

DB2 UDB for z/OS Version 8 Technical Preview

Browse the functional contents of the largest release ever

Understand the prerequisites and the setup for the new functions

Start planning for a smooth migration



Paolo Bruni
Bart Steegmans
Rafael Garcia
Sabine Kaschta
Ravi Kumar



International Technical Support Organization

DB2 UDB for z/OS Version 8 Technical Preview

April 2003

Archived

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

First Edition (April 2003)

This edition applies to Version 8 of DB2 for z/OS (program number 5625-DB2) and Version 8 of DB2 Utilities Suite for z/OS (program number 5655-K63) for use with z/OS Version 1.3 and later versions.

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

© Copyright International Business Machines Corporation 2003. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Figures	ix
Tables	xiii
Examples	xv
Notices	xvii
Trademarks	xviii
Preface	xix
The team that wrote this redbook	xix
Become a published author	xxi
Comments welcome	xxi
Chapter 1. Introduction	1
1.1 The evolution of DB2 UDB for z/OS	2
1.2 Summary of features	3
1.2.1 New DB2 function: msys for set-up DB2 Customization Center	4
1.2.2 No-extra-charge features	4
1.2.3 Charge features	7
1.3 This redbook	7
Chapter 2. DB2 UDB for z/OS V8 at a glance	9
2.1 Scalability	10
2.2 Availability	10
2.3 SQL	12
2.4 e-business	14
2.5 Utilities	16
2.6 Performance	17
2.7 Data sharing	19
2.8 Installation and migration	19
Chapter 3. Scalability	21
3.1 The 64-bit architecture support	22
3.1.1 The z/OS architecture	22
3.1.2 DB2's virtual storage expansion	25
3.1.3 Moving above the 2 GB bar	26
3.1.4 General performance expectations	31
3.1.5 Storage monitoring and tuning	32
3.1.6 DB2 V8 requires z/Architecture and z/OS V1R3	33
3.2 More partitions	33
3.3 More tables in join	35
3.4 More log data sets	36
Chapter 4. Availability	39
4.1 DB2 V8 partition and index related terminology	40
4.1.1 Table-controlled partitioning	40
4.1.2 Index-controlled partitioning terminology	42
4.1.3 Table-controlled partitioning terminology	43
4.2 Data partitioned secondary indexes	46

4.2.1	Creating a data partitioned secondary index	46
4.2.2	The need for DPSIs	48
4.2.3	DPSI considerations	49
4.3	Online schema changes	50
4.3.1	Online schema changes overview	50
4.3.2	Table data type changes	52
4.3.3	Index changes	55
4.3.4	Versioning	56
4.3.5	Usage considerations	59
4.3.6	New DBET states for online schema changes	59
4.3.7	Impact of online schema changes on user tasks	59
4.3.8	Dynamic partitions	60
4.4	System level point-in-time recovery	62
4.4.1	Backing up the system	63
4.4.2	Restoring the system	64
4.5	Online ZPARMs	67
4.6	Other availability enhancements	69
4.6.1	Control intervals larger than 4 KB	69
4.6.2	Monitor system checkpoints and log offload activity	69
4.6.3	Log monitor long running UR backout	69
4.6.4	Improved LPL recovery	70
Chapter 5.	SQL	71
5.1	Long names	72
5.2	SQL statements 2 MB long	73
5.3	Dynamic scrollable cursors	75
5.3.1	Cursor positioning and serialization	79
5.3.2	Considerations	79
5.4	Common table expressions and recursive SQL	80
5.4.1	Example of fullselect	80
5.4.2	Example with CREATE VIEW and INSERT	82
5.4.3	Recursive SQL	82
5.5	Multi-row fetch and insert	84
5.5.1	DECLARE CURSOR	84
5.5.2	FETCH	85
5.5.3	INSERT	87
5.6	Get diagnostics	90
5.7	Scalar fullselect	94
5.7.1	Functional description	95
5.7.2	Restrictions	98
5.8	Select from insert	98
5.8.1	Functional description	98
5.9	Qualified column names in INSERT and UPDATE	103
5.10	Expressions in GROUP BY	104
5.11	Multiple DISTINCT	106
5.12	Sequences	107
5.12.1	Usage considerations	110
5.12.2	Using sequences in applications	114
5.13	Identity columns enhancements	115
5.13.1	SQL statements for identity column enhancements	115
5.14	Sequences and identity columns comparison	117
5.15	Multilevel security	119
5.16	MQSeries UDFs	123

5.17 ASCII flag for compile	126
Chapter 6. e-business	129
6.1 IBM DB2 Universal Driver for SQLJ and JDBC features	130
6.1.1 IBM JDBC Type 4 driver	131
6.1.2 New IBM JDBC Type 2 driver	132
6.1.3 Java API enhancements	132
6.1.4 SQLJ	133
6.1.5 Nested stored procedure result sets for JDBC and ODBC applications	135
6.1.6 Extended DESCRIBE	135
6.1.7 SQLcancel	136
6.1.8 LOB streaming	136
6.2 Unicode support	136
6.2.1 Unicode parser	137
6.2.2 Program preparation with new Unicode precompiler	138
6.2.3 Utility Unicode parser	140
6.2.4 Multiple CCSIDs per SQL statement	140
6.2.5 ODBC Unicode support	142
6.2.6 Unicode and distributed support	142
6.3 ODBC enhancements	143
6.3.1 ODBC SQLConnect user and password support	143
6.3.2 ODBC Unicode support	143
6.3.3 Cursor extensions	144
6.3.4 SQLCancel support	144
6.4 XML publishing functions	144
6.5 CURRENT PACKAGE PATH special register	153
6.6 DDF communication database enhancements	154
6.6.1 Requester database ALIAS	155
6.6.2 Server location alias	156
6.6.3 Member routing in a TCP/IP network	157
6.7 Enhancements for stored procedures and UDFs	159
6.7.1 Maximum failures	159
6.7.2 Exploit WLM server task thread management	160
6.7.3 Enhancements to SQL stored procedure language	160
6.7.4 COMPJAVA stored procedures no longer supported	164
6.7.5 DB2 established stored procedures	164
6.8 Miscellaneous enhancements	165
6.8.1 RRSAP compatibility for CAF applications	165
6.8.2 Roll up accounting data for DDF and RRSAP threads	166
6.8.3 Improved query and result set processing	167
6.8.4 Time out for SNA allocate conversation requests	167
6.8.5 Data stream encryption	167
6.8.6 DISPLAY LOCATION command	168
Chapter 7. Utilities	169
7.1 Online schema changes support	170
7.1.1 More flexibility with partitions	170
7.1.2 Utility support for schema evolution	178
7.1.3 Point-in-time recovery restrictions	186
7.2 Delimited LOAD and UNLOAD	189
7.3 Unicode	194
7.4 Distribution statistics	196
7.4.1 Collecting cardinality and distribution statistics	198

7.4.2	Collecting column correlation statistics	198
7.4.3	Use of work data sets	199
7.4.4	Examples	199
7.5	Backing up and restoring the system	200
7.6	Other changes.	202
7.6.1	New default RESTART	202
7.6.2	New defaults SORTDATA and SORTKEYS	202
7.6.3	COPY and RECOVER tape parallelism	203
Chapter 8.	Performance	205
8.1	Comparing unlike data types.	206
8.2	Materialized query tables	209
8.2.1	Creating an MQT	210
8.2.2	Populating and maintaining an MQT.	212
8.2.3	Automatic query rewrite using Materialized Query Tables	213
8.2.4	Determining if query rewrite occurred	220
8.3	Multi-row INSERT and FETCH	220
8.4	Cost based parallel sort	221
8.5	Data caching and sparse index usage for star join	221
8.6	Long and variable length keys	225
8.7	Support for backward index scan	228
8.8	Trigger enhancements	229
8.9	Reduced lock contention on volatile tables	229
8.10	Table UDF cardinality option and block fetch	230
8.10.1	Table UDF cardinality clause	230
8.10.2	Table UDF block fetch	232
Chapter 9.	Data sharing	235
9.1	CF lock propagation reduction	236
9.2	Reduction of overhead costs for data sharing workloads	237
9.3	Batched updates for index page splits	237
9.4	Improved LPL recovery	238
9.5	Resolution of indoubt units of recovery in restart light	238
9.6	Change to IMMEDIATE BIND option default.	238
9.7	Change to -DISPLAY GROUPBUFFERPOOL output.	239
Chapter 10.	Installation and migration	241
10.1	Currency of versions and migration paths.	242
10.2	Major changes to installation and migration	243
10.2.1	Before migrating	244
10.3	Installation.	246
10.3.1	Major changes to install jobs.	247
10.4	Migration	248
10.4.1	Compatibility Mode	250
10.4.2	Enabling New Function Mode	251
10.4.3	ENMF jobs	254
10.4.4	New Function Mode	256
10.5	Catalog changes	259
10.6	msys for Setup DB2 Customization Center.	259
10.7	Samples	261
Appendix A.	Unicode definitions	263
A.1	Basic conversions	264
A.2	Additional conversions	264

Related publications	267
IBM Redbooks	267
Other resources	267
Referenced Web sites	268
How to get IBM Redbooks	269
IBM Redbooks collections	269
Abbreviations and acronyms	271
Index	273

Archived

Archived

Figures

1-1	DB2 features	4
3-1	The challenge	22
3-2	The 64-bit memory architecture	23
3-3	64-bit virtual address space mapping	24
3-4	Summary of z/OS current versions	25
3-5	Data spaces	27
3-6	The new buffer pool values	29
3-7	RID pool	29
3-8	EDM pool	31
3-9	New number of partitions	34
3-10	Logs data sets increase	37
4-1	Table-controlled partitioning	40
4-2	Index-controlled partitioning terminology — 1	42
4-3	Index-controlled partitioning terminology — 2	43
4-4	Table-controlled partitioning terminology	43
4-5	Partitioned versus non-partitioned index	44
4-6	Partitioning index	45
4-7	Clustering index	45
4-8	Secondary index used as clustering index	46
4-9	Creating a partitioned index	47
4-10	Data partitioned secondary index layout	47
4-11	Online schema evolution	51
4-12	ALTER TABLE SET DATATYPE statement	53
4-13	ALTER INDEX statement	55
4-14	Versioning information in the DB2 catalog	58
4-15	Backup system operations	64
4-16	Restore system operation	66
5-1	Scrollable cursors in DB2 V7	75
5-2	Sensitive and insensitive cursors with DB2 V7	76
5-3	Scrollable DECLARE CURSOR syntax	77
5-4	Cursor types comparison	78
5-5	Common table expression in fullselect	81
5-6	Recursive SQL	83
5-7	Multirow DECLARE CURSOR syntax	84
5-8	Multirow FETCH syntax	85
5-9	Multirow INSERT syntax	88
5-10	GET DIAGNOSTICS statement	91
5-11	GET DIAGNOSTICS syntax	91
5-12	Scalar fullselect — Extension to expression	94
5-13	Scalar fullselect — Extension to CASE expression	95
5-14	Tables used in the scalar fullselect examples	96
5-15	Scalar fullselect — Example of WHERE clause	96
5-16	Scalar fullselect — Example of nesting in a SELECT list	97
5-17	Scalar fullselect — Example of CASE expression	98
5-18	SELECT FROM INSERT — Table specification syntax changes	99
5-19	SELECT FROM INSERT — Order by clause syntax changes	99
5-20	SELECT FROM INSERT — INSERT trigger	101
5-21	SELECT FROM INSERT — Effect of updates and deletes against result table	102

5-22	SELECT FROM INSERT — Ordering sequence example	103
5-23	Sequences — CREATE SEQUENCE statement	108
5-24	Sequences — ALTER SEQUENCE statement	109
5-25	Identity columns — altering attributes	116
5-26	Sequences and identity columns	118
5-27	Multilevel security hierarchy	120
5-28	Row granularity with seclabel	121
5-29	Basic DB2/MQSeries configuration	124
5-30	MQSeries UDF environment	126
6-1	Existing SQLJ preparation process	133
6-2	Universal Client SQLJ preparation process	134
6-3	Nested stored procedure result sets	135
6-4	Program preparation using the NEWFUN keyword	139
6-5	SELECT statement using multiple CCSIDs	142
6-6	Relational data displayed in HTML format	145
6-7	XML2CLOB syntax diagram	147
6-8	XMLELEMENT syntax diagram	147
6-9	XMLFOREST syntax diagram	149
6-10	XMLCONCAT syntax diagram	151
6-11	XMLAGG syntax diagram	152
6-12	Access LUW database without DBALIAS	155
6-13	Access LUW database with DBALIAS	156
6-14	Location alias name	157
6-15	DDF communication record	157
6-16	Member routing in a TCP/IP network	158
6-17	Stored procedure and UDF enhanced failure handling syntax	159
6-18	SIGNAL statement syntax diagram	162
6-19	SIGNAL used in condition handler?	162
6-20	RESIGNAL statement syntax diagram	163
7-1	Online schema — Table space with 59 partitions	170
7-2	Online schema — ALTER TABLE ADD PARTITION syntax	171
7-3	Online schema — Table space after adding a partition	172
7-4	Online schema — Rotate partition overview	173
7-5	Online schema — Rotate partition syntax	174
7-6	Online schema — Rotate partition example	175
7-7	Online schema — Alter partition boundary syntax	176
7-8	Online schema — Alter partition boundary example	176
7-9	Example of REORG TABLESPACE to rebalance partitions	179
7-10	Example of index-based partitioning	181
7-11	A proposed solution to eliminate BUILD2 phase with online REORG	182
7-12	REPAIR VERSIONS syntax	185
7-13	Online schema — Recovery of table space to a point-in-time	187
7-14	Online schema — Recovery of index to a point-in-time	188
7-15	LOAD delimited input syntax	190
7-16	UNLOAD delimited output syntax	192
7-17	DSNUTILU definition	195
7-18	RUNSTATS syntax changes	197
7-19	RUNSTATS — Distribution statistics and key correlation statistics blocks	197
8-1	Mismatched operands — Numeric type	208
8-2	Mismatched operands — Transitive closure	209
8-3	CREATE MQT syntax	210
8-4	Credit card application schema	217
8-5	Decision between workfile caching or sparse index	222

8-6	Data caching in outside-in join phase	223
8-7	Work file sorts prior to sparse index enhancement	223
8-8	Benefits of using a sparse index on workfiles used in a star join plan	224
8-9	Example of star join plan	225
8-10	Table UDF cardinality clause	231
8-11	Predicate using table UDF indexable and stage 1	233
10-1	Currency of DB2 versions	242
10-2	Possible migration paths	243
10-3	Install through TSO or msys for Setup	247
10-4	Modes of operation	249
10-5	DSNTIPA1 — Main panel for ENFM	251
10-6	DSNTIPT — Choose a new name for SDSNSAMP library	252
10-7	DSNTIP00 — Shadow data set allocation	253
10-8	DSNTIP01 — Image copy data set allocations	254
10-9	DISPLAY GROUP command	256
10-10	Checking BSDS conversion	257
10-11	DSNJCNVB sample JCL	257
10-12	DSNJCNVB SYSPRINT	258
10-13	DSNJU004 output indicating new BSDS structure	258
10-14	msys description	260

Archived

Tables

3-1	Maximum number of partitions versus DSSIZE and page size	34
4-1	New subsystem parameters changeable online with DB2 V8	68
5-1	SQL Identifier length limits	72
5-2	Data values for :hva1 and :hva2	93
6-1	Query result — Simple usage of XMLELEMENT and XML2CLOB	148
6-2	Query result — Nested elements	148
6-3	Query result — XMLATTRIBUTES	149
6-4	Query result — XMLFOREST	150
6-5	Query result — XMLCONCAT	151
6-6	Query result — XMLAGG	153
7-1	RECOVER TABLESPACE PIT actions	187
7-2	RECOVER INDEXSPACE PIT actions	189
7-3	Acceptable data type forms for the delimited file format	194
10-1	Evolution of the DB2 catalog	250

Archived

Examples

3-1	New data set naming convention	35
4-1	Index-controlled vs. table-controlled partitioning syntax	41
4-2	Converting from index-controlled to table-controlled partitioning	41
5-1	C program using CLOB for host-variable on EXECUTE IMMEDIATE statement . . .	73
5-2	COBOL program using DBCLOB for host-variable on PREPARE statement	74
5-3	Examples of using scrollable cursors	78
5-4	Common table expression in SELECT	81
5-5	Common table expression in CREATE VIEW or INSERT	82
5-6	Multirow DECLARE CURSOR	84
5-7	FETCH examples	86
5-8	Example 1 of INSERT	90
5-9	Example 2 of INSERT	90
5-10	Expressions in GROUP BY	105
5-11	Multiple COUNT(DISTINCT)	106
5-12	Creating and using a sequence	114
5-13	Sample program with ASCII option	127
6-1	Multiple CCSID SQL statement — 1	141
6-2	Multiple CCSID SQL statement — 2	141
6-3	Multiple CCSID SQL statement — 3	141
6-4	Complex query using XML publishing functions	145
6-5	XMLELEMENT and XML2CLOB usage	148
6-6	Nested elements	148
6-7	Using the XMLATTRIBUTES function	149
6-8	Using XMLFOREST	150
6-9	Alternative query not using XMLFOREST	150
6-10	Using the XMLCONCAT function	151
6-11	Using the XMLAGG function	152
6-12	Using the RETURN statement	161
6-13	Using the SIGNAL statement	163
6-14	Using the RESIGNAL statement	164
7-1	Displaying a four-partition table space	177
7-2	Displaying partitions added with DB2 V8	177
7-3	Displaying ranges of partitions with DB2 V8	178
7-4	Displaying indexes after ALTER	178
7-5	Sample DDL for index partitioned table space	180
7-6	Avoiding BUILD2 phase	181
7-7	Sample LOAD job with delimited input	190
7-8	Sample UNLOAD job with delimited output	192
7-9	Distribution statistics — Example 1	199
7-10	Distribution statistics — Example 2	199
7-11	Distribution statistics — Example 3	200
7-12	Distribution statistics — Example 4	200
7-13	Distribution statistics — Example 5	200
8-1	Sample SELECT statement	206
8-2	Employee table definition	208
8-3	Sample create of a materialized query table	211
8-4	Converting a base table into an MQT	211
8-5	Sample REFRESH TABLE statement	212

8-6	Creating a informational RI constraint	215
8-7	UserQ1.	217
8-8	MQT TRANSCNT	217
8-9	NewQ1.	218
8-10	UserQ2.	218
8-11	TRANSIAB MQT	218
8-12	NewQ2.	218
8-13	UserQ3.	219
8-14	TRANSAVG MQT	219
8-15	NewQ3.	220
8-16	Sample SQL statement	221
8-17	Create a NOT PADDED index.	226
8-18	Create a PADDED index	226
8-19	Comparing non-padded index entries	227
8-20	Alter an index to NOT PADDED	227
8-21	Alter an index to PADDED.	227
8-22	Conditional after trigger	229
8-23	Using the CARDINALITY MULTIPLIER clause in a query.	231
8-24	Using the CARDINALITY clause instead.	232

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:

Affinity™
AIX®
AS/400®
BookMaster®
CICS®
CT™
DB2 Connect™
DB2 Extenders™
DB2®
DFS™
DFSMSshm™
DFSORT™
DRDA®
Enterprise Storage Server™
eServer™
FlashCopy®
Footprint®

GDDM®
Informix®
IBM®
IMS™
iSeries™
Lotus®
MQSeries®
MVS™
Net.Data®
OS/2®
OS/390®
Parallel Sysplex®
Perform™
PAL®
QBIC®
QMFT™
Redbooks™

Redbooks(logo)™ 
RACF®
RAMAC®
RETAIN®
RMF™
S/390®
SecureWay®
System/390®
SOM®
SP™
VTAM®
WebSphere®
Word Pro®
XT™
z/Architecture™
z/OS™
zSeries™

The following terms are trademarks of other companies:

ActionMedia, LANDesk, MMX, Pentium and ProShare 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.

C-bus is a trademark of Corollary, 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.

SET, SET Secure Electronic Transaction, and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.

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 IBM Redbook) is the twelfth and largest release of DB2 for MVS. It brings synergy with the zSeries hardware and exploits 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 major changes and enhancements made available with DB2 V8. It will help you understand the functions offered by DB2 V8, and provides enough information to start evaluating their applicability to your environment, as well as to start planning for the installation of DB2 V8 or the migration from DB2 V7.

The team that wrote this redbook

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

Paolo Bruni is a certified Consultant IT Architect working as a Data Management Project Leader at the International Technical Support Organization, San Jose Center since 1998. In this capacity he has authored several redbooks on DB2 for OS/390 and DM tools, and has conducted workshops and seminars worldwide. During Paolo's many years with IBM, in development, and in the field, his work has been mostly related to database systems.

Bart Steegmans is a DB2 Product Support Specialist from IBM Belgium on assignment as a Data Management for z/OS Specialist at the International Technical Support Organization, San Jose Center. He has over 12 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 6 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 Global Learning Services in Germany as an education consultant. She has 12 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 the insurance industry 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 Learning Services, Australia. He has over 17 years of experience in DB2. He was on assignment at the International Technical Support Organization, 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 the EMEA Development Team, IBM Learning Services.

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

Emma Jacobs
Yvonne Lyon
Deanna Polm
International Technical Support Organization, San Jose Center

Rich Conway
Bob Haimowitz
International Technical Support Organization, Poughkeepsie Center

Meg Bernal
Mengchu Cai
Gayathiri Chandran
Jason Cu
Margaret Dong
Gene Fu
Shivram Ganduri
John Garth
Michelle Guo
Muniza Hasan
Keith Howell
Ming Hu
Koshy John
Le Kha
Heather Lamb
John Lawler
Li-Mey Lee
Dave Levis
Phyllis Marlino
Bruce McAlister
Claire McFeely
Roger Miller
David Moy
Paul Ostler
Mary Petras
Jim Pizor
Mike Shaddock
Sampanna Shanbbhag
Akira Shibamiya
Kalpana Shyam
John Tobler
Yoichi Tsuji

Yumi Tsuji
Grace Tzeng
Koko Yamaguchi
Devon Yu/
Kathy Zagelow
Ruiming Zhou
IBM Silicon Valley Laboratory

Judy Ruby-Brown
IBM Americas Advanced Technical Support

Ian Cook
Sarah Ellis
IBM EMEA Product Introduction Centre

Karen Galloway
Mark Wilson
IBM Americas Product Introduction Center

Samson Tai
IBM AP Product Introduction Center

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

Archived



Introduction

In this chapter we give a brief introduction to DB2 UDB for z/OS Version 8 and describe the main features and the packaging of the product. We then discuss the contents of this redbook.

Note: The reader should bear in mind that DB2 V8 is currently still at the stage of a work in progress. This redbook is a preview of DB2 V8 main functions, and, while accurate at the time of writing, the chances are that it will progressively become less correct as time goes by, until general availability. This is due to normal development detailed changes, and corrections and improvements originating from the experience of early users during the introduction program.

For the most up-to-date version of this redbook, or any other follow-up redbook that replaces it, check the following Web site:

ibm.com/redbooks

1.1 The evolution of DB2 UDB for z/OS

DB2 UDB for z/OS Version 8 makes a fundamental change in many areas, re-engineering much of DB2. Key improvements enhance scalability, application porting, and continuous availability. Management for very large databases is much easier. Support for key vendor applications is compelling. The enhancements for 64-bit virtual storage make management simpler, and improve scalability and availability. This new version breaks through many old DB2 limitations.

DB2 is going through a renaissance and a re-engineering process. For example, the degree of improvement for indexes is similar to the difference between type 1 and type 2 indexes. DB2 is able to use indexes more effectively and in cases that it could not before. DB2 V8 reduces the space in variable-length indexes, is able to have index-only access with variable-length data, and uses the index when the predicates do not match exactly. Partitioning and clustering are independent. This change is important because you will be able to partition without an index and be able to cluster on any index. It means in some cases tables can have one less index and reduce the occurrences of “death by random I/O” using faster sequential processing. This will improve INSERT, DELETE, LOAD, REORG, and UPDATE processing if the index can be removed.

DB2 V8 has more flexibility in indexes with longer index keys, up to 2000 bytes. Longer indexes are needed for Unicode and for longer names. Unicode will be used much more in the future. All platforms are moving to Unicode: Windows, UNIX, Linux and z/OS. It is not a question of *whether* you are moving to Unicode, it is *when*. If you use Java, you are running with Unicode now. Version 7 of DB2 UDB for z/OS and OS/390 delivered support for Unicode-encoded data. You can easily store multilingual data within the same table or on the same DB2 subsystem. DB2 V8 makes Unicode support more flexible. SQL statements and literals are able to be Unicode or EBCDIC. Unicode tables can join with an EBCDIC table. Many of the DB2 catalog character columns are converted to Unicode, so Unicode will be used by everyone.

Improving continuous availability is another key area for our work. A huge enhancement in this area is schema evolution. For some changes to tables or columns today, our users have to drop an object and recreate it. The process can be lengthy and error-prone. In DB2 V8, you can make more changes using an alter in a fraction of a second. For example, you can extend numeric and character columns, you can change between CHAR and VARCHAR. You can add a new partition to an existing partitioned table space. You can rotate the partitions, which will allow you to do such things as keep the most current 36 months of data.

You will find that the SQL consistency across the DB2 family has improved substantially in the past few versions, while significant new function has been added. The *IBM DB2 Universal Database SQL Reference for Cross-Platform Development* defines IBM DB2 Universal Database Structured Query Language (DB2 UDB SQL). It is intended for programmers who want to write portable applications using SQL that is common to the DB2 UDB relational database products and the SQL 1999 Core standard.

The book describes the rules and limits for preparing portable programs. Version 1 of the book showed the dramatic improvements from DB2 V5, 6 and 7. Version 1.1 reflects the common SQL available on DB2 UDB for z/OS V7, iSeries V5R2, and Linux, UNIX, and Windows V8. DB2 V8 breaks many more barriers to DB2 family compatibility and application portability with extensive improvements in the SQL language that include materialized query tables, insert within a SELECT statement, sequences, improvements in identity columns, long names, long statements, multi-row fetch and insert, dynamic scrollable cursors and the ability to group by an expression. We look forward to the next version of the *Reference* reflecting these further SQL enhancements.

The biggest impact of the z/Architecture on DB2 is the ability to have large real memory support. Prior to the zSeries, customers were limited to 2 GB real storage due to the 31-bit addressing of the S/390 architecture. The real storage limit of 2 GB is a leading performance inhibitor for many high end customers. Another performance inhibitor is the 2 GB virtual storage limit for the main DB2 (DBM1) address space. DB2 V3 provided hiper pools to offer some relief, but many customers need more. Version 6 allowed customers to use large real storage by moving to buffer pools in data spaces.

DB2 V8 delivers 64-bit virtual storage addressing. Instead of hiper spaces or data spaces, the single large address space can allow easier management of storage as we transition from multiple 31-bit address spaces to a few 64-bit address spaces, improving both availability and scalability. Think about it! In a single address space, we have addressability up to 16 exabytes. Right now, we do not know anyone who has that big a database. But DB2 will be there when there are. See the *64-bit Virtual Storage Roadmap*, GM13-0076-01, which was updated in June 2002, available from:

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

Migration to DB2 V8 is only from DB2 V7, running on z/OS Version 1 Release 3 or higher, and running on a zSeries processor. If you want to migrate from V6 to this one, then migrate first to V7, zSeries, and z/OS V1R3 or later.

The most exciting feature of DB2 V8 depends upon you and what you want most. This version breaks through many limits, and our many customers face many different limits. Scalability, continuous availability, cross platform compatibility, and usability are some of the many “ities” that are improved. Anyone who is interested in ERM products such as SAP, PeopleSoft, and Siebel should understand that most of these enhancements will help with those large applications. The larger memory, Unicode, and SQL flexibility improvements are at the top of many vendor lists. Longer table and column names, multi-row fetch and insert and raising many other limitations will help DB2’s application vendors.

The biggest problem with DB2 V8 is that there is so much to describe. This redbook provides a good preview, but you will find that this version has more to offer.

More details and more accurate information will be provided in follow-on redbooks.

1.2 Summary of features

The contents of this section reflect the January 28, 2003 Software Announcements of DB2 UDB for z/OS V8, and the DB2 Utilities Programs. Refer to the DB2 for z/OS Web site for these announcements and any updated information:

<http://ibm.com/software/data/db2/os390/>

In Figure 1-1 we summarize the features of DB2 UDB for z/OS Version 8.

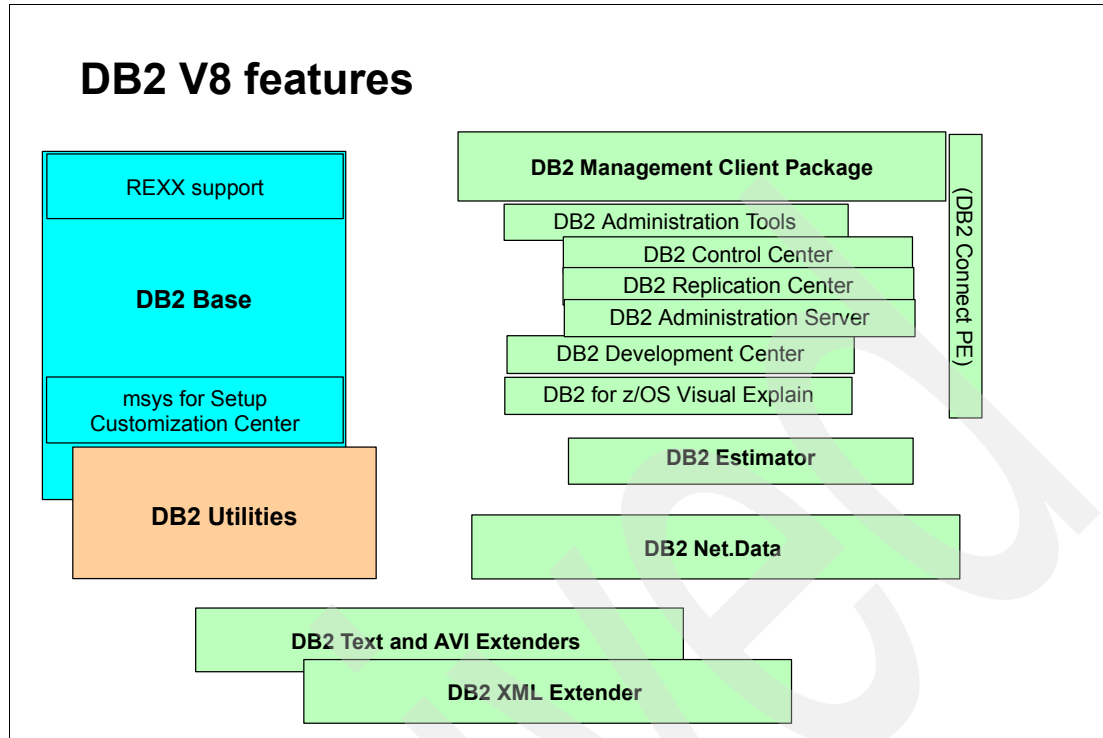


Figure 1-1 DB2 features

Notice that:

- ▶ The REXX Language Support has moved to DB2 base.
- ▶ Net.Data was stabilized as of V7.
- ▶ The DB2 Warehouse Manager has been removed as a priced feature of DB2 V7 in March 2002 and it is now a separate tool.
- ▶ Estimator is available as download from the Web only.
- ▶ Installer has been replaced by the msys plug-in, now in DB2 base.
- ▶ The DB2 Extenders are divided in three groups:
 - Text Extenders
 - Audio, Video, Image Extenders
 - XML Extender

1.2.1 New DB2 function: msys for set-up DB2 Customization Center

msys for set-up is a z/OS initiative with the intent to ease installation tasks. It consists on Web-based interactive dialogs for configuration and guidance through a set of high level questions which use defaults and best practises where possible to reduce user decision making. See 10.6, “msys for Setup DB2 Customization Center” on page 259 for its description.

1.2.2 No-extra-charge features

The features or products that can be obtained at no extra charge are:

- ▶ DB2 Management Clients Package
- ▶ DB2 Estimator
- ▶ DB2 Net.Data

- ▶ DB2 Extenders

DB2 Management Clients Package

The DB2 UDB for z/OS Version 8 Management Clients Package feature is a collection of the following workstation-based tools:

- ▶ **DB2 Development Center:** This new function extends and replaces the capabilities of the Stored Procedure Builder. As well as providing an easy to use development environment for stored procedures, it also supports:
 - Wizards for fast development of SQL and Java stored procedures
 - Advanced import, export, deployment options
 - A read-only server view to look at stored procedures, UDFs, triggers, tables, and views separately from the standard project view
 - Connection pooling and disconnected mode
 - More support for languages and authorizations
 - Integrated and remote multi-platform debugging tool

It also integrates stored procedures with code generated by Microsoft Visual Studio 6.0, ActiveX Data Objects, and Visual SourceSafe.

- ▶ **DB2 Administration Tools:** This category includes DB2 Administration Server (DAS), DB2 Control Center, and DB2 Replication Center:
 - **DB2 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 in the form of an SMP/E installable package with 390 Enablement.

- **DB2 Control Center:** IBM DB2 Control Center provides support to help you manage DB2 databases on an array of operating systems in your workplace. 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 390 Enablement provides these stored procedures, the user-defined function, and batch programs, in the form of an SMP/E installable package.
- **DB2 Replication Center:** The Replication Center is a graphical user interface of DB2 UDB Data Replication V8, used to define replication sources and map sources to targets. It is also used to manage and monitor the Capture and Apply processes on local and remote systems. The Replication Center runs on Windows and UNIX/Linux systems and must have connectivity to both the source and target servers. For information, refer to the DB2 Replication home page:

<http://www.software.ibm.com/data/dpropr>

- ▶ **DB2 Visual Explain:** The latest version of Visual Explain is a totally redesigned and rewritten new release, soon to be available on the Web site:

<http://www.ibm.com/software/db2os390/db2ve/>

Visual Explain is a workstation based feature of DB2 UDB for z/OS that displays:

- An easy-to-understand graph of the access paths of SQL statements
- The 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 to 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 (DB2 subsystem parameters) and DSNHDECP.

- The DB2 Visual Explain subsystem parameter browser requires an activated WLM address space.

Assistance is provided through the IBM Support Centers on all of these tools, just as for the main product.

The DB2 Administration Tools and the DB2 Development Center (which includes the former DB2 Stored Procedure Builder) are also shipped with the DB2 distributed and DB2 Connect V8.1 products. A restricted-use copy of DB2 Connect Personal Edition V8.1 (57324-B56) for Windows is provided in the DB2 Management Package feature of DB2 UDB for z/OS to satisfy the functional dependency.

This feature also contains two components that need to be installed at the host:

- ▶ Enablement for Control Center
- ▶ Enablement for Administration Server

DB2 Extenders

- ▶ Several types of Extenders are available:
 - Text Extenders
 - Audio, Video, Image Extender
 - XML Extender, for:
 - DTD and XML schema validation
 - Stylesheet processing
 - Shredding of XML documents into DB2 tables
 - Composition of XML documents from DB2 tables

1.2.3 Charge features

These are the features or products that can be obtained at an additional cost. They will be detailed with a later announcement, among them are the **DB2 Utilities**. The Utilities provide full support for DB2 UDB for z/OS V8, and enhancements are related to all three programs:

- DB2 Diagnostic and Recovery Utilities for z/OS, V8
- DB2 Operational Utilities for z/OS, V8
- DB2 Utilities Suite for z/OS, V8

The powerful enhancements in each utility package are available, without charge, if you have a license for the Version 7 utility package and also have licensed Subscription and Support. The major functional enhancements are described in Chapter 7, “Utilities” on page 169.

1.3 This redbook

This redbook is the result of an International Technical Support Organization (ITSO) residency and follow-on work done before DB2 UDB for z/OS Version 8 has reached General Availability (GA). It is a best effort on trying to keep up with a complete re-engineering of the product, as well as the addition of many functions, and the never-ending task of making the product the best enterprise server in the market. Its contents are most probably correct over 95%, and this should be sufficient to provide a good base for evaluating the new version for your needs, but by no means can we claim it 100% correct.

The results of the introduction program (ESP), currently under way, will probably continue to tune the product to the customer’s needs until GA and beyond. Please refer to the standard DB2 documentation available with the product for the correct specification of parameters. More redbooks are planned to provide more technical information, recommendations, and catch up with any other enhancements.

The book is structured in the following chapters:

- ▶ Chapter 2, “DB2 UDB for z/OS V8 at a glance” on page 9
- ▶ Chapter 3, “Scalability” on page 21
- ▶ Chapter 4, “Availability” on page 39
- ▶ Chapter 5, “SQL” on page 71
- ▶ Chapter 6, “e-business” on page 129
- ▶ Chapter 7, “Utilities” on page 169
- ▶ Chapter 8, “Performance” on page 205
- ▶ Chapter 9, “Data sharing” on page 235
- ▶ Chapter 10, “Installation and migration” on page 241

Archived

DB2 UDB for z/OS V8 at a glance

DB2 V8 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 memory, providing consistent table and column name lengths, allowing two-megabyte SQL statements, 4096 partitions, and three times the log space.

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 chapter we introduce the main enhancements available to you in New Function Mode. These enhancements are broadly associated to the following categories:

- ▶ Scalability
- ▶ Availability
- ▶ SQL
- ▶ e-business
- ▶ Utilities
- ▶ Performance
- ▶ Data sharing
- ▶ Installation and migration

2.1 Scalability

With V8, DB2 UDB for z/OS breaks through limits and sets new heights for scalability and performance. In this section we introduce the major scalability enhancements.

Virtual storage expansion

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
- ▶ Sort pool
- ▶ RID pool
- ▶ Compression dictionaries

A single large address space of up to 2⁶⁴ 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.

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 value determines the maximum number of partitions that is possible.

More tables in join

In 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. V8 allows 225 tables to be joined in all types of joins.

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 is overwritten is increased from 1,000 to 10,000 per log copy.

2.2 Availability

In this section we cover the various enhancements related to availability.

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 so 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).

Online schema changes

Several changes to DB2 objects can now be implemented without disrupting the availability of DB2.

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

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.

Partitioning and clustering were bundled together in versions prior to V8. Now you can have a partition without an index and can cluster the data on any index. These changes may spare one index and reduce random I/O.

System level point-in-time 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 minimizing the amount of log data that need 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.

Online ZPARMs

V7 introduced the ability to change some of the ZPARMs online using SET SYSPARM command, without the need to recycle DB2 UDB for the changed values to become effective. In V8 you can change more ZPARMs online. However, you cannot change the serviceability and DSNHDECP ZPARMs online.

CI size larger than 4 KB

DB2 V8 introduces the 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.

Monitor system checkpoint and log offload activity

In V8, DB2 monitors system checkpoint and log offload activity and uses the active log switch routine to notify when it detects that the checkpoint process or the offload task may have stalled.

Monitor long running UR backout

In V7 it is not possible to determine how long the backout of a long running UR might take. V8 issues a new progress message every 2 minutes until the backout is completed.

Improved LPL recovery

This enhancement improves availability, performance, usability, and serviceability for DB2 LPL recovery. Specifically, the following functions are added:

- ▶ **Usability and serviceability:** DB2 automatically initiates an LPL recovery processor if pages are added into the LPL, and recovers those pages. This avoids the manual intervention for LPL recovery, such as the START DATABASE command or the RECOVER utility.

- ▶ **Availability and performance:** The improved LPL recovery processor (the START DATABASE command or the automatic LPL recovery) makes a write claim on an object (a page set or partition) instead of draining it. This makes good pages in the object available to SQL users. Also, it improves the performance, since the claim is less disruptive than the drain.

2.3 SQL

There are many enhancements with the SQL language that provide compatibility with the DB2 family. In this section we introduce the major SQL enhancements.

Long 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.

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 Megabytes.

Enhanced scrollable cursors

The DYNAMIC option of scrollable cursors allows scrolling directly on the DB2 table without an intermediate temporary result table. Depending on the access path chosen, scrolling can occur using an Index or doing a table space scan.

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 because using common table expression instead of views saves both users and the system the work of creating and dropping views.

Multi-row fetch and insert

The multi-row FETCH/INSERT is a performance as well as compatibility enhancement that allows a user to execute multiple fetches or inserts with one SQL statement. This feature can be used for read only window scrolling applications, as well by users that want to view specific rows of data and at the same time have the ability to change any of the viewed rows.

Get diagnostics

The new GET DIAGNOSTICS statement is important to provide the information from all of the extended names and new function. Most programmers will need to switch from using SQLCA and use this more standard, more capable facility for diagnostic information. The GET DIAGNOSTICS statement can return much more information about the statement and about conditions and connections. For example, it can return the longer names, the multiple conditions for the multi-row statements, and the error message associated with an error.

Scalar fullselect

V8 allows scalar fullselects to appear wherever expressions are allowed. A scalar fullselect can return a maximum of one row and one column, that is, a single value.

Select from insert

This enhancement supports the requirement to retrieve row or rows inserted into a table in the following situations:

- ▶ Provide a way for an application to find out the value of an automatically generated column (such as a ROWID, identity) when a new row is inserted into a table.
- ▶ Provide a way for an application to request that one or more column values that were inserted be returned because their values were not specified by the invoking application (either the values inserted were the default values, or the values were changed by a trigger).
- ▶ Provide a short hand syntax for returning all values for each row being inserted (that is, do not require specification of all column names).
- ▶ Provide a way to return values for multiple rows being inserted.

Qualified column names in INSERT and UPDATE

DB2 V8 allows the column name to be qualified in the INSERT statement and in the SET clause of the UPDATE statement. This enhancement is introduced for DB2 family compatibility.

