for Windows has been updated for Version 8. However, with DB2 for z/OS Version 8, DB2 Estimator is *only* available as a download from the DB2 for z/OS Web page, and can be found at:

  http://www.ibm.com/software/data/db2/os390/estimate/



12.14.3 Optional chargeable features

A number of optional chargeable features are available with DB2 Version 8:

- The DB2 Utilities Suite
- Query Management Facility editions
- The DB2 Net Search Extender

The DB2 Utilities Suite

With DB2 V7, the DB2 Utilities are separated from the base product and offered as separate products licensed under the IBM Program License Agreement (IPLA). The DB2 Utilities **were** grouped into three categories;

- DB2 Operational Utilities, which included Copy, Load, Rebuild, Recover, Reorg, Runstats, Stospace, and Unload.
- DB2 Diagnostic and Recovery Utilities, which included Check Data, Check Index, Check LOB, Copy, CopyToCopy, Mergecopy, Modify Recovery, Modify Statistics, Rebuild, and Recover.
- DB2 Utilities Suite, which combines the functions of both DB2 Operational Utilities and DB2 Diagnostic and Recovery Utilities in the most cost effective option.

DB2 Version 8 offers all of the utilities in one package. The only DB2 utility product is the DB2 Utilities Suite (product number 5655-K61 FMIDs JDB881K and JDB881M). The DB2 Operational Utilities and the DB2 Diagnostic and Recovery Utilities offerings are no longer available.

The DB2 Utilities are:

- BACKUP SYSTEM
- CHECK DATA
- CHECK INDEX
- CHECK LOB
- COPY
- COPYTOCOPY
- EXEC SQL
- LOAD
- MERGECOPY
- MODIFY RECOVERY
- MODIFY STATISTICS
- REBUILD INDEX
- RECOVER
- REORG INDEX
- REORG TABLESPACE
- RESTORE SYSTEM
- RUNSTATS
- STOSPACE
- UNLOAD

All DB2 utilities operate on catalog, directory, and sample objects, without requiring any additional products.

Query Management Facility

The Query Management Facility (QMF) is the tightly integrated, powerful, and reliable tool for query and reporting within IBM's DB2 family. QMF for OS/390 is also a separately orderable, priced feature of DB2 Version 8.

The DB2 QMF On Demand feature has been greatly enhanced with V8. It includes:

- Support for DB2 UDB V8, including DB2 Cube Views, long names, Unicode, and enhancements to SQL
- Drag-and-drop building of OLAP analytics, SQL queries, pivot tables, and other business analysis and reports
- Visual data "appliances", such as executive dashboards, that offer unique, visually rich, interactive functionality and interfaces specific to virtually any type of information task
- Database explorer for easily browsing and identifying database assets and other objects they may reference
- QMF for WebSphere, allowing ordinary Web browsers to become "zero maintenance" thin clients for visual on demand access to enterprise DB2 business data

With this release, DB2 QMF is offered in several simplified editions, enabling you to more easily apply its DB2 QMF on demand information strategy to single or multiple database and end-user platforms. QMF V8 consists of the following editions:

- **DB2 QMF Enterprise Edition**

  DB2 QMF Enterprise Edition provides the entire DB2 QMF family of technologies, enabling enterprise-wide business information across end-user and database platforms. DB2 QMF Enterprise Edition consists of these components:

  – DB2 QMF for TSO/CICS
  – DB2 QMF High Performance Option (HPO)
  – DB2 QMF for Windows
  – DB2 QMF for WebSphere
  – DB2 QMF Visionary Studio

Other editions of DB2 QMF offer subsets of QMF Enterprise Edition, as follows.

- **DB2 QMF Distributed Edition**

  DB2 QMF Distributed Edition provides components to support end users functioning entirely from Web or Windows clients to access enterprise databases. This edition consists of:

  – DB2 QMF for Windows
  – DB2 QMF for WebSphere
  – DB2 QMF Visionary Studio

- **DB2 QMF Classic Edition**

  DB2 QMF Classic Edition supports end users functioning entirely from traditional mainframe terminals and emulators (including IBM Host On Demand) to access DB2 UDB databases. This edition consists of:

  – DB2 QMF for TSO/CICS

Note that when you are interested in the functions provided by HPO, you must buy the DB2 QMF Enterprise Edition product.

Net Search Extender

DB2 Net Search Extender contains a DB2 stored procedure that adds the power of fast full-text retrieval to Net.Data, Java, or DB2 CLI applications. It offers application programmers a variety of search functions, such as fuzzy search, stemming, Boolean operators, and section search.

Appendixes

A

IBM DB2 DM Tools support

This appendix describes the V8 support provided by the IBM DB2 Data Management Tools.

A.1 DB2 Data management Tools and DB2 UDB for z/OS V8

Table A-1 shows the different IBM DB2 Data management Tools and their V8 readiness. As you can see, most tools fully exploit DB2 Version 8 at GA time. Some tools will support Version 8 in an already planned upcoming release.

For the most up to date information see:

`http://www.ibm.com/support/docview.wss?rs=434&context=SSZJXP&uid=swg21162152&loc=en_US&cs=utf-8&lang=en+en`

Table A-1 DB2 Tools support for DB2 V8

DB2 Tools	Version	DB2 V8 support			Comments
		NS	T	E	
Application Recovery Tool	1.2.0			X	APAR PQ84689
Data Encryption	1.1.0			X	No change required
DB2 Administration Tool DB2 Administration Tool	4.2.0 5.1.0		X	 X	No change required No change required
DB2 Archive Log Accelerator	2.1.0			X	No change required
DB2 Archive Log Compression	1.1.0	X			Replaced by Archive Log Accelerator
DB2 Automation Tool	1.3.0	X			Support planned in future release
DB2 Bind Manager	2.2.0			X	No change required
DB2 Buffer Pool Analyzer	2.1.0			X	APAR PQ81858
DB2 Change Accumulation FEC Common Code DLC Common Code	1.3.0 1.3.0 1.3.0			X X X	APAR PQ84974 APAR PQ84970 No change required
DB2 Data Archive Expert Grouper Component	1.1.0 1.1.0			X X	APAR PQ84256 APAR PQ84063
DB2 Data Export Facility	1.1.0	X			Replaced by DB2 Test Database Generator
DB2 Data Propagator DB2 Data Propagator	7.1.0 8.1.0		X	 X	APAR PQ50953 APAR PQ74674
DB2 High Performance Unload	2.1.0			X	APAR PQ84688
DB2 II Classic Federation	8.1.0	X			Support planned in future release
DB2 Log Analysis Tool FEC Common Code	2.1.0 1.3.0			X X	APAR PQ84373 No change required
DB2 Object Comparison Tool	3.1.0			X	No change required
DB2 Object Restore FEC Common Code	2.1.0 1.3.0			X X	APAR PQ84483 No change required
DB2 Path checker	2.1.0			X	No change required
DB2 Performance Expert	2.1.0			X	APAR PQ81853

DB2 Tools	Version	DB2 V8 support			Comments
		NS	T	E	
DB2 Performance Monitor	8.1.0			X	APAR PQ81857
DB2 Query Monitor	1.1.0	X			Support planned in future release
DB2 Replidata	3.1.0	X			Support planned in future release
DB2 Row Archive Manager	1.1.0	X			Replaced by DB2 Data Archive Expert
DB2 SQL Performance Analyzer	2.1.0	X			Support planned in future release
DB2 Table Editor	4.3.0			X	APAR PQ80771 and FixPak 5
DB2 Test DB Generator Grouper Component	2.1.0 1.1.0			X X	APAR PQ84475 APAR PQ84063
DB2 Utilities Suite	8.1.0			X	No change required
DB2 Web Query Tool	1.3.0			X	FixPak 5

NS = Not Supported
T = Toleration support
E = Exploitation of V8 functions

B

Unicode setup

This appendix describes the minimal Unicode definitions you need, in order for your DB2 V8 system to work properly.

B.1 CCSID setup

DB2 Version 8 requires you to provide a DSNHDECP module that specifies valid, non-zero CCSIDs for single-byte character sets (SBCS) for both EBCDIC and ASCII. For Far East languages like Chinese, and Korean, you must also specify valid, non-zero CCSIDs for mixed-byte (MBCS) and double-byte (DBCS) character sets for both EBCDIC and ASCII. See Appendix A of the *DB2 Installation Guide,* GC18-7418 for information on choosing valid CCSIDs for ASCII and EBCDIC.

DB2 uses the same CCSIDs for Unicode regardless of language. DB2's Unicode CCSID are:

- 367 for SBCS data
- 1208 for MBCS data
- 1200 for DBCS data.

Unicode UTF-16 (CCSID 1200) supports the same group of characters as Unicode UTF-8 (CCSID 1208), so they are compatible and can convert to and from any EBCDIC/ASCII SBCS/MBCS/DBCS CCSIDs.

DB2 requires the z/OS Unicode Services and appropriate conversion definitions to perform most Unicode conversions. For additional information on setup, read the Information APARs II13048 and II13049.

B.1.1 Required Unicode conversions

These z/OS Unicode Services conversion definitions are required by all DB2 V8 systems.

Required conversion for Unicode code pages

In planning your z/OS Unicode Services setup, you need to begin with a set of basic conversions between DB2's SBCS, DBCS, MBCS Unicode CCSIDs, as follows:

- CONVERSION 367, 1200, ER;
- CONVERSION 367, 1208, ER;
- CONVERSION 1200, 367, ER;
- CONVERSION 1200, 1208, ER;
- CONVERSION 1208, 367, ER;
- CONVERSION 1208, 1200, ER;

Required conversions to use the DB2 V8 samples

Some DB2-supplied sample programs require precompiler option CCSID(37) or CCSID(1047). Therefore, the following conversions are also needed:

- CONVERSION **00037**, 00367, ER;
- CONVERSION 00037, 01200, ER;
- CONVERSION 00037, 1208, ER;
- CONVERSION 00367, 0037, ER;
- CONVERSION 01200, 00037, ER;
- CONVERSION 1208, 00037, ER;

- CONVERSION **01047**, 00367, ER;
- CONVERSION 01047, 01200, ER;
- CONVERSION 01047, 1208, ER;
- CONVERSION 00367, 1047, ER;
- CONVERSION 01200, 1047, ER;
- CONVERSION 1208, 1047, ER;

For completeness, add conversions between 37 and 1047:

- CONVERSION 00037, 01047, ER;
- CONVERSION 001047, 0037, ER;

Required conversion definitions for EBCDIC CCSID 500

For some work, DB2 relies on CCSID 500. Therefore you must set up conversions between CCSID 500 and the different Unicode CCSIDs:

- CONVERSION **500**, 00367, ER;
- CONVERSION 500, 01200, ER;
- CONVERSION 500, 01208, ER;
- CONVERSION 00367, 500, ER;
- CONVERSION 01200, 500, ER;
- CONVERSION 01208, 500, ER;

Required conversion definitions for EBCDIC CCSIDs

If your DSNHDECP specifies an EBCDIC SBCS CCSID (SCCSID) other than 37 or 1047, you need to set up the following additional conversions:

- CONVERSION ***<your sccsid>***, 00367, ER;
- CONVERSION <*your sccsid*>, 01200, ER;
- CONVERSION <*your sccsid*>, 01208, ER;
- CONVERSION 00367, <*your sccsid*>, ER;
- CONVERSION 01200, <*your sccsid*>, ER;
- CONVERSION 01208, <*your sccsid*>, ER;

For completeness, also add conversions between your SCCSID and 37, and between SCCSID and 1047:

- CONVERSION 00037, <*your sccsid*>, ER;
- CONVERSION <*your sccsid*>, 00037, ER;
- CONVERSION 01047, <*your sccsid*>, ER;
- CONVERSION <*your sccsid*>, 01047, ER;

When using Far East languages, also add conversions between your EBCDIC MBCS CSSID (MCCSID) and EBCDIC DBCS CCSID (GCCSID), and each Unicode CCSID:

- CONVERSION ***<your mccsid>***, 00367, ER;
- CONVERSION <*your mccsid*>, 01200, ER;
- CONVERSION <*your mccsid*>, 01208, ER;
- CONVERSION 00367, <*your mccsid*>, ER;
- CONVERSION 01200, <*your mccsid*>, ER;
- CONVERSION 01208, <*your mccsid*>, ER;

- CONVERSION ***<your gccsid>***, 00367, ER;
- CONVERSION <*your gccsid*>, 01200, ER;
- CONVERSION <*your gccsid*>, 01208, ER;
- CONVERSION 00367, <*your gccsid*>, ER;
- CONVERSION 01200, <*your gccsid*>, ER;
- CONVERSION 01208, <*your gccsid*>, ER;