Expressions in GROUP BY

DB2 V8 supports expressions to be specified in the GROUP BY clause. This enhancement not only accomplishes DB2 family compatibility but also facilitates many vendor developed applications running without any changes.

Multiple DISTINCT

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
```

Sequences

V8 provides support for a new data object called *sequence*. A sequence is a user-defined object that generates a sequence of numeric values according to user-specifications. Sequences provide an ideal way for applications to have the DBMS generate fast, recoverable, guaranteed-unique, sequential numeric key-values, and to coordinate keys across tables and in data sharing environments. A sequence is created with the CREATE SEQUENCE statement. Its attributes are altered with the ALTER SEQUENCE statement. Other SQL statements related to sequences are GRANT/REVOKE ON SEQUENCE, DROP SEQUENCE and COMMENT ON SEQUENCE. The current and next value of a sequence are retrieved with the PREVIOUS VALUE FOR SEQUENCE and NEXT VALUE FOR SEQUENCE expressions.

Identity columns enhancement

With DB2 V8 identity column enhancements extend the ALTER TABLE ALTER COLUMN statement to include identity column specifications, allow dynamic alter of the attributes of an existing identity column, and provide support for some additional keywords for the column specification.

Multilevel security

A high priority requirement is for row-level security for applications that need more granularity in their security schemes. Traditionally, views and joins have been the application solution to this requirement. The multilevel security introduced with DB2 V8 supports hierarchical security schemes and combines extensions to SQL with RACF access control to provide row granularity.

MQSeries UDF

DB2 and MQSeries can be used to construct applications that combine messaging and database access. It is now possible to integrate MQSeries messaging operations within SQL statements.

2.4 e-business

DB2 UDB for z/OS is more than a large storehouse for your enterprise data. DB2 V8 helps you to leverage your enterprise for e-business. In this section we introduce the major enhancements that help you to achieve this.

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. This enhancement 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.

Unicode support

Architectural changes to DB2 V8 expand the DB2 catalog with full 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.

ODBC enhancements

Several enhancements have been provided to ODBC. Among them:

- ▶ **Unicode support:** ODBC now supports Unicode formats UTF-8 and UCS-2. In addition, a new ODBC initialization keyword, CURRENTAPPENSCH, lets you specify the encoding scheme that you want the ODBC driver to use for input and output of host variable data, SQL statements, and all character string arguments of the ODBC application programming interfaces. You can specify Unicode, EBCDIC, or ASCII encoding scheme.
- ▶ **ODBC SQL Connect user and password support:** With V7, the DB2 ODBC driver validates the userid and password, but their argument values are not used for end user authentication. DB2 V8 implements the new support for USER/USING SQL CONNECT statement, and the ODBC driver makes use of the userid and password values provided on the APIs to perform authentication.

XML publishing functions

XML has increasingly become the de facto data format language on the internet, on corporate intranets, and for data exchange. However, in many cases applications have to generate XML data from traditional relational databases using application packages or middleware. This enhancement provides a set of SQL built-in-functions that allow applications to generate XML data from relational data with high performance. This reduces application development efforts in generating XML data for data integration, information exchange, and Web services.

CURRENT PACKAGE PATH special register

Package switching and versioning for static SQL applications is critical. SQLJ access increases the need for these types of control. This enhancement introduces a new special register, CURRENT PACKAGE PATH, as a means to specify a list of collections to search for the appropriate package. The semantics are similar to the PKLIST bind option, except that the PACKAGE PATH list is processed at the server. This new special register provides this control to applications that do not run under a DB2 plan.

DDF enhancements

The enhancements provided to DDF include:

- ▶ **Requester database ALIAS:** A new column DBALIAS, which has been added to the SYSIBM.LOCATIONS table, now points to the correct TCP/IP address for the workstation you want to access. You can now connect to databases with the same name on every LINUX/UNIX/Windows system.
- ▶ **Server location alias:** To ease the migration of location names when consolidating in a data sharing group, you can now update the BSDS to add server location aliases.
- ▶ **Member routing in a TCP/IP network:** By combining the server location alias and a definition in the new SYSIBM.IPLIST table, you can route to a specific member rather than letting WLM do the balancing.

Enhancements for stored procedures and UDFs

Currently, you can specify a maximum abend value for all stored procedures and user defined functions on a single DB2 subsystem. However, this may not be always satisfactory because you can have a mix of established applications and applications under testing. This enhancement allows you to set a limit on how many times a stored procedure or a user defined function can fail before it is stopped. This enhancement also exploits the z/OS WLM function designed to determine appropriate resource utilization and provide a method of changing the number of tasks within a stored procedure address space.

DB2 V8 no longer supports COMPJAVA stored procedures.

SQL procedure enhancements

Among other improvements, the SIGNAL and RESIGNAL SQL statements allow the SQL procedure to specify a specific SQLSTATE and message text to raise a condition within the SQL procedure. If the condition is not handled by the procedure body, this information is returned to the caller. This capability is useful to packaged applications such as the extenders which have their own SQLSTATEs that they want to return to the invoking application.

2.5 Utilities

Significant enhancements have been introduced in the area of DB2 Utilities. In this section we introduce the major ones.

Online schema changes

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.

Delimited LOAD and UNLOAD

This enhancement allows LOAD to process input files where the columns are identified by delimiters such as “,” rather than having fixed positions within the record and/or 2-byte length fields for VARCHAR input. This allows LOAD to process files unloaded, for example, from the workstation DB2, Oracle, Sybase, or Informix.

Conversely, UNLOAD unloads the data in the output file with delimiters and this data set can be used to load the data in another system.

Unicode parser

You can provide the utility control statements entirely in EBCDIC characters or entirely in Unicode characters. DB2 automatically detects which is being used and therefore there is no OPTIONS control statement or system parameter setting required in order to submit utility control statements in Unicode.

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 multicolumn cardinality for non-indexed column groups and update the catalog.

Back up and restore system

These two new utilities provide system level, point-in-time, level of recovery. They activate new functionalities available with the new z/OS V1R5 DFSMSHsms, which allow a much easier and less disruptive way for fast volume-level backup and recovery to be used for disaster recovery and system cloning. This function is of great interest for ERP solutions where recovery copies of large number of disk volumes for data and indexes need to be synchronized for application related consistency, but it could be used as cornerstone for any recovery solution.

REORG enhancements

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 ALTER INDEX statement before executing the REORG utility. You can specify DISCARD with SHRLEVEL CHANGE. You can avoid BUILD2 phase during online REORG by using the new data partitioned secondary indexes.

2.6 Performance

In this section we introduce the major enhancements that help improve the performance of your applications:

Compare unlike data types

This enhancement improves the performance of DB2 V8 by allowing the predicates to be stage 1, even when comparing columns/values of different data types, as long as their data types are compatible.

Materialized query tables

This enhancement provides a set of functions which allow DB2 applications to define, populate, and make use of materialized query tables. The elements of these functions are as follows:

- ▶ A source table is either a base table, view, table expression, or a user-defined table function.
- ▶ A materialized query table is a table that is used to contain materialized data that is derived and summarized from one or more source tables specified by a fullselect.
- ▶ A materialized query table can be either a system-maintained materialized query table, which does not allow user update, or a user-maintained materialized query table, which allows user update.
- ▶ The extended CREATE TABLE SQL statement is used to define a materialized query table to DB2. It specifies a fullselect associated with the table (much in the way a fullselect is used to define a global temporary table), and specifies the mechanisms that are to be used to refresh the materialized query table and to keep the data in the materialized query table synchronized with the data in the source tables from which the materialized query table was derived. The extended CREATE TABLE SQL statement can also be used to define a base table by specifying a fullselect for DEFINITION ONLY to derive the column definitions of the table.
- ▶ The extended ALTER TABLE SQL statement can be used to register an existing base table as a materialized query table to DB2. It can specify a fullselect associated with a table so that the table can be used in automatic query rewrite. It can also be used to enable or disable a materialized query table for automatic query rewrite and to switch the materialized query table types between the system-maintained and the user-maintained. The extended ALTER SQL statement can also be used to change a materialized query table into a base table.
- ▶ The REFRESH TABLE SQL statement is used to refresh a named materialized query table. This statement deletes the data currently in the materialized query table and then executes the fullselect associated with the materialized query table to repopulate it.
- ▶ Automatic query rewrite is a process that examines a submitted query that references source table(s), and if appropriate, rewrites the query so that it executes against a materialized query table derived from the source tables. The automatic query rewrite results in a significant reduction in query execution time in most cases.
- ▶ The CURRENT REFRESH AGE special register is used to control, at the SQL statement level, whether a materialized query table with a certain refresh timestamp can be used in the automatic query rewrite.
- ▶ The CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION special register is used to control, at the SQL statement level, which types of materialized query tables (system-maintained, user-maintained, both, or none) are exploited in the automatic query rewrite.

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.

Parallel sort

The sort process has been enhanced to be able to run multi-table sorts in parallel. Based on a cost model decision, sort parallelism can exploit CPU resources and reduce elapsed time.

Sparse index for star join

The star join implementation in DB2 UDB for z/OS has to deal with, potentially, 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.

Long and variable length keys

DB2 V8 extends support for long index keys, variable length index keys, and long predicates. The maximum length of a long index key is 2000 bytes, increased from 255 in DB2 V7. True variable length keys are supported in the index. That is, variable length key columns are not padded to the maximum length of the index key. This reduces the storage requirements for the index, since only actual data is stored. More importantly, DB2 allows index-only access to variable length index keys. The maximum column length for predicate operands is 32704 (the maximum size of a VARCHAR column), increased from 255 in DB2 V7.

Backward index scan

With this enhancement it is no longer necessary to create an ascending and a descending index on the same table columns in order to satisfy scans in both directions.

Trigger enhancement

Each time an AFTER trigger with a WHEN clause 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. This enhancement saves the changes in memory if there are only few, instead of creating and deleting the workfile each time.

Reduced lock contention on volatile tables

Volatile (or cluster) tables, used primarily in the SAP application environment, 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.

Table UDF cardinality option

This enhancement introduces the cardinality option for a user defined table function reference in the SQL language level. Queries involving user defined table functions, in general, could benefit from this feature, when users can estimate the number of rows returned by the function before the queries are run. The performance improvement of such queries could be achieved in conjunction with the features introduced by table UDF block fetch. This option is a nonstandard SQL extension, and it is specific to DB2 UDB for z/OS implementation.

2.7 Data sharing

Data sharing includes CF lock propagation reduction and other performance and usability enhancements.

CF lock propagation reduction

This enhancement allows in a data sharing environment, 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.

Other performance and usability enhancements

- ▶ Reduction of overhead costs for data sharing workloads
- ▶ Batched updates for index page splits
- ▶ Improved LPL recovery
- ▶ Resolution of indoubt units of recovery in restart light
- ▶ Change to IMMEDIATE BIND option default
- ▶ Change to -DISPLAY GROUPBUFFERPOOL output

2.8 Installation and migration

The two key items we anticipate in this brief section are prerequisites and migration changes.

Prerequisites

The DB2 V8 environment is available only for the z/OS platform, that is, you need to run DB2 on a zSeries processors with z/OS V1R3 operating system or later. For some specific function a later release of z/OS might be necessary.

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. Fall back to V7 in case of problems is allowed.
2. **Enable 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.

Archived



Scalability

DB2 for z/OS V8 offers zSynergy and breaks through some of the previous operating system limitations that affect 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 fields 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 tables in join
- ▶ More log data sets

3.1 The 64-bit architecture support

In this section we first describe objectives and functions of the z/OS architecture as a level set, and then we look at the 64-bit virtual support with DB2 UDB for z/OS Version 8.

3.1.1 The z/OS architecture

Figure 3-1 is meant to demonstrate why the new 64-bit architecture was introduced. It shows that, for several storage starved environments, adding CPU capacity was not allowing any further growth, and resulted 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 had provided an excellent interim solution, and the G5/G6 experienced some relief with the implementation of the enhanced MOVE PAGE instruction, but systems with large and variable workloads needed the storage constraint removed.

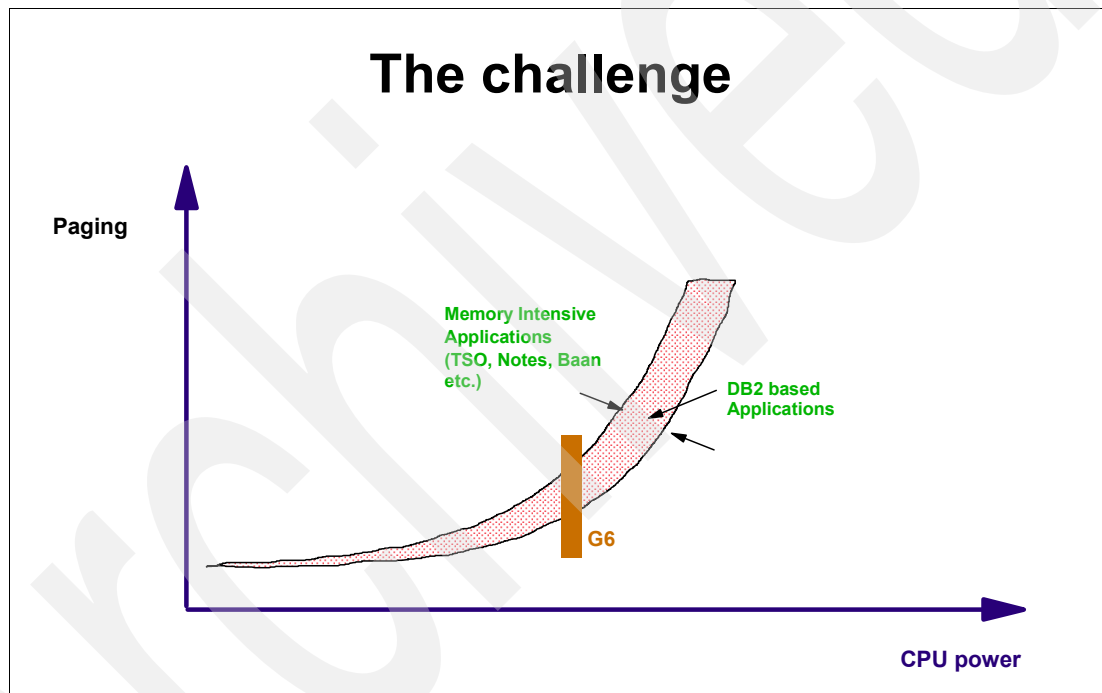


Figure 3-1 The challenge

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.

OS/390 R10 and z/OS have provided the 64-bit *real storage* addressability needed to scale in real memory addressing. OS390 R10 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. z/OS 1.2 and later releases provide *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, with the z900 offering 64 GB) 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 z900 and z800 memory as REAL. The elimination of Expanded Storage support has been handled by z/OS with minimal customer impact while reducing memory management overhead. It has also enabled the 16-way multi-processors in z900 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/>

In z/OS V1.2, IBM has delivered the initial *64-bit virtual storage* management support. With the new z/OS 64-bit operating environment now 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.

S/390 hardware and software have a long and outstanding history of support for large scale computing environments. High capacity and flexible scalability in both hardware and software are the main characteristics of the architecture, and the OS/390 Parallel Sysplex and Workload Manager have provided outstanding horizontal growth for subsystems and application servers. z/OS is providing 64-bit virtual storage addressability to increase the capacity and throughput of applications in various e-business environments.

Figure 3-2 gives a pictorial representation of the evolution of the memory management from the 24-bit to the 31-bit to the 64-bit support.

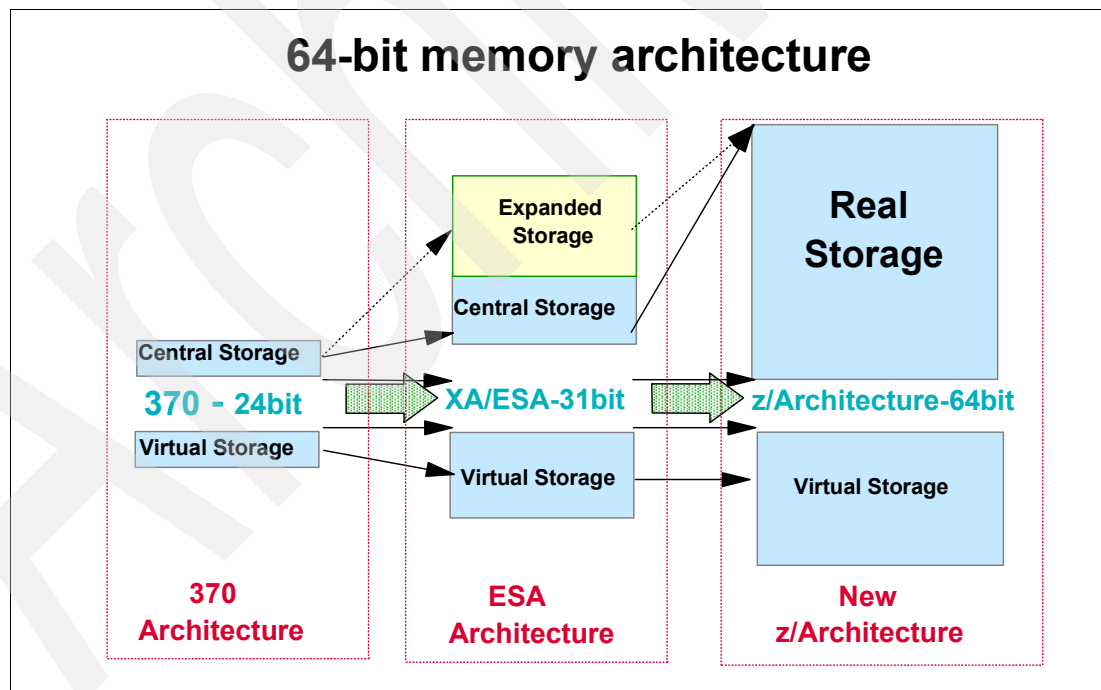


Figure 3-2 The 64-bit memory architecture

Figure 3-3 shows in more detail 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 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 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.

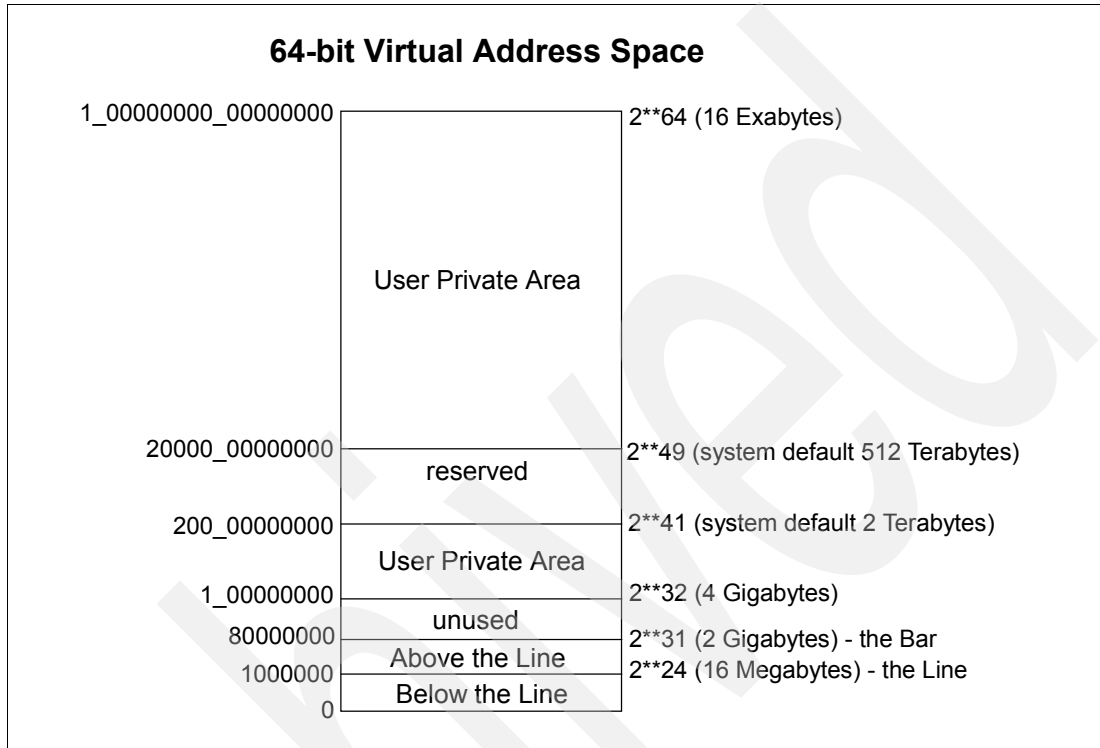


Figure 3-3 64-bit virtual address space mapping

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.htm>

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), and future DB2 applications can benefit transparently from DB2's 64-bit virtual storage support. The benefit is derived from the efficiencies with the local availability of data made possible through 64-bit data addressability. DB2 UDB for z/OS V8, with 64-bit virtual address support, can only execute on IBM @server zSeries 800/900 including the zSeries 900 turbo, or equivalent, running z/OS V1R3, or later. DB2 V6 (5645-DB2) and V7 (5675-DB2) already support 64-bit real storage addressing for data space buffers, providing improved scalability and performance in a zSeries processor running in 64-bit real mode. Using 64-bit real provides a significant storage relief for most customers.

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. For a brief summary of the main functions of the z/OS recent versions, see Figure 3-4.

z/OS 1.1 & 1.2	z/OS 1.3	z/OS 1.4
<ul style="list-style-type: none"> • z/Architecture -64-bit real storage (1.1) -Intelligent Resource Director (1.1) • Interoperability with Linux -IRD extensions, HiperSockets (1.2) • Availability -Parallel Sysplex coupling enhancements (1.1, 1.2) • Networking -HiperSockets (1.2) -TCP/IP enhancements (1.2) • Security -Industry and international standards (1.2) -Extended support for SSL and Digital Certificates (1.2) • Open Applications -Java, C++ and File System enhancements (1.2) • Cost of Ownership -Simplified setup and operations (1.1, 1.2) -New tools and utilities for DB2 and IMS • 64 bit virtual support 	<ul style="list-style-type: none"> • Application Flexibility -zSeries File System improvements -UNIX System Services availability enhancement • eLiza Self Optimizing -WLM improved for distributed DB2® and WebSphere applications • Improved Storage and I/O Management (DFSMS) -Business continuity and business efficiency improvements • Security -PKI Services for Digital Certificate Life-Cycle Management -Enhanced Crypto Algorithms and Standards -More granular access control for UNIX file systems with Access Control Lists 	<ul style="list-style-type: none"> • Tools to manage your e-business -Increase scalability by new extended network addressing with IPv6 -Enable clock synchronization between clients & servers with SNTP -Add systems to your sysplex without sysplex wide IPL -Enhanced System SSL, Firewall and LDAP Technologies for e-business security • Application Flexibility and Autonomic Computing extensions -Enhancement of dynamic balancing of business workloads -Self optimization of WebSphere for z/OS by more granular reporting -Ease system setup & support multiple users with msys for setup framework -Simplify management of Unix System Services (USS) identities -Performance and Usability improvements for zFS and SMB File/Print Server • Availability & Serviceability improvements -Server activity logged by the LDAP Server -z/OS Service through the internet with ShopzSeries

Figure 3-4 Summary of z/OS current versions

3.1.2 DB2's virtual storage expansion

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.

3.1.3 Moving above the 2 GB bar

DB2 V8 has made massive changes to its code, and now provides a solution to the current virtual storage constraint by utilizing 64-bit virtual addressing to move the following data areas above the 2 GB bar (2^{31}) in the DBM1 address space:

- ▶ Buffer pools and buffer pool control blocks
- ▶ Sort pool
- ▶ RID pool
- ▶ EDM pool
- ▶ Compression dictionaries
- ▶ Castout buffers
- ▶ IRLM locks
- ▶ Materialized LOB values

Larger buffer pools

With the very large and ever-cheaper main memories that are available on the current and upcoming z/Architecture machines (currently 10s of GB, moving towards 100s of GB), 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 far 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 due to DBM1 virtual storage constraints. VSAM control blocks and compression dictionaries are other sizeable contributors to the demands on DBM1.
- ▶ DB2 V7 limits the total size of hiperpools to 8 GB. This limit could be raised, however hiperpools have several 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 (HWBs) 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.

Data spaces provided a good short term solution by exploiting the 64-bit REAL memory support introduced OS/390 V2R10. DB2 V6 could place buffer pools and statement caching in data spaces freeing up space for other work in the DBM1 address space. A performance penalty would be paid when such buffering was not 100% backed by real storage.

The advantages of data spaces over hiperpools were:

- ▶ 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 z900 and 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.

The total size of data space virtual pools is limited to 32 GB (4 KB page size). This limit is imposed by a maximum of 8 million “page manipulation blocks” (PMBs) which reside in the DBM1 address space. Also, the lookaside pool resides in DBM1. 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 (see Figure 3-5).

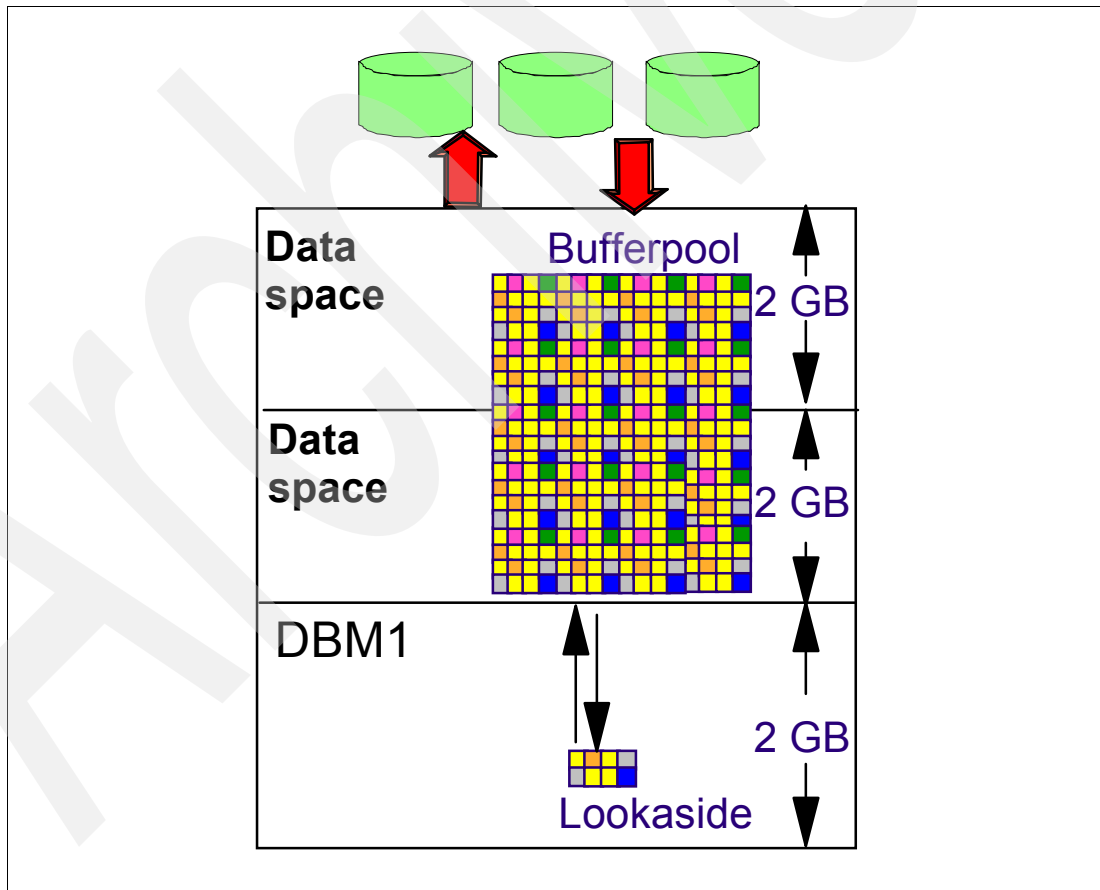


Figure 3-5 Data spaces

The use of 64-bit virtual addressing greatly increases the maximum buffer pool sizes. DB2 V8 is 64-bit exclusive, and therefore always allocates the buffer pools above the 2 GB bar. This effectively eliminates the need for hiperpools and data space pools and simplifies DB2 systems management and operations tasks. Therefore, hiperpools and data space 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. 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 real storage, or the normal allocation of a buffer pool not previously used will cause an aggregate size which exceeds the real storage. Either request will then be limited to 8 MB (2000 pages for 4 KB, 1000 pages for 8 KB, 500 pages for 16 KB, and 250 pages for 32 KB).

As mentioned, DB2 limits the allocation of buffer pools if the aggregated (allocated) pool size exceeds the value of 2 x REAL storage size of system image.

DB2 V8 increases the maximum buffer pool sizes to the limit of the architecture, 1 TB, however the limitation is the given by the real storage available:

- ▶ Maximum size for a single buffer pool is 1 TB
- ▶ Maximum size for summation of all active buffer pools is 1 TB

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.

The buffer pool names (BP0, BP1, etc.) do not change. Neither do the page sizes: The options are still 4 KB, 8 KB, 16 KB, or 32 KB. The values change as listed in Figure 3-6.

Parameter	Value
BP0	Minimum and default 2000
BP8K0	Minimum and default 1000
BP16K0	Minimum and default 500
BP32K	Minimum and default 250
CTHREAD	Limit increased from 2000 (check REAL storage availability)

Figure 3-6 The new buffer pool values

The ALTER BUFFERPOOL command parameters no longer supported are VPTYPE, HPSIZE, HPSEQT, CASTOUT. If they are specified, just a warning message DSNB539I is issued.

The other parameters remaining unchanged are VPSEQT, VPPSEQT, VPXPSEQT, DWQT, VDWQT, and PGSTEAL.

LSTATS report removes the references to hiperpool related counters.

RID pool

The RID Pool is split into two parts. A small part of the RID Pool remains below the 2 GB bar and the majority (about 75%) is moved above; see Figure 3-7. The RID Pool below the 2 GB bar stores the RID Maps which are small in number, and the RID Pool above the 2 GB bar contains the RID Lists which comprise the bulk of the RID Pool storage.

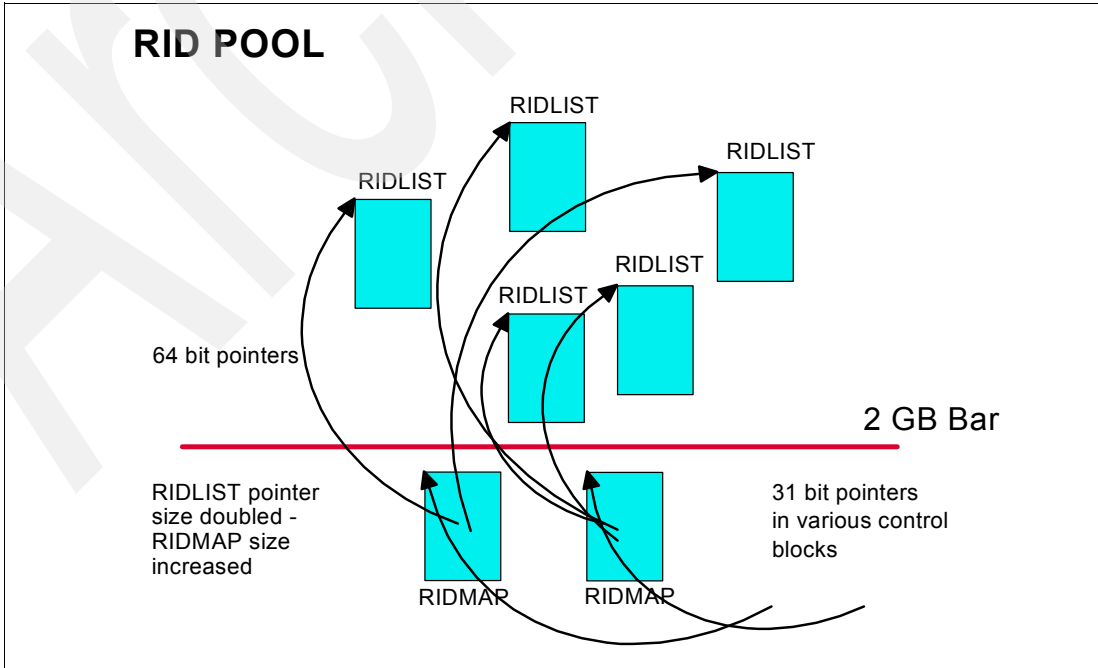


Figure 3-7 RID pool

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.

Sort pool

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 Sort to store various control structures and data records. One pool is an agent-related local storage pool, and the other is a global sort pool. To take advantage of the 64-bit addressability for a larger storage pool, some high level sort control structures remain in agent-related storage below the 2 GB bar, but these structures contain 64-bit pointers to areas in the global sort pool above the 2 GB bar. The sort pool above the 2 GB bar contains sort tree nodes and data buffers.

Compression dictionaries

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) therefore moving the dictionary above the 2 GB bar provides significant storage relief for many customers.

For some customers, those who have a large number of compressed table spaces, the compression dictionaries can use up as much as 500 megabytes. This can further increase compression dictionary storage requirement for some systems, depending upon how many of these data sets contain compressed table spaces. DB2 V8 also implements support for 4096 partitions for a single table or index, this is another driver for moving compression dictionaries above the 2 GB bar. With 4096 partitions, customers might choose to have a larger number of smaller partitions resulting in a corresponding increase in the total number of compression dictionaries in those partitioned database.

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.

EDM pool

A new dynamic statement cache is created above the 2 GB bar. Today if “cache dynamic” is on, the statements are cached in the data space, if one is defined, or in the normal EDM pool if a data space is not defined. Now, cached, dynamic statements are always cached in the dynamic statement cache pool above the 2 GB bar, see Figure 3-8.

A new EDM DBD cache is created above the 2 GB bar. This gives the DBDs the needed space to grow and relieves contention with other objects in the EDM pool. Three parameters are now used to allocate the EDM pool: EDMPOOL storage size, EDM storage cache, and EDM DBD cache.

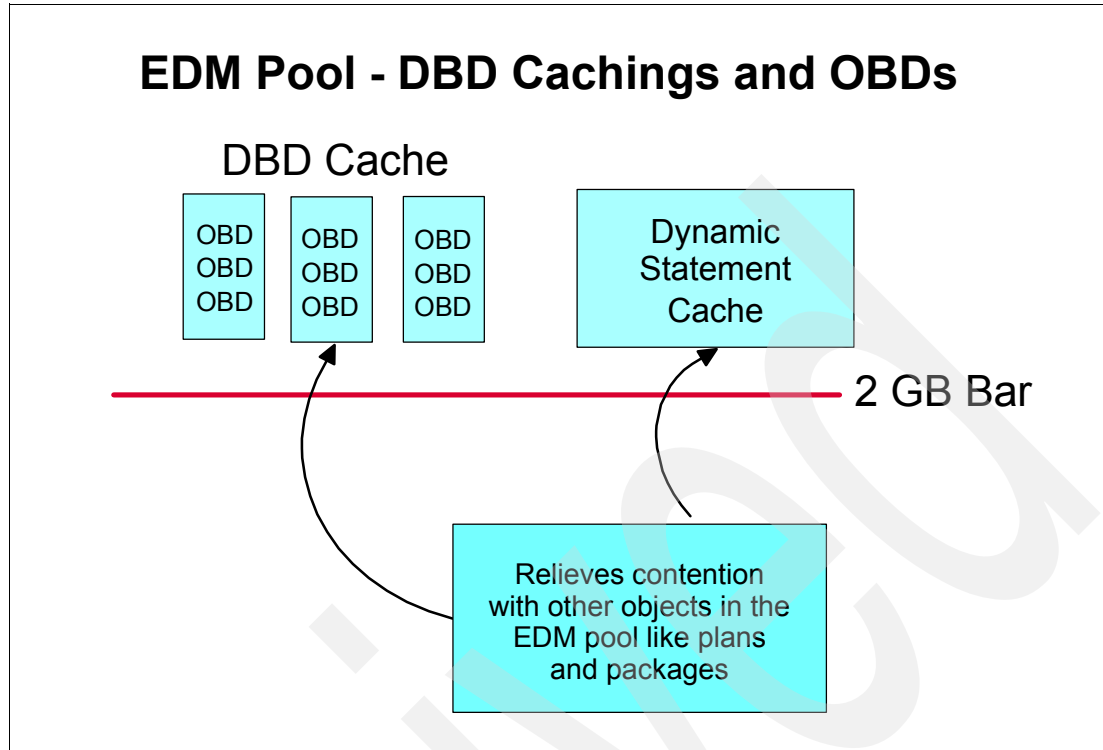


Figure 3-8 EDM pool

LOB data

LOBs are now materialized, depending on the application requirements, above the 2 GB bar in DBM1 address space, and allocated in areas limited by the system parameters previously used for allocating data spaces:

- ▶ LOBVALA (the size per user):
The default is 2048 KB, the limit value is 2097152 KB.
- ▶ LOBVALS (the size per system):
The default is 2048 KB, the limit value is 512 MB.

Other changes

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. The locks now reside above the 2 GB bar, and the default for IRLM has changed in V2.2, it was PC=NO, now PC=YES is forced.

3.1.4 General performance 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 vs. 2 byte instructions
- ▶ More expensive linkage between modules

Here is the expected impact when comparing to DB2 V7 BPOOLS with same size and backed by REAL storage:

- ▶ Up to 10% CPU degradation for intensive page processing workloads is likely.
- ▶ Other workloads are expected to be within 5%.

3.1.5 Storage monitoring and tuning

With DB2 V8's exploitation of 64-bit virtual storage, the following capabilities are possible:

- ▶ Buffer pool monitoring and tuning becomes simpler:
 - Hiperpools and data space pools are eliminated, thus reducing complexity. There is now only one type of buffer pool. EDM pool and LOB data spaces have 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.
 - 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 groups to fewer members.
- ▶ To handle the expected increases in workload, the maximum number of deferred write and castout engines are increased in order to decrease *engine not available* conditions.

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

VSTOR 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, RMF produces summary reports
- ▶ Specify D, 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:

3.1.6 DB2 V8 requires z/Architecture and z/OS V1R3

The main focus of DB2 V8 and virtual storage constraint relief is to utilize 64-bit virtual addressing to move the buffer pools and their associated buffer control blocks above the 2 GB bar in the ssnmDBM1 address space. DB2's data access modules are enhanced to access the 64-bit addressable buffers in place, without any data movement as is done today for data space buffer pools.

DB2 V8 requires z/Architecture machines and also requires that those machines must be running in 64-bit addressing mode. If an attempt is made to start DB2 V8 on a non-64-bit machine, then DB2 issues an error message during startup and terminates.

DB2 V8 needs 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 that these prerequisite have implications for disaster recovery and sysplex cross-system restart scenarios.

Summarizing the key points:

- ▶ zSeries processor running 64-bit mode and z/OS 1.3 (or later) are prerequisite:
 - This requires preliminary migrations for test and development environments.
 - You should also consider the disaster recovery implications.
 - Parallel Sysplex and cross system restart are possible.
- ▶ DB2 now allocates space above 2 GB bar in DBM1:
 - This frees up significant storage in the 31 bit addressable area.
 - There are more concurrent threads.
 - Higher transaction throughput is possible.
- ▶ z/OS provides a new MEMLIMIT JCL keyword which controls how much VSTOR above 2 GB bar is available in each address space.

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 should be available in its 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.

Start by running a DB2 UDB for OS/390 Version 7 subsystem under a 64-bit virtual O/S 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.

3.2 More partitions

Customers have a need for more partitions in their partitioned table spaces for certain types of applications. One example of these applications is 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 wanted 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 in many partitions to reduce the size of their partition data sets. For a 16 terabyte table, the maximum values allowed for page size 4 KB, each partition has to be 64 gigabytes and that may be too unwieldy to manage. More partitions would allow to reduce the data set size by making the data more granular (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 value is dependent on the page size and the LARGE (not recommended) or DSSIZE (preferred) parameter. 4096 partitions would help the applications mentioned above and eliminate the work around solutions. You can define new partitioned table spaces with the new value of partitions, or you can use online schema changes (see 4.3.8, “Dynamic partitions” on page 60) 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; see Figure 3-10.

Maximum number of partitions

- The maximum number of PARTs depends on the **DSSIZE** and the **page size** in the CREATE TABLESPACE statement
- USE DSSIZE when creating table space for partitions of 4GB and larger
 - ▶ LARGE clause is only for compability with previous release to identify each partition of a partitioned table space has a maximum partition size of 4 GB.
 - ▶ Use DSSIZE clause as a preferred method instead
- DB2 creates default DSSIZE (if LARGE or DSSIZE keywords are omitted) based on the page size value

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

Figure 3-9 New number of partitions

The maximum number allowed in NUMPARTS is dependent on the page size and DSSIZE specified for the table space, see Table 3-1 for details. If DSSIZE (or LARGE) keywords are not specified, 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, for 32 KB page size, DSSIZE default is 32 GB.

Table 3-1 Maximum number of partitions versus DSSIZE and page size

DSSIZE	Page size 4 KB	Page size 8 KB	Page size 16 KB	Page size 32 KB
1-4 GB	4096	4096	4096	4096

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

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 4.3.8, “Dynamic partitions” on page 60.)

For example, if a customer wanted daily data segments, the table space could be created with 365 partitions and then partitions could 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 needs to be one LOB table space, one auxiliary table and one auxiliary index per partition per LOB. 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.

New data set naming convention

Currently, DB2 names the DB2 data sets with the convention of 'Axxx' 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 3-1.

Example 3-1 New data set naming convention