Required conversion definitions for ASCII CCSIDs

You need these additional conversions for your ASCII SBCS CCSID (ASCCSID):

- CONVERSION ***<your asccsid>***, 00367, ER;
- CONVERSION <*your asccsid*>, 01200, ER;
- CONVERSION <*your asccsid*>, 01208, ER;

- CONVERSION 00367, <*your asccsid*>, ER;
- CONVERSION 01200, <*your asccsid*>, ER;
- CONVERSION 01208, <*your asccsid*>, ER;

For completeness, also add conversions between your ASCCSID and 37, and between your ASCCSID and 1047:

- CONVERSION 00037, <*your asccsid*>, ER;
- CONVERSION <*your asccsid*>, 00037, ER;
- CONVERSION 01047, <*your asccsid*>, ER;
- CONVERSION <*your asccsid*>, 01047, ER;

When using Far East languages, you need to add conversions between your ASCII MBCS CCSID (AMCCSID) and ASCII DBCS CCSID (AGCCSID), and each Unicode CCSID, as follows:

- CONVERSION **<*your amccsid*>**, 00367, ER;
- CONVERSION <*your amccsid*>, 01200, ER;
- CONVERSION <*your amccsid*>, 01208, ER;
- CONVERSION 00367, <*your amccsid*>, ER;
- CONVERSION 01200, <*your amccsid*>, ER;
- CONVERSION 01208, <*your amccsid*>, ER;

- CONVERSION **<*your agccsid*>**, 00367, ER;
- CONVERSION <*your agccsid*>, 01200, ER;
- CONVERSION <*your agccsid*>, 01208, ER;
- CONVERSION 00367, <*your agccsid*>, ER;
- CONVERSION 01200, <*your agccsid*>, ER;
- CONVERSION 01208, <*your agccsid*>, ER;

B.2 Required ASCII - EBCDIC conversions

The following z/OS Unicode Services conversion definitions are required to convert between your ASCII and EBCDIC CCSIDs:

If your DSNHDECP specifies an SCCSID other than 37 or 1047, you need these additional conversions:

- CONVERSION <*your sccsid*>, <*your asccsid*>, ER;
- CONVERSION <*your asccsid*>, <*your sccsid*>;

For Far East languages, add conversions between your MCCSID and AMCCSID and between your GCCSID and AGCCSID:

- CONVERSION <*your mccsid*>, <*your amccsid*>, ER;
- CONVERSION <*your amccsid*>, <*your mccsid*>
- CONVERSION <*your gccsid*>, <*your agccsid*>, ER;
- CONVERSION <*your agccsid*>, <*your gccsid*>

C

Instrumentation enhancements

This appendix contains an overview of the instrumentation enhancements grouped together. Most of these are discussed in more detail throughout this publication.

Note: To make sure that you have an accurate description of the V8 trace records in member DSNWMSGS, you should have the PTF for APAR PQ86477 applied (still open at the time of writing). You should also be aware that DSNWMSGS is moving. The PTF for PQ79140 copies this member from the SDSNSAMP library into the SDSNIVPD data set. The PTF for PQ90629 (still open at the time of writing) will delete the member from the SDSNSAMP data set.

C.1 Statistics trace enhancements

In this section we discuss the IFC enhancements related to trace records that can be obtained through the statistics trace.

C.1.1 Additional high water mark counters

V8 adds the following high water mark counters to the statistics record:

- Q3STHWIB - High water mark for IDBACK, the number of background connections
- Q3STHWIF - High water mark for IDFORE, the number of foreground connections
- Q3STHWCT - High water mark for CTHREAD, the number of active threads

These fields also show up in DB2 Performance Expert (DB2 PE) statistics reports, in the “subsystem services” section, as shown in Example C-1:

Example: C-1 Enhanced subsystem services reporting

```
SUBSYSTEM SERVICES           QUANTITY  /SECOND  /THREAD  /COMMIT
---------------------------  --------  -------  -------  -------
IDENTIFY                            0     0.00     0.00     0.00
CREATE THREAD                       2   261.75     1.00     1.00
SIGNON                              0     0.00     0.00     0.00
TERMINATE                           2   261.75     1.00     1.00
ROLLBACK                            0     0.00     0.00     0.00

COMMIT PHASE 1                      0     0.00     0.00     0.00
COMMIT PHASE 2                      0     0.00     0.00     0.00
READ ONLY COMMIT                    0     0.00     0.00     0.00

UNITS OF RECOVERY INDOUBT           0     0.00     0.00     0.00
UNITS OF REC.INDBT RESOLVED         0     0.00     0.00     0.00

SYNCHS(SINGLE PHASE COMMIT)         2   261.75     1.00     1.00
QUEUED AT CREATE THREAD             0     0.00     0.00     0.00
SUBSYSTEM ALLIED MEMORY EOT         0     0.00     0.00     0.00
SUBSYSTEM ALLIED MEMORY EOM         0     0.00     0.00     0.00
SYSTEM EVENT CHECKPOINT             0     0.00     0.00     0.00

HIGH WATER MARK IDBACK              1   130.87     0.50     0.50
HIGH WATER MARK IDFORE              1   130.87     0.50     0.50
HIGH WATER MARK CTHREAD             1   130.87     0.50     0.50
```

C.1.2 Enhanced DBM1 and z/OS storage usage reports

Additional fields were added to storage related IFCIDs 225 and 217 for 64 bit addressing. A sample DB2 PE statistics trace is shown in Example C-2. This report is based on IFCID 225, which is what most people use. IFCID 217 has more detailed information, but is normally only used under the guidance if the IBM Support team.

Example: C-2 Sample virtual storage report section of the DB2 PE statistics trace

```
DBM1 AND MVS STORAGE BELOW 2 GB                       QUANTITY    DBM1 AND MVS STORAGE BELOW 2 GB   CONTINUED          QUANTITY
--------------------------------------------  ----------------    ------------------------------------------  ----------------
TOTAL DBM1 STORAGE BELOW 2 GB          (MB)             227.16    24 BIT LOW PRIVATE                     (MB)              0.21
 TOTAL GETMAINED STORAGE               (MB)             148.14    24 BIT HIGH PRIVATE                    (MB)              0.43
```

```
   VIRTUAL BUFFER POOLS                   (MB)                 N/A
   VIRTUAL POOL CONTROL BLOCKS            (MB)                 N/A
   EDM POOL                               (MB)              146.48
   COMPRESSION DICTIONARY                 (MB)                 N/A
   CASTOUT BUFFERS                        (MB)                 N/A
   DATA SPACE LOOKASIDE BUFFER            (MB)                 N/A
   HIPERPOOL CONTROL BLOCKS               (MB)                 N/A
   DATA SPACE BP CONTROL BLOCKS           (MB)                 N/A
 TOTAL VARIABLE STORAGE                   (MB)               67.11
  TOTAL AGENT LOCAL STORAGE               (MB)               45.89
   TOTAL AGENT SYSTEM STORAGE             (MB)                2.81
     NUMBER OF PREFETCH ENGINES                               7.00
     NUMBER OF DEFERRED WRITE ENGINES                        12.00
     NUMBER OF CASTOUT ENGINES                                0.00
     NUMBER OF GBP WRITE ENGINES                              0.00
     NUMBER OF P-LOCK/NOTIFY EXIT ENGINES                     0.00
   TOTAL AGENT NON-SYSTEM STORAGE         (MB)               43.08
     TOTAL NUMBER OF ACTIVE USER THREADS                    100.00
  RDS OP POOL                             (MB)                 N/A
  RID POOL                                (MB)                0.19
  PIPE MANAGER SUB POOL                   (MB)                0.00
  LOCAL DYNAMIC STMT CACHE CNTL BLKS      (MB)                3.93
  THREAD COPIES OF CACHED SQL STMTS       (MB)               15.70
    IN USE STORAGE                        (MB)                3.33
    STATEMENTS COUNT                                        438.00
    HWM FOR ALLOCATED STATEMENTS          (MB)                3.33
      STATEMENT COUNT AT HWM                                438.00
      DATE AT HWM                                         03/25/05
      TIME AT HWM                                      19:48:36.43
  BUFFER & DATA MANAGER TRACE TBL         (MB)                 N/A
 TOTAL FIXED STORAGE                      (MB)                0.17
 TOTAL GETMAINED STACK STORAGE            (MB)               11.74
STORAGE CUSHION                           (MB)              134.53

DBM1 STORAGE ABOVE 2 GB                                   QUANTITY
-------------------------------------------    ----------------
FIXED STORAGE                             (MB)                2.54
GETMAINED STORAGE                         (MB)             2118.84
   COMPRESSION DICTIONARY                 (MB)                0.00
   CACHED DYNAMIC SQL STATEMENTS (MAX)    (MB)             1024.00
   DBD CACHE (MAX)                        (MB)             1024.00
VARIABLE STORAGE                          (MB)               68.31
VIRTUAL BUFFER POOLS                      (MB)              468.75
VIRTUAL POOL CONTROL BLOCKS               (MB)                0.14
CASTOUT BUFFERS                           (MB)                0.00
STAR JOIN MEMORY POOL                     (MB)                0.00

31 BIT EXTENDED LOW PRIVATE               (MB)               30.63
31 BIT EXTENDED HIGH PRIVATE              (MB)              242.98
EXTENDED REGION SIZE (MAX)                (MB)             1628.00
EXTENDED CSA SIZE                         (MB)              256.88

AVERAGE THREAD FOOTPRINT                  (MB)                0.64
MAX NUMBER OF POSSIBLE THREADS                             1924.51

REAL AND AUXILIARY STORAGE                                QUANTITY
-------------------------------------------    ----------------
REAL STORAGE IN USE                       (MB)              352.51
AUXILIARY STORAGE IN USE                  (MB)                0.00
```

Note that more fields were added than just fields for storage allocated above the bar.

The real and auxiliary storage statistics (also available in V7), as well as storage information about the *local* dynamic statement cache were added by APAR/PTF PQ91101/UQ92441, and supported by DB2 Performance Expert via the PTF for PQ94872. This DB2 PE PTF also introduces the:

```
AVERAGE THREAD FOOTPRINT
MAX NUMBER OF POSSIBLE THREADS
```

fields. This information is derived by DB2 PE from other fields in the report. The info is not provided by DB2 itself.

The *local* dynamic statement cache storage information is stored in the following fields in IFCID 225:

QW0225LC	Number of statements in cached SQL statements pools.
QW0225LS	Allocated storage for thread copies in cached SQL statements.
QW0225H	High water mark allocated storage for thread copies in cached SQL statements pools.
QW0225HC	High water mark # of statements in cached SQL statements pools at high storage time.
QW0225HT	Timestamp at high water.

In addition, IFCID 225 and 217 have been added to monitor class 1 and are available via IFI READS requests, but you have to make sure the return area is large enough.

In addition for V8 only, a field in DSNDQISE (EDM statistics - IFCID 2 section 7) that was previously unused is now defined to be the number of SQL statements in the EDM statement cache pool.

QISESTMT Number of statements in EDM *global* dynamic statement cache

C.1.3 Package information added to deadlock trace records

As deadlock trace records are part of class 3 of the statistics trace, we discuss this enhancement next.

When a deadlock (IFCID 172) occurs, DB2 will now add package/DBRM related information about the holder and blocker that allows you to identify the culprit and victims more easily. The following information is added for both the blockers and holders of the locks involved in the deadlock:

- Location name
- Collection name
- Program or package name
- Consistency token

This information is also available from DB2 PE, as shown in Example C-3:

Example: C-3 Enhanced deadlock trace record

```
LOCATION: PMODB2A                          DB2 PERFORMANCE EXPERT (V2)                               PAGE: 1-3
       GROUP: N/A                                  RECORD TRACE - LONG                         REQUESTED FROM: NOT SPECIFIED
      MEMBER: N/A                                                                                          TO: NOT SPECIFIED
   SUBSYSTEM: SDA1                                                                              ACTUAL FROM: 04/16/04 09:53:59.42
 DB2 VERSION: V8                                                                                  PAGE DATE: 04/16/04
 PRIMAUTH CONNECT   INSTANCE      END_USER       WS_NAME                         TRANSACT
 ORIGAUTH CORRNAME  CONNTYPE      RECORD TIME    DESTNO ACE IFC  DESCRIPTION   DATA
 PLANNAME CORRNMBR                TCB CPU TIME              ID
 -------- -------- ----------- ----------------- ------ --- --- ------------- ----------------------------------------------------
 N/P      N/P      00BB46094741 N/P              N/P                         N/P
 N/P      N/P      'BLANK'     09:53:59.43270268 103075   3 172 DEADLOCK DATA  NETWORKID:  G998C442  LUNAME:  HF05      LUWSEQ:     1
 N/P      N/P                  N/P
         |-------------------------------------------------------------------------------------------------------------------------
         |                                                  DEADLOCK HEADER
         |INTERVAL COUNT:     152949   WAITERS INVOLVED:         2   TIME DETECTED: 04/16/04 09:53:59.424504
         |.........................................................................................................................
         |                                                    UNIT OF WORK
         |                                                  R E S O U R C E
         |LOCK RES TYPE: PAGESET LOCK                        DBID: DSNDB06            OBID: SYSDBASE
         |LOCK HASH VALUE: X'00000906'
         |                                                   B L O C K E R
         |PRIMAUTH    : AND         PLAN NAME : DISTSERV     CORR ID : javaw.exe          CONN ID : SERVER
         |NETWORKID   : G998C442    LUNAME    : HF05         OWNING WORK UNIT:       223    UNIQUENESS VALUE: X'00BB46094741'
         |MEMBER      : SDA1        DURATION  :   COMMIT     STATE   : INTENT EXCLUSIVE      ACE     :   3
         |TRANSACTION : javaw.exe                            WS_NAME : B55Z5WHO              END_USER: and
         |PROGRAM NAME: SYSSH200    LOCATION  : 'BLANK'                                      PCKG/COLL ID: NULLID
         |CONS TOKEN  : X'5359534C564C3031'
         |STATUS     :  HOLD
         |QW0172HF: X'12'
         |                                                    W A I T E R
         |PRIMAUTH    : CHM         PLAN NAME : DSNESPRR     CORR ID : CHM                CONN ID : TSO
         |NETWORKID   : DEIBMIPS    LUNAME    : IPSASDA1     OWNING WORK UNIT:       132    UNIQUENESS VALUE: X'BB13C6E7AD25'
         |MEMBER      : SDA1        DURATION  :   COMMIT     STATE   : SHARED INTENT EXCLUSIVACE     :   2
         |TRANSACTION : 'BLANK'                              WS_NAME : 'BLANK'               END_USER: 'BLANK'
         |PROGRAM NAME: DSNESM68    LOCATION  : 'BLANK'                                      PCKG/COLL ID: DSNESPRR
         |CONS TOKEN  : X'149EEA901A79FE48'
         |DB2S ASIC   :     5008    REQ WORK UNIT:    132    EB PTR : X'206F8540'           REQ FUNCTION: CHANGE
         |WORTH       : X'12'       QW0172WG: X'32'
         |
         |                                                  R E S O U R C E
         |LOCK RES TYPE: DATA BASE LOCK                      DBID: DSNDB04            OBID: 0
         |LOCK HASH VALUE: X'00000080'
         |                                                   B L O C K E R
```

```
|PRIMAUTH    : CHM        PLAN NAME : DSNESPRR     CORR ID : CHM                    CONN ID : TSO
|NETWORKID   : DEIBMIPS   LUNAME    : IPSASDA1     OWNING WORK UNIT:        132     UNIQUENESS VALUE: X'BB13C6E7AD25'
|MEMBER      : SDA1       DURATION  :   COMMIT     STATE   : EXCLUSIVE              ACE     :   2
|TRANSACTION : 'BLANK'                             WS_NAME : 'BLANK'                END_USER: 'BLANK'
|PROGRAM NAME: DSNESM68   LOCATION  : 'BLANK'                                       PCKG/COLL ID: DSNESPRR
|CONS TOKEN  : X'149EEA901A79FE48'
|STATUS     :  HOLD
|QW0172HF: X'10'
|                                                      W A I T E R
|PRIMAUTH    : AND        PLAN NAME : DISTSERV     CORR ID : javaw.exe              CONN ID : SERVER
|NETWORKID   : G998C442   LUNAME    : HF05         OWNING WORK UNIT:        223     UNIQUENESS VALUE: X'00BB46094741'
|MEMBER      : SDA1       DURATION  :   COMMIT     STATE   : EXCLUSIVE              ACE     :   3
|TRANSACTION : javaw.exe                           WS_NAME : B55Z5WHO               END_USER: and
|PROGRAM NAME: SYSSH200   LOCATION  : 'BLANK'                                       PCKG/COLL ID: NULLID
|CONS TOKEN  : X'5359534C564C3031'
|DB2S ASIC   :     5008   REQ WORK UNIT:    223    EB PTR : X'2027DFB8'             REQ FUNCTION: LOCK
|WORTH       : X'11'      QW0172WG: X'30'
|------------------------------------------------------------------------------------------------------------------------
```

C.1.4 Lock escalation trace record

Many installations go to great lengths to avoid or minimize lock escalation. However, DB2 does not produce an IFCID record when lock escalation occurs, thus making it difficult for performance monitors to report on lock escalation. You can use the existing IFCID 20 to obtain some information about lock escalation, but it is not written until commit and it is written for every thread, regardless of whether escalation occurred or not, and is therefore not really suitable for the purpose of tracking lock escalations. Information can also be derived from the locking section of IFCID 3, but again that is after the facts, and only at the plan level, as the locking section of the accounting record is at the plan level, not the package level (at least in Version 7).

As an alternative, you can trap console message DSNR031I today. It is produced whenever lock escalation occurs. However, performance monitors prefer a single interface for all performance related information through the DB2 Instrumentation Facility Component.

For this reason, DB2 Version 8 introduces a new IFCID 337 to report on lock escalations. IFCID 337 is added to the statistics trace class 3 and performance trace class 6. If activated, IFCID 337 is written whenever lock escalation occurs. The record contains a superset of the information that is reported in message DSNI031I and contains the following information:

- QW0337DB — Database ID
- QW0337OB — Page set ID or table OBID
- QW0337LS — Lock state that was escalated to (not used with Selective Partition Locking)
  - Intented share
  - Intented exclusive
  - Shared
  - Update
  - Shared intent exclusive
  - Exclusive
- QW0337LL — Type of lower level lock used where we escalate from
  - Page lock
  - Row lock
  - LOB lock
- QW0337LH — The number of lower level locks held that were released by escalation
- QW0337SN — Statement number
- QW0337ST — Waiters cached statement ID, or zero
- QW0337CI — Collection ID (in EBCDIC or Unicode)

- QW0337PK— Package name field (in EBCDIC or Unicode)

Message DSNI031I is enhanced to include the collection ID in order to be consistent with IFCID 337.

```
DSNI031I csect-name - LOCK ESCALATION HAS OCCURRED FOR RESOURCE NAME = name, LOCK STATE
= state, PLAN NAME:PACKAGE NAME = id1:id2, COLLECTION-ID = id11, STATEMENT NUMBER
= id3, CORRELATION-ID = id4, CONNECTION-ID = id5, LUW-ID = id6, THREAD-INFO =
id7:id8:id9:id10
```

C.1.5 Detect long running readers

In order to be able to detect long-running readers, DB2 V8 introduces a new DSNZPARM (LRDRTHLD — as the number of minutes an application can hold a read claim against an object before a trace record is written) and an additional type that was added to IFCID 313. IFCID 313 is written when statistics trace class 3 is active. For more information on this enhancement, see 3.18.4, "Detecting long readers" on page 208.