```

catname.DSNDBx.dbname.pname.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, and
lnnn is E000-E096 for partitions 4000 through 4096

```

3.3 More tables in join

The documented limit on the number of tables that can be joined is 15 for DB2 versions prior to V8. Many customers need to run queries that join more than the 15 tables in their ERP or CRM applications, typically designed with large numbers of tables to be joined. For queries

which qualify for star join processing, the limit is up to 225 tables in the FROM clause. To get around the 15 table limit, customers can evaluate the use of a “hidden” ZPARM 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 reason the default limit on the number of tables joined has stayed at 15 is because there has been a risk that a large query could cause DB2 to consume extra amounts of resources (storage and CPU) when evaluating the cost of each possible join sequence. This in turn can cause critical storage shortages and negatively impact the DB2 subsystem. See APAR PQ31326 for more details.

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 us 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.

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 threshold is 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.

3.4 More 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 MB 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 are creating so many archive log data sets that the 1,000 archive log volume maximum only allows them to register a few days of log data in the BSDS.

Having more log data available in the active log data sets 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, queuing for archive log tape volumes would be virtually eliminated.

The enhancements are summarized in Figure 3-10. DB2 V8 increases the maximum number of archive log volumes recorded in the BSDS from 1,000 volumes per log copy to 10,000 volumes. The maximum number of active log data sets is also increased from 31 per log copy to 93 log data sets.

Active and archive logs are increased

- The maximum size of active or archive log data set is 4 GB
 - ▶ APAR PQ48126 for DB2 V6 and V7
- The maximum number of archive log volumes increases from 1000 to 10,000 per log copy
 - ▶ DB2 system can remain recoverable without having to take frequent image copies
- The maximum number of active log data sets increases from 31 to 93 per log copy
 - ▶ more active logs means less chances for DB2 to require the archive log data sets for an extended rollback or media recovery
 - ▶ faster recovery as active log read is generally faster than archive log read
- Changes are required in the BSDS to allow the new maximum values
 - ▶ convert BSDS with the new DSNJCNVB utility
- MAXARCH (RECORDING MAX on install panel DSNTIPA) now allows a maximum value of 10,000
 - ▶ if DB2 detects the a value > 1000 without BSDS conversion
 - DSNJ155I is issued at startup
 - the value is reset to 1000 and processing continues
- Preconversion tasks
 - ▶ save and rename BSDS
 - ▶ redefine a larger BSDS using statements in DSNTIJIN
 - ▶ copy old BSDS to new

Figure 3-10 Logs data sets increase

DB2 V8 increases the maximum number of active log data sets to 93 per log copy. DB2 V8 is able to support 10,000 archive log data sets per log copy.

Increasing the maximum number of active log data sets and archive log volumes requires conversion of the BSDS to allow more data sets entries. To do the conversion, the user runs a new 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 Mode (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 during a new installation, however, the user must still convert the BSDS by running the new conversion utility job DSNJCNVB once in NFM. When migrating to V8, the user should follow the documented pre-conversion procedure to manually redefine a larger BSDS before converting with DSNJCNVB.

Prior to running the conversion utility, you need to do the following steps:

1. Rename existing BSDS data set to save the original in case conversion fails.
2. Allocate larger BSDS using original BSDS name using VSAM DEFINE statements in DSNTIJIN.
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 “Increase number of active and archive log data sets in BSDS” on page 256.

Archived

Availability

DB2 UDB for z/OS breaks through many limitations 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 access your data through more partitions, gain greater availability and management through data partitioned secondary indexes (DPSIs), and minimize partition management for historical data with support for rolling partitions. Version 8 also introduces a new technique to back up and recover an entire DB2 subsystem, as well as improvements to CI size support, the LPL recovery process, and better monitoring of the system checkpoint process, log offload activity, and UR backouts.

In this chapter we discuss the following topics:

- ▶ Online schema changes
- ▶ System level point-in-time recovery
- ▶ Data partitioned secondary indexes
- ▶ Online ZPARMs
- ▶ Control intervals larger than 4 KB
- ▶ Monitor system checkpoints and log offload activity
- ▶ Log monitor long running UR backout
- ▶ Improved LPL recovery

4.1 DB2 V8 partition and index related terminology

Up to V7, terms such as secondary index (on a partitioned table), non-partitioning index, or non-clustering index, were often used interchangeably. With all the enhancements related to partitioning and indexes in DB2 V8, it is important that we make sure to use the correct terminology. It is important to distinguish between:

- ▶ Index-controlled and table-controlled partitioning
- ▶ Partitioned and non-partitioned indexes
- ▶ Partitioning and non-partitioning or secondary indexes

4.1.1 Table-controlled partitioning

Before V8, only *index-controlled partitioning* was supported. With index-controlled partitioning, the 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 partition boundaries can be specified on the CREATE TABLE statement, as shown in Figure 4-1.

Both types of partitioning are supported with DB2 V8, but several new enhancements are only supported when using table-controlled partitioning.

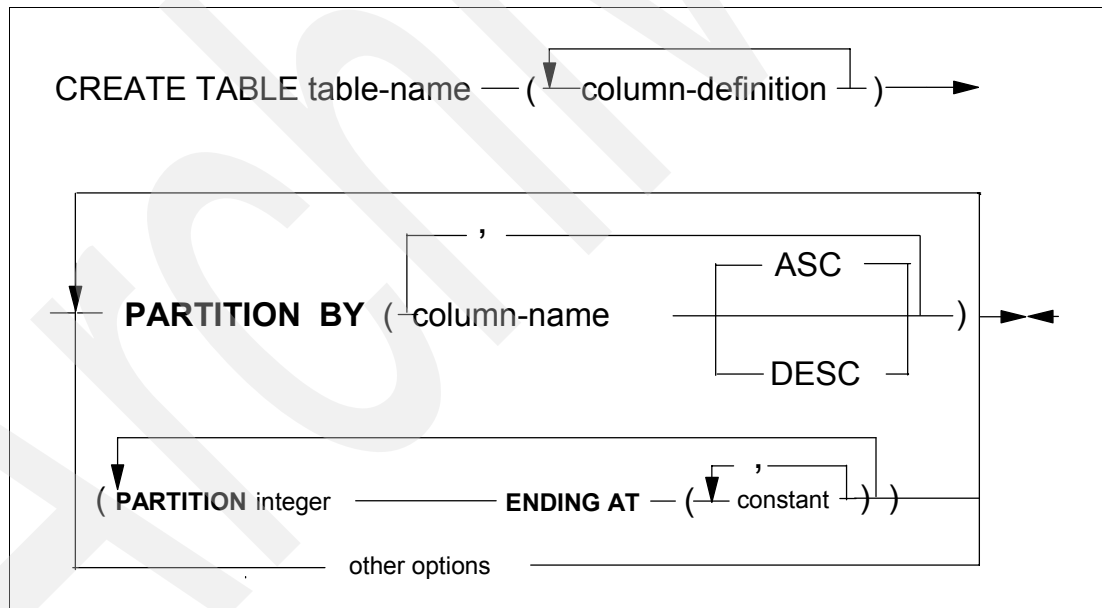


Figure 4-1 Table-controlled partitioning

Example 4-1 compares the index-controlled partitioning syntax with the new table-controlled partitioning syntax. Notice that when using table-controlled partitioning, an index is not required.

Example 4-1 Index-controlled vs. table-controlled partitioning syntax

Create table space statement has not changed

```
CREATE TABLESPACE tsname NUMPARTS n
  ( PART 1 USING PQTY...,
    PART n ...      );
```

Index-controlled partitioning

```
CREATE TABLE tname (col1,col2,col3...) ...;
CREATE INDEX ixname on tname (col1,col2) CLUSTER
  ( PART 1 VALUES (bnd1a,bnd1b) USING PQTY ...,
    PART n VALUES (bndna,bndnb) ...      );
```

Table-controlled partitioning

```
CREATE TABLE tname (col1,col2,col3...) ....
  PARTITION BY (col1,col2) ...
  (PARTITION 1 ENDING AT (bnd1a,bnd1b),
   PARTITION n ENDING AT (bndna,bndnb));
```

Partitioning control specifics

When creating a partitioned table (that is, the table space has a NUMPARTS specification), table-controlled partitioning is initiated by specifying the new PARTITION BY clause, which identifies the columns and values used to delimit the partition boundaries. When the new clause is used, the definition of the table is complete and data can be inserted into the table. Once establishing 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 keyword is disallowed.

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 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.

Converting to table-controlled partitioning

For tables that use index-controlled partitioning created in DB2 V8 or in previous releases, the use of any of the statements listed in Example 4-2 automatically converts the table to table-controlled partitioning.

Example 4-2 Converting from index-controlled to table-controlled partitioning

```
DROP partitioning index
ALTER INDEX NOT CLUSTER (on the partitioning index)
ALTER TABLE ADD PARTITION
ALTER TABLE ALTER PARTITION ROTATE
ALTER TABLE ALTER PARTITION part
CREATE INDEX PARTITIONED
CREATE INDEX ENDING AT... omitting CLUSTER keyword
```

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

1. Creating a Data Partitioned Secondary Index (CREATE INDEX PARTITIONED) with DEFER YES. The CREATE INDEX PARTITIONED syntax triggers conversion to table-controlled partitioning, and the index is left in RBDP. There is no loss of availability.
2. Dropping the newly created (empty) index.

4.1.2 Index-controlled partitioning terminology

In DB2 V7 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. This occurs because the index that defines the key ranges for the different partitions is the partitioning index, it is necessarily partitioned (made up of different physical partitions), and it is also the clustering index, as shown in Figure 4-2.

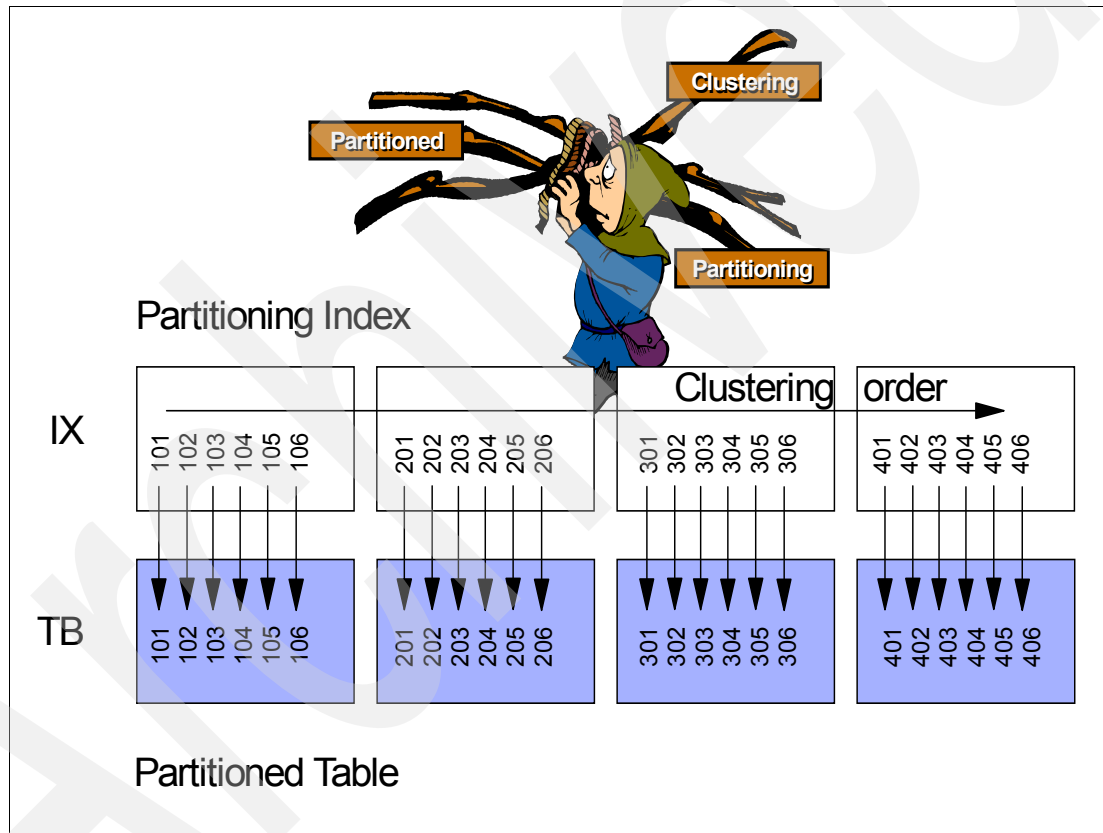


Figure 4-2 Index-controlled partitioning terminology — 1

Any other index on an index-controlled partitioned table is called a secondary index, and it is non-partitioned, that is, not split up into multiple partitions (although you can define pieces for those indexes to improve I/O performance). This is shown in Figure 4-3. Notice that logical partitions do exist for the secondary index, but they are only used by utilities to claim and drain.

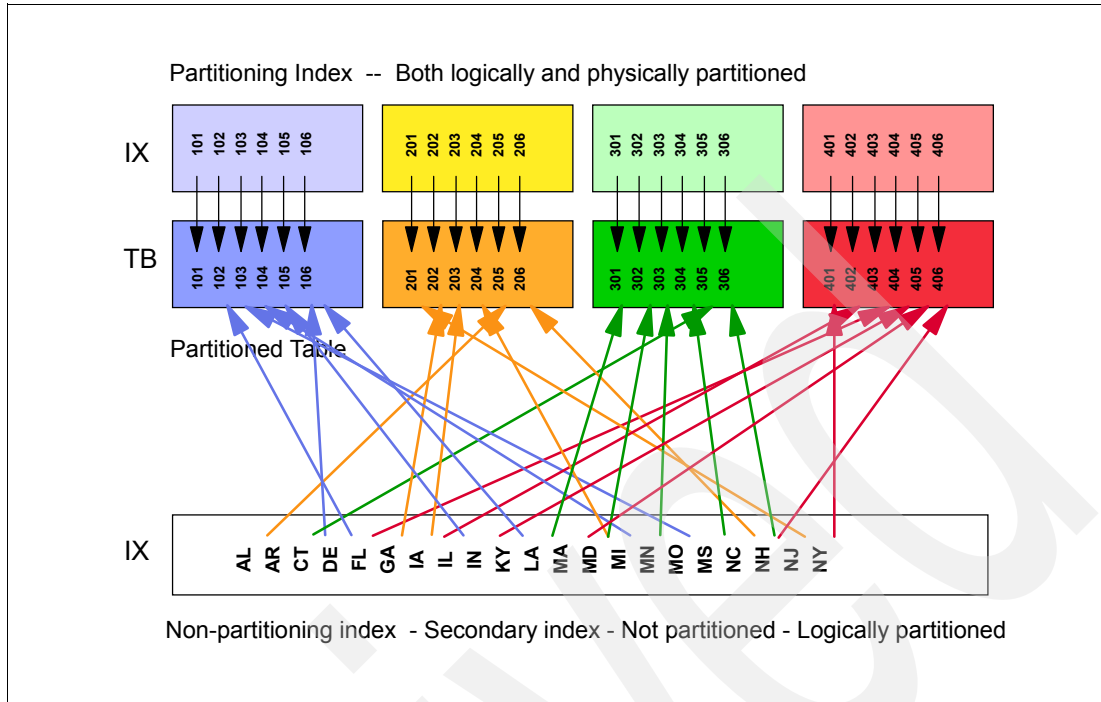


Figure 4-3 Index-controlled partitioning terminology — 2

4.1.3 Table-controlled partitioning terminology

By using table-controlled partitioning, clustering, being partitioned and being the partitioning index are 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, as shown in Figure 4-4.

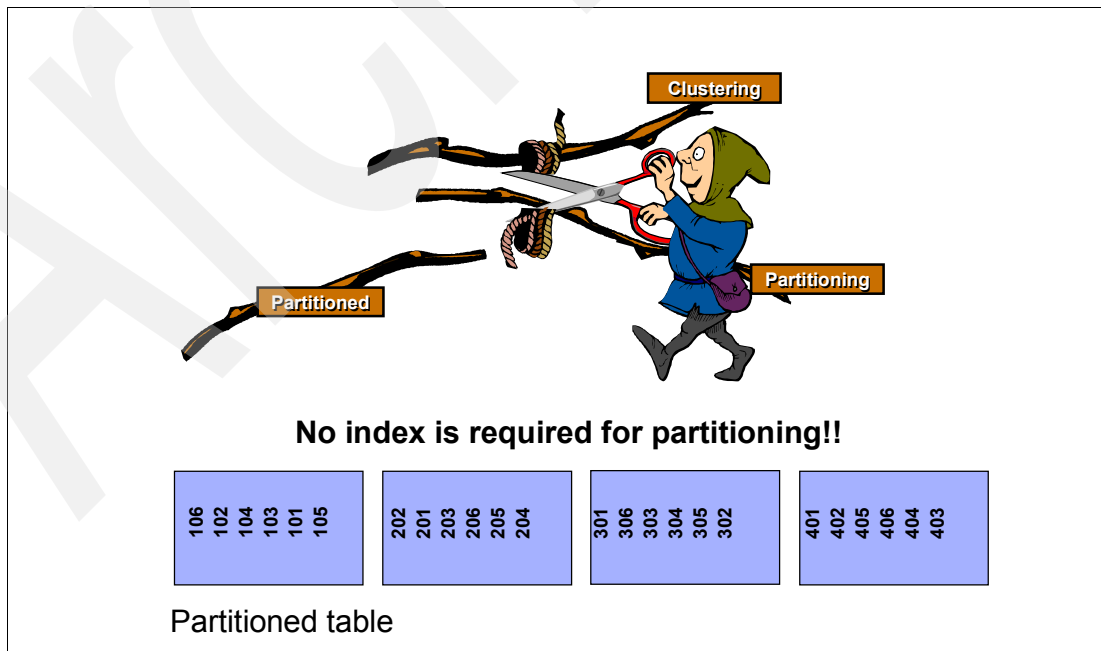


Figure 4-4 Table-controlled partitioning terminology

Index classification

Indexes on table-controlled partitioned tables can be classified as follows:

- ▶ Based on whether or not an index is physically partitioned:
 - Partitioned index** The index is made up of multiple physical partitions (not just index *pieces*)
 - Non-partitioned index** The index is a single physical data set (or multiple pieces)
- ▶ Based on whether or not the columns in the index correlate with the partitioning columns of the table (the leftmost partitioning columns of the index are those specified in the PARTITION BY clause of the table):
 - Partitioning index** The columns in the index are the same as (and have the same collating sequence), or start with the column(s) in the PARTITION BY clause of the CREATE TABLE statement.
 - Secondary index** Any index where the columns do not coincide with the partitioning columns of the table. These are dealt with in more detail in 4.2, “Data partitioned secondary indexes” on page 46.
- ▶ Based on whether or not the index determines the **clustering** of the data. Please note that when using table-controlled partitioning:
 - Clustering index** The index determines the order in which the rows are stored in the partitioned table.
 - Non-clustering index** The index does not determine the data order in the partitioned table.

Now let us look at a few examples to illustrate their characteristics.

Figure 4-5 shows the difference between a partitioned and a non-partitioned index. The partitioned index is made up of multiple partitions, one per data partition.

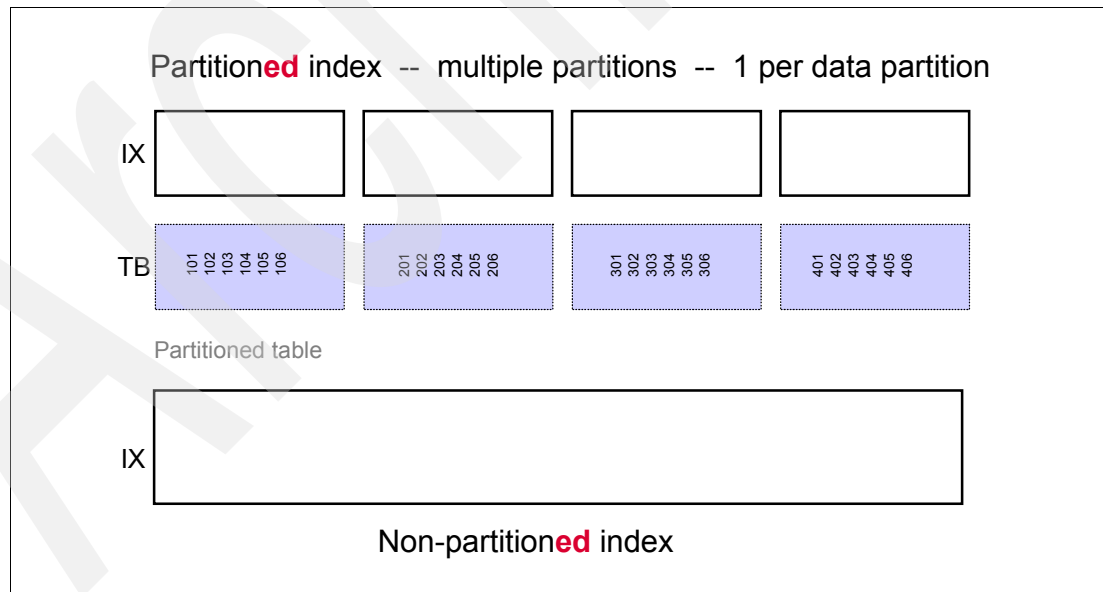


Figure 4-5 Partitioned versus non-partitioned index

Figure 4-6 shows what a partitioning (and in this case partitioned) index looks like. The left-most columns in the index have to be the same as the columns in the PARTITION BY clause, including the collating sequence, to qualify as a partitioning index. Note that starting in Version 8, a partitioning index can be partitioned (part_IX_1) or non-partitioned (part_IX_2).

A **partitioning** index has the same left-most columns, in the same collating sequence, as the columns which partition the table

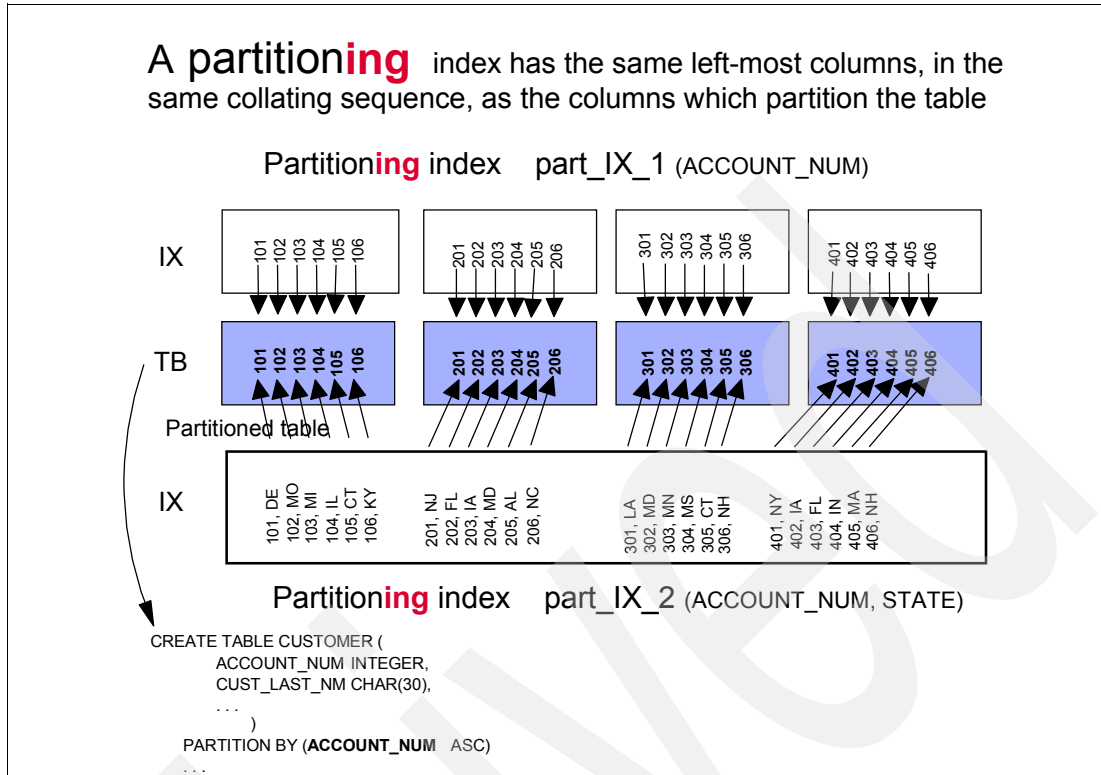


Figure 4-6 Partitioning index

Figure 4-7 shows a clustering index. In this example, the clustering index is also the partitioning index (in previous versions of DB2, this was always the case.) and it is also partitioned. It could also be defined as UNIQUE.

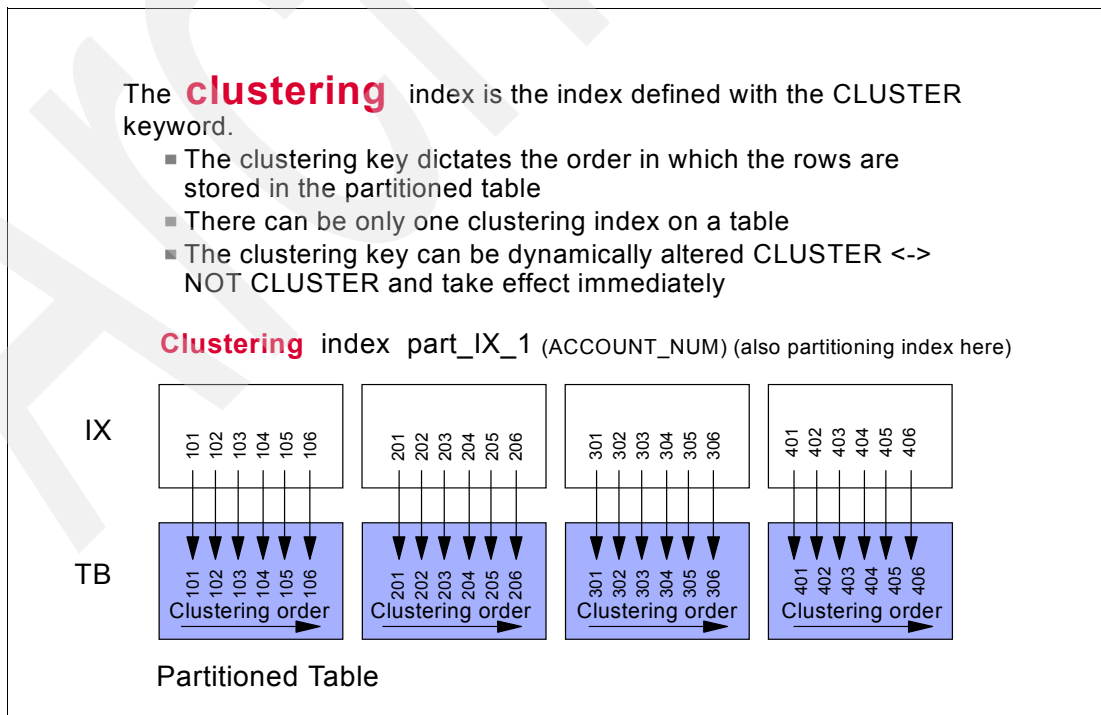


Figure 4-7 Clustering index

However, starting with DB2 V8, a secondary index can also be defined as the clustering index. This is shown in Figure 4-8. In this case we choose 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 column.

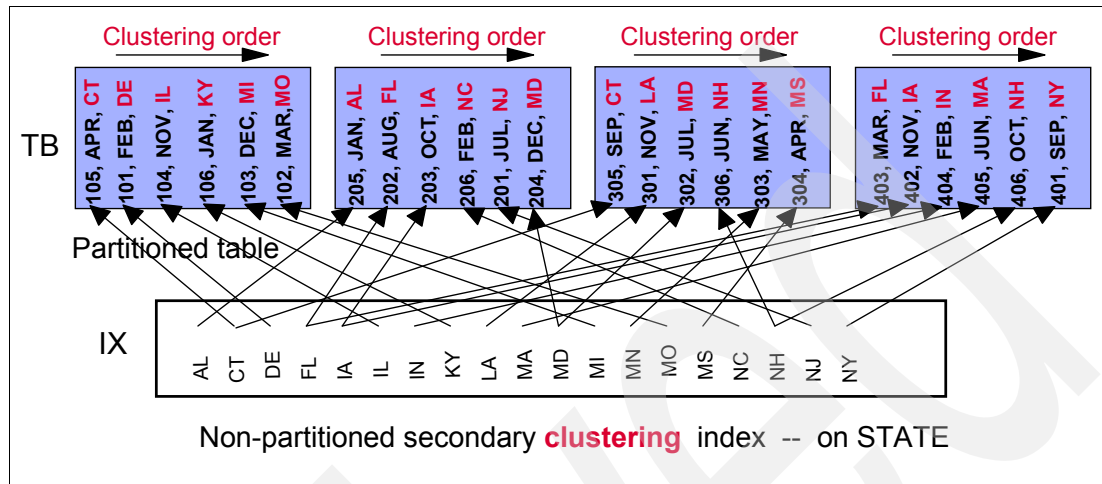


Figure 4-8 Secondary index used as clustering index

Keep in mind that with the unbundling of partitioning and clustering, all the combinations are now possible for any index.

4.2 Data partitioned secondary indexes

With DB2 V7, secondary indexes on partitioned tables are non-partitioned indexes. Secondary indexes cannot be physically partitioned. They can only be allocated across multiple *pieces* to reduce I/O contention.

DB2 V8 introduces 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 partitions in the secondary index as table space partitions, and index keys in partition 'n' of the index reference, but only data in partition 'n' of the table space. Such an index is called a *Data Partitioned Secondary Index (DPSI)*.

4.2.1 Creating a data partitioned secondary index

You create a DPSI by specifying the new *PARTITIONED* keyword in the CREATE INDEX statement, shown in Figure 4-9, and defining keys on columns that do not coincide with the partitioning columns of the table. When defining a DPSI, the 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.

After creation, a DPSI looks somewhat like Figure 4-10. In this example, the index "data_part_si_1" is a data partitioned secondary index. In our example, the DPSI is also the clustering index, but that is not necessarily 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 related to the rows in the corresponding data partition.

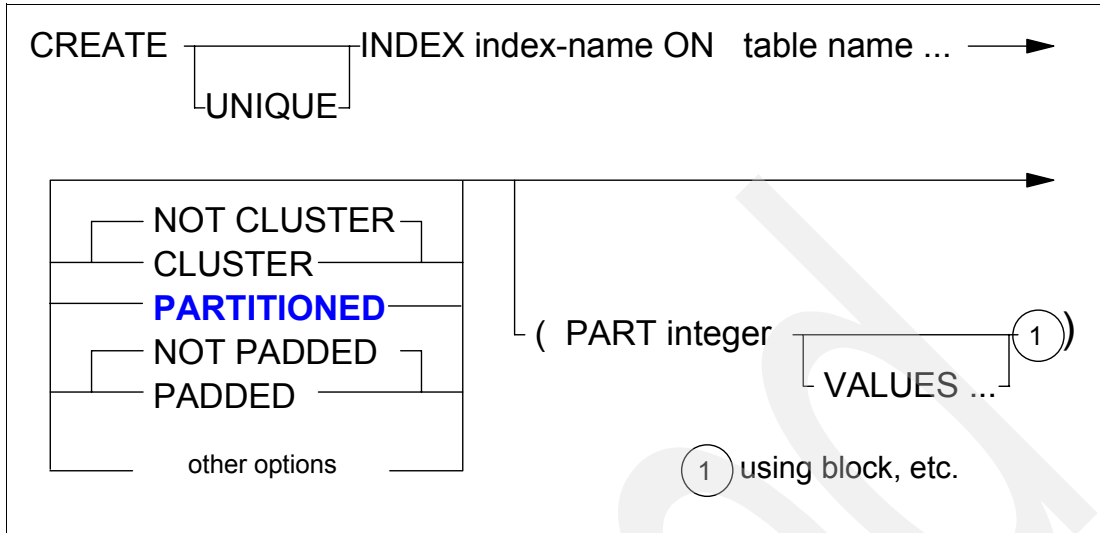


Figure 4-9 Creating a partitioned index

Figure 4-10 also shows a *Non-Partitioned Secondary Index (NPSI)* "non_part_si_2". 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. This diagram also illustrates that a single table may support a mix of non-partitioned and data-partitioned secondary indexes. However, for utility performance, it is best to have all data partitioned secondary indexes rather than non-partitioned secondary indexes.

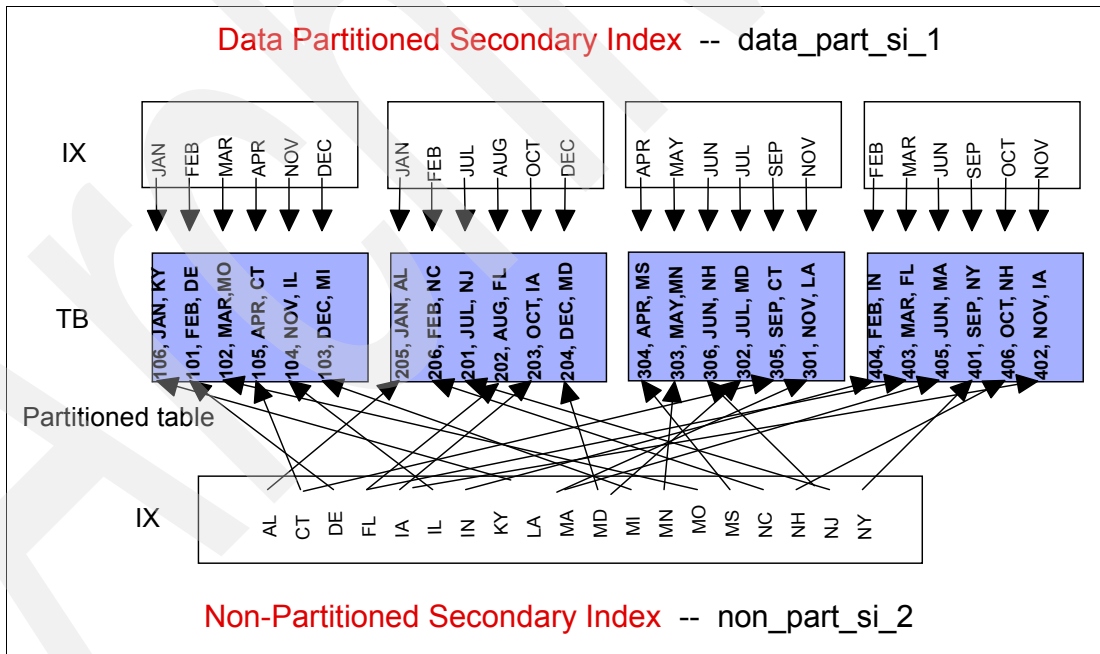


Figure 4-10 Data partitioned secondary index layout

4.2.2 The need for DPSIs

Indexes may be created on a table for one or several reasons — to enforce a uniqueness constraint, to achieve data clustering, but most likely, to provide access paths to data for queries or referential constraint enforcement. And 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 non-partitioned index to a table in a partitioned table space. This is because there are areas where non-partitioned indexes can cause performance and contention problems.

Availability

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

- ▶ **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.
- ▶ **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 be recovered. Furthermore, if partitioning is performed along lines meaningful to applications, any logical damage (by a wayward application, for example) can be isolated to certain partitions, again limiting the data outage and recovery scope.

These positive aspects of partitioning begin to deteriorate if there are non-partitioned indexes present. Here are some examples:

- ▶ A BUILD2 phase is performed during online REORG of a partition, when non-partitioned indexes exist on the table space. This phase uses the shadow index for the index's logical partition to correct RID values in the global index. 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, running 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 which is slower than append logic.
- ▶ Recovery from media failure is available for the entire index only. No piece-level rebuild or recovery is available.

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:

- ▶ Using DPSIs eliminates the need for a BUILD2 phase. 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.
- ▶ Using DPSIs also eliminates LOAD PART job contention and enables 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 (loads) DPSI keys into a separate index structure, in key order. This allows the insertion logic to follow an efficient append strategy.
- ▶ DPSIs 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.

Partition-level operations

One common partitioning scheme is to partition by date. Typically, the oldest data resides in part 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 you now want to do one of two things:

1. To discard the oldest partition's worth of data and reuse 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 reuse 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.

Partition-level operations become less clean if there are secondary (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 secondary indexes 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.

The use of DPSIs also streamlines partition-level operations such as adding and rotating partitions, also introduced in DB2 V8 (see 4.3, "Online schema changes" on page 50 and 7.1, "Online schema changes support" on page 170).

Data sharing overhead

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 belong to different data partitions are spread throughout the non-partitioned index.

Also in this case, DPSIs can come to the rescue. Because P-locking occurs at the physical partition level, affinity routing is effective for DPSIs because the parts of the DPSI are aligned with the data parts in the table. This is most beneficial during batch type applications with heavy insert/update/delete activity.

4.2.3 DPSI considerations

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.

For example, if you are aware of a correlation between DPSI and PI key values, you need to code the PI restriction explicitly when supplying a DPSI predicate to facilitate partition pruning. Let's 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, write queries on ORDERNO as queries on ORDERNO and DATE, with DATE restricted to reflect this policy. This allows the pruning of uninteresting partitions from the query.

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

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 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.

A more detailed account of utility and query processing of partitioned secondary indexes is given in Chapter 7, "Utilities" on page 169.

4.3 Online schema changes

Traditionally, customers needing to change object attributes have been subjected to periods of unavailability because of the methods that were required to implement those changes. DB2 V8 makes great strides in implementing schema changes without impacting data availability.

4.3.1 Online schema changes overview

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

- ▶ Better *application maintenance* is now possible through the use of DB2 packages and package versioning since Version 2 Release 3. By using package versions, you can prepare DB2 packages for the new version of the application in advance, and once the new application code gets activated, it automatically picks up the new version of the DB2 package, without any service interruption.
- ▶ Easier *code maintenance* is available with the introduction of DB2 data sharing. You can stop and start individual members (DB2 subsystems) to activate maintenance (PTFs) or a new DB2 release, while applications continue to use the active members of the data sharing group.
- ▶ Another area in which much work has been accomplished is to have the data available as much as possible. *Data* requires *maintenance* every so often. 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 (see Figure 4-11).

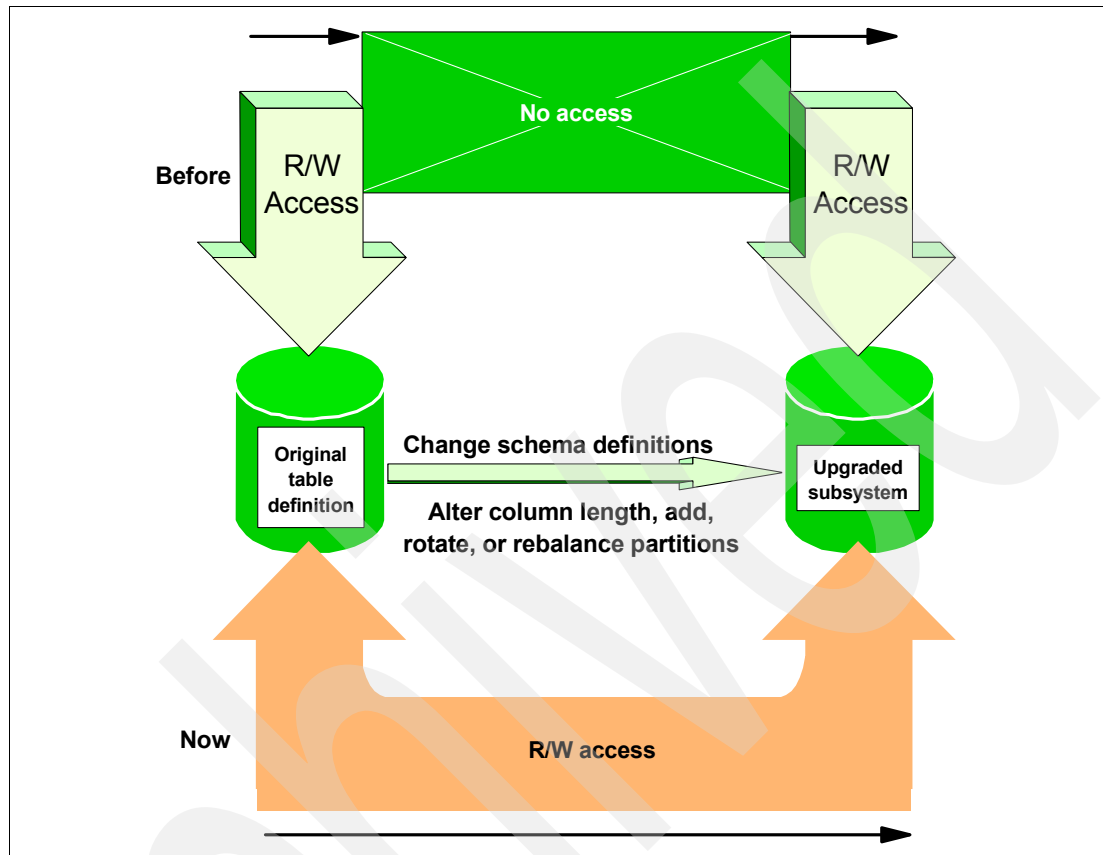


Figure 4-11 Online schema evolution

In the past, DB2 releases have implemented most DDL ALTER enhancements without actively addressing the problem of data unavailability while modifying object attributes.

Many changes to table, table space, and index schemas (DDL) in today's DB2 V7 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.
2. Drop the object
3. Create the object with the new definition
4. Reestablish authorization for the object
5. Recreate all dependent objects like views and indexes, etc. and their authorizations
6. Reload the data
7. Rebind plans and packages
8. Test that all is OK

However, some schema changes can already be done without having to drop and recreate an object, or stopping and starting the object, such as adding a column to a table, renaming a table, altering the primary and secondary quantity of an object, or changing partition boundaries.

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, DOUBLE, DECIMAL)
- ▶ Change type 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/alter an index to have non-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
- ▶ Support REORG of parts in REORG pending states
- ▶ Loosen the restrictiveness of indexes in recover or rebuild pending states

4.3.2 Table data type changes

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. Likewise, when a data row is modified or inserted, the entire row is saved using the new catalog definition.

When the object is reorganized, all rows are converted into the format of the latest version (see 4.3.4, "Versioning" on page 56).

To support changing the column data type of a column in an existing table, the SET DATATYPE clause of the ALTER TABLE ALTER COLUMN was enhanced to support these additional changes. The command is shown in Figure 4-12.

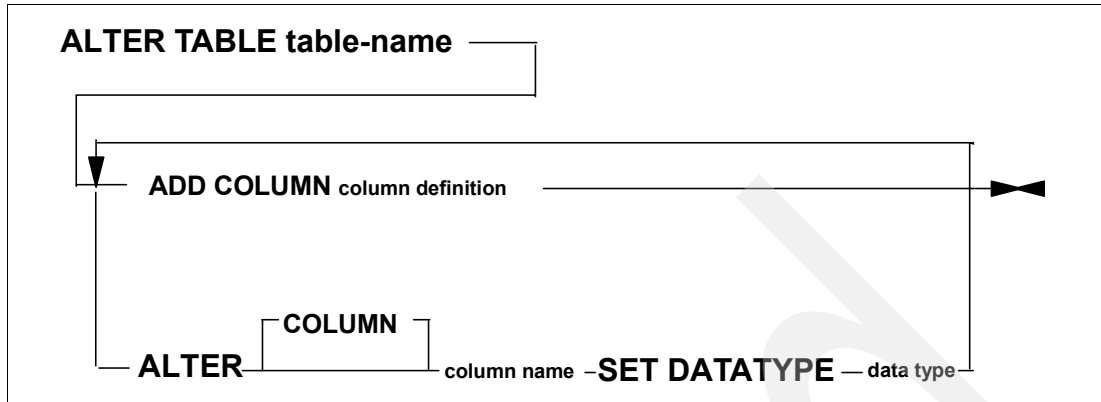


Figure 4-12 ALTER TABLE SET DATATYPE statement

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.

Impact on dependent objects

The ALTER TABLE SET DATATYPE has different implications for different types of objects affected by it.

Table spaces

When the ALTER completes, the table space is placed in an Advisory Reorg Pending (AREO*) state. See 4.3.6, “New DBET states for online schema changes” on page 59 for more details on this new database exception state. Access to a table can continue with rows containing columns in multiple version formats, but there is a performance degradation since altered columns have to be converted to the new format when they are accessed, and the full row is always logged, until the table space is reorganized. To reduce the performance impact, it is recommended to schedule a REORG after issuing the ALTER statement that changes the data type of a column in a table.

Indexes

When the data type or length of a column is altered on a table and that column is defined for an index, the index is altered accordingly. 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. For more information on versioning, see 4.3.4, “Versioning” on page 56. If a table has multiple indexes, the change of 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.

All new keys inserted are in the new index format. Changed columns which are included as part of an index key affect availability of the index according to the column data type. 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, 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.

With DB2 V8 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.

Changes for character data type columns result in immediate index availability. This includes columns defined as CHAR, VARCHAR, GRAPHIC, or VARGRAPHIC.

► **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.

Changes for numeric data type columns are immediate with delayed index availability. This includes columns defined as SMALLINT, INTEGER, DECIMAL or NUMERIC, FLOAT, REAL, or DOUBLE.

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 materializes all keys to the format of the latest version unless the index is in RBDP. In this state, access to the data is required to get the length of the key.

Scope of unavailability

To limit the scope of unavailability for dynamic SQL:

- 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 queries, DB2 does not choose an index in RBDP for an access path.

Runstats

Some of the statistics get converted at the time of the ALTER (for instance, HIGH2KEY, LOW2KEY). Invalidation is done for distribution statistics in SYSCOLDISTSTATS and SYSCOLDIST, and DB2 sets the STATSTIME in SYSCOLUMNS to January 1, 0001, which signals the optimizer to ignore the distribution frequency statistics.

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 previously.

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 which 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 is returned.

4.3.3 Index changes

In DB2 V8, some index attributes can also be changed “in-flight” using an ALTER INDEX statement, without causing the index to become unavailable, as shown in Figure 4-13.

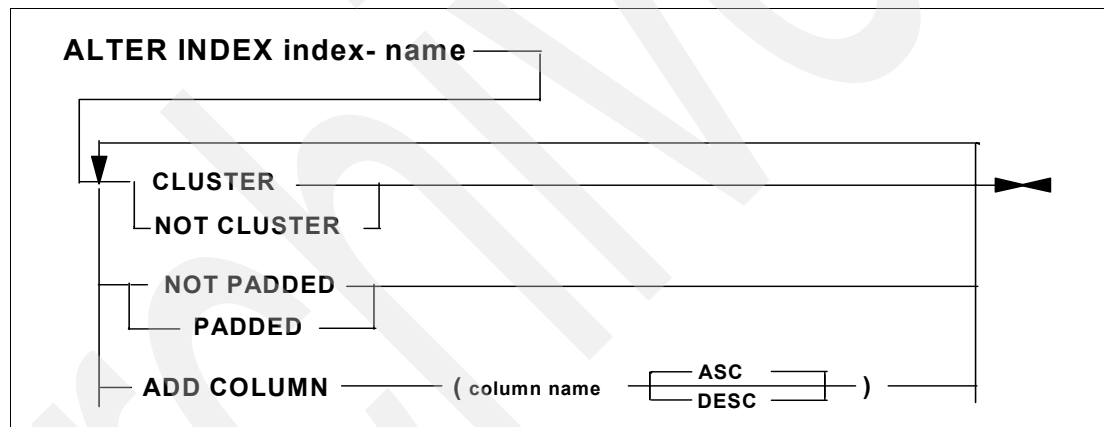


Figure 4-13 ALTER INDEX statement

Adding index columns

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 for the table is added 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 left in a Rebuild Pending state (RBDP). This could be the case when you add an existing column to an index.

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 optionally dropping an index, and then creating a new index with the column. When creating a new index, there is always a period of unavailability while the index is being created.

Varying length index keys

In DB2 V8, 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 or altering 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 for new V8 installations, while PADDED is the default when migrating from V7 for compatibility reason. A new ZPARM, PADIX, can change the default. The application and performance implications of this enhancement are explained in 8.6, “Long and variable length keys” on page 225.

An index can be changed from *PADDED* to *NOT PADDED* using ALTER INDEX, as shown in the diagram in Figure 4-13. If the index has varying length columns, it is placed in a pending state and a value of 'N' is placed in the PADDED column of SYSINDEXES. Once the index has been rebuilt, all the keys are varying length, and the pending state is reset.

In a similar way, an index can also be changed from *NOT PADDED* to *PADDED* using ALTER INDEX. If the index has varying length columns, it is placed in a pending state and a value of 'Y' is placed in the PADDED column of SYSINDEXES. Once the indexed has been rebuilt, all the keys are padded to the maximum length, and the pending state is reset.

Changing the table clustering

In V8, there are two enhancements related to clustering:

- ▶ Specifying the CLUSTER keyword for a secondary index in a partitioned table space. Historically, the partitioning index for partitioned tables also had to be the clustering index. These two attributes are now unbundled so that the clustering attribute can be assigned to a secondary index. Also see 4.1.1, “Table-controlled partitioning” on page 40 for more information.
- ▶ Changing the clustering order in a partitioned or non-partitioned table space without dropping the index. The clustering attribute of an index can be modified by using the *CLUSTER* and *NOT CLUSTER* options of *ALTER INDEX*. As before, only one clustering index is allowed for any table.

If no explicit clustering index is specified for a table, the REORG utility now 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.

When the clustering index is changed, new INSERTs are immediately placed using the new clustering order. Preexisting data rows are not affected until the next reorganization rearranges them all to be in clustering order.

4.3.4 Versioning

To support online schema evolution, DB2 has implemented a new architecture, called *versioning*, 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 which 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 in some cases 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 NOT PADDED
ALTER INDEX index-name PADDED
ALTER INDEX index-name ADD COLUMN column-name
```