C.1.6 Monitor system checkpoints and log offload activity

To increase system availability, it is important to know that "warning flags" come on when important system activities that can cause outages are stalled. For example, when DB2 stops taking system checkpoints or gets stuck during its log offload activity, the system will grind to a halt. DB2 for z/OS V8 provides you with new messages, as well as an IFCID 335 record to help you to identify problems faster. An IFCID 335 record is produced if the statistics class 3 trace is active. See also 3.18.2, "Monitoring system checkpoints and log offload activity" on page 207 for more information about the console message that are produced in this case.

C.2 Accounting enhancements

There are a number of significant enhancements to DB2 accounting. First of all, a lot of work has been done to reduce the overhead associated with gathering class 2 and class 3 accounting information. Next we describe the other enhancements in more detail.

C.2.1 Accounting rollup for DDF and RRSAF

To avoid flooding the system with accounting records, DB2 V8 allows applications coming into DB2 through DDF or RRS to roll up the accounting information of individual "transactions" into a single record, that is written out at a user defined interval. This enhancement is described in more detail in 7.3, "Rollup of DB2 accounting data for DDF and RRSAF threads" on page 583.

C.2.2 Package level accounting

With this enhancement, valuable tuning information is added to the package information in the DB2 accounting records. DB2 Version 8 adds SQL, buffer pool, and locking counters to package level accounting. As before, the gathering of package level accounting level information is triggered by activating accounting trace classes 7 and 8. This is especially important for applications that come in through the network via DRDA, as they only execute packages, not plans, and the amount of accounting related information for those packages.

With this enhancement, the following additional information is gathered at the package level.

SQL counters

These are stored in a new section called QXPK. It contains information on the following SQL statements that were executed by the package:

- The number of SELECT statements
- The number of INSERT statements
- The number of UPDATE statements
- The number of DELETE statements
- The number of DESCRIBE statements
- The number of PREPARE statements
- The number of OPEN statements
- The number of CLOSE statements
- The number of FETCH statements
- The number of LOCK TABLE statements
- The number of SQL CALL statements

Buffer pool counters

The buffer pool counters at the package level are identical to those that you have in today's plan level accounting record. However, there is only one section for all buffer pools used by the package. For example, if the package did 5 getpages for objects in BP1 and 9 getpages for objects in BP2, in the package level buffer pool section you will find 14 getpages.

Note, however, that adding up the numbers in the buffer pool sections of all packages touched by a plan, may not add up to the same number that is shown at the plan level. This can be the case because DB2 does some work while the plan is running but before the package is invoked; for example, the loading of the package into the EDM pool, or work that is done via DBRMs in the plan that do not have a package.

Locking counters

The locking information at the package level is identical to the information that you have in today's plan level accounting record.

It is worth noting for those of you who write their own monitoring programs that in V8 package related accounting information is always stored in IFCID 239, an no longer as part of IFCID 3 (plan level accounting).

C.2.3 Writing accounting records with KEEPDYNAMIC(YES)

If the DB2 DSNZPARM CMTSTAT is set to INACTIVE (new default for DB2 V8), DB2 writes an accounting record when a transaction commits and the connection qualifies to become inactive. When using the KEEPDYNAMIC(YES) bind option, DB2 cannot disconnect the connection from the thread (which is what happens when the connection becomes inactive), because the thread contains information about locally cached statements. Therefore DDF threads always (have to) remain active when using KEEPDYNAMIC(YES). As a consequence accounting records are not cut at transaction boundaries. Likewise, DDF does not re-establish WLM enclaves at transaction boundaries.

Although KEEPDYNAMIC(YES) still prevents DDF threads from becoming inactive in DB2 V8, using KEEPDYNAMIC(YES) still allows DB2 to cut accounting records at transaction boundaries. That is, at commit time, when DB2 evaluates whether a DDF connection is eligible for inactivation, if the only reason why it cannot become inactive is the presence of cached dynamic SQL statements due to KEEPDYNAMIC(YES), DDF still cuts an accounting record, and also completes the WLM enclave (as if KEEPDYNAMYIC(YES) is not specified).

When a new request arrives from the client system over the connection (that is still active), a new enclave is created and a new accounting interval is started. With this new behavior, you may now consider period based WLM goals for threads that commit frequently, but cannot become inactive only because they use KEEPDYNAMI(YES). Threads that cannot become inactive for other reasons, do not reset the WLM enclave, and period based goals are probably not right for them (as was the case in the past).

This new behavior is supported for DRDA clients. DB2 for z/OS clients that use the DB2 private protocol are not affected by this change. As in V7, the presence of held cursors or declared temporary tables keeps the threads active and does not allow accounting intervals or WLM enclaves to complete.

C.3 Performance trace enhancements

In this section we discuss enhancements to IFCIDs that are related to DB2 performance trace classes.

C.3.1 Tracing the complete SQL statement text

With this enhancement, DB2 V8 delivers another long standing requirement. The previous version of DB2 limits the SQL statement text to a maximum of 5,000 bytes (the maximum length of a trace field. DB2 Version 8 provides a new IFCID 350, similar to IFCID 63 which traces the entire SQL statement, potentially in a set of trace records.

Since there is a 5,000 byte limit on a single data item being traced, a “repeating group" of 5,000 byte items are placed in each IFCID 350 record up to a maximum of 6. If that is not enough, multiple IFCID 350 records will be produced. IFCID 350 is written under the same circumstances that trigger the writing of IFCID 63. Example C-4 shows a sample output from DB2 Performance Expert beta release (still indicating that IFCID 350 is an “unknown record”, for both IFCID 63 and 350.

Example: C-4 IFCID 63 and IFCID 350

```
PRIMAUTH CONNECT   INSTANCE      END_USER       WS_NAME                      TRANSACT
ORIGAUTH CORRNAME  CONNTYPE      RECORD TIME    DESTNO ACE IFC  DESCRIPTION   DATA
PLANNAME CORRNMBR                TCB CPU TIME              ID
-------- -------- ----------- ----------------- ------ --- --- -------------- ------------------------------------------------------
BART     TSO      BAE8EF7995FC 'BLANK'          'BLANK'                      'BLANK'
BART     BART     TSO         08:03:26.50426289   1135  17  63 SQL STATEMENT  NETWORKID:  USIBMSC   LUNAME:  SCPDB8A   LUWSEQ:     1
DSNESPCS 'BLANK'                     0.01394512                              OPTIONS:  X'04' HOST LANG: N/A         SQL STATEMENT:
                                                                             SELECT TYPE, COUNT(DISTINCT(DBNAME)), COUNT(DISTINCT
                                                                             (TSNAME)), COUNT(DISTINCT(DBID)) FROM SYSIBM.SYSTABLES
                                                                             GROUP BY TYPE
                              08:03:26.50427200   1136  17 350 UNKNOWN RECORD 'BLANK'
                                     0.01394894                              NETWORKID:  USIBMSC   LUNAME:  SCPDB8A   LUWSEQ:     1
         |-------------------------------------------------------------------------------------------------------------------------
         |0000  0400C000  00000120                                                                    | ..{.....
         |.........................................................................................................................
         |0000  01224040  E2C5D3C5  C3E340E3  E8D7C56B   40C3D6E4  D5E34DC4  C9E2E3C9  D5C3E34D  | ..  SELECT TYPE, COUNT(DISTINCT(
         |0020  C4C2D5C1  D4C55D5D  6B404040  40404040   40404040  40404040  40404040  40404040  | DBNAME)),
         |0040  40404040  40404040  40404040  40404040   40404040  40404040  40C3D6E4  D5E34DC4  |                           COUNT(D
         |0060  C9E2E3C9  D5C3E34D  E3E2D5C1  D4C55D5D   6B404040  40404040  40404040  40404040  | ISTINCT(TSNAME)),
         |0080  40404040  40404040  40404040  40404040   40404040  40404040  40404040  40404040  |
         |00A0  40C3D6E4  D5E34DC4  C9E2E3C9  D5C3E34D   C4C2C9C4  5D5D4040  40404040  40404040  |  COUNT(DISTINCT(DBID))
         |00C0  40404040  40404040  40404040  40404040   40404040  40404040  40404040  40404040  |
         |00E0  C6D9D6D4  40E2E8E2  C9C2D44B  E2E8E2E3   C1C2D3C5  E240C7D9  D6E4D740  C2E840E3  | FROM SYSIBM.SYSTABLES GROUP BY T
         |0100  E8D7C540  40404040  40404040  40404040   40404040  40404040  40404040  40404040  | YPE
         |0120  4040                                                                          |
         |-------------------------------------------------------------------------------------------------------------------------
```

C.3.2 Tracing PREPARE attributes

Since DB2 Version 7, you can change the attributes of a cursor or SELECT statement on the PREPARE statement. To do so, you use the ATTRIBITES clause on the PREPARE. For example, it allows you to change a cursor's isolation level by providing a new attribute string. However, in V7 the attribute string is not always present in IFCID records, which makes problem determination and performance analysis more complex.

V8 will add the attribute string to the following trace records:

- IFCID 63 (which contains the first 5000 bytes of the SQL statement text) will now also contain the attribute string on a short prepare.
- IFCID 124 (which is used to obtain information about the currently executing SQL statement through a READS request) will now also contain the attributes string.
- IFCID 317 (which is used to retrieve the SQL statement string of a statement in the dynamic statement cache) will now also contain the attributes string.

C.3.3 Add cached statement ID to IFCID 124

As mentioned before, IFCID 124 provides information about the currently executing SQL statement through the READS interface. IFCID 124 has been enhanced in DB2 Version 8 and now also includes the dynamic cached SQL statement ID. The 4 byte field QW0124ST was added to contain the cached dynamic SQL statement identifier.

This enhancement will work nicely with another enhancement in DB2 V8 that allows you to explain SQL statements from the dynamic statement cache. You can retrieve the information about the currently executing SQL statement via IFCID 124. That now contains the SQL statement ID of the statement in the dynamic statement cache, and then explain the SQL statement from the dynamic statement cache using that statement ID to have a look at the access path of the currently executing SQL statement.

For more information on how to explain a statement from the dynamic statement cache, see 4.20, "EXPLAIN STMTCACHE" on page 368.

C.3.4 Agent level workfile tracing

Another long standing requirement that is satisfied in DB2 Version 8 is related to the ability to monitor space usage in the workfile and TEMP database at the application. This allows you to take corrective action, such as adding space or cancelling run away query that is using lot of space in these databases, and preventing other queries from running successfully even they only need very little space.

DB2 Version 8 introduces a new IFCID 342 to record workfile and TEMP database usage at an application level.

When active, IFCID 0342 is written whenever space is allocated or deallocated in the workfile or TEMP database. It gives information about the total amount of space being used by the agent currently and the maximum space that agent may have used at any given point in time in the workfile or TEMP database. The trace also records total amount of space being used for indexes on Declared Temporary Tables.

This can be a very interesting IFCID for monitors to trigger exceptions, for example, to catch runaway queries before the monopolize the system and cause other applications to fail, because the workfile database and TEMP database are shared amongst all DB2 users.

C.3.5 Obtaining secondary authorization ID information

This enhancement allows applications to obtain the authorization identifiers that are associated with an application (such as QMF) that is executing. The primary authorization ID, the CURRENT SQLID, and secondary auth IDs can be retrieved with a synchronous read of IFCID 234. This enhancement is also available in DB2 V6 and V7 via APAR PQ47973, PTF UQ57178 for V7 and PTF UQ57177 for V7.

IFCID 0234 is always started implicitly by DB2 at startup time. It is not contained in any DB2 trace class and cannot be specified as an IFCID value on the -START TRACE command.

C.3.6 Auditing for multi-level security

Audit record (IFCID 142) is produced if the table with security label is created, altered or dropped.

The user must be identified to Security Server with a valid seclabel. If not, an authorization error and audit record are produced (IFCID 0140).