Multiple ALTER COLUMN SET DATA TYPE statements in the same unit of work 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 ALTER PARTITION ROTATE FIRST TO LAST
ALTER TABLE table-name ADD PARTITIONING KEY column-name
ALTER INDEX index-name NOT CLUSTER
ALTER INDEX index-name CLUSTER
```

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.

Version limits

A table space can have up to 256 different active versions while an index can have up to 16 different “active” versions (“active” versions include those within the pageset and all available image copies).

The range of *active versions*, which are 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 an SQLCODE -4702.

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 system pages.

Version information in the DB2 catalog

As can be seen in Figure 4-14, 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, when there is more than one active version, contains one row for each OBD or index that can be recovered to an image copy that was made before the first version was generated for that OBD or index.

Oldest for pageset and all available copies - updated by utilities such as MODIFY and REORG

Used to allocate next number

	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 table in a segmented table space will be assigned a different version number

Figure 4-14 Versioning information in the DB2 catalog

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. Storing the version information inside the page set, makes the objects self-defining. System pages can be included in incremental image copies if the SYSTEMPAGES YES keyword is specified.

Reclaiming versions

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 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, 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.

4.3.5 Usage considerations

As mentioned before, DB2 V8 takes the first steps to avoid outages due to schema changes. Certain restrictions are still in place, such as:

- ▶ 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, DATE, TIME, TIMESTAMP and FOR BIT DATA columns.
- ▶ Data type and lengths cannot be altered when:
 - The column is part of a referential constraint
 - An EDITPROC or VALIDPROC exists on the table
 - The table is referenced in the definition of a materialized query table
 - The column is defined as an identity column

When using this feature, please make sure that you not forget to assess which programs need to be changed. Host variables, for example, may need to be extended to cater for the extra length.

To minimize any performance degradation, schedule a REORG as soon as possible after ALTER.

Rebuild any affected indexes and rebind plans and packages. This prevents the system from picking an inefficient access path during automatic rebind because the best suited index is currently not available.

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

4.3.6 New DBET states for online schema changes

In support of online schema changes, DB2 V8, besides using the existing RBDP state, introduces a new Database Exception Tables (DBET) state, Advisory Reorg (AREO*). AREO* indicates that the table space, index, or partition identified should be reorganized for optimal performance.

The DISPLAY DATABASE command now shows the new DBET state AREO* for all objects.

4.3.7 Impact of online schema changes on user tasks

In this section we discuss the effects of online schema changes on the major tasks involved in using DB2.

Database design and implementation

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. When designing applications, you 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.

Up until DB2 V7, for an application that requires inserting time based data, a separate partition is often assigned to store the data for each month. The design usually leans toward allocating the maximum number of partitions allowed up front. This probably meant 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.

With DB2 V8, it is much easier to add partitions at a later date, so the plan may be to start out with 24 partitions, and then reevaluate partition needs within the next 12 to 18 months. This results in “managing” many fewer objects which today may be preallocated without being immediately used. The use of templates and LISTDEFS in your utilities, by dynamically adding new partitions and objects, will also contribute to the overall flexibility and manageability of the schema changes.

When designing an application that requires storing the data for only a certain amount of time, for example for legal reasons, consider a rolling partitions design. Now that there is the ability to easily ROTATE and reuse partitions over time, it is easier to manage a limited partitions which are set up based upon dates.

Operational considerations

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.

When rotating partitions of a partitioned table, consider the time needed to complete the DDL statement. The reset operation requires that the keys for these deleted rows 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 elapsed 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 row at a time. Individual delete row processing is required for referential integrity relationships, when DATA CAPTURE is enabled, or when there are delete triggers.

Application programming

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.

4.3.8 Dynamic partitions

DB2 V8 has the ability to immediately add partitions, rotate partitions, and change the partitioning key values for table-controlled partitioned tables via the ALTER TABLE statement. Here we give a brief description of these enhancements. More detailed information can be found in 7.1, “Online schema changes support” on page 170.

Adding partitions

With Version 8, users are able to dynamically add partitions to a partitioned table. You can add partitions up to the maximum limit which is determined by the parameters specified when the partitioned table was initially created (see Table 3-1 on page 34). 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 predefined. The data sets for the data partition and all partitioned indexes must be defined using the VSAM access method services 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 part number is selected by DB2 based on the current number of partitions of the table.

The newly added partition is immediately available. However, you have to stop the table space and partitioned index before adding the partition.

The table is quiesced and all related plans, packages and cached statement are invalidated. This is necessary as access path may be optimized to read only certain partitions. Automatic rebinds will occur (if allowed), but you may wish to issues rebinds manually.

Since you cannot specify attributes like PRIQTY, the values of the previous logical partition are used. Therefore you probably want to run an ALTER TABLESPACE statements afterwards to provide accurate space parameters, before starting to use the newly added partition.

Rotating partitions

Rotating partitions allows old data to “roll off” while reusing the partition for new data with the ALTER TABLE ALTER PARTITION ROTATE FIRST TO LAST statement. A typical case is where 13 partitions are used to continuously keep the last 12 months of data. When rotating, one can specify that all the data rows in the oldest (or logically first) partition is to be deleted, and then specify a new table space high boundary so that the partition essentially becomes the last logical partition in sequence ready to hold the data which is added. Because the data of the partition being rolled off is deleted, you may want to consider running an unload job before rotating the partition.

If this REUSE option is specified, a logical reset of the partition is done instead of deleting and redefining data sets.

The partition that was rolled off is immediately available after the SQL statements is successfully executed. No REORG is necessary.

After using the new ALTER PARTITION ROTATE 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 which 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.

Changing partition boundaries

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 PARTITION ENDING AT statement. The affected data partitions are placed into Reorg Pending state (REORP) until they have been reorganized:

Rebalancing partitions

Rebalancing partitions is done through the 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 REORG status (like in the case of ALTER TABLE ALTER PARTITION... ENDING AT statement.)

You cannot specify REBALANCE with REORG TABLESPACE *SHRLEVEL CHANGE*. Also, do not specify partitioned table spaces with LOB columns. Also note that when the clustering sequence does not match the partitioning sequence, REORG must be run twice; once to move rows to the right partition; and secondly to sort in clustering sequence. DB2 leaves the table space in AREO* (Advisory Reorg Pending) state after the first REORG, indicating that a second one is recommended. (More information about this new pending state can be found in 4.3.6, “New DBET states for online schema changes” on page 59.)

Upon completion, DB2 invalidates plans, packages, and the dynamic statement cache that reference the reorganized object.

For examples using the various functions implemented with dynamic partitions support, please refer to 7.1.2, “Utility support for schema evolution” on page 178.

4.4 System level point-in-time recovery

DB2 V8 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.

Two new utilities provide the vehicle for system-level point-in-time recovery (see also 7.5, “Backing up and restoring the system” on page 200):

- ▶ 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 is described in 4.4.1, “Backing up the system” on page 63. 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 32 KB page writes (for page sets that are not using a 32 KB CI size) and data set extensions.

The BACKUP SYSTEM and RESTORE SYSTEM utilities rely on new DFSMSshm services, and SMS constructs in z/OS V1R5 that automatically keep track of which volumes need to be copied.

You can specify the number of copy versions to be maintained on disk (max 15) by using the VERSIONS attribute.

Prerequisites for this feature

To use the BACKUP SYSTEM and RESTORE SYSTEM utilities, all data sets that you want to recover must be SMS-managed data sets. Additionally, you must meet the following requirements:

- ▶ z/OS V1R5 or above
- ▶ Disk control units that support the ESS FlashCopy ® API
- ▶ HSM constructs defined by using the DB2 naming convention
- ▶ Defined SMS copy target storage groups
- ▶ DB2 has to be running in NFM

4.4.1 Backing up the system

As mentioned before, 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*. When taking a FULL backup, the copies of the log are always taken after the database copies.
- ▶ **DATA ONLY:** This type of copy only contains the “database” data. Notice that a BACKUP SYSTEM DATA ONLY does NOT copy the “log” data. Such a copy is referred to as a *data-only system backup*.

Full or data only system backups can be used in conjunction with the new RESTORE SYSTEM utility to recover the system to an arbitrary PIT between system copies, or any RBA of the active log following the last backup. Full system backups could be used to recover the system to the point-in-time (PIT) at which the copy was taken, using normal DB2 restart recovery.

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 an LRSN (data sharing — the RBLP LRSN determined by taking minimum of all member level RBLP values) of the time the most recent system backup ran.

Figure 4-15 gives an overview of how the BACKUP SYSTEM utility operates.

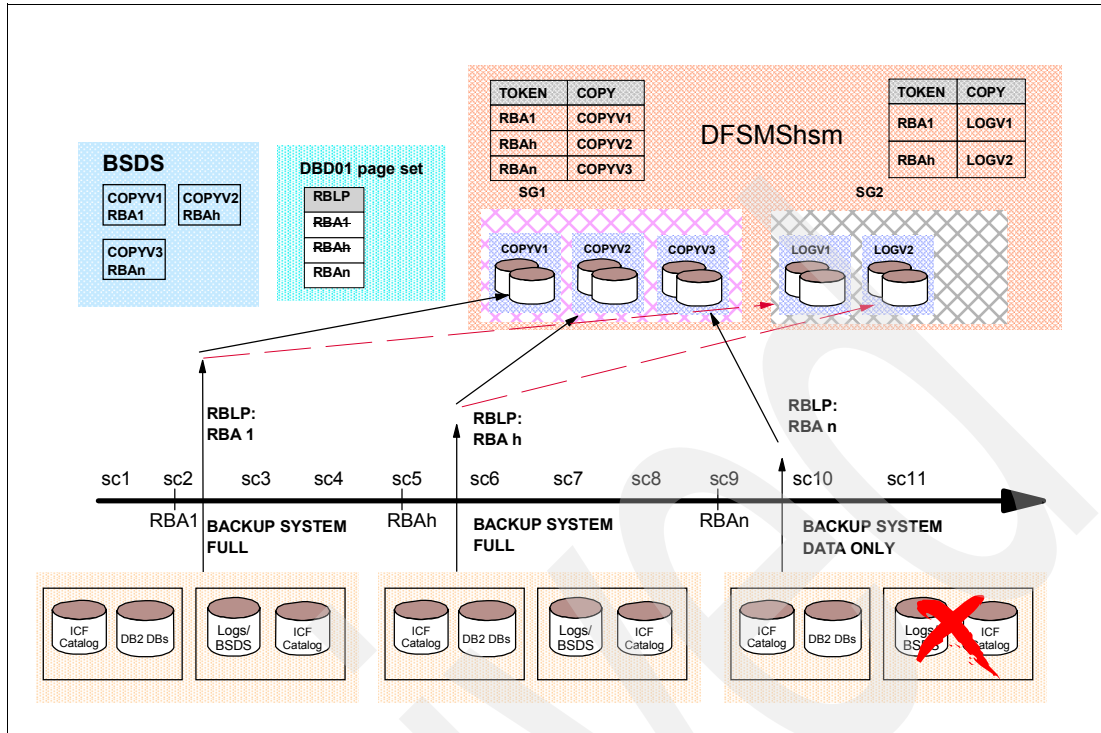


Figure 4-15 Backup system operations

BACKUP SYSTEM invokes DFSMSShsm services to either take a data-only or full system backup via volume copy functions. The backups may (and probably do) contain uncommitted data. This is not a problem since 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, the following DB2 system activities are quiesced:

- ▶ System checkpoints
- ▶ 32 KB page writes (if CIs are not 32 KB)
- ▶ Writing close page set control log records (PSCRs)
- ▶ Data set creation, extensions, renaming and deletions

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.

In a data sharing environment, the BACKUP SYSTEM utility fails 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 within a few seconds. Taking copies of the volumes is done in parallel.

4.4.2 Restoring the system

Depending on the type of system backup that is available, and the point-in-time (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 issuing the DFSMSHsm commands manually
 - iii. Issuing the SET LOG RESUME command
2. Stop DB2. In case of a data sharing group, stop all members.
3. Use HSM commands to restore the database and log data.
4. If data sharing, delete the CF structure.
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 “PITR” conditional restart control record (CRCR).

Creating a PITR 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 on 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.

You can specify a single option on the RESTORE SYSTEM utility. These are the possibilities:

- ▶ **No option specified:** In this case, RESTORE SYSTEM first restores the version of the database data (created by the BACKUP SYSTEM utility) that was taken immediately prior to the specified logpoint (on 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. It uses the recovery base log point that is recorded in DBD01 to determine the starting point of the log apply phase.

Figure 4-16 shows how a RESTORE SYSTEM utility is executed. In this example, the LOGONLY option is not used.

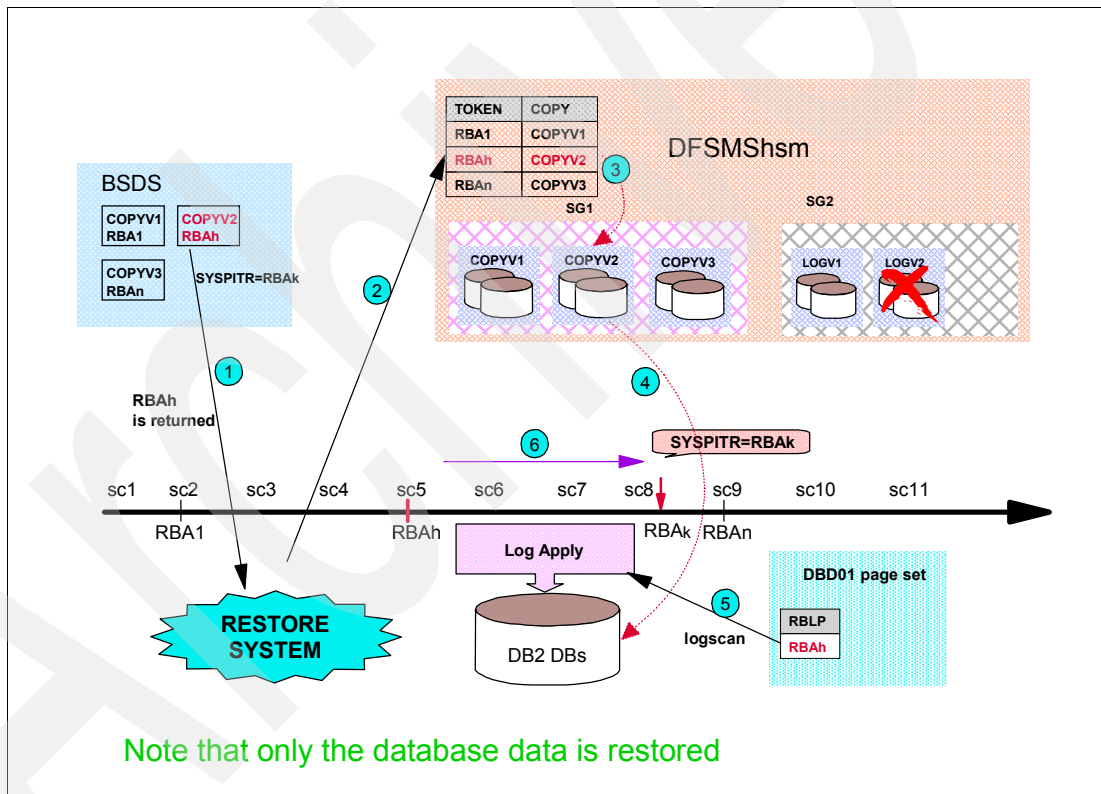


Figure 4-16 Restore system operation

Note: The RESTORE SYSTEM utility does not restore the log data. 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 DFMSHsm services, as shown in "Restoring a DB2 system to the PIT of a prior system level backup" on page 65.

As indicated above, each backup of data can have multiple VERSIONS. You cannot specify a specific desired version, other than implicitly via the log truncation point. RESTORE SYSTEM automatically recovers from the latest version prior to the log truncation point. The restore of the database volume from the backup copy 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 be put in RECP state, index spaces will be either put in RECP or in RBDP state, depending on their COPY attribute. These objects should than be recovered to a different point-in-time, prior to the log truncation point using image copies, or rebuild from the data in the case of a COPY NO index.

Note: With DB2 V8, 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, like an application or a single table space.

4.5 Online ZPARMs

In order to minimize the events which require recycling your DB2 subsystem to make ZPARM changes effective, DB2 V7 introduced online change for subsystem parameters. 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*, SC26-9934 for a list of online changeable subsystem parameters.

DB2 V8 adds some more online changeable parameters. Table 4-1 lists the ZPARMs made online changeable in Version 8, including those new to Version 8.

Table 4-1 New subsystem parameters changeable online with DB2 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
MAINTYPE			Materialized query table
PADNTSTR			Pad null terminating strings in output host variables
REFSHAGE			Current refresh age
EDMDBDC		DSNTIPC	Min DBD cache size
EDMSTMTC			Size of the statement cache
SRTPOOL			Sort pool size
XLKUPDLT		DSNTIPI	X lock for searched U/D
MAXKEEPD		DSNTIPE	Max kept dyn stmt
PADIX			Pad indexes (default)
PARTKEYU		DSNTIP4	Update part key cols
RESYNC		DSN6FAC	DSNTIPR
MAXTYPE1	Max type 1 inactive		
IDTHTOIN	Idle thread timeout		
POOLINAC	DSNTIP5		Pool thread timeout
TCPKPALV			TCP/IP keepalive
TCPALVER			TCP/IP already verified
IMMEDWRI	DSN6GRP	DSNTIP4	Immediate write
ACCUMACC	DSN6SYSP	DSNTIPN	DDF / RRSF ACCUM
UIFCIDS			Unicode IFCIDs
EXTRAREQ		DSNTIP5	Extra blocks — Req
EXTRASRV			Extra blocks — SRV
IXQTY		DSNTIP7	Index space allocation
TSQTY			Table space allocation
DSVCI			New CI size
SVOLARC		DSN6ARVP	DSNTIPA

For most parameters, online change is transparent, with the change taking effect immediately or as soon as DB2 can make them effective.

4.6 Other availability enhancements

DB2 V8 also takes actions to avoid a loss of availability, by trying to anticipate and warn you about potential availability problems, as well as to take more automatic action to reduce the outage to a minimum. We list here the following enhancements:

- ▶ Control intervals larger than 4 KB
- ▶ Monitor system checkpoints and log offload activity
- ▶ Log monitor long running UR backout
- ▶ Improved LPL recovery

4.6.1 Control intervals larger than 4 KB

DB2 table spaces and index spaces are defined as VSAM linear data sets. Up to DB2 V7 every page set has been allocated in control intervals of 4 KB, even though VSAM allows CI sizes multiple of 4 up to 32 KB for linear data sets, and DB2 has chained the CIs up to the required page size.

DB2 V8 introduces support for CI sizes of 8, 16, and 32 KB, activated by the default of the new DSVCI ZPARM in panel DSNTIP7 (see also “Use CI size > 4 KB” on page 244). This is valid for user defined and DB2 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 migrating will be later converted by the execution of Loads or Reorgs. 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 ENFM phase.

The new CI sizes reduce integrity exposures, relieve some restrictions on concurrent copy and the use of striping, and provide the potential for reducing elapsed time for table space scans.

4.6.2 Monitor system checkpoints and log offload activity

With DB2 V7, you had 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 to DB2 V6. Sometimes, however, there are situations where DB2 stopped taking system checkpoints or was stuck during its log offload activity. Since both situations might cause problems during operations, DB2 V8 provides you with new messages, which help you to identify problems faster. In addition a new 0335 IFCID record is produced if a statistics class 3 is active.

4.6.3 Log monitor long running UR backout

During restart processing, DB2 issues progress messages during the forward and backward phase of the restart. Message DSNR0311 is issued in 2 minute intervals, showing the current log RBA being processed and the target RBA to complete the restart phase.

With DB2 V8, a new progress message is issued during the *backout of postponed 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.

For *in-abort URs*, there is also a new progress message that is issued after every two minutes.

4.6.4 Improved LPL recovery

The LPL (logical page list) is a list of pages that are logically in error, kept in DBET by DB2. These pages 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.

Prior to DB2 V8, once a page is entered into the LPL, that page is inaccessible until it is recovered. There are two ways to recover pages which are in LPL:

- ▶ Use the `-START DATABASE` command with `SPACENAM` option
- ▶ Use the `RECOVER` utility to recover the page set or partition in question to the current point-in-time

However, during the execution of both the `START DATABASE` command, as well as the `RECOVER` utility, the entire page set or partition is unavailable.

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.

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 `STARTDATABASE` 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. To enforce the requirement that only one LPL recovery process is running at a time for a given pageset or partition, this is now done by using a new “LPL recovery” lock, instead of the drain(all) lock.

SQL

In this chapter we discuss the following topics:

- ▶ Long names
- ▶ SQL statements 2 MB long
- ▶ Dynamic scrollable cursors
- ▶ Common table expressions and recursive SQL
- ▶ Multi-row fetch and insert
- ▶ Get diagnostics
- ▶ Scalar fullselect
- ▶ Select from insert
- ▶ Qualified column names in INSERT and UPDATE
- ▶ Expressions in GROUP BY
- ▶ Multiple DISTINCT
- ▶ Sequences
- ▶ Identity columns enhancements
- ▶ Multilevel security
- ▶ MQSeries UDFs
- ▶ ASCII flag for compile

5.1 Long names

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.

Table 5-1 shows the new SQL identifier length limits in DB2 V8.

Table 5-1 SQL Identifier length limits

Item	Limit in bytes
Alias name	128
Authorization name	128
Auxiliary table name	128
Buffer pool name	8
Catalog name	8
Collection id	128
Column name	30
Condition name	128
Constraint name	128
Correlation name	128
Cursor name	30 for a DECLARE CURSOR WITH RETURN or 128 in any other context.
Database name	8
Distinct type name	Two part name, both parts are 128
External Java routine name	1305
Function name	Two part name, both parts are 128
Host identifier	64
Index key	2000
Index name	128
Location name	16
Package id	8 for a package and 128 for a trigger
Parameter name	128
Plan name	8
Predicates	32704
Procedure name	128
Program name	8

Item	Limit in bytes
Schema name	128
Sequence name	128
Specific name	128
SQL variable name	64
Statement name	128
Stogroup name	128
String constant	32704
Savepoint name	128
Synonym	128
Table name	128
Table space name	8
Trigger name	128
Version ID	64
View name	128

5.2 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. To do this, SQL statements passed to DB2 on PREPARE and EXECUTE IMMEDIATE statements will be passed in Character Large Objects (CLOBs). The catalog has had the ability to accommodate longer than 32 KB statements for a long time, since SQL statements are broken up into pieces, and stored in succeeding records with a sequence number to keep them in order.

EXECUTE IMMEDIATE: *host-variable* may now be specified as a CLOB or DBCLOB. When *host-variable* is specified as a VARCHAR or VARGRAPHIC, the maximum length of the contained SQL statement remains 32,767 bytes or 16,383 DBCS characters (32767 bytes), respectively. However, when a CLOB or DBCLOB is used, the maximum length of the SQL statement contained in the CLOB or DBCLOB is 2 MB or 1 MB respectively. If the SQL statement exceeds the limit described above, then SQLCODE of -101 is returned.

In Example 5-1 we illustrate how to use a CLOB host variable in the EXECUTE IMMEDIATE statement of a sample C program.

Example 5-1 C program using CLOB for host-variable on EXECUTE IMMEDIATE statement

```
main() {
    EXEC SQL BEGIN DECLARE SECTION;
        ....
    SQL TYPE IS CLOB(4K) string1;100
        ....
    EXEC SQL END DECLARE SECTION;
        ....
}
```

```

strcpy(string1.data,"UPDATE DSN8610.EMP SET SALARY = SALARY * 1.1");
string1.length = 44;
.....
EXEC SQL EXECUTE IMMEDIATE :string1;
.....

```

PREPARE: FROM *host-variable* may now be specified as a CLOB or DBCLOB. When *host-variable* is specified as a VARCHAR or VARGRAPHIC, maximum length of the contained SQL statement remains 32,767 bytes or 16,383 bytes, respectively. However, when a CLOB or DBCLOB is used, the maximum length of the SQL statement contained in the CLOB or DBCLOB is 2 Megabytes or 1 Megabyte respectively. If the SQL statement exceeds the limit described above, then SQLCODE of -101 is returned.

In Example 5-2 we illustrate the use of a DBCLOB host variable in the PREPARE statement of a sample COBOL program.

Example 5-2 COBOL program using DBCLOB for host-variable on PREPARE statement

```

IDENTIFICATION DIVISION.
.....
DATA DIVISION.
.....
WORKING-STORAGE SECTION.
01 USTRING SQL TYPE IS DBCLOB(3K).100
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 that CLOB or DBCLOB cannot be used for ATTRIBUTES host-variable

When the precompiler option of NEWFUN(YES) is specified, the V8 Precompiler produces a DBRM in V8 format and support for greater than 32 KB SQL statements. Since V8 DBRMs are unusable by prior releases due to the Unicode enhancement, the fact that an application program precompiled by the V8 Precompiler with NEWFUN(YES) cannot be used in prior releases is moot. An application program cannot be run until its DBRM has been bound, and a V8 DBRM cannot be bound on a prior release.

When the precompiler option of NEWFUN(NO) is used, the V8 Precompiler produces a DBRM in V7 format and with no support for greater than 32 KB SQL statements.

Distributed data

Remote support for large SQL statements requires requester and server support for the new database protocol version, *The Open Group DRDA Version 3 Technical Standard*. DB2 and DB2 Connect is adding support for this new version of the database protocol and can flow large SQL statements.

SQL statements longer than 32 KB prepared at the requester can be sent to the server. However, distributed private protocol does not provide support for large objects.

Sample programs, tools, and utilities

With DB2 V8, the sample programs DSNTEP2 and DSNTIAUL are modified to handle greater than 32 KB SQL statements. DSNTIAD has not been modified since it is used by the installation procedure in compatibility mode. Utilities which imbed SQL statements are also modified, as well as the REXX support, so that they can handle greater than 32 KB SQL statements. SPUFI has also been modified to handle greater than 32 KB SQL statements.

5.3 Dynamic scrollable cursors

DB2 V7 introduced static scrollable cursors; see Figure 5-1. 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. Scrolling is performed in both the forward direction and backward direction. DB2 deletes the result table when the cursor is closed. DB2 V7 only supports static scrollable cursors.

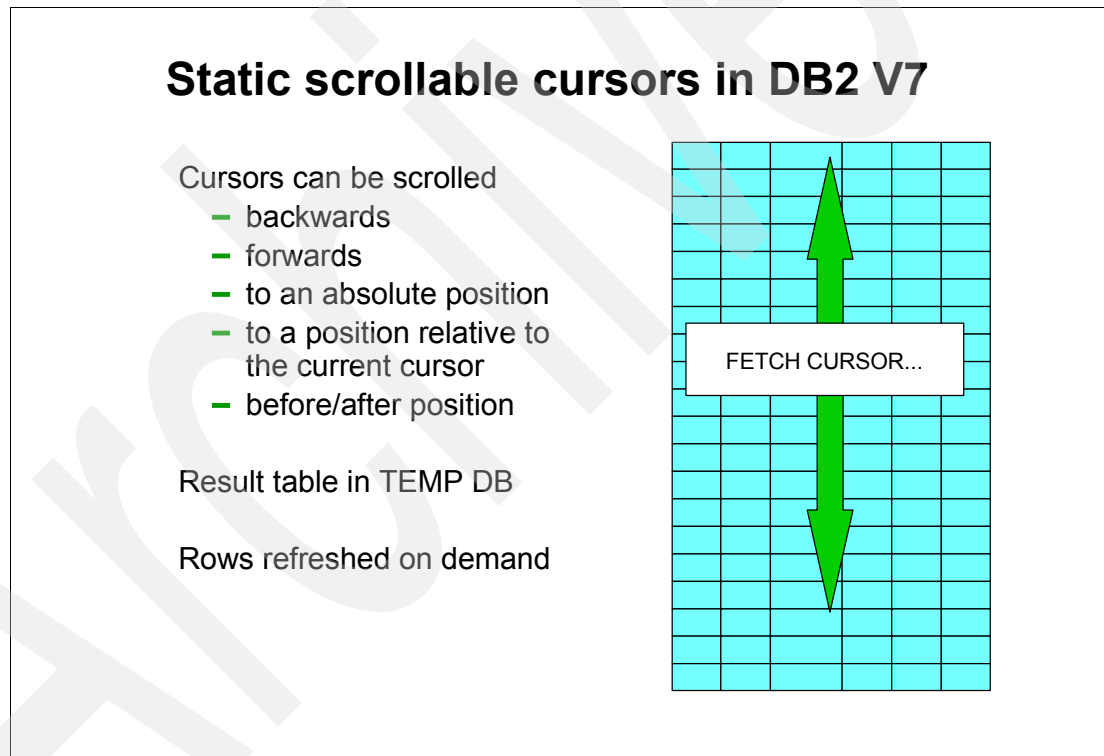


Figure 5-1 Scrollable cursors in DB2 V7

V7 introduced keywords to control whether the data in the result set is maintained with the actual rows in the base table.

You can declare a scrollable cursor in one of the following two levels of awareness dictated by the combination of SENSITIVE STATIC in the DECLARE CURSOR statement and whether INSENSITIVE or SENSITIVE is defined in the FETCH statement; see Figure 5-2.

► **INSENSITIVE:**

This means that the cursor is read-only and not interested in changes made to the base data once the cursor is opened.

- The number and content of the rows stored in the result table is fixed when the cursor is opened and does not change with changes to the base data.
 - FETCH processing on the result table is insensitive to changes made to the base table after the result table has been built.
 - The cursor cannot be used to issue positioned updates and deletes
- SENSITIVE STATIC:
- This means that the cursor is interested in changes which may be made after the cursor is opened.
- The number rows stored in the result table is fixed when the cursor is opened. However, the content of the rows can change.
 - FETCH processing on the result table is sensitive (to varying degrees) to changes made to the base table after the result table has been built.
 - The cursor can be used to issue positioned updates and deletes

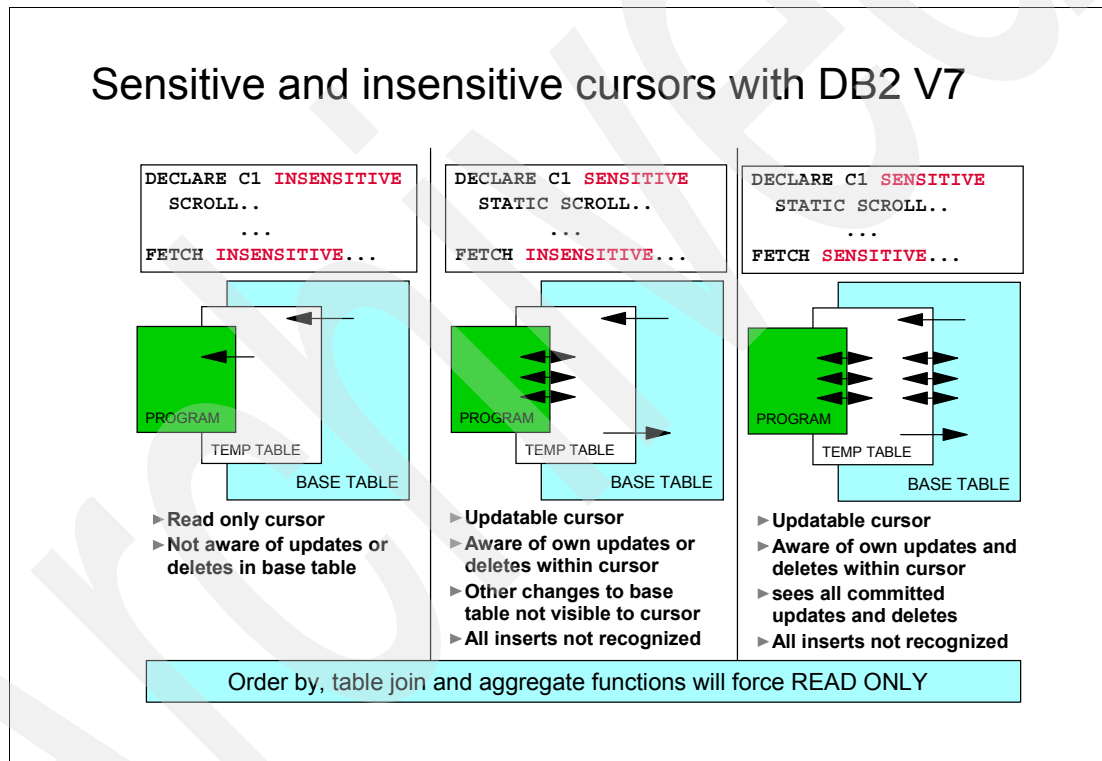


Figure 5-2 Sensitive and insensitive cursors with DB2 V7

V7 provides the static scrollable cursor function in which the result table of the SELECT statement associated with the cursor was materialized in a declared global temporary table and the scrolling was performed on the temporary table.

DB2 V8 introduces *dynamic scrollable cursors*. A dynamic scrollable cursor enables backward index scan and backward sequential detect to avoid sort.

The dynamic scrollable cursor is defined as:

```

DECLARE C1 SENSITIVE DYNAMIC SCROLL CURSOR
FOR SELECT C1, C2 FROM T1;
```

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 access path and the table space scan access path. The data-partitioned secondary indexes (DPSI) also support dynamic scrolling.

You can declare a scrollable cursor as INSENSITIVE, SENSITIVE STATIC, or SENSITIVE DYNAMIC. The syntax is reported in Figure 5-3.

Note: Dynamic scrolling does not apply if a result table has to be materialized at the time cursor is opened.

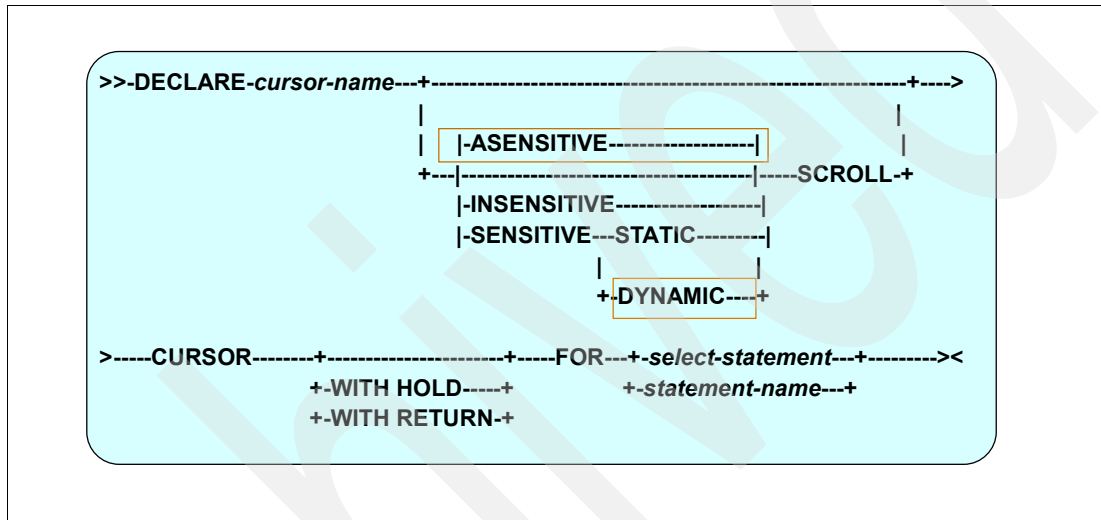


Figure 5-3 Scrollable DECLARE CURSOR syntax

For client applications that do not care whether or not the server supports the sensitivity or scrollability, DB2 uses the default option ASENSITIVE to determine whether the cursor behaves as SENSITIVE DYNAMIC or INSENSITIVE depending on the complexity of the associated SELECT statement. If the cursor is specified as ASENSITIVE and is read-only, it behaves as an INSENSITIVE cursor. If the cursor is specified as ASENSITIVE and is not read-only, DB2 provides the maximum allowable sensitivity, which is SENSITIVE DYNAMIC.

Figure 5-4 summarizes the cursor types characteristics.

Cursor type	Result table	Visibility of own changes	Visibility of others' changes	Updatability
Non-Scrollable (SQL contains a Join or Sort or 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

Figure 5-4 Cursor types comparison

Regarding updatability, note that a cursor can become read-only if the SELECT statement references more than one table, or contains a GROUP BY. In Example 5-3 we list some examples of syntax for scrollable cursors.

Example 5-3 Examples of using scrollable cursors

-
- ▶ **Declare cursor**

```
DECLARE CURSOR ORDERSCROLL SENSITIVE DYNAMIC SCROLL FOR
  SELECT ORDERNUM, CUSTNAME, ORDERAMT, ORDERDATE FROM ORDERS
  WHERE ORDERAMT > 1 FOR UPDATE OF COMMENTS;
```
 - ▶ **Open cursor**

```
OPEN CURSOR ORDERSCROLL;
```
 - ▶ **Fetch forward**

```
LOOP-TO-FILL SCREEN
DO 3 TIMES
  FETCH FROM ORDERSCROLL INTO :hv1, :hv2, :hv3, :hv4;
END
```
 - ▶ **Fetch RELATIVE**

```
SKIP-FORWARD-1-ROWS
  FETCH RELATIVE +1 FROM ORDERSCROLL INTO :hv1, :hv2, :hv3, :hv4;

SKIP-BACKWARD-5-ROWS
  FETCH RELATIVE -5 FROM ORDERSCROLL INTO :hv1, :hv2, :hv3, :hv4;
```
 - ▶ **Fetch ABSOLUTE**

```
RE-READ-THE-THIRD-ROW
  FETCH ABSOLUTE + 3 FROM ORDERSCROLL INTO :hv1, :hv2, :hv3, :hv4;
```
 - ▶ **Execute a positioned Update through scrollable cursor**

```
UPDATE-THE-CURRENT-ROW
  UPDATE ORDERS SET COMMENTS = "Expedite"
  WHERE CURRENT OF ORDERSCROLL;
```
 - ▶ **Close the scrollable cursor**

```
CLOSE CURSOR ORDERSCROLL;
```
-

5.3.1 Cursor positioning and serialization

At OPEN CURSOR, the cursor is positioned before the first row. After FETCH, the fetched row becomes the current row and the cursor is positioned on the current row. On any cursor, scrollable or non-scrollable, the cursor is positioned on the most recently fetched row. At this point, the UPDATE or DELETE WHERE CURRENT OF statement operates on the current row under the existing rules of positioned updates/deletes.

When FETCH returns 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 executed). With dynamic scrollable cursors, the most recently fetched row from the base table remains locked in order to maintain its position for a positioned update or delete, and there is no optimistic locking.

Dynamic scrollable cursor is useful when it is important to the application to see the updated rows and the newly inserted rows, and there is no need to see deleted rows. If the application requires constant result table, a SENSITIVE STATIC scrollable cursor with ISOLATION CS is recommended.

Optimistic locking was introduced with SENSITIVE STATIC in DB2 V7. With the SENSITIVE STATIC scrollable cursor, under ISOLATION CS, DB2 does not keep locks on rows or pages after the result table is materialized into the declared temporary table. DB2 also releases subsequently acquired locks for FETCH SENSITIVE requests. Locks are not held on the current row because DB2 is optimistic that no positioned update/delete will occur. Even if it does occur, DB2 is optimistic that the values in SELECT list will not have changed, so it compares temporary result table values with current values under a new lock to ensure data integrity before allowing positioned update/delete. With static scrollable cursors, with ISOLATION CS, there are no locks held after the cursor is opened, and the row is unlocked after each FETCH. Subsequent positioned updates or deletes use optimistic locking.

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.

5.3.2 Considerations

You should be aware of the following important considerations:

- ▶ A dynamic scrollable cursor is useful when it is important to see updated as well as newly inserted rows.
- ▶ For maximum concurrency with dynamic scrollable cursors, use ISOLATION CS, and row level locking.
- ▶ Cursors requiring use of a work file cannot be declared SENSITIVE DYNAMIC. For example, you cannot associate the following SQL statement with a dynamic scrollable cursor, as the entire result must be materialized to a work file:

```
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 )
```

- ▶ The use of a non-deterministic function (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.
- ▶ Rollback to an external savepoint works the same way as rollback works with forward only cursors.
- ▶ 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.
- ▶ Parallelism is not supported with scrollable cursors.
- ▶ Dynamic scrollable cursors can be associated with views.
- ▶ There are no special considerations for dynamic scrollable cursors on summary tables.
- ▶ DRDA support for dynamic scrollable cursors is provided. Since these cursors are not DRDA block fetched, one row at the time will be fetched from a remote server. You might consider declaring the cursor for rowset positioning to obtain a similar blocking effect.
- ▶ 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 (functions may not be meaningful because size of result set can change)

5.4 Common table expressions and recursive SQL

With this enhancement, DB2 V8 introduces a new SQL feature called *common table expressions and recursive SQL*. A common table expression is similar to a *temporary view*, defined and used only for the duration of the SQL statement. There can be many common table expressions in a single SQL statement, and each common table expression can be referenced many times in the statement. All references to a common table expression in a given SQL statement share the same result table. This is unlike regular views or nested table expressions (another way of avoiding the creation of views), which are derived each time they are referenced.

A common table expression takes the form of a *WITH* clause at the beginning of an SQL statement with a syntax similar to a view definition. A common table expression can be used:

- ▶ To avoid creating a view (and when positioned updates and deletes are not required)
- ▶ When the result table is based on host variables
- ▶ When the same result table is shared in a fullselect
- ▶ When the result is derived via recursion

5.4.1 Example of fullselect

The fullselect syntax diagram is changed by adding an optional WITH clause for common table expression prior to the fullselect, as reported in Figure 5-5. A common table expression permits defining a result table with a table name that can be referenced as a table name in any FROM clause of the fullselect that follows. Multiple common table expressions can be specified following the single WITH keyword. Each common table expression specified can also be referenced by name in the FROM clause of subsequent common table expressions.

The table name of a common table expression must be different from any other common table expression table name in the same statement. If a list of columns is specified, it must consist of as many names as there are columns in the result table of the fullselect. Each column-name must be unique and unqualified. If these column names are not specified, the names are derived from the select list of the fullselect used to define the common table expression.

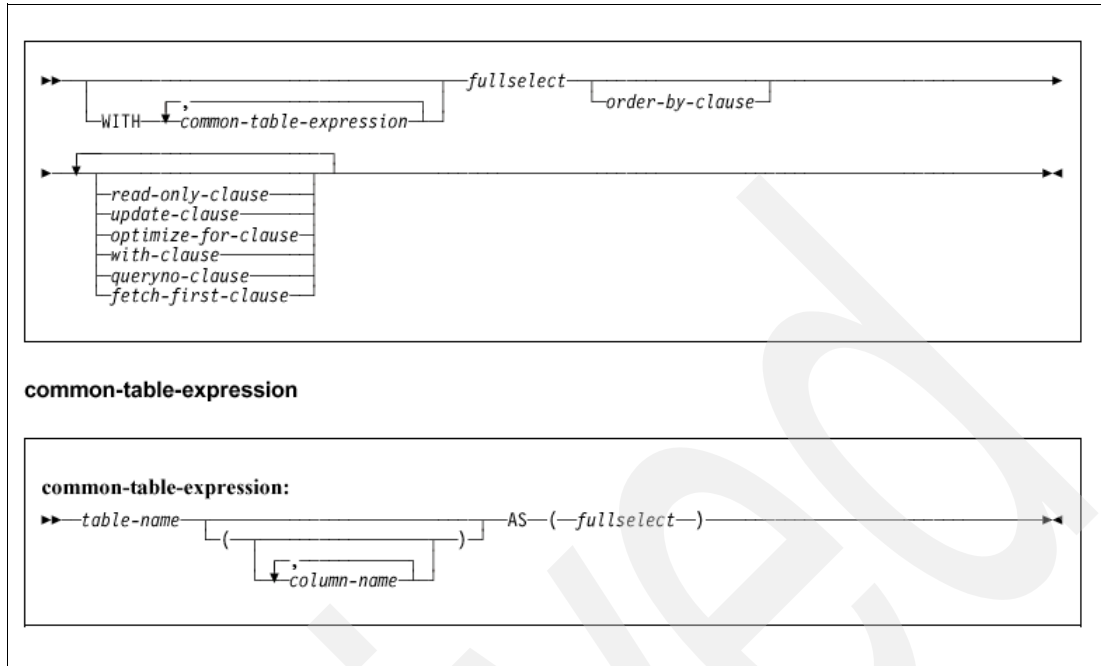


Figure 5-5 Common table expression in fullselect

As an example, imagine that you need to write a query to find the department with highest total pay. This query involves two levels of aggregations: first the total pay for each department is computed using `GROUP BY` and `SUM` function; then the maximum total pay is found, based on the result of the first step.

We can write this query by first defining a view to calculate the total pay by department (`DTOTAL`) and then selecting from it, or we can use the common table expression clause thereby saving the overhead of creating and dropping the `DTOTAL` view. See Example 5-4. Avoiding the view creation, especially for occasional queries, can make the life of the programmer much easier.

Example 5-4 Common table expression in SELECT

```
WITH DTOTAL (deptno,totalpay) AS
  (SELECT deptno,sum(salary+bonus)
   FROM employee GROUP BY deptno)
SELECT deptno FROM DTOTAL
WHERE totalpay =(SELECT max(totalpay) FROM DTOTAL)
```

As for regular table expressions, common table expressions can contain references to host variables.

5.4.2 Example with CREATE VIEW and INSERT

Common table expressions may also be used prior to the fullselect nested immediately inside a CREATE VIEW or an INSERT statement. Example 5-5 shows their usage.

Example 5-5 Common table expression in CREATE VIEW or INSERT

```
CREATE VIEW RICH_DEPT (deptno) AS
  WITH DTOTAL (deptno,totalpay) AS
    (SELECT deptno,sum(salary+bonus)
     FROM employee GROUP BY deptno)
  SELECT deptno FROM DTOTAL
  WHERE totalpay >(SELECT AVG(totalpay) FROM DTOTAL);
INSERT INTO vital_mgr (mgrno) AS
  WITH vitaldept (deptno,se_count) AS
    (SELECT deptno,count()
     FROM employee
     WHERE job ='senior engineer' GROUP BY deptno)
  SELECT d.manager
  FROM Department d, DTOTAL s
  WHERE d.deptno =s.deptno
  AND s.se_count >(SELECT AVG(se_count)
  FROM vitaldept)
```

If the common table expression is specified in a CREATE VIEW statement the table name cannot be the same as the view name being created. If the common table expression is specified in an INSERT statement, the table name cannot be the same as the table or view name that is the object of the insert.

5.4.3 Recursive SQL

If the fullselect of a common table expression contains a reference to itself in a FROM clause, the common table expression is a recursive common table expression. Queries using recursion are useful in supporting applications such as bill of materials, reservation systems, and network planning.

Each fullselect that is part of the recursion cycle must start with SELECT or SELECT ALL. Use of SELECT DISTINCT is not allowed. The unions must be UNION ALL.

When developing recursive common table expressions, remember that an infinite recursion cycle (loop) can be created. Check that recursion cycles will terminate. This is especially important if the data involved is cyclic. A recursive common table expression is expected to include a predicate that will prevent an infinite loop. The recursive common table expression is expected to include:

- ▶ In the iterative fullselect, an integer column incremented by a constant.
- ▶ A predicate in the where clause of the iterative fullselect in the form "counter_col < constant" or "counter_col < :hostvar".

To illustrate the capability of a recursive common table expression for bill of material applications, consider the example in Figure 5-6, where the objective is to find all descendents of AAA.

The WITH statement at the top defines a temporary table called PARENT.

The upper part of the UNION ALL is only invoked once. It does an initial population of the PARENT table with the three rows that have an immediate parent key of AAA.

The second part of the UNION ALL is run recursively until there are no more matches to the join. In the join, the current child value in the temporary PARENT table is joined to the related parent values in the HIERARCHY table. Matching rows are added to the (temporary) PARENT table. This recursive processing will stop when all of the rows in the PARENT table that match rows in the HIERARCHY table are found.

The SELECT statement at the bottom of the statement returns the final contents of the PARENT table back to the user.

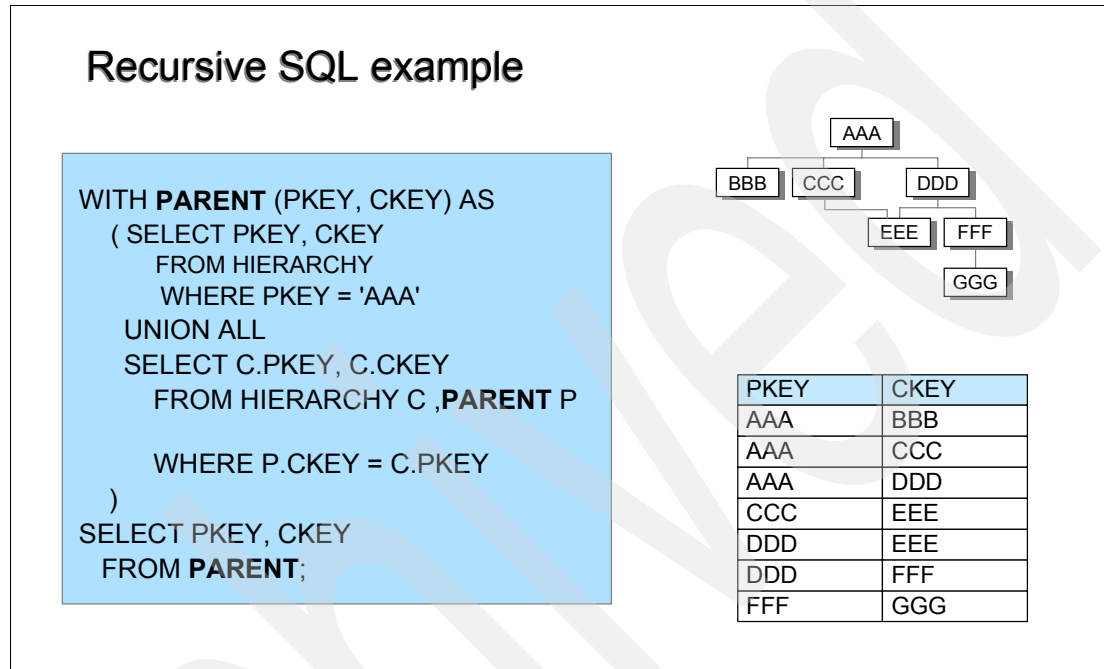


Figure 5-6 Recursive SQL

Common table expressions and nested table expressions follow the same set of rules for determining whether they are deletable, updatable, insertable, or read-only.

How to define a temporary table

A temporary result table can now be derived through a nested table expression, a common table expression, or a view. How to define a temporary table depends in part upon how often, and for how long, you intend to use it:

- ▶ For a single use within a query, you can use either a common table expression or a nested table expression for this case. Using a common table expression makes the main query smaller and easier to read.
- ▶ For multiple uses within a query, a common table expression is more suitable.
- ▶ For multiple queries, you need to use views.

5.5 Multi-row fetch and insert

DB2 V8 introduces support for multiple-row processing for both the FETCH and INSERT statements. Remember that in prior versions of DB2, an application had to execute multiple SQL FETCH statements, one for each row to be retrieved from a table and multiple SQL INSERT statements, as well as one for each row to be inserted into a table. The DB2 V8 enhancement helps to lower the 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 will be fewer send and receive messages over the network. This capability increases the usability and the portability of SQL. Combined with scrollable cursors, this makes it easier to code browsing applications.

In the following sections we introduce the changes to the SQL statements and provide some examples to introduce the enhanced SQL FETCH and INSERT statements.

5.5.1 DECLARE CURSOR

Figure 5-7 shows the syntax of the DECLARE CURSOR.

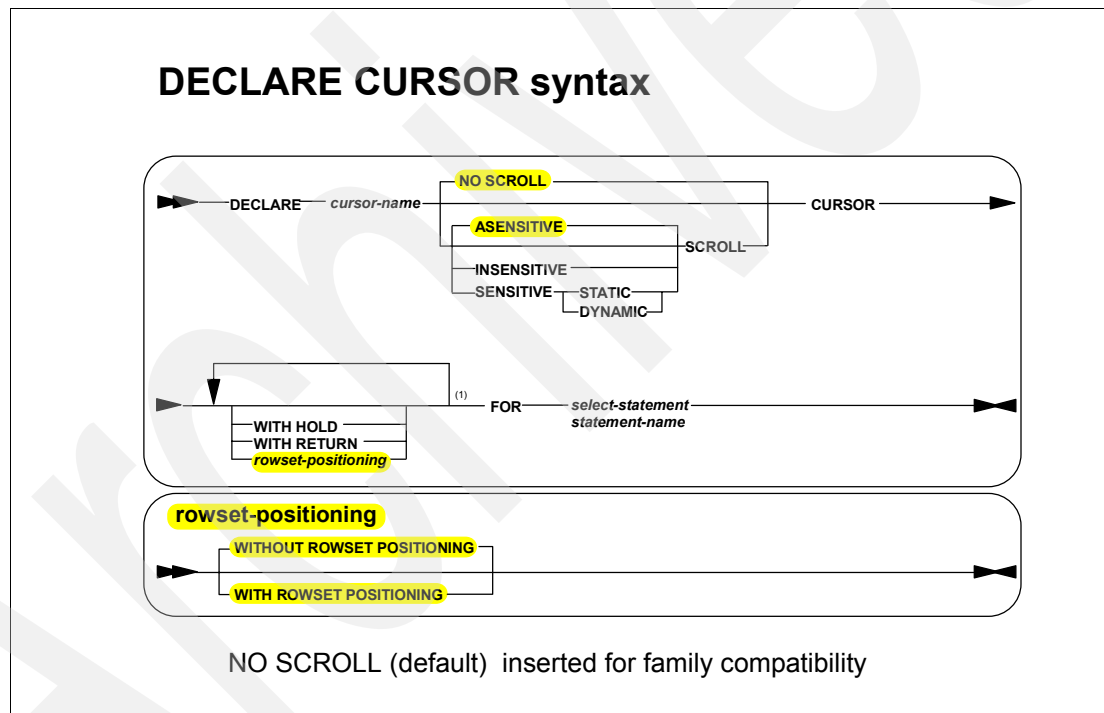


Figure 5-7 Multirow DECLARE CURSOR syntax

Example 5-6 shows how to define the cursor of a query to retrieve a rowset from the table DEPT. The prepared statement is MYCURSOR.

Example 5-6 Multirow DECLARE CURSOR

```

EXEC SQL
  DECLARE CURSOR CURS1 CURSOR
  WITH ROWSET POSITIONING
  FOR MYCURSOR;
  
```

Rowset positioning specifies whether multiple rows of data can be accessed as a rowset on a single FETCH statement. The default is WITHOUT ROWSET POSITIONING.

5.5.2 FETCH

Figure 5-8 shows the multirow FETCH syntax with the new options shaded.

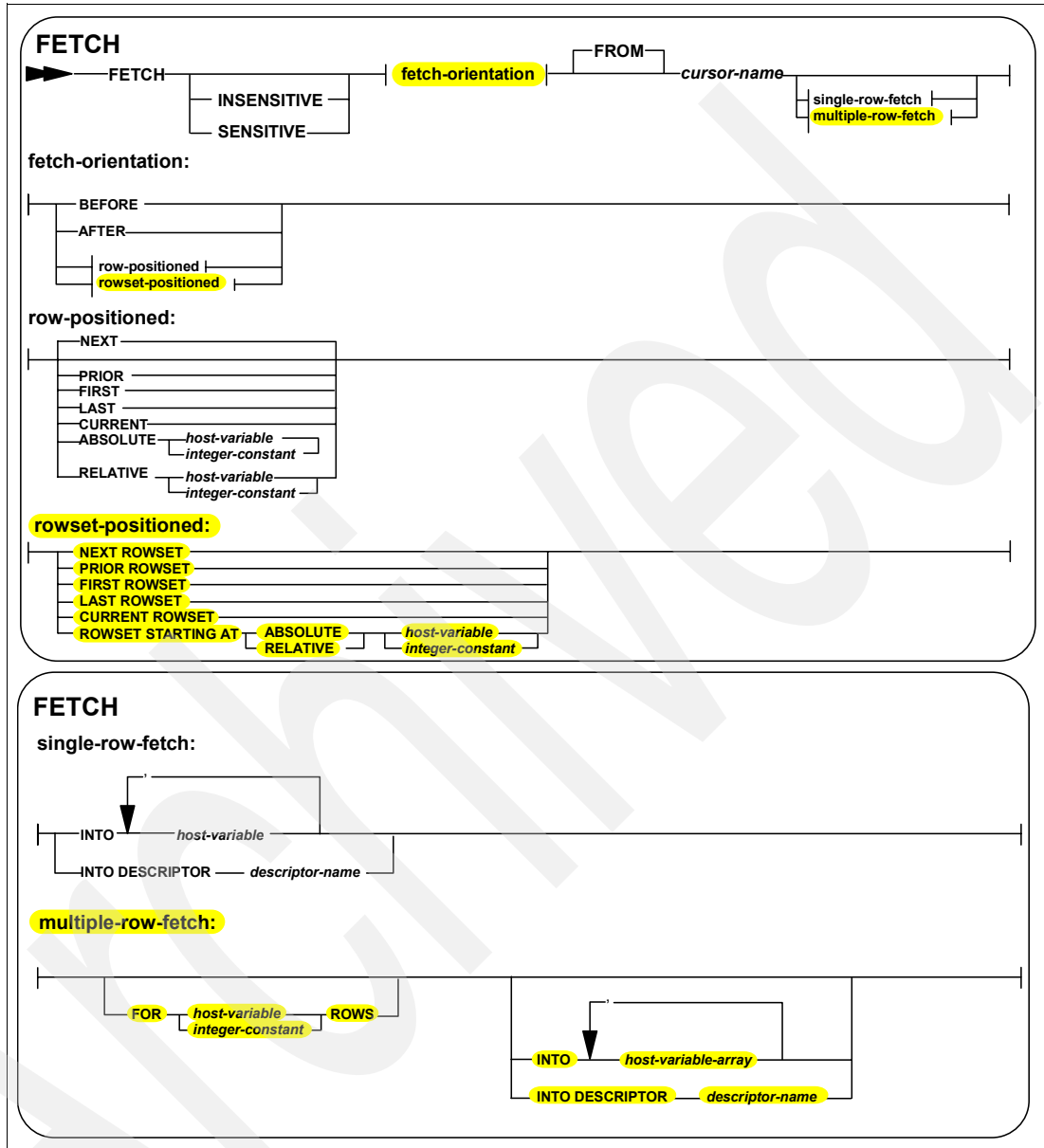


Figure 5-8 Multirow FETCH syntax

Two new blocks, multiple-row-fetch and rowset-positioned fetch, have been introduced.

The multiple-row fetch block is identical to the existing single-row-fetch block in DB2 V7, except that there is an additional clause FOR *n* ROWS.

The rowset-positioned block is similar to the existing row-positioned block in DB2 V7. The row-positioned clause specifies positioning of the cursor with row-positioned fetch orientations NEXT, PRIOR, FIRST, LAST, CURRENT, ABSOLUTE, and RELATIVE, whereas 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.

The multiple-row FETCH is implemented as a static SQL statement. 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, or update the current rowset. Fetching multiple rows of data can be done with both scrollable and nonscrollable cursors. New syntax on the FETCH statement allows specification of the number of rows to be returned in the rowset. The maximum rowset size is 32767.

Fetch works with a host variable array in which each element of the array contains a value for the same column. Changes to allow host variable arrays have been made to COBOL, PL/1, C++. Assembler support is limited to cases where USING DESCRIPTOR is allowed. Multiple row fetch is not supported in REXX, FORTRAN, Java, or SQL procedures.

Example 5-7 shows some simple examples of FETCH being used.

Example 5-7 FETCH examples

Given the cursor C1 is defined as:

```
DECLARE C1 CURSOR WITH ROWSET POSITIONING FOR SELECT * FROM EMP
```

- Fetch the previous rowset, and have the cursor positioned on that rowset.

```
FETCH PRIOR ROWSET FROM C1 FOR 3 ROWS INTO .... or  
FETCH ROWSET STARTING AT RELATIVE -3 FROM C1 FOR 3 ROWS INTO ....
```

- 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 ....
```

- Fetch the first x rows and leave the cursor positioned on that rowset at the completion of the fetch.

```
FETCH FIRST ROWSET FROM C1 FOR x ROWS INTO .....  
FROM C1 FOR 3 ROWS INTO...
```

In the foregoing example:

- ▶ The clause *FOR n ROWS* specifies that with a single FETCH statement in the application program, DB2 fetches *n* rows starting from the positioned cursor. It determines the ROWSET size.
- ▶ In the clause *INTO :hva1, :hva2, ...* the first host variable array corresponds to the first column's output, the second host variable array corresponds to the second column's output, and so on.
- ▶ The *ROWSET* is the group of rows for the result table of the query which are returned by the single FETCH statement.

Single row and multiple row fetches can be mixed for a multi-fetch cursor. If *FOR n ROWS* is NOT specified and the cursor is declared for rowset positioning, then the size of rowset will be 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.

Note that the clause specifying the desired number of rows can be specified in either the SELECT statement of the cursor, or on a FETCH statement for the cursor, or in both. However, these clauses have a different effect:

- ▶ 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 then an end of data condition occurs.
- ▶ In a FETCH statement, the FOR *n* ROWS clause controls the number of rows that are returned for a single FETCH statement.

Both these clauses can be specified.

OPEN CURSOR, ALLOCATE CURSOR, and DESCRIBE CURSOR have been extended to comply with multirow support.

Positioned UPDATE

The new syntax allows you to specify in the UPDATE statement the following clause to update a specified row in the rowset:

```
FOR CURSOR cursor-name FOR ROW row-number OF ROWSET
```

Instead, if you specify the existing WHERE CURRENT OF cursor-name, all the rows in the rowset are updated. For example:

```
Update all the rows in the rowset that cursor CSR1 is positioned on.  
UPDATE T1  
SET C1 = 5  
WHERE CURRENT OF CSR1
```

Positioned DELETE

The new syntax allows you to specify in the DELETE statement the following clause to delete a specified row in the rowset:

```
FOR CURSOR cursor-name FOR ROW row-number OF ROWSET
```

Instead, if you specify the existing WHERE CURRENT OF cursor-name, all the rows in the rowset are deleted. The following example deletes the fourth row in the rowset that cursor CSR1 is positioned on.

```
DELETE FROM T1  
FOR CURSOR CSR1 FOR ROW 4 OF ROWSET
```

5.5.3 INSERT

With DB2 V7 there are two ways to insert a row into a table:

- ▶ INSERT via VALUES is used to insert a single row into the table or view using values provided or referenced.
- ▶ INSERT via SELECT is used to insert one or more rows into the table or view using values from other tables or views.

With DB V8, a third way has been added with the syntax shown in Figure 5-9.

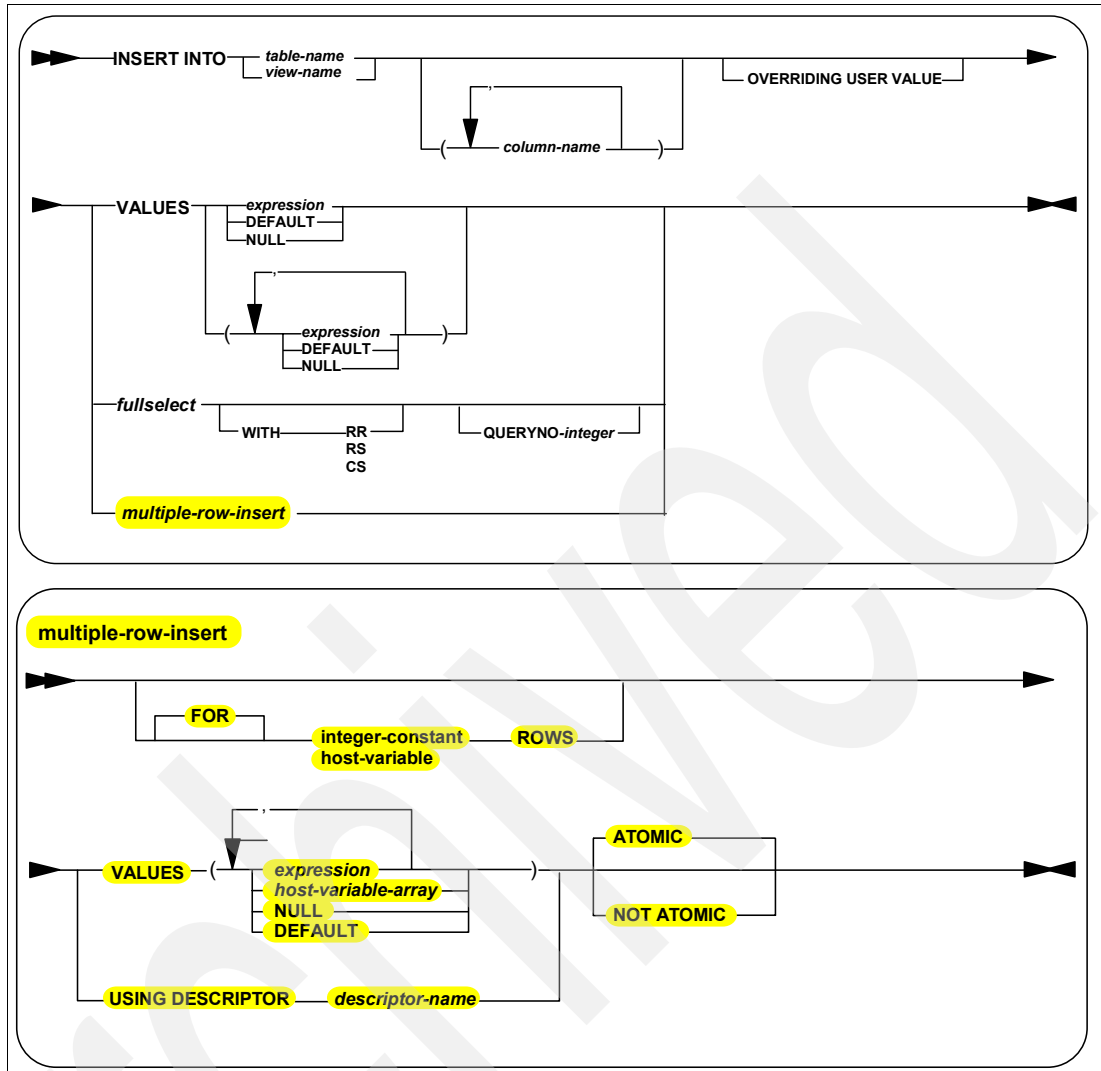


Figure 5-9 Multirow INSERT syntax

- ▶ INSERT via *FOR n ROWS* form is used to insert multiple rows into a table or view using values provided in the host variable array.

There are two forms of multiple row insert:

- *Static SQL* INSERT with host variable arrays:

```
INSERT INTO T FOR :n ROWS
VALUES(:hva1, :hva2) ATOMIC
```

- *Dynamic SQL* INSERT with host variable arrays:

```
stmt = 'INSERT INTO T VALUES(?,?)'
attrvar = 'FOR MULTIPLE ROWS ATOMIC'
PREPARE my_insert ATTRIBUTES :attrvar FROM :stmt
EXECUTE my_insert FOR :hv ROWS USING (:hva1, :hva2)
```

The *FOR n ROWS* clause is supplied on the EXECUTE statement for the dynamic SQL INSERT instead of on the INSERT statement itself, as in the case of static SQL INSERT. If *n* is greater than or equal to 1, then the parameter marker represents a host variable array. The maximum number of rows that can be inserted with a single INSERT statement is 32767. The input data for these multiple rows is provided with new host variable arrays, where each array represents the multiple rows of a single column.

The *VALUES* or *USING DESCRIPTOR* clause allows specification of multiple rows of data. For example, assuming :hva1 and :hva2 represent host variable arrays, the *VALUES (:hva1, :hva2)* clause may be used to specify the multiple values for an insert statement.

The *ATOMIC* or *NOT ATOMIC* 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 (one or more rows) failure. *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* 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 changes made during the execution of the statement are not undone. Size and number of rows should be considered when deciding, since logging and roll back are affected.

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. 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.

To handle nulls, you can also 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.
```

SQLCA

After a multiple-row INSERT or multiple-row FETCH statement, information is returned to the program through the SQLCA as follows:

- ▶ **SQLSTATE:** SQLSTATE of last error
- ▶ **SQLCODE:** SQLCODE of last error
- ▶ **SQLERRD3:** actual number of rows inserted (in the case of INSERT), or the number of rows returned in the case of FETCH
- ▶ **SQLWARN:** accumulation of flags set during any single insert

SQLDA

SQLDA must contain a valid description of the host variable arrays or buffers which contain the values to be inserted. Each SQLVAR describes a host variable array or buffer which contains a value for a column of target table. SQLDA must have enough storage to contain SQLVAR for each target column for which values are provided, plus an additional SQLVAR entry for use by DB2 UDB for z/OS. Prior to the multi-row insert, the SQLDA fields must be set correctly to include number of SQLVAR occurrences, number of variables used, pointer to arrays, indicator variables etc.

Examples of INSERT

In Example 5-8 we insert a variable number of rows using host variable arrays for column values. Assume that the table T1 has one column and that a variable (:hv) number of rows of data are to be inserted into the table T1.

Example 5-8 Example 1 of INSERT

```
EXEC SQL
INSERT INTO T1 FOR :hv ROWS
VALUES (:hva:hvind) ATOMIC ;
```

In this example, :hva represents the host variable array and :hvind represents the array of indicator variables.

In Example 5-9 we insert multiple rows using host variable arrays for column values. Assume table T2 has 2 columns, C1 is SMALL INTEGER and C2 is INTEGER. INSERT 10 rows of data into T2. Values to be inserted are in host variable arrays :hva1 (array of SMALL INTEGER) and :hva2 (array of INTEGER values).

Example 5-9 Example 2 of INSERT

```
EXEC SQL INSERT INTO T2 (C1, C2) FOR 10 ROWS
VALUES (:hva1 :hvind1, :hva2 :hvind2) NOT ATOMIC;
```

PREPARE

A new block cursor-width is introduced to specify whether multiple rows of data can be inserted as a rowset on a single dynamic INSERT statement and use of keywords ATOMIC and NOT ATOMIC.

Assume that the table PROG has 9 columns. Prepare a dynamic INSERT statement which inserts 5 rows into this table.

```
stmt='INSERT INTO PROG(C1, C2, C3, C4, C5, C6, C7, C8, C9)
VALUES(?,?,?,?,?,?,?,?)' ;
attrvar='FOR MULTIPLE ROWS' ;
NROWS=5 ;
EXEC SQL PREPARE ins_stmt ATTRIBUTES :attrvar FROM :stmt ;
```

EXECUTE

A new block multiple-row-insert is introduced to specify whether multiple rows of data can be inserted with a single dynamic INSERT statement. For example:

```
Execute the dynamic SQL statement prepared above.
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.

5.6 Get diagnostics

The *GET DIAGNOSTICS* statement enables applications to retrieve diagnostics information about statements that have been executed. This statement complements and extends the diagnostics that are available in the SQLCA. See Figure 5-10.

GET DIAGNOSTICS statement

- It enables more diagnostic information to be returned than it can be contained in SQLCA
- It returns SQL error information
 - for overall statement
 - for each condition, when multiple conditions occur
- It supports SQL error message tokens larger than 70 bytes (SQLDA size limitation)

```

INSERT INTO T1 FOR 5 ROWS VALUES (:array);
GET DIAGNOSTICS :errcount = NUMBER;
DO || = 1 TO ERR_COUNT;
  GET DIAGNOSTICS FOR CONDITION :||
  :rc = RETURNED_SQLSTATE;
END;
  
```

Figure 5-10 GET DIAGNOSTICS statement

In Figure 5-11 we show the syntax for the GET DIAGNOSTICS statement.

GET DIAGNOSTICS syntax

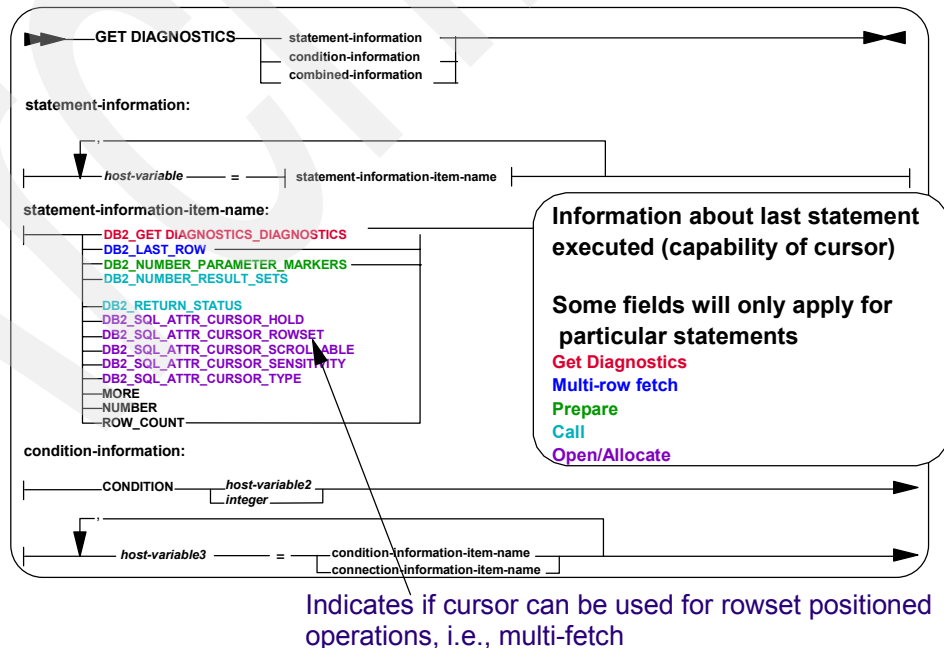


Figure 5-11 GET DIAGNOSTICS syntax

The following C language program example demonstrates the use of this new statement:

In an application, use GET DIAGNOSTICS to determine how many rows were updated.

```
long rcount;
EXEC SQL UPDATE T1 SET C1 =C1 +1;
EXEC SQL GET DIAGNOSTICS :rcount = ROW_COUNT;
```

After execution of this code segment, *rcount* contains the number of rows that were updated.

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 fetch 10 rows with a single FETCH statement.

▶ Example 1:

Assume that an error, SQLCODE -802, is detected on the 5th row. SQLERRD3 is set to 4 for the 4 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;
Would result in num_row = 4 and num_cond = 1 (1 condition).
```

```
GET DIAGNOSTICS CONDITION 1 :sqlstate = RETURNED_SQLSTATE,
:sqlcode = DB2_RETURNED_SQLCODE, :row_num = DB2_ROW_NUMBER;
```

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. For example:

Diagnostic information for multi-row insert

When NOT ATOMIC 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 multiple 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 multiple 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 multiple-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 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 via the GET DIAGNOSTICS statement.