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 1055. Note that some of the documents referenced here may be available in softcopy only:

- *DB2 for OS/390 Version 5 Performance Topics*, SG24-2213
- *DB2 for OS/390 Version 6 Performance Topics*, SG24-5351
- *DB2 for z/OS and OS/390 Version 7 Utilities Suite*, SG24-6289
- *DB2 for z/OS and OS/390 : Squeezing the Most Out of Dynamic SQL*, SG24-6418
- *DB2 for z/OS and OS/390: Ready for Java*, SG24 6435
- *DB2 UDB for z/OS Version 8 Performance Topics*, SG24-6465
- *MLS / DB2 Implementation*, SG24-6480
- *DB2 for z/OS Stored Procedures: Through the CALL and Beyond*, SG24-7083

Other publications

These publications are also relevant as further information sources:

- *z/OS ICSF Administrator's Guide*, SA22-7521
- *Managed System Infrastructure for Setup User’s Guide*, SC33-7985
- *z/OS Planning for Multilevel Security*, GA22-7509
- *Enterprise COBOL for z/OS Compiler and runtime migration guide*, GC27-1409

DB2 UDB for z/OS Version 8 manuals

- *DB2 Administration Guide,* SC18-7413
- *DB2 Application Programming and SQL Guide,* SC18-7415
- *DB2 Application Programming Guide and Reference for Java,* SC18-7414
- *DB2 Command Reference,* SC18-7416
- *DB2 Data Sharing: Planning and Administration,* SC18-7417
- *DB2 Diagnosis Guide and Reference,* LY37-3201
- *DB2 Diagnostic Quick Reference Card,* LY37-3202
- *DB2 Image, Audio, and Video Extenders Administration and Programming,* SC18-7429
- *DB2 Installation Guide,* GC18-7418
- *DB2 Messages and Codes,* GC18-7422
- *DB2 ODBC Guide and Reference,* SC18-7423

- *An Introduction to DB2 Universal Database for z/OS,* SC18-7419
- *DB2 Program Directory,* GI10-8566
- *DB2 UDB for z/OS RACF Access Control Module Guide,* SC18-7433
- *DB2 Reference for Remote DRDA Requesters and Servers,* SC18-7424
- *DB2 Reference Summary,* SX26-3853
- *DB2 Release Planning Guide,* SC18-7425
- *DB2 SQL Reference,* SC18-7426
- *DB2 Text Extender Administration and Programming,* SC18-7430
- *DB2 Utility Guide and Reference,* SC18-7427
- *DB2 What's New?,* GC18-7428
- *DB2 XML Extender for z/OS Administration and Programming,* SC18-7431

Program directories for DB2 UDB for z/OS Version 8

- *DB2 UDB for z/OS Version 8,* GI10-8566
- *DB2 UDB for z/OS Version 8 - Application Connectivity,* GI10-8585
- *DB2 UDB for z/OS Version 8 - DB2 Management Clients Package*, GI10-8562
- *DB2 UDB for z/OS Net.Data Version 7*, GI10-8565
- *DB2 Net Search Extender Version 8 for DB2 for z/OS*, GI10-8582

DB2 Query Management Facility (QMF) Version 8 manuals

- *Developing QMF Applications*, SC18-7651
- *Getting Started with QMF for Windows and QMF for Websphere*, SC18-7449
- *Installing and Managing QMF for TSO/CICS*, GC18-7444
- *Installing and Managing QMF for Windows and QMF for Websphere*, GC18-7448
- *Introducing QMF*, GC18-7443
- *QMF High Performance Option User's Guide for TSO/CICS*, SC18-7450
- *QMF Messages and Codes*, GC18-7447
- *QMF Reference*, SC18-7446
- *Using QMF*, SC18-7445
- *Visionary Studio Developer's Guide*, SC18-9093
- *Visionary Studio Getting Started Guide*, GC18-9092

Program Directories for QMF Version 8

- *QMF Classic Edition Version 8*, GI10-8517
- *QMF Enterprise Edition Version 8*, GI10-8519
- *QMF Classic Edition Version 8 National Language Version*, GI10-8562
- *QMF Distributed Edition Version 8 National Language Version,* GI10-8518
- *QMF Enterprise Edition Version 8 National Language Version*, GI10-8564

IBM Press books

- *DB2 SQL Procedural Language for Linux, UNIX, and Windows*, ISBN 0-13-100772-6

Online resources

These Web sites and URLs are also relevant as further information sources:

- DB2 UDB for z/OS Internationalization Guide

  http://www.ibm.com/software/data/db2/zos/pdf/ccmstr.pdf

- z/OS Managed System Infrastructure for Setup DB2 Customization Center User's Guide

  ftp://ftp.software.ibm.com/software/db2storedprocedure/db2zos390/techdocs/msysdb2cc.pdf

- DB2 Development Center — The Next-Generation AD Tooling for DB2

  http://www7b.software.ibm.com/dmdd/library/techarticle/0207alazzawe/0207alazzawe.html

- DB2 Integrated SQL Debugger IBM DB2 Stored Procedure Builder V7.2

  http://www7b.software.ibm.com/dmdd/library/techarticle/alazzawe/0108alazzawe.html

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

B

D

M

P

Q

R

S

V

Z

IBM

Redbooks

DB2 UDB for z/OS Version 8: Everything You Ever Wanted to Know, ... and More

(2.0” spine)
2.0” <-> 2.498”
1052 <-> 1314 pages

DB2 UDB for z/OS Version 8: Everything You Ever Wanted to Know, ... and More

Detailed description of all Version 8 enhancements

Release planning and migration information

Presentation guide style for easy reference

IBM DATABASE 2 Universal Database Server for z/OS Version 8 is the twelfth and largest release of DB2 for MVS. It brings new synergy with the zSeries hardware and is the first major subsystem to exploit the z/OS 64-bit virtual addressing capabilities. DB2 V8 offers data support, application development, and query functionality enhancements for e-business, while building upon the traditional characteristics of availability, exceptional scalability, and performance for the enterprise of choice.

DB2 Version 8 has been re-engineered for e-business, with many fundamental changes in architecture and structure. Key improvements enhance scalability, application porting, security, and continuous availability. Management for very large databases is made much easier, while 64-bit virtual storage support makes management simpler and improves scalability and availability. This new version breaks through many old limitations in the definition of DB2 objects, including SQL improvements, schema evolution, longer names for tables and columns, longer SQL statements, enhanced Java and Unicode support, enhanced utilities, and many more advantages.

This IBM Redbook introduces the many enhancements made available with DB2 V8. It will help you understand the functions offered by DB2 V8, and provides information to help you to evaluate their applicability to your environment, as well as to plan for the installation of DB2 V8 or the migration from DB2 V7.

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

SG24-6079-00 ISBN 0738497738