As an example, assume that you are inserting multiple rows using host variable arrays for Column Values. The table T1 has 2 columns, C1 is a SMALL INTEGER column, and C2 is an INTEGER column. INSERT 10 rows of data into the 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 5-2.

Table 5-2 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
10	35	88

```
EXEC SQL
INSERT INTO T1 (C1, C2) FOR 10 ROWS VALUES (:hva1:hvind1, :hva2:hvind2)
NOT ATOMIC;
```

After execution of the INSERT statement, we have the following in the 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;
```

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;
```

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;
```

Would result in sqlstate = 22003, sqlcode = -302, and row_num = 8

5.7 Scalar fullselect

DB2 on the UNIX and Windows platforms provides support for scalar fullselect. This allows applications that use scalar fullselects to be portable without change and also conform with the SQL standards.

Figure 5-12 shows the extended syntax of scalar fullselect to expression.

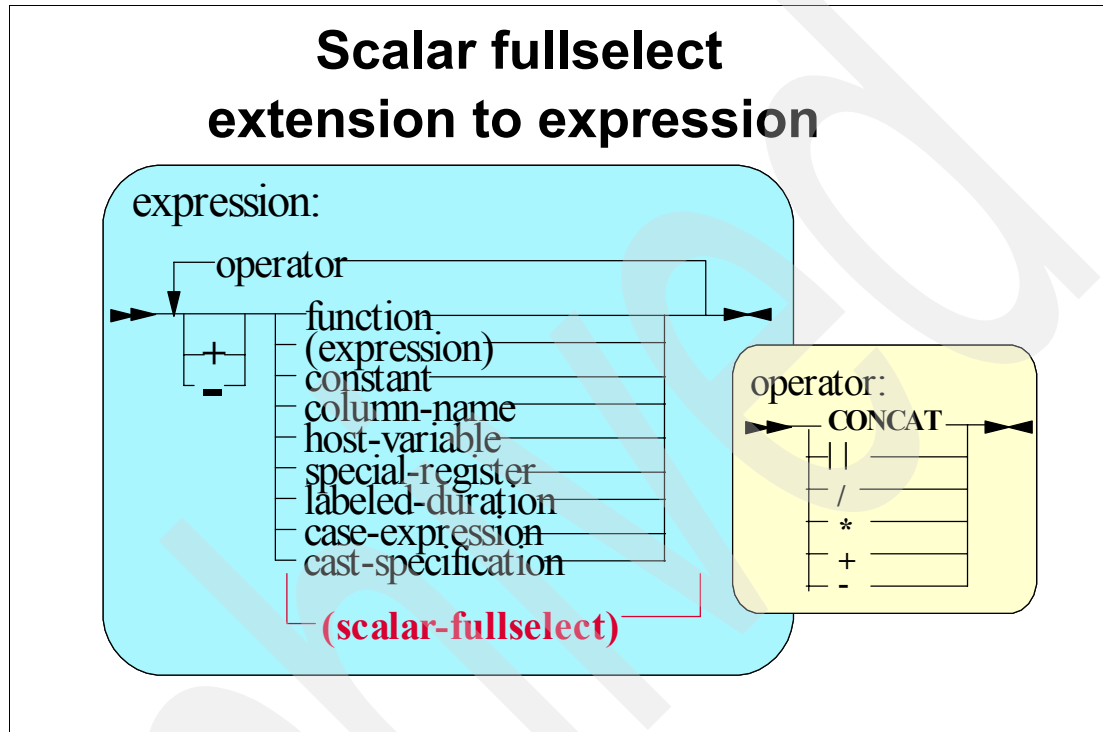


Figure 5-12 Scalar fullselect — Extension to expression

Figure 5-13 shows the extended syntax of scalar fullselect to CASE expression.

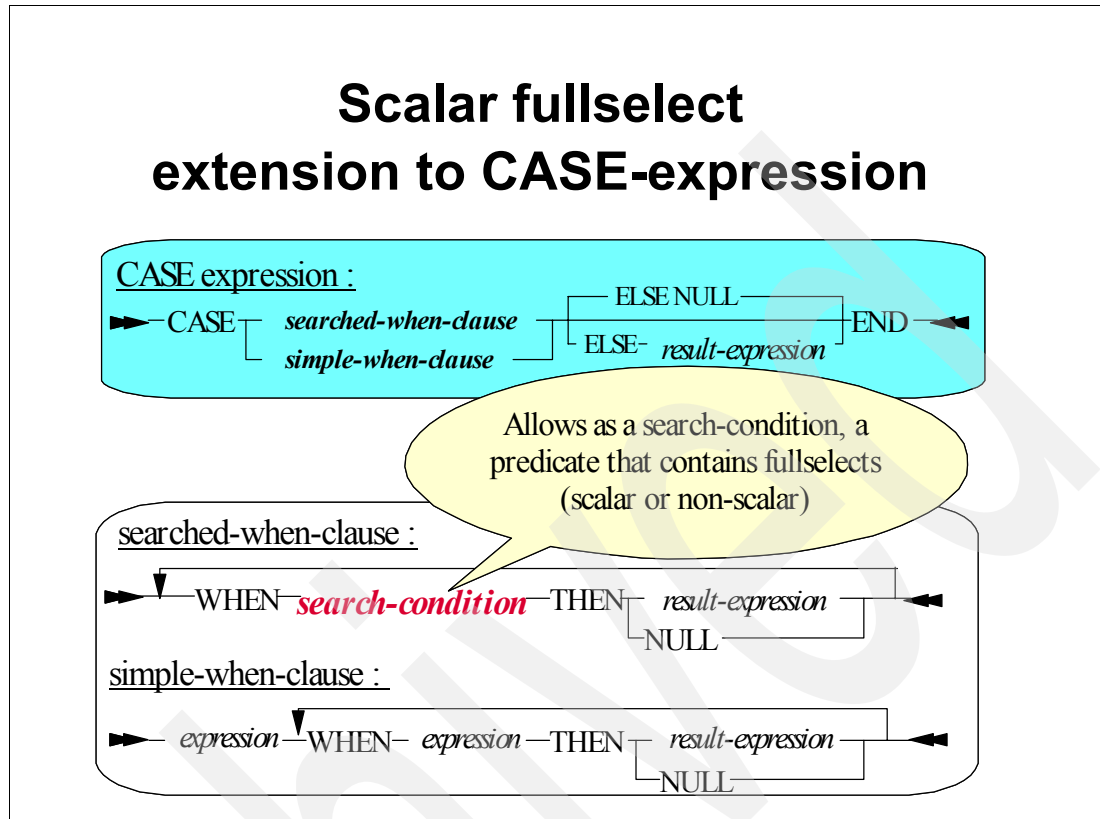


Figure 5-13 Scalar fullselect — Extension to CASE expression

5.7.1 Functional description

A scalar fullselect enclosed in parentheses 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.

You can code a SELECT statement as follows:

```

SELECT (sfs1) AS COL1,
       (SELECT (sfs2) FROM T3 WHERE ... ) AS COL2
FROM T1,
     TABLE(TUDF(T1.C1 + (sfs3))) T2
WHERE (sfs4) + (sfs5) = (sfs6)
  
```

In this example, sfs1 to sfs6 are any correlated or noncorrelated scalar fullselects, and TUDF represents a table user defined function.

Next we explain the enhancement with the help of a few examples.

Figure 5-14 shows the four tables, PARTS, PRODUCTS, PARTPRICE, and PARTINVENTORY, used in the examples.

Tables used in the examples

<u>PARTS</u>			<u>PARTPRICE</u>			
<u>PART</u>	<u>PROD#</u>	<u>SUPPLIER</u>	<u>PART</u>	<u>PROD#</u>	<u>SUPPLIER</u>	<u>PRICE</u>
WIRE	10	ACWF	WIRE	10	ACWF	3.50
OIL	160	WESTERN_CHEM	OIL	160	WESTERN_CHEM	1.50
MAGNETS	10	BATEMAN	MAGNETS	10	BATEMAN	59.50
PLASTIC	30	PLASTIC_CORP	PLASTIC	30	PLASTIC_CORP	2.00
BLADES	205	ACE_STEEL	BLADES	205	ACE_STEEL	8.90

<u>PRODUCTS</u>			<u>INVENTORY</u>			
<u>PROD#</u>	<u>PRODUCT</u>	<u>PRICE</u>	<u>PART</u>	<u>PROD#</u>	<u>SUPPLIER</u>	<u>ONHAND#</u>
505	SCREWDRIVER	3.70	WIRE	10	ACWF	8
30	RELAY	7.55	OIL	160	WESTERN_CHEM	25
205	SAW	18.90	MAGNETS	10	BATEMAN	3
10	GENERATOR	45.75	PLASTIC	30	PLASTIC_CORP	5
			BLADES	205	ACE_STEEL	10

Figure 5-14 Tables used in the scalar fullselect examples

Figure 5-15 shows the use of scalar fullselect in the *WHERE* clause.

Example of scalar fullselects in a WHERE clause

Find which products have the prices in the range of at least twice the lowest price of all the products and at most half the price of all the products.

```
SELECT PRODUCT, PRICE
FROM PRODUCTS A
WHERE PRICE BETWEEN 2 * (SELECT MIN(PRICE) FROM PRODUCTS)
AND .5 * (SELECT MAX(PRICE) FROM PRODUCTS) ;
```

<u>PRODUCT</u>	<u>PRICE</u>
RELAY	7.55
SAW	18.90

Figure 5-15 Scalar fullselect — Example of WHERE clause

Figure 5-16 shows an example which uses 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 *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 this statement. The column *COST* is passed through the derived table *X*.

Example of nested scalar fullselects in a *SELECT* list

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(COST),0) AS INV_COST
   FROM (SELECT (
             (SELECT PRICE FROM PARTPRICE WHERE PART = B.PART)
            *(SELECT ONHAND# FROM INVENTORY WHERE 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

Figure 5-16 Scalar fullselect — Example of nesting in a *SELECT* list

Figure 5-17 shows an example of the use of scalar fullselect in *CASE* expression.

Example of scalar fullselect in a 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

Figure 5-17 Scalar fullselect — Example of CASE expression

5.7.2 Restrictions

The scalar fullselects are not supported in the following cases:

- ▶ CHECK constraint
- ▶ Grouping expression
- ▶ View created with the WITH CHECK OPTION
- ▶ CREATE FUNCTION (SQL scalar)
- ▶ Column function
- ▶ ORDER BY clause
- ▶ Join conditions of the ON clause for INNER and OUTER joins

5.8 Select from insert

In applications that use triggers, timestamp columns, identity columns, and ROWID columns, the data values generated and inserted automatically by DB2 are not visible immediately, and the applications have to query the tables to retrieve these values. The **SELECT FROM INSERT** new statement provides this capability. The **INSERT** statement can be used in the **FROM** clause of a **SELECT** statement that is a subselect and the **SELECT INTO** statement.

5.8.1 Functional description

Changes have been done to the SQL syntax in the table-spec and order-by-clause to support this enhancement. The keywords **FINAL TABLE** in the table-spec and **INPUT SEQUENCE** in the order-by-clause are introduced.

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.

In the following sections we discuss various scenarios, illustrating them with examples.

Result table rows from the INSERT statement

The result table of the INSERT contains all of the rows that are inserted. 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.

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.

Figure 5-20 shows the BEFORE INSERT trigger statement followed by the SELECT FROM INSERT statement.

INSERT trigger

```
CREATE TRIGGER TRIG1
NO CASCADE BEFORE INSERT ON EMPLOYEE
REFERENCING NEW AS NEWSALARY
FOR EACH ROW MODE DB2SQL
WHEN (NEWSALARY.LEVEL = 'Associate')
BEGIN ATOMIC
  SET NEWSALARY.SALARY = NEWSALARY.SALARY + 5000.00
END;

SELECT NAME,SALARY INTO :name_hv, :salary_hv
FROM FINAL TABLE
(INsert INTO EMPLOYEE(NAME,SALARY,LEVEL)
VALUES('New Hire',35000.00,'Associate'));
```



```
:name_hv =
'New Hire'
:salary_hv =
40000.00
```

Figure 5-20 SELECT FROM INSERT — INSERT trigger

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 successfully inserted till the failure are undone and the result table is empty.

What happens if the result table has 100 rows and the 90th row being fetched fails? The result table still contains 100 rows and the related SQLCODE is returned to the application. The application can decide whether to commit all the rows inserted or undo all the rows inserted.

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'))
```

Effect of updates and deletes against result table rows

If the application declares a nonscrollable 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, 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.

Figure 5-21 shows an example.

```
SELECT FROM INSERT  
effect of updates and deletes  
against result table rows  
  
DECLARE CS1 CURSOR FOR  
SELECT SALARY  
FROM FINAL TABLE  
(INSERT INTO EMPLOYEE (NAME, SALARY, LEVEL)  
SELECT NAME, INCOME, BAND FROM OLD_EMPLOYEE);  
  
OPEN CS1;  
FETCH CS1 INTO :DECHV; <--- returns the first row  
UPDATE EMPLOYEE  
SET SALARY = SALARY + 500; <--- give employee $500 raise  
DO WHILE (SQLCODE = 0);  
  FETCH CS1 INTO :DECHV; <--- values determined prior  
  <--- to the UPDATE  
  <--- statement are returned  
END;
```

Figure 5-21 *SELECT FROM INSERT* — Effect of updates and deletes against result table

Returning rows in the same sequence as they are inserted

If there is a requirement in the application to retrieve the rows in the same sequence as they are inserted, the application may use the INPUT SEQUENCE keywords in the ORDER BY.

Figure 5-22 shows the example of a multi-row INSERT. In this example, HVA1 and HVA2 are host variable arrays, each representing multiple rows, with each entry for each array representing a single row.

SELECT FROM INSERT ordering sequence example

```
DECLARE CS2 CURSOR WITH ROWSET POSITIONING FOR  
SELECT EMPNO  
FROM FINAL TABLE  
(INSERT INTO EMPLOYEE (NAME, TELE)  
VALUES(:HVA1, :HVA2))  
ORDER BY INPUT SEQUENCE;
```

INTEGER
GENERATED
ALWAYS AS
IDENTITY

Input		Result
HVA1	HVA2	EMPNO
Liz	555-1212	1
David	555-9876	2
Jessica	555-0110	3

Figure 5-22 SELECT FROM INSERT — Ordering sequence example

Considerations

You should be aware of the following considerations:

- ▶ 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 underlying base table of the INSERT statement must not have any AFTER INSERT triggers that have a dependency on the target table.
- ▶ 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.
- ▶ The INPUT SEQUENCE clause can only be specified if the table-spec is included in a SELECT statement that contains an INSERT statement.

5.9 Qualified column names in INSERT and UPDATE

DB2 V7 does not allow column names to be qualified in INSERT and UPDATE statements. This hampers application portability, as DB2 on UNIX and Windows platforms does not have this restriction.

DB2 V8 allows column names to be qualified with a table name, or a schema followed by a table name in INSERT statements.

Also, DB2 V8 allows column names in the SET clause of an UPDATE statement to be qualified. Consider the following examples:

- ▶ A correlation name is not specified for the table (or view) name, and the table name 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 the table name, 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 the table name, but it is not used to qualify the column name. Instead, the table name 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 is not specified, and the qualifier for the column name is not the table name. That is, the qualifier is not a valid name in this context. This results in SQLCODE -206 being returned:

```
UPDATE T1 SET T.C1 = C1 + 10 WHERE C1 = 4;
```

- ▶ The table name is qualified following the UPDATE verb, and the qualifier is also used in qualifying the column name. This is allowed:

```
UPDATE MY.T1 SET MY.T1.C1 = C1 + 10 WHERE C1 = 4;
```

- ▶ The table name is qualified following the UPDATE verb, and the qualifier is not used in qualifying the column name. This is allowed:

```
UPDATE MY.T1 SET T1.C1 = C1 + 10 WHERE C1 = 4;
```

- ▶ The table name is not qualified after the UPDATE verb, but the implicit qualifier is used as an explicit qualifier for the column name. myid is the authid 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;
```

5.10 Expressions in GROUP BY

DB2 V7 does not allow expressions to be specified in the GROUP BY clause which is supported by DB2 on the UNIX, Windows, and AS/400 platforms. The current work-around is for the applications to code nested table expressions or a view to first provide a result table with the expression as a column of the result, then specify the column in the GROUP BY clause.

The GROUP BY clause specifies an intermediate result table that consists of a grouping of the rows of R. R is the result of the previous clause of the subselect.

A grouping-expression is an expression used in defining the grouping of R. In its simplest form, a grouping-expression contains a single column-name. Each column-name included in a grouping-expression must unambiguously identify a column of R. A grouping-expression cannot include a scalar-fullselect or any function that is non-deterministic or is defined to have an external action. Columns with a LOB data type (or distinct type for which the source type is a LOB) cannot be used in a grouping-expression. Correlated columns or host variables cannot be used in a grouping-expression.

The length attribute of each grouping-expression cannot exceed 255 for a character expression or 127 for a graphic expression.

The result of the GROUP BY clause is a set of groups of rows. In each group of more than one row, all values of each grouping-expression are equal, and all rows with the same set of values of the grouping-expressions are in the same group. For grouping, all null values for a grouping-expression are considered equal.

Grouping-expressions can be used in a search condition in a HAVING clause, in the SELECT clause, or in a sort-key-expression of an ORDER BY clause. In each case, the reference specifies only one value for each group. The grouping-expression specified in these clauses must exactly match the grouping-expression in the GROUP BY clause, except that blanks are not significant (HAVING clause, and SELECT clause and sort-key-expression of an ORDER BY clause). For example, a grouping-expression of

```
SALARY * .10
```

matches the expression in a having-clause of

```
HAVING SALARY * .10
```

but does not match

```
HAVING .10 * SALARY
```

or

```
HAVING (SALARY*.10)+100
```

If the grouping-expression contains varying-length strings with trailing blanks, the values in the group can differ in the number of trailing blanks and may not all have the same length. In that case, a reference to the grouping-expression still specifies only one value for each group, but the value for a group is chosen arbitrarily from the available set of values. Thus, the actual length of the result value is unpredictable.

We demonstrate use of this enhancement in Example 5-10.

Example 5-10 Expressions in GROUP BY

```
CREATE TABLE T1(
C1 VARCHAR(5),
C2 DECIMAL(9,2),
C3 INT,
C4 DATE);
SELECT SUBSTR(C1,LENGTH(C3), 1), <- match to grouping expression SUBSTR
C2 + C3, <- match to grouping expression C2+C3
C3 + C2, <- does not match to any grouping expression or column
C2 + C3 + 4, <- does not match to any grouping expression or column
C1||CHAR(C2), <- match to grouping columns C1 and C2
MAX(C2), <- match to grouping column C2
SUBSTR(C1,LENGTH(C3),1)||'A', <- does not match to any grouping expression or column
C4||CURRENT_TIMESTAMP <- does not match to any grouping expression or column
FROM T1
GROUP BY C1, <- grouping column
C2, <- grouping column
```

```
C2 + C3, <- grouping expression
SUBSTR(C1,LENGTH(C3),1), <- grouping expression
HAVING C2 + 2 = 177.1 AND <- match to grouping column C2
C2 + C3 = 277.2; <- match to grouping expression C2+C3
```

5.11 Multiple DISTINCT

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
```

With DB2 V7 you can also have multiple DISTINCT on the same column only, as in:

```
SELECT COUNT(DISTINCT(A1)), SUM(DISTINCT A1) FROM T1
```

However, you cannot specify multiple DISTINCT on different columns, you get SQLCODE -127 for the reason "DISTINCT IS SPECIFIED MORE THAN ONCE IN A SUBSELECT" if you try to issue a query such as :

```
SELECT COUNT(DISTINCT A1), SUM(DISTINCT A2) FROM T1
```

In general, DB2 V7 only allows one DISTINCT keyword on the SELECT or HAVING clause of any given query. This restriction causes multiple queries to be executed to retrieve the same information thereby resulting in complexity and performance degradation.

DB2 V8, in NFM, removes this restriction, and allows more than one DISTINCT keywords 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.

Executing a query with multiple distinct columns can be costly in terms of number of sorts performed and work files created. Therefore, whenever possible, some optimization may be done to the query by eliminating unnecessary DISTINCT 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 list of queries in Example 5-11.

Example 5-11 Multiple COUNT(DISTINCT)

```
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 example would result in SQLCODE -127.

5.12 Sequences

The need for generating sequential numbers was addressed by introducing support for identity columns in DB2 V6, and enhanced in V7. However, the identity column is a column of a table and as such is associated and tied with the table. There is a strong requirement to have efficient stand-alone, sequential-number-generating objects. In 5.14, “Sequences and identity columns comparison” on page 117 we summarize the differences.

To address this requirement, DB2 DB2 V8 introduces a new SQL data object called a *sequence*. This also facilitates the porting of applications across DB2 platforms as well as non-DB2 platforms.

In the absence of support for sequences, one common-application level implementation is to maintain a one-row table that contains the sequence number., have each transaction lock this table, increment the number, and then commit to unlock the table. That is, only one transaction at a time can increment the sequence number. A variation on this theme would be to use, for example, `SELECT MAX() + 1 WITH RR` followed by `INSERT` using the retrieved key. Performance bottlenecks can easily occur with this method if high concurrency is required in updating the sequence number.

In a data sharing environment, the page that contains the counter can constitute a hot spot in the database, resulting in unpredictable transaction delays caused by inter-system P-lock negotiation for that page and by buffer invalidation and refresh. This contention inhibits transaction throughput and the application’s processing power. Moreover, 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.

All the above problems are solved by the introduction of the support for sequences. A sequence is a user-defined object that generates a sequence of numeric values according to the specifications with which the sequence is created. This provides a means for the applications to have the unique numeric key values generated by DB2 and to coordinate keys across multiple rows and tables. A sequence can be accessed and incremented by many applications concurrently without waiting.

DB2 does not wait for an application that has incremented a sequence to commit before allowing the sequence to be incremented again by another application. Applications can use one sequence for many tables, or create multiple sequences for use of each table requiring generated key values. In either case, the applications control the relationship between the sequences and the tables. In addition, the failure of one DB2 member in a data sharing group never prevents read or update access to the sequence from the surviving members. There are no retained locks to prevent access to the sequence.

We now look at some examples of the SQL statements to support sequences and the major parameters.

CREATE SEQUENCE

The `CREATE SEQUENCE` statement creates a sequence at the application server. This statement can be issued interactively, embedded in an application program, or dynamically prepared. DB2 stores the information in the new catalog tables added to support the sequences: `SYSIBM.SYSSEQUENCES`, `SYSIBM.SYSSEQUENCEDEP` and `SYSIBM.SYSSEQUENCEAUTH`.

Figure 5-23 shows the attributes of the CREATE SEQUENCE statement.

```
CREATE SEQUENCE statement  
  
CREATE SEQUENCE <sequence-name>  
AS < data type >  
START WITH <numeric value>  
INCREMENT BY <numeric value>  
NO MINVALUE / MINVALUE <numeric value>  
NO MAXVALUE / MAXVALUE <numeric value>  
NO CYCLE / CYCLE  
NO CACHE / CACHE <integer value>
```

Figure 5-23 Sequences — CREATE SEQUENCE statement

The attributes are as follows:

► **Sequence name:**

The sequence name is a qualified or unqualified name that designates a sequence. A qualified name is a two-part name consisting of the schema name and an identifier, up to 128 bytes each, separated by a period.

► **AS data type**

This specifies the data type to be used for the sequence value with default INTEGER.

► **START WITH**

This specifies the first value for the sequence.

► **INCREMENT BY**

This specifies interval between consecutive values of the sequence with default 1.

► **MINVALUE**

This specifies the minimum end point of the range of values for the sequence.

► **MAXVALUE**

This specifies the maximum end point of the range of values for the sequence.

► **CYCLE**

This specifies whether the sequence should wrap around and repeat after reaching its maximum or minimum value.

► **CACHE**

This specifies whether to keep some pre allocated values in memory for better access. This is a performance and tuning option: The integer value that optionally follows specifies the number of values of the sequence for DB2 to preallocate in memory. Pre-allocating values in the cache reduces synchronous I/O to the catalog table SYSIBM.SYSSEQUENCES when new sequence numbers are requested.

► **NO CACHE**

When this option is specified, every request for a new sequence number results in synchronous I/O to the catalog table SYSIBM.SYSSEQUENCES, since sequence values are not logically pre allocated in a cache. This is a performance consideration.

Note on effect of caching in data sharing: DB2 always generates sequence numbers in order of request; however, in DB2 data sharing environments where multiple caches could be active simultaneously when the CACHE option is used, it is possible that the requests for next value assignments from different DB2 members may not result in the assignment of values in strict numeric order. For example, if members DB2A and DB2B are using the same sequence, and DB2A gets the cache of values 1-20 and DB2B gets the cache of values 21-40, the actual order of values assigned would be 1,21,2 if DB2A requested for next value first, then DB2B, and then DB2A again. So, if sequence numbers must be generated in strict numeric order for multiple DB2 members using the same sequence concurrently in a data sharing environment, then use NO CACHE.

ALTER SEQUENCE

Use the ALTER SEQUENCE statement to change the attributes of a sequence.

Figure 5-24 shows what can be done with the ALTER SEQUENCE statement.

This statement can be issued interactively, embedded in an application program, or dynamically prepared. You can change the INCREMENT BY, MINVALUE, MAXVALUE, CACHE and CYCLE attributes of a sequence, and optionally restart the sequence from a point that is different from where it would otherwise have continued.

- ▶ Only future values of the sequence are affected by the ALTER statement.
- ▶ The data type of a sequence cannot be altered. To change the data type, the sequence must be dropped and recreated.
- ▶ The unused cache values for the sequence may be lost when the sequence is altered.

ALTER SEQUENCE statement

```
ALTER SEQUENCE <sequence-name>  
INCREMENT BY <numeric value>  
NO MINVALUE / MINVALUE <numeric value>  
NO MAXVALUE / MAXVALUE <numeric value>  
NO CYCLE / CYCLE  
NO CACHE / CACHE <integer value>  
RESTART / RESTART WITH <numeric value>
```

Figure 5-24 Sequences — ALTER SEQUENCE statement

ALTERing a sequence results in the update of relevant columns of the row that describes the altered sequence in the SYSIBM.SYSSEQUENCES catalog table. The row is updated to reflect the new values for parameters explicitly specified with the ALTER statement. Values for other parameters of the sequence not explicitly specified with the ALTER statement remain unchanged in the catalog table.

The value of the START field in SYSIBM.SYSSEQUENCES table row is never modified. The changes to the attributes of a sequence become effective only after the ALTER SEQUENCE is committed. If the ALTER SEQUENCE request is rejected or rolled back, it would be as if it never took place. However, unused cache values may be lost.

DROP SEQUENCE

Use the DROP SEQUENCE statement to drop a sequence.

This statement can be issued interactively, embedded in an application program, or dynamically prepared. You specify the sequence name and the key word RESTRICT.

RESTRICT prevents the sequence from being dropped if any of the following dependencies exists:

- ▶ A trigger exists such that a NEXT VALUE or PREVIOUS VALUE expression in the trigger specifies the sequence.
- ▶ An in-line SQL routine exists such that a NEXT VALUE or PREVIOUS VALUE expression in the routine body specifies the sequence.

DROP of a distinct type should fail if the distinct type is being used by a sequence.

When a sequence is dropped, all privileges on the sequence are also dropped, and plans and packages that refer to the sequence are invalidated.

Dropping a sequence, even if the DROP is rolled back, results in the loss of the still-unassigned cache values for the sequence.

5.12.1 Usage considerations

You should be aware of the following considerations when using sequence objects:

Sequences that cycle

A sequence can be explicitly defined to cycle by specifying the CYCLE keyword. A NO CYCLE option (the default) can be altered to be CYCLE at any time during the life of the sequence.

When the sequence reaches one end point of the logical range, and the cycle option is in effect, the sequence wraps around to the other end point of the range. The 'end points' of the range, in this context, are defined by the (user defined or default) MINVALUE and MAXVALUE values.

Regardless of whether the CYCLE option is in effect or not, a sequence, either a user-defined sequence or an implicit sequence associated with an identity column, can be RESTARTed from any point of the defined or redefined range with the ALTER SEQUENCE statement or the ALTER TABLE (ALTER COLUMN) statement for an identity column.

When defining a sequence with CYCLE, any application conversion tools (for converting applications from other vendor platforms to DB2) should also explicitly specify MINVALUE, MAXVALUE and START WITH.

Range of sequence values and cycles

The word 'range' is used in this document to denote a sequence or series between two end points. The range of a sequence is determined by the combination of the data type and the MINVALUE, MAXVALUE, START WITH and INCREMENT BY values, either user-specified or default. The actual maximum value generated for an ascending sequence or minimum value generated for a descending sequence may not be the same as the MAXVALUE or MINVALUE (user-defined or default), if the INCREMENT BY value is some thing other than 1 or -1. For example, for a sequence defined with MINVALUE = 1, MAXVALUE = 9, START WITH = 1 and INCREMENT BY = 3, the logical range of values is from 1 to 7 although the defined range is from 1 to 9, since 7 is the maximum valid value that can be generated for this sequence.

The first cycle (first set of values) for the sequence always start with the `START WITH` value. Subsequent cycles start with `MINVALUE` for ascending sequence, and `MAXVALUE` for descending sequence.

If `START WITH < MAXVALUE` for an ascending sequence, then the first cycle (first set of values) for the sequence would contain the values starting from `START WITH` up to the maximum possible value within the logical range and the subsequent cycles would contain the values from `MINVALUE` up to the maximum possible value within the logical range.

If `START WITH > MINVALUE` for a descending sequence, then the first cycle (first set of values) for the sequence would contain the values starting from `START WITH` down to the minimum possible value within the logical range and the subsequent cycles would contain the values from `MAXVALUE` down to the minimum possible value within the logical range.

If `START WITH >= MAXVALUE` for an ascending sequence, then the first cycle (first set of values) for the sequence would contain only the `START WITH` value, and the subsequent cycles would contain the values from `MINVALUE` up to the maximum possible value within the logical range.

If `START WITH <= MINVALUE` for a descending sequence, then the first cycle (first set of values) for the sequence would contain only the `START WITH` value, and the subsequent cycles would contain the values from `MAXVALUE` down to the minimum possible value within the logical range.

When `START WITH` falls at the starting end point of the defined range (at `MINVALUE` for ascending sequence or `MAXVALUE` for descending sequence), all cycles of the sequence will have the same number of values.

Defining a constant sequence

It is possible to define a sequence that would always return a constant value. A constant sequence can be used as a numeric global variable. `ALTER SEQUENCE` can be used to change the constant value or to change the constant sequence into a non-constant sequence.

If `INCREMENT BY` is 0, and any of the following conditions is true, then the sequence would generate a single constant value (= `START WITH`) repeatedly.

- ▶ `START WITH` value is within the range defined by `MINVALUE` and `MAXVALUE`
- ▶ `START WITH` value is less than `MINVALUE` for an ascending sequence

This is a constant sequence which does not require the `CYCLE` option to be in effect for the constant value to repeat.

If `INCREMENT BY` is #, but `START WITH` value is greater than `MAXVALUE` for an ascending sequence, which means that `START WITH` is beyond the destination end-point of the range defined by `MINVALUE` and `MAXVALUE`, then the sequence would generate the `START WITH` value once; and then, when the next value is requested, if the `NO CYCLE` option is in effect, it would return the out-of-range error. Otherwise it would wrap around to the other end-point (`MINVALUE` for ascending sequence; `MAXVALUE` for descending sequence), generate the first value of the range, and then generate this last value repeatedly.

If `INCREMENT BY` is not 0, but `MINVALUE = MAXVALUE = START WITH`, then the sequence would first generate the `START WITH` value. In order for this sequence to be a repeating constant sequence, it requires the `CYCLE` option to be in effect. For example, if `MINVALUE = MAXVALUE = START WITH = 1`, then the sequence first generates 1 value, = 1. If the `NO CYCLE` option is in effect for this sequence, then after generating the first value, when the next value is requested it returns the out-of-range error.

Consumed values of a sequence

Once DB2 generates a value for a sequence, that value is said to be *consumed* regardless of whether that value is utilized by the application or not, and is not reused within the current cycle. This is true for both sequences and identity columns.

A consumed value may be unutilized when the statement which caused the value to be generated fails for some reason or is ROLled BACK after the value was generated.

Generated but unused values may constitute gaps in a sequence.

Gaps in a sequence

Consecutive values in a sequence (either a user-defined sequence or an implicit sequence associated with an identity column) differ by the constant INCREMENT BY value specified for the sequence. If it is a sequence associated with an identity column, we refer to DB2-generated values only. However, *gaps* can occur in a sequence. Following is a list of SOME examples of how gaps can be introduced in a sequence:

- ▶ A transaction has advanced the sequence and then rolls back.
- ▶ The SQL statement leading to the generation of the next value fails after the value was generated.
- ▶ The NEXT VALUE expression is used in the SELECT statement of a cursor in a DRDA environment where:
 - The client uses block-fetch, and not all retrieved rows are FETCHed by the application.
 - The sequence or an identity column associated with a sequence is altered and then the ALTER is rolled back
 - The sequence or an identity column table is DROPPed and then the DROP is rolled back.
 - The SYSIBM.SYSSEQ table space is stopped, leading to the loss of unused cache values.
 - DB2 is stopped, leading to the loss of unused cache values.
 - The DB2 subsystem goes down, leading to the loss of unused cache values.

Values of such gaps are not available for the current cycle, unless the sequence is altered and restarted in a specific way as to make them available.

Since a sequence is incremented independently of the transaction, a given transaction incrementing a sequence two times may see a 'gap' in the two numbers that it received if there are other transactions concurrently incrementing the same sequence. Most applications can tolerate these, since these are really not "gaps".

Duplicate sequence values

DB2-generated sequence values are guaranteed to be unique except under the following circumstances, when duplicate values can happen:

- ▶ The CYCLE option is in effect, causing a set of values to be repeated once the sequence reaches one end of its logical range of values. (The sequence could also be a CONSTANT-sequence.)
- ▶ The sequence is RESTARTed WITH a value that has already been generated.
- ▶ The ascending/descending direction of a sequence is reversed by the ALTER statement (by changing INCREMENT BY value from a positive number to a negative number or vice versa). This could cause duplicate sequence values.

- ▶ The system crashes, followed by a COLD START or a CONDITIONAL RESTART that skips forward recovery, leaving the SYSIBM.SYSSEQUENCES table in an inconsistent state.
- ▶ A point-in-time recovery of the SYSIBM.SYSSEQ table space regresses the SYSIBM.SYSSEQUENCES table to a prior point-in-time, causing MAXASSIGNEDVAL to become inconsistent with the actual current point of the sequence.

CACHE considerations

Sequence number allocation can happen faster with caching than without caching, since a range of sequence numbers can be virtually allocated in DB2 memory when caching is in effect.

When a cache is virtually allocated, the first value of this cache is assigned to the sequence, the MAXASSIGNEDVAL column of the SYSIBM.SYSSEQUENCES table is updated to contain the last value of this cache, and then the subsequent values assigned to the sequence come from the set of the cache values that have not been assigned yet, until all the cache 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; and when caching is in effect, the Catalog is only updated when the cache is refreshed.

Choosing a value for CACHE that is not too small allows you to access more successive sequence numbers with fewer I/Os to the catalog table. However, if DB2 goes down in the event of a system failure or shut-down, all still unassigned values in the cache are lost. Such “lost” values represent a gap in the sequence. To remove such a gap in case of a system crash or shut-down, the user would need to determine the actual last value assigned for the sequence, and then ALTER the sequence to RESTART WITH the next logical value.

Effect of CACHE in data sharing

DB2 always assigns the value in order of request. However, when the CACHE option is used, caches are also reserved in order of request. In a data sharing environment where transactions from different members can request numbers from a single sequence, each DB2 member gets its own CACHE containing the next available set of n consecutive numbers to assign, where n is the value specified for CACHE. This means that each member gets the sequence values from its own cache. Thus, since DB2 generated numbers are assigned in order of request but possibly from different simultaneously-existing caches, in the data sharing environment the numbers may not be in strict numeric order, although guaranteed to be unique.

For example, assume a sequence named SEQ1 that was defined with START WITH = 1, INCREMENT BY = 1, CACHE = 20, in a data sharing environment. A transaction from member DB2A requests the first value for SEQ1. It gets the value 1 from its cache of values 1 to 20. Now another transaction from member DB2B requests the next value for SEQ1. Since each DB2 member gets its own separate cache, DB2B gets the value 21 from its cache of values 21 to 40. If DB2A issues the next request it would get the value 2. Each member gets the values from its own cache in numeric order until the cache is exhausted and the next available cache is allocated; but the numbers assigned are 1, 21, 2, ... in this example.

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 are always in strict numeric order, since there is only one active cache at any given time.

5.12.2 Using sequences in applications

To create a sequence, use the CREATE SEQUENCE statement. After you create the sequence, GRANT usage/alter privileges to all users who should be able to use the sequence.

An application can retrieve the next value in a sequence, or see what its current sequence value is.

To retrieve the first value and the successive values one by one, use the NEXT VALUE expression specifying the sequence name. A new sequence number is generated every time the NEXT VALUE expression is invoked, until all values of the range are exhausted. However, if NEXT VALUE is invoked multiple times within a query, the number generation happens only once for each row of the result.

To retrieve the value generated previously within the current session, (after at least one value has been generated), use the PREVIOUS VALUE expression specifying the sequence name. The current sequence number can be repeatedly referenced using PREVIOUS VALUE. There may be multiple instances of a PREVIOUS VALUE expression specifying the same sequence name within a single statement.

Once defined, sequences can be efficiently used by many users; 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.

Example 5-12 shows how to create and use a sequence named “order_seq” and use it for a table named “orders”.

Example 5-12 Creating and using a sequence

```
CREATE SEQUENCE order_seq
START WITH 1
INCREMENT BY 1
NOMAXVALUE
NOCYCLE
CACHE 20
INSERT INTO orders (orderno, custno)
VALUES (NEXT VALUE FOR order_seq, 123456);
```

or,

```
UPDATE orders
SET orderno = NEXT VALUE FOR order_seq
WHERE custno = 123456;
```

or,

```
SELECT NEXT VALUE FOR order_seq INTO :hv_seq from orders;
```

A program can use the same sequence number as a unique key value in two separate tables by referencing the sequence number with a NEXT VALUE expression for the first row (this generates the sequence value), and a PREVIOUS VALUE expression for the other rows (the instances of PREVIOUS VALUE refer to the sequence value most recently generated).

```
INSERT INTO orders (orderno, custno)
VALUES (NEXT VALUE FOR order_seq, 123456);
INSERT INTO line_items (orderno, partno, quantity)
VALUES (PREVIOUS VALUE for order_seq, 987654, 1);
```

If NEXT VALUE is invoked in the same statement as the PREVIOUS VALUE then, regardless of their order in the statement, PREVIOUS VALUE returns the previous non-incremented value, and NEXT VALUE returns the next value.

To change the attributes of a sequence object, use the ALTER SEQUENCE statement.

5.13 Identity columns enhancements

Some of the new features introduced in sequences are also made available to identity columns in DB2 V8.

Here we provide a list of the identity column enhancements:

- ▶ Allow dynamic ALTER of identity column attributes. The ALTER TABLE (ALTER COLUMN) SQL statement is extended to allow modifying the attributes of an existing identity column.
- ▶ Support the following keywords as part of identity column attribute specification in order to aid porting from other vendor implementations and stay in sync with sequences:
 - NO MINVALUE
 - NO MAXVALUE
 - NO ORDER
 - ORDER

The single-word keywords NOMINVALUE, NOMAXVALUE, NOCYCLE, NOCACHE, and NOORDER can be used as alternatives to the corresponding two-word variations.

- ▶ Allow separator commas between identity column attribute specifications to be optional (instead of required) when the identity column is defined.
- ▶ Allow INCREMENT BY to be 0 for identity columns.
- ▶ Allow MINVALUE to be less than OR equal to MAXVALUE (instead of less than only) for identity columns.

5.13.1 SQL statements for identity column enhancements

The ALTER COLUMN clause of the ALTER TABLE SQL statement is extended to include identity column specifications, to allow modifying the attributes of an existing identity column, as described below.

ALTER TABLE statement

Use the “ALTER COLUMN” clause of the “ALTER TABLE” SQL statement to modify the attributes of an existing identity column, and optionally, to specify continuation of the sequence associated with the 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 the column will be affected by the changes made using the ALTER TABLE statement with the ALTER COLUMN clause.

The data type of an identity column CANNOT be altered. To change the data type, the table containing the column must be dropped and recreated.

The unused cache values of an identity column may be lost when the column attributes are altered.

ALTERing an identity column's sequence attributes results in the update of relevant columns of the row that describes the sequence associated with the identity column in the

SYSIBM.SYSSEQUENCES catalog table. The row is updated to reflect the new values for the parameters explicitly specified with the ALTER statement.

Values for other parameters of the sequence not explicitly specified with the ALTER statement remain unchanged in the catalog table. The value of the START field in SYSIBM.SYSSEQUENCES table row is never modified.

The user is responsible for the effects of ALTERing attributes of the identity column. No check is done by DB2, at ALTER time, in the context of the existing sequence associated with the column. For example, if an ascending sequence is now altered to become a descending sequence (which could create the possibility of duplicate values if NO CYCLE), or the range of values is extended or shortened by the ALTER, DB2 issues no warning for the ALTER.

The changes to the attributes of an identity column become effective only after the ALTER statement is committed.

If the ALTER request is rejected or is ROLLED BACK, it would be as if it never took place. However, unused cache values may be lost.

Any of the attributes of an identity column, except data type, can be specified as part of the "ALTER TABLE" statement with the "ALTER COLUMN <IdentityColumnName>" clause.

Figure 5-25 shows the syntax for the ALTER TABLE ALTER COLUMN enhancement.

Altering identity column attributes

```
ALTER TABLE ALTER COLUMN <identity-column-name>  
SET GENERATED ALWAYS / BY DEFAULT  
SET INCREMENT BY <numeric value>  
SET NO MINVALUE / MINVALUE <numeric value>  
SET NO MAXVALUE / MAXVALUE <numeric value>  
SET NO CYCLE / CYCLE  
SET NO CACHE / CACHE <integer value>  
RESTART / RESTART WITH <numeric value>
```

Figure 5-25 Identity columns — altering attributes

If SET GENERATED ALWAYS or SET GENERATED BY DEFAULT is explicitly specified in the ALTER statement, then the information is recorded in the DEFAULT column of the SYSIBM.SYSCOLUMNS table.

If SET GENERATED ALWAYS and SET GENERATED BY DEFAULT are both NOT explicitly specified in the ALTER statement, then the information in the DEFAULT column of the SYSIBM.SYSCOLUMNS table remains unchanged.

If INCREMENT BY *numeric-constant* is explicitly specified in the ALTER statement, then the value in the INCREMENT column of the SYSIBM.SYSSEQUENCES table row is updated to contain the new value.

If *MINVALUE numeric-constant* is explicitly specified in the ALTER statement, then the value in the MINVALUE column of the SYSIBM.SYSSEQUENCES table row is updated to contain the new value.

If *MAXVALUE numeric-constant* is explicitly specified in the ALTER statement, then the value in the MAXVALUE column of the SYSIBM.SYSSEQUENCES table row is updated to contain the new value.

If *CYCLE* is explicitly specified in the ALTER statement, then the value in the CYCLE column of the SYSIBM.SYSSEQUENCES table row is updated to indicate that the CYCLE option is in effect.

If *CACHE* is explicitly specified in the ALTER statement, then the value in the CACHE column of the SYSIBM.SYSSEQUENCES table row is updated to contain the new cache value.

If *RESTART* is specified without the *WITH numeric-constant*, then the sequence associated with the identity column that is being altered is restarted at the *START WITH* value specified implicitly or explicitly when the identity column was defined.

The value of the *START* field in SYSIBM.SYSSEQUENCES table row is not modified when the sequence is altered to *RESTART* (with or without a value).

Using the *RESTART* option could cause sequence numbers to be duplicates of values generated by the sequence previously.

Other identity column enhancements

In addition to allowing dynamic ALTER of identity column attributes described above, the following identity column enhancements are also implemented:

- ▶ Support the following keywords as part of identity column attribute specification (during CREATE TABLE AND ALTER-ADD of an identity column) in order to aid porting from other vendors and stay in sync with sequences:
 - NO MINVALUE
 - NO MAXVALUE
 - NO ORDER
- ▶ The single-word keywords *NOMINVALUE*, *NOMAXVALUE*, *NOCYCLE*, *NOCACHE*, and *NOORDER* as alternatives to the corresponding two-word variations.
- ▶ Allow separator commas between identity column attribute specifications to be optional (instead of required) when the identity column is defined.
- ▶ Allow *INCREMENT BY* to be 0 for identity columns.
- ▶ Allow *MINVALUE* to be less than OR equal to *MAXVALUE* (instead of less than only) for identity columns.
- ▶ Prevent loss of unused cache values at the end of a utility LOAD for an identity column. There is no LOAD involvement with sequences.

5.14 Sequences and identity columns comparison

Sequences are stand-alone sequence-objects created at user request. They are used by users for whatever purpose they may choose, while identity columns are sequence-objects generated and maintained by DB2, and are associated with a particular table.

Here is a list of the differences:

- ▶ The NEXTVAL/PREVVAL expressions (used for retrieving the next/previous value of a sequence) cannot be used for identity columns. An identity column value can either be SELECTed, or retrieved using the IDENTITY_VAL_LOCAL function invocation.
- ▶ The ALTER SEQUENCE statement cannot be used for identity columns. Identity column attributes can be altered by using the “ALTER TABLE(ALTER COLUMN)” statement.
- ▶ The DROP SEQUENCE, COMMENT ON SEQUENCE, and GRANT/REVOKE...SEQUENCE statements cannot be used for identity columns.
- ▶ Sequences and DB2 V8 identity columns allow INCREMENT BY to be 0, whereas V6 and V7 identity columns do not.
- ▶ Sequences and DB2 V8 identity columns allow MINVALUE to be equal to MAXVALUE. For V7 identity columns, MINVALUE must be less than MAXVALUE. V6 identity columns do not support these keywords.
- ▶ The set of keywords with which identity column attributes are defined in V6 and V7, with the exception of the GENERATED keyword, is a subset of the full set of keywords for defining the attributes of a sequence and DB2 V8 identity columns.
 - In V6, identity columns do not support the CYCLE/NO CYCLE, MINVALUE/NO MINVALUE, MAXVALUE/NO MAXVALUE.
 - The V7 identity columns do not support NO MINVALUE and NO MAXVALUE keywords.

In Figure 5-26 we summarize these differences.

Applications can use sequences to avoid the concurrency and performance problems that can result when they generate their own sequence numbers.

Sequence	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

Figure 5-26 Sequences and identity columns

5.15 Multilevel security

Security, privacy, and auditing have become more important in the last few years. One high priority requirement is for row-level security for applications that need more granularity in their security schemes. For example, in organizational hierarchies, it is desirable to set up a corresponding security hierarchical scheme in which employees can see their own payroll data, a first line manager can see payroll information on all of the reporting employees, and so on. In addition, government security schemes often include a security hierarchy such as TOP SECRET, SECRET, or UNCLASSIFIED.

Also, Web hosting companies need to store multiple customers' data into a single DBMS, and security and laws on privacy demand row level security. The granularity must be extended from table level to row level for individual user access to be restricted to a specific set of rows. Traditionally, views and joins have been the application solution to limit access to selected rows and columns, but they are cumbersome to construct with the desired level of granularity and not very effective for update/insert/delete. Triggers, database constraints, and stored procedures are often needed for update control.

DB2 V8 has several additional security related functions, the RACF exit has additional capabilities for sequences, the additional REFRESH privilege for MQTs, and the change in the interface necessary to handle long names.

A major security enhancement with DB2 V8 is the multilevel security (MLS) with row granularity introduced to support the types of hierarchical security schemes mentioned above. This support combines with new RACF access control functions available with z/OS V1R5.

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.

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 example in see Figure 5-27, an application wants to have a hierarchy representing the colors of the rainbow.

At the top of the hierarchy, RAINBOW would be a security label that includes all the colors (RED, ORANGE, YELLOW, GREEN, BLUE, INDIGO, VIOLET). At the middle of the hierarchy, you could have other security labels: PASTEL (BLUE, INDIGO, VIOLET) and SUNSET (RED, ORANGE, YELLOW).

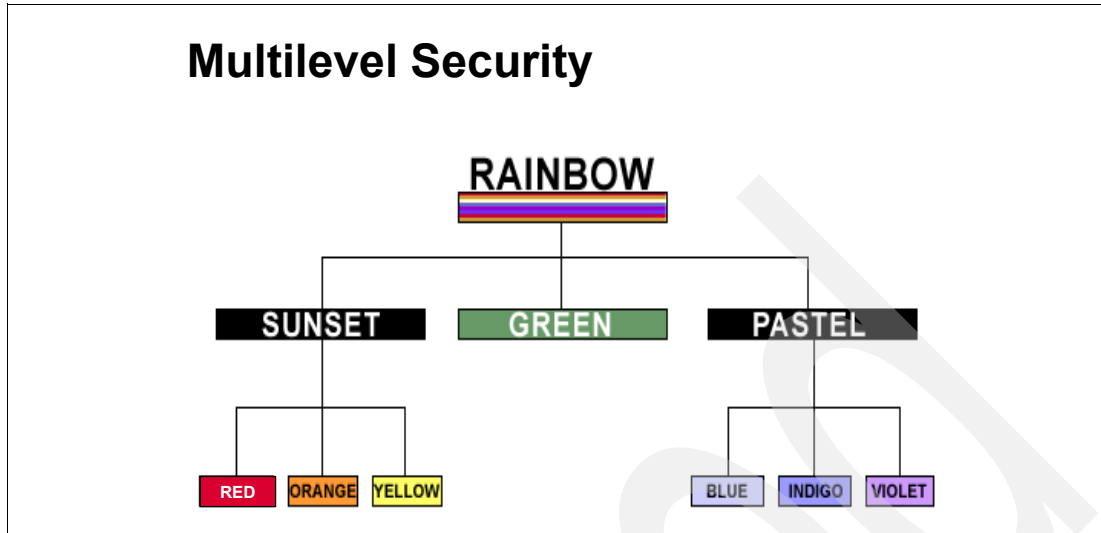


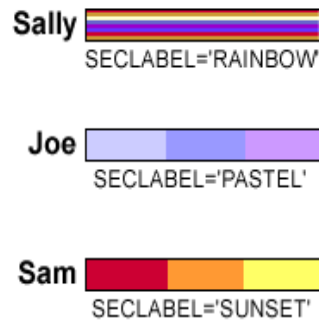
Figure 5-27 Multilevel security hierarchy

Basically, you are able to activate this support on a table base, and at row level, by adding a column that acts as the security label. See Figure 5-28.

The first step is to establish the MLS structure in RACF with the security labels. You then add a column to your table for the security label or *seclabel*. You can now use the SecureWay Security Server functions available with z/OS V1R5 for MLS access control. Each row value has a specific security label value provided by RACF and saved in rows for INSERT, UPDATE, LOAD. The seclabel in each row is used to compare with the seclabels for the DB2 users for authority checking. If access is allowed, then normal access to the data row takes place. If access is not allowed, then data is not returned. Runtime user seclabel values for data checking are cached to minimize CPU usage.

With the hierarchy established in RACF, the system understands that users with authority to access RAINBOW can access anything. Users with authority to access PASTE information can access any row associated with BLUE, INDIGO, VIOLET, or PASTE. Users with SUNSET can access SUNSET, RED, ORANGE, YELLOW. This is much more powerful than just having an exact match on security label, that is, user's label must exactly match the data's label, since it has the notion of *groups* which makes security administration easier to manage and is an expansion of the existing security concepts.

MLS with row granularity



DB2 SECURITY LABEL_EXT	COL1	COL2	COL2
RAINBOW	56	7	76
RAINBOW	24	56	65
RAINBOW	42	6	45
BLUE	3	456	7
INDIGO	113	456	56
VIOLET	3	456	4
BLUE	4	456	7
RED	4	76	567
ORANGE	33	7	567
RED	5455	76	567
YELLOW	999	65	45

Figure 5-28 Row granularity with seclabel

With this additional capability, DB2 is able to implement that type of security scheme without requiring the application to access the data using special views or predicates.

The terms for comparing seclabels differ from relational algebra, and the combination of hierarchies and non-hierarchical categories means that the result of a comparison for valid seclabels has four possible values:

- ▶ Dominate: greater than or equal to
- ▶ Reverse dominate: less than or equal to
- ▶ Equivalence: equal to

Equivalent means that the seclabels are the same or have the same level and set of categories. One way to check this is to determine whether both dominance and reverse dominance are true.

- ▶ Null: none of the above

SECLABEL definition

To enable the row level security the table must have a column defined in the CREATE/ALTER TABLE statement with the column-option

AS SECURITY LABEL

Once created with seclabel, the table cannot be disabled. Any column name can be the security label, but the same column name cannot be used more than once in the same table. Only one security label is allowed in a table.

The security label column must be data type single byte character, char(8), NOT NULL WITH DEFAULT. This column cannot have field procedures, edit procedures or check constraints.

Audit records IFCID 0142 are produced every time the table with security label is created, altered or dropped.

The security mechanism on a table with MLS row level granularity is mandatory and automatic, each user has to be identified to SecureWay Security Server with a valid seclabel, if the user is not known, authorization error occurs and an audit record is produced. The verification works as follows on DML:

► **SELECT:**

If the user has a valid seclabel, its value is compared with the data seclabel of the row to be selected, and if the user seclabel dominates the data seclabel, the row is returned.

If the user seclabel does not dominate the data seclabel, then the row is not included in data returned, but no error is reported.

► **INSERT:**

For user with valid seclabel, then the value of the data seclabel column for that row to be inserted is set to the value of the user seclabel. If a user does not have the write-down privilege, then the seclabel of inserted rows will be exactly the current seclabel. If the user does have the write-down privilege, then he or she can set the value of the seclabel column to any value.

► **UPDATE:**

For user with valid seclabel, it is compared with the data seclabel of the row to be updated, if the seclabels are the same then the row updates, if the seclabels are not the same, then both dominance and reverse dominance are checked. Update is permitted if both are true and the value of the original data seclabel in the updated row is set to the value of the user seclabel. A user who has write down authority can access and delete down-level (dominance) rows, but not up-level (reverse dominance) rows.

► **DELETE:**

For user with valid seclabel, it is compared with the data seclabel of the row to be deleted if the seclabels are the same, then row deletes, if the seclabels are not the same, then both dominance and reverse dominance are checked. The delete is permitted only if both are true. A user who has write down authority can access and delete down-level (dominance) rows, but not up-level (reverse dominance) rows.

Utilities have been updated to reflect MLS; the user must be identified to RACF and have a valid ACEE:

► **LOAD RESUME (similar to INSERT)**

Without write down permission, seclabel is set to the current seclabel. With write down permission, the user is permitted to specify the seclabel.

► **LOAD REPLACE**

This deletes all rows, so write down authority is required.

► **UNLOAD and REORG UNLOAD EXTERNAL (similar to SELECT)**

Only rows can be unloaded if the user seclabel dominates the data seclabel. No error is returned if this is not true, but the row is not unloaded.

► **UNLOAD and REORG UNLOAD EXTERNAL (similar to SELECT)**

Rows can only be unloaded if the user seclabel dominates the data seclabel. No error is returned if this is not true, but the row is not unloaded.

► **REORG DISCARD (similar to DELETE)**

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 then the row is discarded. If the check for dominance and reverse dominance of the two seclabels are true, then the row is discarded; otherwise, the row is not discarded.

Note that there are requirements for z/OS V1R5 and the Security Server (RACF) V1R5 (or equivalent function).

MLS scope and requirements

The use of MLS requires z/OS V1R5 and the Security Server (RACF) V1R5 (or equivalent function).

The following restrictions also apply:

- ▶ Referential constraints cannot be defined on a security label column.
- ▶ Sysplex parallelism is not used for queries that access a table with a security label column.
- ▶ Field procedures are not allowed on a security label column.
- ▶ Edit procedures are not allowed on a security label column.
- ▶ Trigger transition tables do not have security labels.

Row-level access controls can be used with native DB2 access controls or with RACF access controls. If you use RACF access controls, then you can define multilevel security for other objects. Security labels can be used also to define the access controls on most DB2 objects such as subsystem or data sharing group, database, table space, table, view, schema, plan, package, collection, stored procedure, UDFs, Java ARchive (JAR), distinct type, sequence, storage group, and buffer pool. Then, access will require both the discretionary access control (PERMIT) and the mandatory access control (seclabel comparison). The grouping will depend upon the hierarchy of the objects. In general, so you will define the seclabel of an object higher in the object hierarchy to dominate all objects within it.

5.16 MQSeries UDFs

DB2 and MQSeries can be used to construct applications that combine messaging and database access. It is now possible to integrate MQSeries messaging operations within SQL statements. This is accomplished via a set of User Defined Functions (UDFs) which incorporate the MQSeries Application Messaging Interface (AMI).

The same functionality is available also for DB2 V7 by applying the PTF for APAR PQ59549.

The capabilities of MQSeries UDFs are:

- ▶ Send and forget
- ▶ Read or receive
- ▶ Request/response

In Figure 5-29 we show a basic MQSeries configuration.

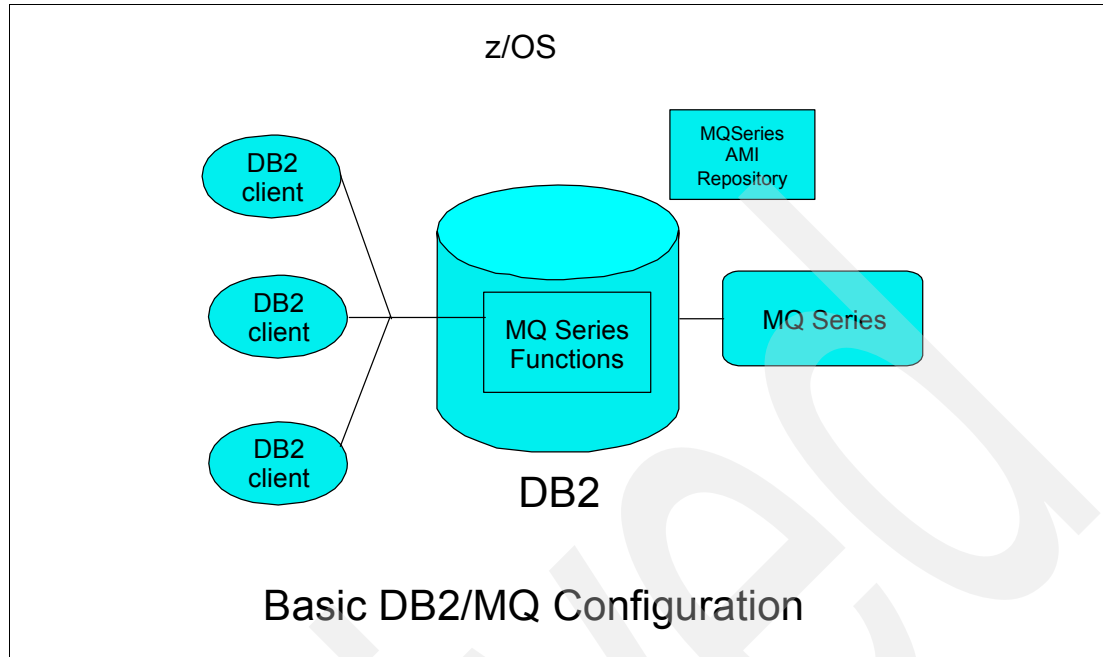


Figure 5-29 Basic DB2/MQSeries configuration

The names of the DB2 MQSeries UDFs are as follows:

- ▶ `MQRead('receive-service' , 'service-policy')`
 - Returns message at the head of the queue specified by *receive-service*
 - Uses the quality of service policy defined in *service-policy*
 - Returns a VARCHAR(4000) containing the message
 - Does not remove the message from the queue
- ▶ `MQReadAll('receive-service' , 'service-policy' , 'num-rows')`
 - Returns a table containing the messages and message metadata from the MQSeries location specified by *receive-service*
 - Uses the quality of service policy *service-policy*
 - Does not remove the message from the queue associated with *receive-service*
- ▶ `MQReadClob('receive-service' , 'service-policy')`
 - Returns a message at the head of the queue specified by *receive-service*
 - Uses the quality of service policy defined in *service-policy*
 - Does not remove the message from the queue
 - Returns a CLOB of 1MB maximum length, containing the message
- ▶ `MQREADAllClob('receive-service' , 'service-policy' , 'num-rows')`
 - Returns a table containing the messages and message metadata from the MQSeries location specified by *receive-service*
 - Uses the quality of service policy *service-policy*
 - Does not remove the messages from the queue
 - Returns a maximum of *num-rows* messages if *num-rows* is specified
 - Returns all messages if *num-rows* is not specified

- ▶ MQSend('send-service' , 'service-policy' , 'msg-data' , 'correl-id')
 - Sends the data contained in msg-data to the MQSeries location specified by *send-service*
 - Uses the quality of service policy defined by *service-policy*
 - Specifies a *correl-id* for a message (optional)
 - Returns a value of '1' if successful or a '0' if unsuccessful
- ▶ MQReceive('receive-service' , 'service-policy' , 'correl-id')
 - Returns a message from the MQSeries location specified by *receive-service*
 - Uses the quality of service policy *service-policy*
 - Returns the message at the head of the queue if *correl-id* is not specified
 - Return value is a VARCHAR(4000) containing the message
 - Returns null if no messages are available
 - Removes the message from the queue associated with *receive-service*
- ▶ MQReceiveAll('receive-service' , 'service-policy' , 'correl-id' , 'num-rows')
 - Returns a table containing the messages and message metadata from the MQSeries location specified by *receive-service*
 - Uses the quality of service policy *service-policy*
 - Removes the messages from the queue associated with *receive-service*
 - Returns messages with a matching correlation identifier if *correl-id* is specified
 - Returns a maximum of *num-rows* messages if *num-rows* is specified
- ▶ MQReceiveClob('receive-service' , 'service-policy' , 'correl-id')
 - Returns a message from the MQSeries location specified by *receive-service*
 - Uses the quality of service policy *service-policy*
 - Removes the message from the queue associated with *receive-service*
 - Returns the first message with a matching correlation identifier if *correl-id* is specified
 - Returns a CLOB with a maximum length of 1MB containing the message
 - Returns NULL if no messages are available
- ▶ MQReceiveAllClob('receive-service' , 'service-policy' , 'correl-id' , 'num-rows')
 - Returns a table containing the messages and message metadata from the MQSeries location specified by *receive-service*
 - Uses the quality of service policy *service-policy*
 - Removes the messages from the queue associated with *receive-service*
 - Returns a CLOB with a maximum length of 1MB containing the message
 - Returns a maximum of *num-rows* messages if *num-rows* is specified

DB2 provides two flavors of these MQSeries functions. One flavor supports only single-phase commit and is identified by a schema name of DB2MQ1C. The second flavor supports two-phase commit and is identified by a schema name of DB2MQ2C. Each flavor of the functions should run under a separate WLM environment and WLM will need to be customized for running MQSeries UDF support. In Figure 5-30 you can see a high level view of this environment.

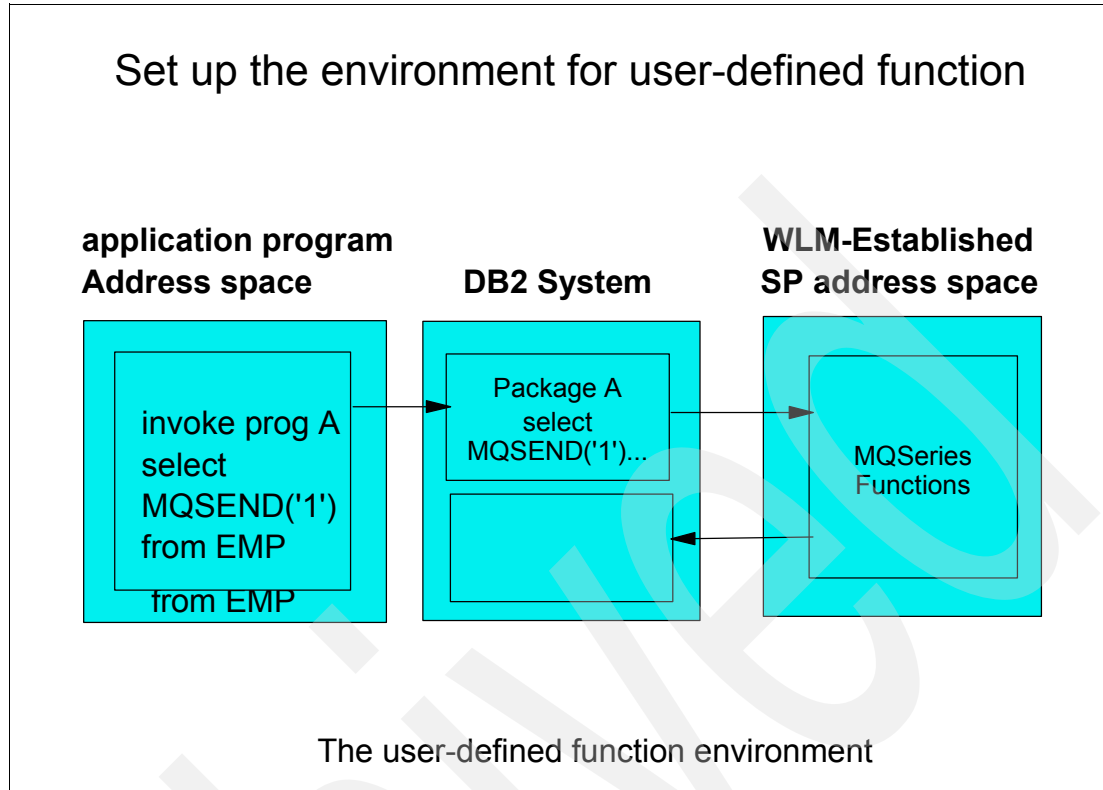


Figure 5-30 MQSeries UDF environment

5.17 ASCII flag for compile

z/OS V1R2 C and C++ compilers have introduced a new compiler option called the ASCII option. This option, when specified, sets a flag in the control structure produced by the compiler and used by the LE environment and tells LE that the program is an ASCII program, although the program itself is written in EBCDIC. The argv input to main is in ASCII and library functions expect ASCII arguments. Thus, the compiler converts all of character constants and string literals that appear in the program from the codepage that the source is written in to ASCII (ISO8859-1 codepage for character constants and literals or UCS-2 for wide constants and literals).

The precompiler generates some character string literals in the modified source program. Character string literals currently generated by the Precompiler are:

- ▶ Program name in RDI
- ▶ SQLDA ID
- ▶ Location-name in the CONNECT statement
- ▶ Procedure-name in the CALL statement
- ▶ Cursor-name in the ALLOCATE CURSOR statement

In order for DB2 for z/OS to function properly, these precompiler generated string literals will not be converted to ASCII by the compiler even when the ASCII option is specified. These string literals will always be generated by the precompiler as hexadecimal string literals instead of character string literals. In Example 5-13 we can see the effects of the ASCII option on a program.

Example 5-13 Sample program with ASCII option

Here is the sample program after being processed by the precompiler and with the ASCII option turned on

```
/***/  
EXEC SQL CONNECT TO "SANTA_TERESA_LAB"  
$$$***/  
strcpy( SQLTEMP,  
"\xE2\xC1\xD5\xE3\xC1\x6D\xE3\xC5\xD9\xC5\xE2\xC1\x6D\xD3\xC1\xC2");  
{  
SQLPLIST SQLPLIST1 =  
{64, 16384, 30, "\xD7\xD4\xC7\xF8\xF4\xF1\x40\x40", 0, 0, 0, 0,  
0, 0, 0, 0, 0, 769, 23};  
SQLELTS_PTR SQLELTS_PTR1;  
struct  
{ char SQLDAID??(8??);  
long SQLDABC;  
short SQLN;  
short SQLD;  
char SQLPVLT??( (sizeof(SQLELTS) * 1) ??);  
} SQLPVAR1;  
SQLELTS_PTR1 = (SQLELTS *) &SQLPVAR1.SQLPVLT;  
SQLELTS_PTR1->SQLTYPE = 460;  
SQLELTS_PTR1->SQLLEN = 17;  
SQLELTS_PTR1->SQLADDR = (char *)  
&( SQLTEMP );  
SQLELTS_PTR1->SQLIND = NULL;  
SQLELTS_PTR1->LENGTH = 8;  
memcpy(SQLELTS_PTR1->DATA,  
"\x00\x00\x03\xA2\x00\x00\x00\x00", 8);  
SQLELTS_PTR1 = SQLELTS_PTR1 + 1;  
strcpy(SQLPVAR1.SQLDAID, "\xE2\xD8\xD3\xC4\xC1\x4E\x40\x08");  
SQLPVAR1.SQLDABC = 60;  
SQLPVAR1.SQLN = 1;  
SQLPVAR1.SQLD = 1;  
SQLPLIST1.SQLVPARAM = (char *) &SQLPVAR1.SQLDAID;  
SQLPLIST1.SQLCODEP = (char *) &sqlca;  
SQLPLIST1.SQLTIMES??( 0 ??) = 0x16FC;  
SQLPLIST1.SQLTIMES??( 1 ??) = 0xDECB;  
SQLPLIST1.SQLTIMES??( 2 ??) = 0x1E53;  
SQLPLIST1.SQLTIMES??( 3 ??) = 0x200C;  
DSNHLI ( (unsigned int *) &SQLPLIST1);  
}
```

Archived



e-business

In this chapter we discuss the following topics:

- ▶ IBM DB2 Universal Driver for SQLJ and JDBC features
- ▶ Unicode support
- ▶ ODBC enhancements
- ▶ XML publishing functions
- ▶ CURRENT PACKAGE PATH special register
- ▶ DDF communication database enhancements
- ▶ Enhancements for stored procedures and UDFs
- ▶ Miscellaneous enhancements

6.1 IBM DB2 Universal Driver for SQLJ and JDBC features

The current client (prior to DB2 V8) for Linux, UNIX, Windows and OS/2 platforms provides 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, Windows application server, or the client for a z/OS database server.

Currently, access to a Linux/UNIX/Windows (LUW) server and a z/OS server use 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, and 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 Driver. The first deliverable of this new architecture is the *IBM DB2 Universal Driver for SQLJ and JDBC Version 1.0*, also known as the IBM Java Combined Client.

The Universal Driver is architected as an abstract JDBC processor that is independent of driver-type connectivity or target platform. (More information on driver types, see 6.1.1, “IBM JDBC Type 4 driver” on page 131.) The IBM DB2 JDBC 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 defined by Sun. Because the Universal Driver is an abstract machine, driver types become connectivity types.

This abstract JDBC machine architecture is independent of any particular JDBC driver-type connectivity or target platform, allowing for both all-Java connectivity (Type 4) or 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.

Objectives

The new common runtime environment fulfills the following key requirements:

- ▶ Have a single driver for Linux, UNIX, Windows and z/OS. This eliminates the major cause of today’s Java porting problems and also enables to deliver performance and functional enhancements quicker (since they only need to be developed once).
- ▶ Enhance the current API to provide a fully compliant JDBC 2.0 driver, for both a Type 2 and Type 4 JDBC driver. The enhanced functionality will not be in the current Type 2 driver. It is only made available in the new Universal Client based Type 2 driver. The current Type 2 driver will be shipped for compatibility reasons, but will not be enhanced.
- ▶ Reduce the client footprint. Footprint reduction is achieved by eliminating the multiple layers of processing which reduces both disk and memory consumption on the client. An additional gain is made by partitioning the client into three distinct components:
 - A C based client supporting all SQL and CLI/ODBC access (this will not be discussed any further in this section, as we are focusing on the Java part).
 - A Java based client supporting all SQLJ and JDBC access.

- An administrative client providing a consistent set of administrative function, including replication, across all platforms. This is not discussed any further, as we concentrate on the Java client.

Each of the components above can be installed separately, or as any combination of the three, to allow consistent access to both a LUW server, and a z/OS server without any additional installation steps.

- ▶ Provide a full Java application development process for SQLJ, by:
 - Providing a fully portable customized SQLJ profile
 - Enabling the bind of DB2 packages from the client (using the Type 4 driver)
- ▶ Trace improvements, by allowing:
 - Turning traces on and off dynamically, and
 - Allowing multiple levels of tracing, with different levels of detail

The new Universal Driver for SQLJ and JDBC is made available in DB2 V7 as well via the maintenance stream.

Benefits for DB2 UDB for z/OS

At first glance this change might look like something that only impacts DB2 UDB for LUW and DB2 Connect users. This is certainly not the case, as explained hereafter:

- ▶ First, and most importantly, because of a common code base, the functions provided on DB2 UDB for LUW and DB2 UDB for z/OS is exactly the same, not just similar. This largely improves DB2 Family compatibility. For example, it enables users to develop on LUW, and deploy on z/OS without having to make any change.
- ▶ As mentioned before, there are also many functionality enhancements to the Java API for the (Universal Driver based) Type 2 and the new Type 4 JDBC driver, to make it fully compliant with the JDBC 2.0 standard.
- ▶ With the elimination of the “private protocols” used by the LUW clients in previous versions, using DRDA will render better performance.
- ▶ 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.

Functional enhancements

Not only will the new DB2 Universal Driver bring more consistent application behavior, it also introduces several new functions, making it fully JDBC 2.0 compliant, such as:

- ▶ Java API enhancements
- ▶ Nested stored procedure result sets for JDBC and ODBC applications
- ▶ Extended DESCRIBE
- ▶ SQLcancel
- ▶ LOB streaming

6.1.1 IBM JDBC Type 4 driver

Not only does DB2 V8 provide a new JDBC driver architecture, known as the IBM DB2 Universal Driver for SQLJ and JDBC. IBM also delivers a JDBC Type 4 driver for the first time. This driver is also shipped with DB2 UDB for LUW Version 8.

As a reminder, we give a brief description of the JDBC driver architectures 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.
- Type 2:** Drivers that are written partly in the Java programming language, and partly in native code. The drivers use a native client library specific to the data source to which they connect. Again, because of the native code, their portability is limited. Notice that a Type 2 has a native component that is part of the driver and is separate from the database access API.
- Type 3:** Drivers that use a pure Java client and communicate with a middleware server using a database independent protocol. 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 the Universal Driver, the DRDA protocol is used to talk directly to the data source.

Note: The number in the driver type has no meaning whatsoever. 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.

Using the Type 4 driver, a client application can now talk directly to DB2 UDB for z/OS, without going through DB2 Connect (although a DB2 Connect licence is required to be able to use the Type 4 driver).

The support of a Type 4 driver, combined with a fully portable SQLJ customized profile, will allow WebSphere to provide better tooling to support the development of SQLJ applications.

6.1.2 New IBM JDBC Type 2 driver

In addition to a brand new Type 4 driver, DB2 V8 delivers a new Type 2 driver as well. It is also based on the same common code base of the Universal Driver for SQLJ and JDBC. The current Type 2 driver (available since DB2 V5) will still be shipped with V8 for compatibility reasons, but will not be enhanced. All enhancements described hereafter are only available with the new Type 2 driver delivered with Universal Driver for JDBC and SQLJ.

6.1.3 Java API enhancements

The current API has been enhanced to provide a fully compliant JDBC 2.0 driver, for both the Type 2 and Type 4 JDBC driver. The enhanced functionality will not be made available in the current Type 2 driver. It will only be made available in the new DB2 Universal Driver based, Type 2 driver. Among these enhancements are:

- ▶ Scrollable cursor support (exploiting DB2 engine scrolling).
- ▶ Batch updates support.
- ▶ Improved security for DB2 authentication.
- ▶ Improved Java SQL error information via the `DB2Diagnosticable` class. This class allows reporting of the contents of the SQLCA and SQL error message text.

- ▶ “Native” DB2 server SQL error messages can be returned when explicitly requested by the `getSQLErrorMessage()` method. Each DB2 server provides an “error message” stored procedure to allow a DB2 Client to retrieve the “native” message text from the target DB2 server.
- ▶ Java API for Set Client Information (SQLESETI).

In the meantime, development is underway to deliver a JDBC 3.0 compliant Universal Driver.

6.1.4 SQLJ

SQLJ support has been around for a while now. It provides superior performance, because it uses static SQL (in contrast to JDBC that uses dynamic SQL), and 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.

Existing SQLJ program preparation process

Figure 6-1 shows the existing SQLJ program preparation process. After creating the serialized profile by means of the SQLJ translator, you have to execute the `db2prof` 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 `db2prof` utility. This is because `db2prof` creates DBRMs, which are a DB2 UDB for z/OS and OS/390 only feature.

The `db2prof` utility also creates a customized serialized profile. Unfortunately, after customization, the profile is no longer portable.

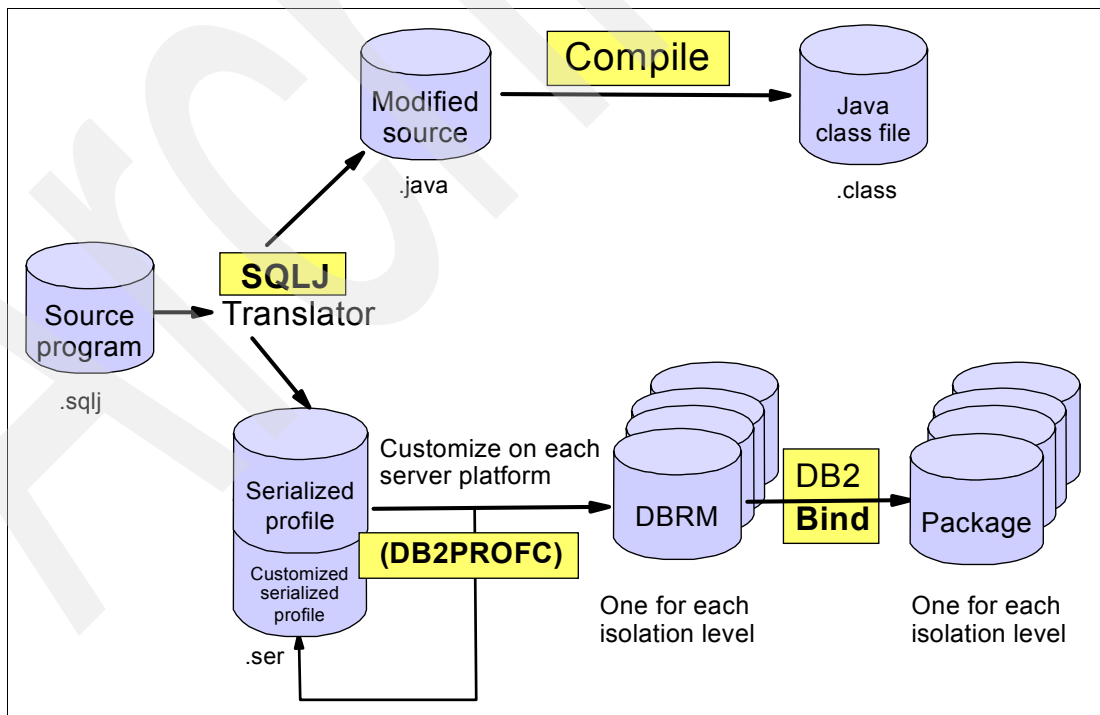


Figure 6-1 Existing SQLJ preparation process

Universal Driver SQLJ program preparation process

Using the new Universal Driver, DBRMs (or .bnd files) are no longer required, as shown in Figure 6-2. 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.

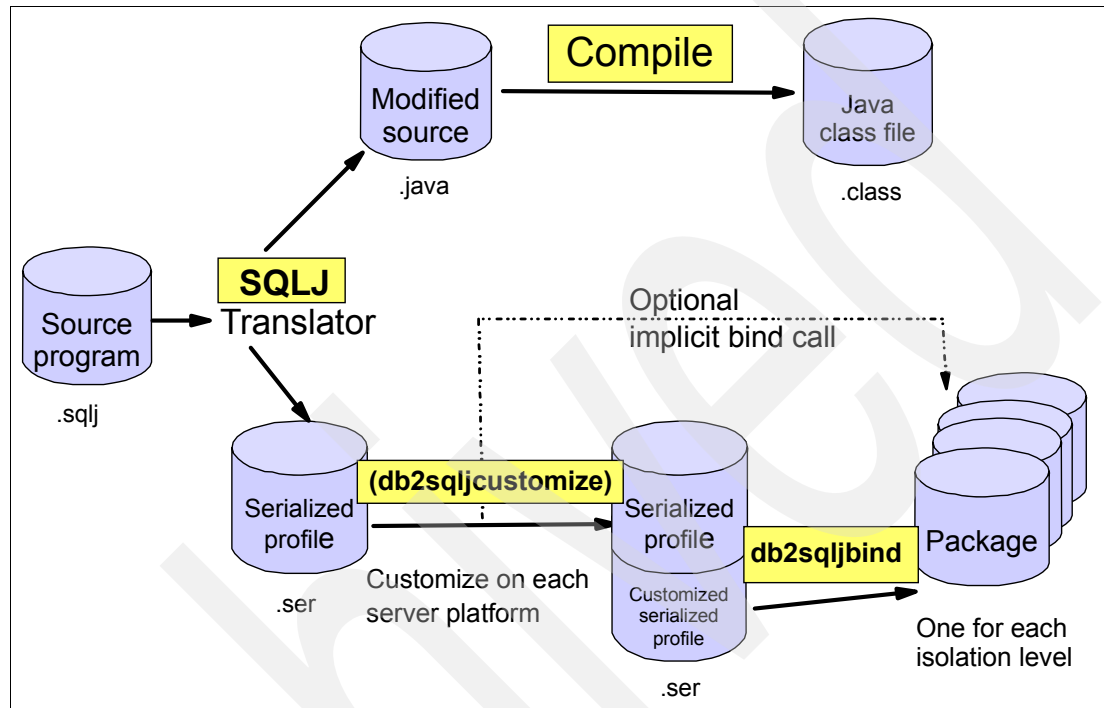


Figure 6-2 Universal Client SQLJ preparation process

For example, when you develop on the workstation, 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 `db2sqljbind` utility was used to connect to the new location and bind the correct program packages.

WSAD Version 5 will provide support for this new application development scheme used by the Universal Driver for SQLJ and JDBC.

6.1.5 Nested stored procedure result sets for JDBC and ODBC applications

Up to V7 it is not possible for a JDBC or a CLI application to have more than one instance of an open result set cursor. Figure 6-3 shows that an attempt to open a cursor that is already open from the previous call of the same procedure fails within DB2 UDB for OS/390 and z/OS. DB2 returns an error because the cursor is already open from the previous call.

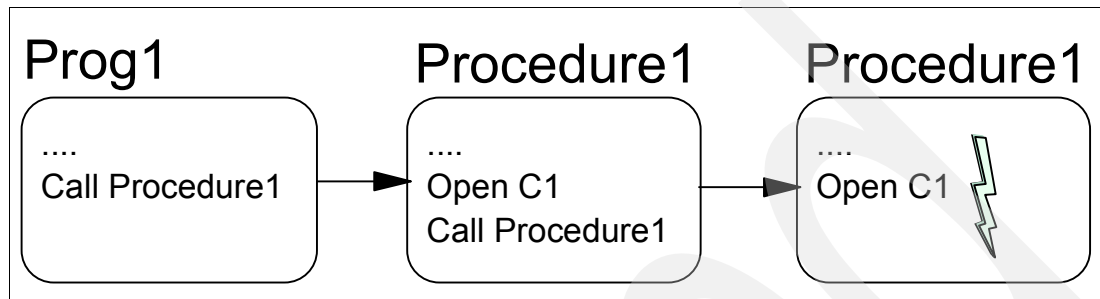


Figure 6-3 Nested stored procedure result sets

With the new Universal Java Client, as well as using ODBC/CLI applications on Linux, UNIX Windows, and z/OS, you can now have multiple instances of the same result set cursor open concurrently. This nesting of instances is possible for up to 16 instances. The DB2 server now provides a unique identifier to the requester for each open cursor or result set. The request can then manage the multiple instances using the unique cursor identifier.

6.1.6 Extended DESCRIBE

Some applications require a great deal of descriptive information to be returned from the server. With this enhanced function, the requester can control the amount and the type of information returned from a prepare, a describe, a query or an execution of a stored procedure.

ODBC/CLI (on LUW) and JDBC applications can request database metadata through a standard set of APIs. In order for ODBC/CLI and JDBC drivers to accurately return this information, they must be sensitive to the underlying database server which the application is requesting the information about.

The extended describe feature is enabled by the DESCSTAT DSNZPARM (as well as the enhancements to the DRDA flow). It provides:

- ▶ Additional descriptive information for a cursor or a result set
- ▶ Information about whether or not a column can be updated
- ▶ Information about whether or not a column is a primary key and/or a preferred candidate key member
- ▶ Information about whether or not a column is an expression or an actual column
- ▶ Information about whether or not a column is a generated column or a real table column
- ▶ The fully qualified view or table name, location.schema.name
- ▶ The fully qualified column name, location.schema.name

For example, a DB2 server can provide extended descriptive information to support the JDBC 2.0 `updateRow` and `deleteRow` methods.

6.1.7 SQLcancel

The SQL cancel() statement allows an ODBC/CLI or JDBC application to cancel an SQL request *long* running on a DB2 server. Note that SQL cancel() is at a more granular level than the DB2 -CANCEL THREAD command. SQLcancel() only rolls back the currently executing SQL statement, not the entire unit of work. In addition, the thread is not destroyed in the process, but is allowed to continue processing.

If the database server is not in an interruptible state, the request completed before DB2 can interrupt, or the request is not interruptible, then DB2 returns a DRDA reply message back to the client confirming the attempt. A new sqlcode -952 is returned if the SQL statement was interrupted (roll back worked). If the SQL statement just reads the data (not write), then we issue -952 even if rollback did not work.

Currently only dynamic SQL statements are interruptible. Stored procedures cannot be interrupted. Transaction level SQL statements like connect, commit and rollback cannot be interrupted. Even bind package cannot be interrupted.

6.1.8 LOB streaming

In prior releases, Universal Clients and Linux/UNIX/Windows servers had to read the entire LOB to determine its length prior to flowing it to DB2.

With this improvement, when using the DRDA flow, DB2 V8 is able to read in the LOB immediately without knowing the exact length up-front.

The main benefit is performance. But there is another advantage. It is not always possible to determine a LOB's attributes (like length and nullness) up-front, for example when a client is acting on behalf of a CLI application that is being supplied with chunks of a LOB value in a piecemeal fashion using the SQLPutData API. Therefore this improvement provides a more flexible mechanism whereby the sender can defer indicating the nullness and/or the length of a LOB value until a chunk is ready to be sent.

The new Universal Driver for SQLJ and JDBC also benefits from this enhancement for Type 4, and Type 2 drivers on both z/OS and distributed platforms.

6.2 Unicode support

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 encoding systems, because no single encoding scheme is adequate for all the letters, punctuations, and technical symbols in common use. These encoding systems also conflict with one another, because two encoding schemes can use the same codepoints for different characters.

In order to get rid of these problems, the support of the Unicode encoding scheme was introduced in DB2 V7. Refer to the redbook *DB2 UDB Server for OS/390 and z/OS Version 7 Presentation Guide*, SG24-6121 to learn more about Unicode and the functions which have been introduced with DB2 V7.

The Unicode support that was introduced in DB2 V7 allows you 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, in V7, does not allow you to join tables with different encoding schemes. You cannot join an EBCDIC table with a Unicode table.

With DB2 V8, considerable new functionality is added to improve the use of the Unicode encoding scheme. Those improvements include:

- ▶ Unicode parsing of SQL and utility statements
- ▶ Unicode catalog
- ▶ Possibility of using multiple CCSIDs per SQL statement
- ▶ ODBC Unicode support

These improvements require that your z/OS Conversion Services and your DB2 CCSID and Unicode definitions are properly set up. See 10.2, “Major changes to installation and migration” on page 243, and Appendix A, “Unicode definitions” on page 263 for details.

6.2.1 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 if a single source of SQL statements includes string constants from multiple locales, you must use one of the following techniques to handle it:
 - Host variables (or parameter markers for dynamic SQL) and DECLARE VARIABLE (or a descriptor)
 - 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 characters. Those characters include “\$@#|~”. For example, some EBCDIC CCSIDs represent the “~” character as a different hex code point than CCSID 37 uses. The DB2 V7 parser always uses CCSID 37, irrespective of the system EBCDIC code page you specified during installation. The result is that you have to use alternate syntax in your statements:

```
SELECT C1 FROM T1 WHERE C1<> 'A';
```

instead of:

```
SELECT C1 FROM T1 WHERE C1 ~='A';
```

As stated previously, the support of Unicode encoding in DB2 V7 allows you to store your data in Unicode. DB2 V8 introduces the Unicode parser. This adds some of the key functionalities which will lead you from a basic Unicode implementation (data only), to full Unicode exploitation.

Unicode parsing in V8 transforms the traditional code page 37 EBCDIC parser into a parser which accepts either syntax regardless of the EBCDIC system CCSID. The Unicode parser converts all SQL statements that are not currently encoded as Unicode UTF-8 to that format before parsing.

6.2.2 Program preparation with new Unicode precompiler

When migrating your DB2 UDB for OS/390 and z/OS from V7 to V8, you must go through three different modes. Independent of the question in which mode you are currently running your DB2 subsystem, any mode of DB2 V8 now uses a Unicode precompiler (or precompiler services). 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.

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 (NFM).

NEWFUN precompiler/coprocessor option

Only NFM allows the use of new SQL functions. Since the precompiler executes outside DB2, it cannot ascertain the current mode of DB2. 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 contains new V8 SQL syntax. A successful precompilation produces a DBRM that is marked as V8-dependent and therefore not compatible with V7. This happens regardless of whether the program contains new syntax or not. As a consequence it can neither be bound on a V7 nor on a V8 subsystem which is not yet running in NFM. The DBRM which 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 EFN mode.

For a new V8 subsystem or for a subsystem which has successfully been converted to NFM, the default changes to *YES*.

Important: 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 in Figure 6-4. As you can see, regardless of whether *NEWFUN* is set to *YES* or *NO*, DB2 invokes the V8 parser. The V8 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.

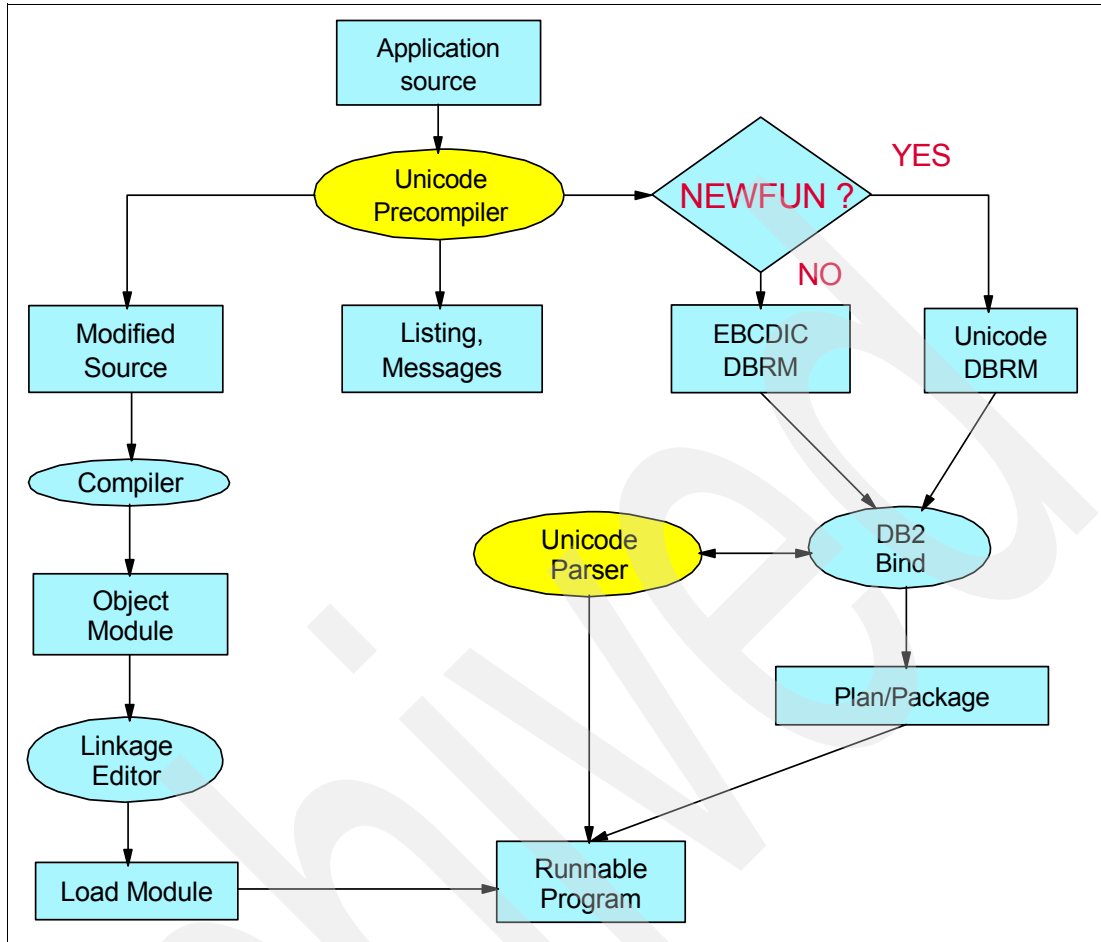


Figure 6-4 Program preparation using the NEWFUN keyword

CCSID(n) precompiler/coprocessor option

In addition to NEWFUN, a second new precompiler option (*CCSID(n)*) has been added. The new option enables the DB2 precompiler to prepare application programs written in any CCSID. Until now, the DB2 precompiler expected the source programs to be coded in the CCSID specified on the panel DSNTIPF during installation. Therefore the portability of application programs developed on a system other than the default encoding scheme used to be a problem.

In Version 8, the precompiler first converts the complete source program from the specified code page to Unicode UTF-8. Then, the 'real' precompiling is done in UTF-8. After that, the modified source is converted back to the specified CCSID, so the final result of the precompilation is a modified source program file in specified CCSID.

Since CCSID(n) is only valid in combination with NEWFUN(YES), the generated DBRM will always be Unicode.

Although this new precompiler option now provides you the possibility to use source programs written in any available CCSID, you have to make sure that the language compiler can handle that codepage correctly. For example, if you specify CCSID(500) for the precompile, the COBOL compilation should also be done using codepage 500.

Hexadecimal string constants

Since it is not easy to enter Unicode or Graphic characters unless you are equipped with a correct keyboard, DB2 V8 now supports two new types of hexadecimal graphic string constants. Hexadecimal Unicode or graphic strings allow you to enter them from any keyboard.

UX'xxxx'

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. Each group of 4 digits represents a single UTF-16 character.

GX'xxxx'

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 DBCS graphic character.

Length of string constant

Up to DB2 UDB for OS/390 and z/OS V7, the maximum length of a string constant was 255 bytes. To accommodate the lengthening of keys and predicates, in V8 this limit is increased to 32704 bytes in most places.

6.2.3 Utility Unicode parser

As stated previously, DB2 V8 extends its Unicode support in many different areas. One effect can be that Unicode object names contain characters that can not be translated back to EBCDIC. In this case, to avoid problems, the utility control statements must be written in Unicode. The DB2 V7 utility parser is not able to interpret utility statements written in Unicode.

DB2 V8 on the other hand, accepts utility statements written completely in Unicode UTF-8 or EBCDIC. A mixture of both encoding schemes is not allowed in utility control statements.

The primary use of this parser probably has to be seen in conjunction with UNIX and Intel based platforms products and features such as CC/390 and ERM applications. For use of remote utility processing, a new stored procedure DSNUTILU has been created. Refer to 7.3, "Unicode" on page 194 for additional information regarding this stored procedure.

6.2.4 Multiple CCSIDs per SQL statement

Initially DB2 did not need to store CCSID values. In an isolated EBCDIC world, it was sufficient to know whether data was single, mixed or double byte. With the introduction of distributed data in DB2 for MVS V2.2, it became important for DB2 to know the specific CCSID value associated with the subtypes, even though only a single EBCDIC encoding scheme was supported. DB2 for MVS V5 added the ability to store character data with an ASCII encoding scheme. However, you were still restricted to one set of EBCDIC or ASCII CCSIDs per DB2 subsystem.

Unicode was introduced to DB2 UDB for OS/390 and z/OS V7 to address the problems of users in many different geographies interacting with one DB2 server, because it is able to represent the characters of many different geographics and languages. Though DB2 UDB for OS/390 and z/OS now supports all three encoding schemes, DB2 V7 still does not allow you to reference table objects defined with different encoding schemes in the same SQL statement.

With DB2 V8, this restriction is removed. Once your DB2 subsystem runs in ENFM, it allows you to access multiple CCSID sets per SQL statement. Apart from the general improvement that this new feature introduces, it is absolutely necessary for you if you currently have applications where you, or a vendor product, join DB2 catalog tables with your own EBCDIC tables. Without this feature, all of these statements would run into errors. Most of the other new features of DB2 V8 are only available once you successfully migrate to NFM. This feature is an exception, because already during ENFM the catalog tables are step by step migrated to Unicode.

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 to it. Example 6-1 shows the importance of knowing the CCSID associated with the string constants in the predicates.

Example 6-1 Multiple CCSID SQL statement — 1

```
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'.

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.

Assuming the application encoding scheme is EBCDIC, both hexadecimal constants use the EBCDIC SBCS CCSID set. This means that in our example, the SQL statement would look like Example 6-2.

Example 6-2 Multiple CCSID SQL statement — 2

```
SELECT ET1.C1, UT1.C1
FROM ET1,UT1
WHERE ET1.C1 = 'ABC'
AND UT1.C1 = 'ää'
```

This is not what was intended with this SELECT statement. The intent here is to compare the contents of the EBCDIC (ET1) and the Unicode (UT1) table with value 'ABC'. In EBCDIC C'ABC' is X'C1C2C3'. To make sure that both tables are compared with string 'ABC', in a multi-CCSID set comparison, you must code the hexadecimal values based on the application encoding scheme. This means that although the comparison of UT1.C1 = X'C1C2C3' in our example is done in Unicode, you must code the statement as shown in Example 6-3.

Example 6-3 Multiple CCSID SQL statement — 3

```
SELECT ET1.C1, UT1.C1
FROM ET1,UT1
WHERE ET1.C1 = X'C1C2C3'
AND UT1.C1 = X'C1C2C3'
```

DB2 then translates the EBCDIC value of the hexadecimal string to Unicode and compares this to column C1 of the Unicode table. That is, DB2 searches for the corresponding code

point in Unicode, converts it to Unicode hexadecimal representation and compares it to the existing hexadecimal values in UT1.c1 afterwards.

Important: In a multiple CCSID scenario X' ' (hexadecimal) constants should always be coded based on the application encoding scheme. You should code the same way, no matter where the X' ' constant is used.

With statements containing one CCSID set, the X' ' constant is interpreted based on the encoding scheme of the SQL statement and not of the application encoding scheme.

There are many rules that influence this decision. We illustrate some using the sample query in Figure 6-5.

```
SELECT a.name, a.creator, b.charcol, 'ABC',
       :hvchar, X'C1C2C3'
FROM SYSIBM.SYSTABLES a,
     ebcdictable b
WHERE a.name = b.name AND
      b.name > 'B' AND
      a.creator = 'SYSADM'
ORDER BY b.name;
```

Result or evaluated:
EBCDIC
Unicode
Application Encoding Scheme

Assuming a *Unicode* catalog, the result will contain multiple CCSIDs and the comparisons and ordering will be dependent on the context

Figure 6-5 SELECT statement using multiple CCSIDs

6.2.5 ODBC Unicode support

DB2 V8 provides you with the ability to update, insert, delete and fetch Unicode data through ODBC application variables. In addition to that, with the ODBC application programming interface, you are now able to use Unicode character strings. For more information, see 6.3.2, “ODBC Unicode support” on page 143.

6.2.6 Unicode and distributed support

DB2 may not connect to its remote partner with EBCDIC CCSID defined in DECP. DB2 will attempt to connect with UTF-8 (mixed) CCSIDs when creating a connection to a server system. If the server rejects the CCSIDs, then it will reconnect with its EBCDIC CCSIDs. If that fails, then the connect will fail. If DB2 gets a connection request from a requester system, it will reply with the Unicode CCSIDs if the requester supports Unicode. Otherwise, DB2 will connect with its EBCDIC CCSIDs.

Also, if we connect to a server system with the mixed Unicode CCSID (even if the local DECP only defines a single-byte EBCDIC CCSID), and the server also is only a single-byte system, then the server may respond with an informational 863 warning SQLCODE in response to the

CONNECT. The installation may not have received this SQLCODE before, but in case they do, they should not be concerned as this is due to the use of Unicode CCSIDs to connect.

6.3 ODBC enhancements

In this section we describes the ODBC/CLI enhancements in DB2 V8:

- ▶ ODBC SQLConnect user and password support
- ▶ ODBC Unicode support
- ▶ Cursor extensions
- ▶ SQLCancel support

6.3.1 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 is an entire DB2 subsystem.)

Actually, the DB2 thread is created when allocating the connection handle. After you created the connection handle, the ODBC application can now 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. They were however 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 userID 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 reasons to 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 any values in your existing applications, 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).

6.3.2 ODBC Unicode support

As we mentioned in 6.2, "Unicode support" on page 136, one of the major enhancements of DB2V8 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.
- ▶ 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:

- ▶ 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. Wide APIs accept Unicode UCS-2 string arguments only, and require that the `CURRENTAPPENSCH` keyword is set to Unicode. The equivalent wide API for the `SQLConnect ()` function call is `SQLConnectW()`. Wide APIs are enabled in V6 with PTF `UQ60475` and in V7 with PTF `UQ60476`.
- ▶ The non-wide functions, for example `SQLSolumnPrivileges`, 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`.

6.3.3 Cursor extensions

With Version 8, DB2 allows an ODBC application to be able to specify:

- ▶ Whether the server should release read locks when a query is closed. This is governed by using the `SQL_CLOSE_BEHAVIOR` (ODBC 2.0) `SQL_ATTR_CLOSE_BEHAVIOR` (ODBC 3.0) attribute.
- ▶ Whether the server should close a query implicitly when there are no more rows for a non-scrollable cursor, regardless of whether the cursor has the `HOLD` attribute. This function enabled by using the `SQL_ATTR_EARLYCLOSE` attribute.

Prior to V8, it was the server that always determined when to release the locks and whether or not a cursor could be closed early.

6.3.4 SQLCancel support

This allows an application to cancel the currently executing SQL statement. For more information, see 6.1.7, “SQLcancel” on page 136.

6.4 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. Up to DB2 UDB for OS/390 and z/OS V7, if you need XML data from traditional relational databases, you must create application packages that convert the DB2 data to the XML format. DB2 V8 provides six new built-in functions to help with this conversion. These functions reduce your application development efforts in generating XML data from relational data with high performance, and enable the development of lightweight applications.

As mentioned before, the 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. 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.

Figure 6-6 gives you an impression of what the result of using these XML publishing functions can look like in your Web applications.

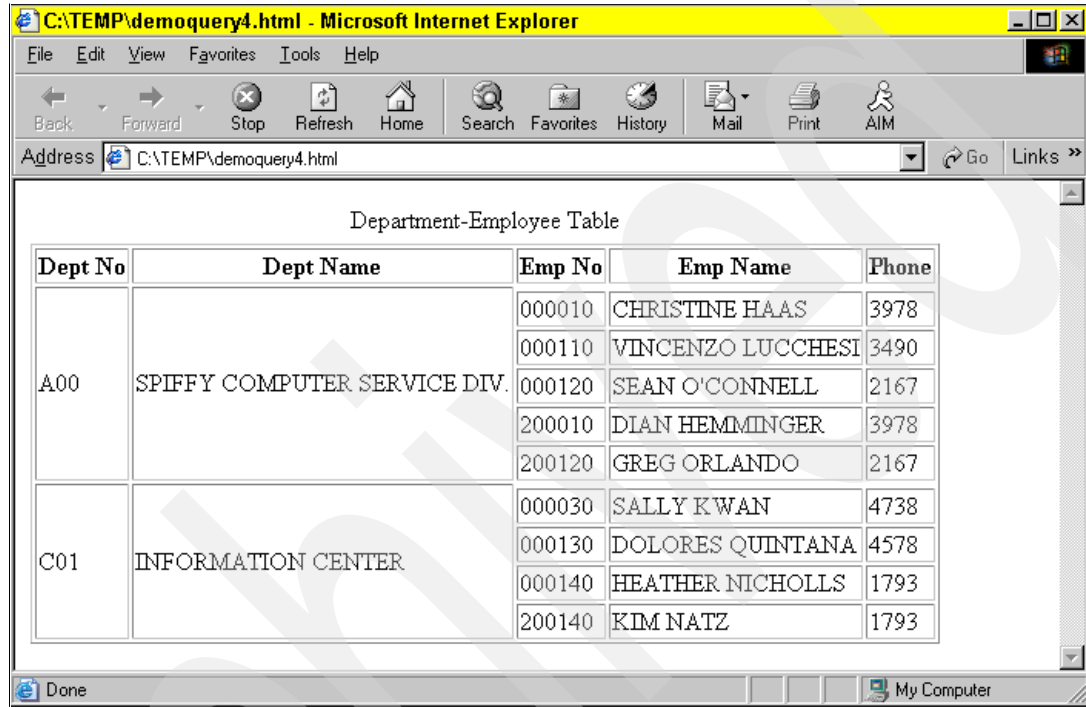


Figure 6-6 Relational data displayed in HTML format

The SQL statement in Example 6-4 was used to generated the XML data directly, using DB2 V8 XML publishing functions and data stored in DB2 tables.

At this point, you should not try to understand every part of this query. We come back to it at the very end of this section.

Example 6-4 Complex query using XML publishing functions

```
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,
```

```

FIRSTNAME || ' ' || 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');

```

Refer to subsequent pages within this chapter for more information about how to use the new XML functions.

Six new built-in functions related to XML publishing can be used with DB2 V8.

- ▶ Cast function
 - XML2CLOB
- ▶ Scalar functions
 - XMLELEMENT
 - XMLATTRIBUTES
 - XMLFOREST
 - XMLCONCAT
- ▶ Aggregate function
 - XMLAGG

These are described here in more detail.

XML data type

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.

Valid values for the XML data type include the following:

- ▶ An element
- ▶ A forest of elements
- ▶ The textual content of an element
- ▶ An empty XML value

There are restrictions for the use of the XML data type:

- ▶ A query result cannot contain this data type
- ▶ Columns of a view cannot contain this data type
- ▶ XML data cannot be used in SORT, that is GROUP BY / ORDER BY and predicates
- ▶ XML data type is not compatible with any other data type
- ▶ The only CAST function that can be used is XML2CLOB

XML2CLOB

The XML2CLOB function returns a CLOB representation of an XML value. The XML2CLOB function provides applications with an interface, so it can access the transient XML data type.

Figure 6-7 shows the syntax diagram of the XML2CLOB cast function.

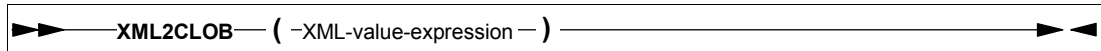


Figure 6-7 XML2CLOB syntax diagram

XMLELEMENT

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. Refer to Figure 6-8 for the syntax diagram.

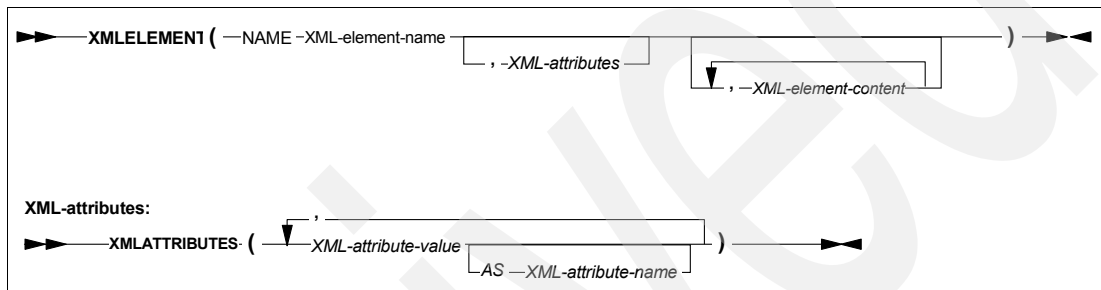


Figure 6-8 XMLEMENT syntax diagram

Let us now take a look at the components of the XMLEMENT function.

- ▶ **NAME:**
The NAME keyword marks the identifier that is supplied to XMLEMENT for the element name.
- ▶ *XML-element-name:*
Specifies an identifier that is used as the XML element name.
- ▶ *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
 - 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.

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 XMLEMENT is an element with empty content.

Refer to the SELECT statement shown in Example 6-5 for a short and simple example of the use of the XML2CLOB and XMLEMENT function.

Example 6-5 XMLELEMENT and XML2CLOB usage

```
SELECT e.id,XML2CLOB(  
           XMLELEMENT (NAME "Emp",e.fname || ' ' ||e.lname)  
           ) AS "Result"  
FROM employee e;
```

The format of the result of this query is shown in Table 6-1:

Table 6-1 Query result — Simple usage of XMLELEMENT and XML2CLOB

ID	Result
1001	<Emp> John Smith</Emp>
1206	<Emp> Mary Martin</Emp>

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 fname and lname.

Example 6-6 shows a more complex SELECT statement using multiple elements.

Example 6-6 Nested elements

```
SELECT e.id, XML2CLOB(  
           XMLELEMENT( NAME "Emp",  
           XMLELEMENT ( NAME "name", e.fname || ' ' ||e.lname ),  
           XMLELEMENT ( NAME "hiredate", e.hire )  
           )  
           ) AS "Result"  
FROM employee e;
```

As you can see, element <Emp> itself contains two nested elements <name> and <hiredate>. Table 6-2 shows the result of this SELECT statement:

Table 6-2 Query result — Nested elements

ID	Result
1001	<Emp> <name>JohnSmith</name> <hiredate>2000-05-24</hiredate></Emp>
1206	<Emp> <name>Mary Martin</name> <hiredate>1996-02-01</hiredate></Emp>

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:

An expression that specifies the value of the attribute. 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, 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. The partially escaped mapping from an SQL identifier to an XML name is used.

If XML-attribute-name is not specified, the expression must be a column name, and the attribute name is created from the column name using the fully escaped mapping from a column name to an XML attribute name. The attribute names for an element must be unique for the XML element to be well-formed. The result of XMLELEMENT cannot be null. Refer to Figure 6-8 for the syntax diagram.

Example 6-7 shows a sample of how to use the XMLATTRIBUTES function:

Example 6-7 Using the XMLATTRIBUTES function

```
SELECT e.id, XML2CLOB(
    XMLELEMENT(NAME "Emp",
        XMLATTRIBUTES(e.id, e.fname || ' ' || e.lname AS "name")
    )
) AS "result"
FROM employee e ;
```

In the foregoing example, the result is an empty XML element named Emp with two attributes. One attribute is the ID column, and the other one, "name", is the concatenation of the fname and lname column of the employee table. The result is shown in Table 6-3.

Table 6-3 Query result — XMLATTRIBUTES

ID	result
1001	<Emp ID='1001' name='John Smith'></Emp>
1206	<Emp ID='1206' name='Mary Martin'></Emp>

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.

The syntax for XMLFOREST function is shown in Figure 6-9.



Figure 6-9 XMLFOREST syntax diagram

► *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. The partially escaped mapping is used 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.

The result type of XMLFOREST is a transient XML data type. The result of the function is the concatenation of the elements, each of which is an XML element from an argument, or the null value if all the arguments are null.

Refer to Example 6-8 for an example of how to use this built-in function:

Example 6-8 Using XMLFOREST

```
SELECT e.id, XML2CLOB(
    XMLELEMENT(NAME "Emp",
        XMLATTRIBUTES( e.fname || ' ' || e.lname AS "name"),
        XMLFOREST(e.hire, e.dept AS "department")
    )
) AS "result"
FROM employee e ;
```

This sample generates an Emp element for each employee. It uses the employee name as its attribute and two subelements which are generated from columns HIRE and DEPT by using XMLFOREST as its content. The element names for the two subelements are 'HIRE' and 'department'. The result is shown in Table 6-4.

A query equivalent to the one shown in Example 6-8 is shown below in Example 6-9.

Example 6-9 Alternative query not using XMLFOREST

```
SELECT e.id, XML2CLOB(
    XMLELEMENT(NAME "Emp",
        XMLATTRIBUTES( e.fname || ' ' || e.lname AS "name"),
        XMLELEMENT(NAME "HIRE",e.hire),
        XMLELEMENT(NAME "department",e.dept)
    )
) AS "result"
FROM employee e ;
```

Instead of using the XMLFOREST function, we used two separate XMLELEMENT functions.

Table 6-4 Query result — XMLFOREST

ID	result
1001	<Emp name="John Smith"> <HIRE>2000-05-24</HIRE> <department>Accounting</department> </Emp>
1206	<Emp name="Mary Martin"> <HIRE>1996-02-01</HIRE> <department>Shipping</department> </Emp>

Attention: As you can see in Table 6-4, the generated element names are folded to uppercase. If you want them to be lowercase or mixed, you must use quotes (“department”)

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 Figure 6-10.

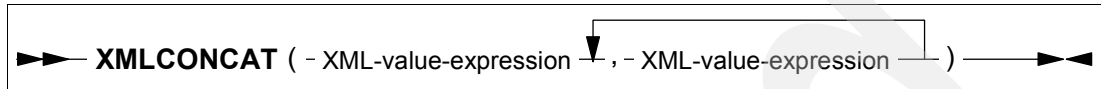


Figure 6-10 XMLCONCAT syntax diagram

Where:

► *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 6-10 shows how to use XMLCONCAT:

Example 6-10 Using the XMLCONCAT function

```
SELECT XML2CLOB(
    XMLCONCAT (
        XMLELEMENT (NAME "first",e.fname),
        XMLELEMENT (NAME "last",e.lname)
    )
)as "Result"
FROM employee e ;
```

Table 6-5 shows the result of the query above.

Table 6-5 Query result — XMLCONCAT

ID	Result
1001	<first>John</first><last>Smith</last>
1206	<first>Mary</first><last>Martin</last>

XMLAGG

The XMLAGG function returns a concatenation of XML elements from a collection of XML elements. A syntax diagram is shown in Figure 6-11.

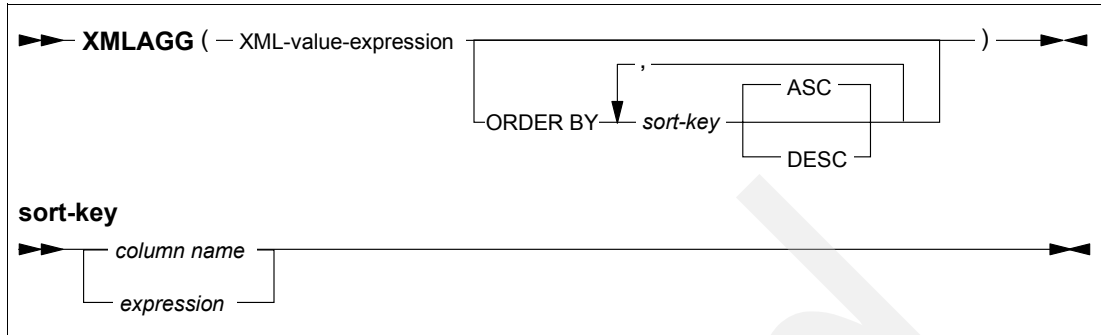


Figure 6-11 XMLAGG syntax diagram

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.

Let us look at Example 6-11 to become more familiar with the way you can use this function:

Example 6-11 Using the XMLAGG function

```

SELECT XML2CLOB(
    XMLELEMENT(NAME "Department",
        XMLATTRIBUTES (e.dept AS "name"),
        XMLAGG (
            XMLELEMENT (NAME "emp",e.lname)
            ORDER BY e.lname
        )
    )
)AS "dept_list"
FROM employee e
GROUP BY dept ;
  
```


The layout of the result of the query may look like Table 6-6. The result column is a CLOB. The contents are formatted for your convenience.

Table 6-6 Query result — XMLAGG

dept_list
<pre><Department name="Accounting"> <emp>Smith</emp> <emp>Yates</emp> </Department> <Department name="Shipping"> <emp>Martin</emp> </Department></pre>

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.

Now that you have learned about all the built-in functions, let us get back to the complex query which we showed at the very beginning of this section (see Example 6-4 on page 145). Apart from the XML publishing functions, the sample also contains many HTML tags, which are needed to format the output the way it is. The *as "border"* option or the *as "rowspan"* option are examples for these HTML tags.

These XML publishing functions complement the functionality delivered by the XML Extender product. For more information, see *DB2 UDB for OS/390 and z/OS Version 7 XML Extender Administration and Programming, SC26-9949*.

6.5 CURRENT PACKAGE PATH special register

DB2 V7 provides the ability to specify a list of collection IDs in which DB2 will look for packages at execution time. This is done via the BIND PLAN *PKLIST* option. DB2 will search for a package in each of the collections in the PKLIST. However, not all execution environments have this PKLIST capability. The new CURRENT PACKAGE PATH may help the following cases:

- ▶ DB2 UDB for z/OS Application Requestors connected via DRDA, requesting the execution of a package on a remote DB2 UDB for z/OS.

When OS/390 and z/OS requestors are using DRDA to access remote OS/390 and z/OS servers, today these applications have the PKLIST bind option that allows them to search package collections at runtime. The PKLIST option is a "requester" function, that is the remote server does not know the value set for PKLIST, because it is part of the PLAN and only packages are used at the remote servers. When you specify PKLIST(*.PROD.*, *.TEST.*, *.MIGR.*), the DB2 requester uses the PKLIST values to resolve package references when CURRENT PACKAGESET is blank. The algorithm is pretty expensive. In the case of the package list specified above:

- A message is sent to the current server first requesting a package (for example PROGA) with the first collection from the PKLIST. In our example, the request would be for PROD.PROGA.
- If this results in SQLCODE -805 (package not found), another message is sent requesting for the package within the next collection in the PKLIST. In our example,

TEST.PROGA would be requested, and so on, until the requester receives a result indicating that a requested package is found.

Although DB2 caches the fully qualified package names at the requester to avoid resending all these network messages on subsequent SQL statements issued from this same package, this algorithm still involves much network traffic when there is a long list of collections in the PKLIST. Considerable network traffic is generated if the requester application has many separately compiled modules, because the resolution algorithm has to be performed for each package.

The new *CURRENT PACKAGE PATH* special register reduces network traffic and improves CPU and elapsed time for applications that use DRDA from a DB2 UDB for z/OS Application Requester. The network is only crossed once to resolve the package collection ID at the server, instead of one message per collection ID to perform the resolution at the Application Server.

- ▶ Stored procedures and user-defined functions

In DB2 UDB for OS/390 and z/OS, you cannot identify package resolution schemes for stored procedures and UDFs independent from the rules established by the caller's plan. Although the SET CURRENT PACKAGESET statement, or the COLLID option for a routine can be used within a procedure or function to change the package resolution rule from the invokers, these can only specify one collection. The new *CURRENT PACKAGE PATH* special register allows a stored procedure or user-defined function to be implemented without concern for the invoker's runtime environment, and also allows multiple collections to be specified.

- ▶ Programs that run from the workstation, for example Windows, to access DB2 UDB for z/OS do not use a plan. Currently, those applications must issue a SET CURRENT PACKAGESET statement each time they want to use a package from a different collection (package set).

To give you more flexibility regarding the specification of packages you want to use during program execution, you can now use the new *CURRENT PACKAGE PATH* special register. You can issue a SET CURRENT PACKAGE PATH statement at the beginning of your application. This way you can obtain similar functionality than the PLKLIST bind option on DB2 UDB for z/OS. This new function is especially helpful for those of you who are using SQLJ and JDBC applications, since there is a strong tendency to use a single plan for all applications.

Using the new SET CURRENT PACKAGE PATH, you can implement versioning in a similar way for non-DB2 UDB for z/OS ARs than you have today for DB2 UDB for z/OS ARs, or local DB2 UDB for z/OS applications. This can for example be a very useful technique in helping to control changes made in production environments. You can for instance use versioning to keep a backup version of a package in a different collection. A previous version of a package is used in same way as a backup application load module or executable during a fallback.

6.6 DDF communication database enhancements

In this section we examine the following enhancements:

- ▶ Requester database ALIAS
- ▶ Server location alias
- ▶ Member routing in a TCP/IP network

6.6.1 Requester database ALIAS

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*. If the requester is a DB2 UDB for z/OS, 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 V7, there is always a one-to-one mapping between location name and database name, as there is a unique index on the LOCATION column. As you can see in Figure 6-12, prior to DB2 V8, there is no way to access multiple DB2 UDB for LUW databases that have the same name (even when they reside on different machines).

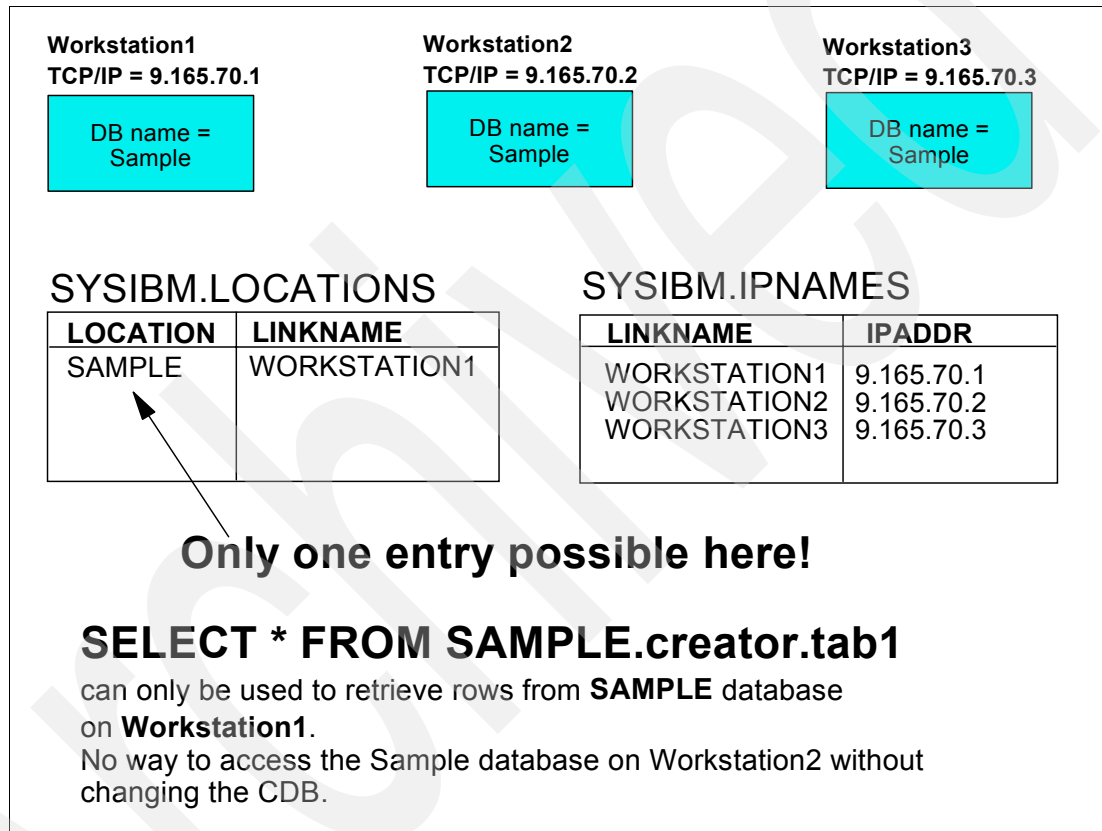


Figure 6-12 Access LUW database without DBALIAS

This restriction is removed in DB2 V8. A new column DBALIAS is added to the SYSIBM.LOCATIONS table. As reported in Figure 6-13, you continue to specify the LOCATION name as the first qualifier of your three part table name in your SELECT statement [1]. The mapped LINKNAME links you to the corresponding entry in SYSIBM.IPNAMES [2], 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 LUW 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.

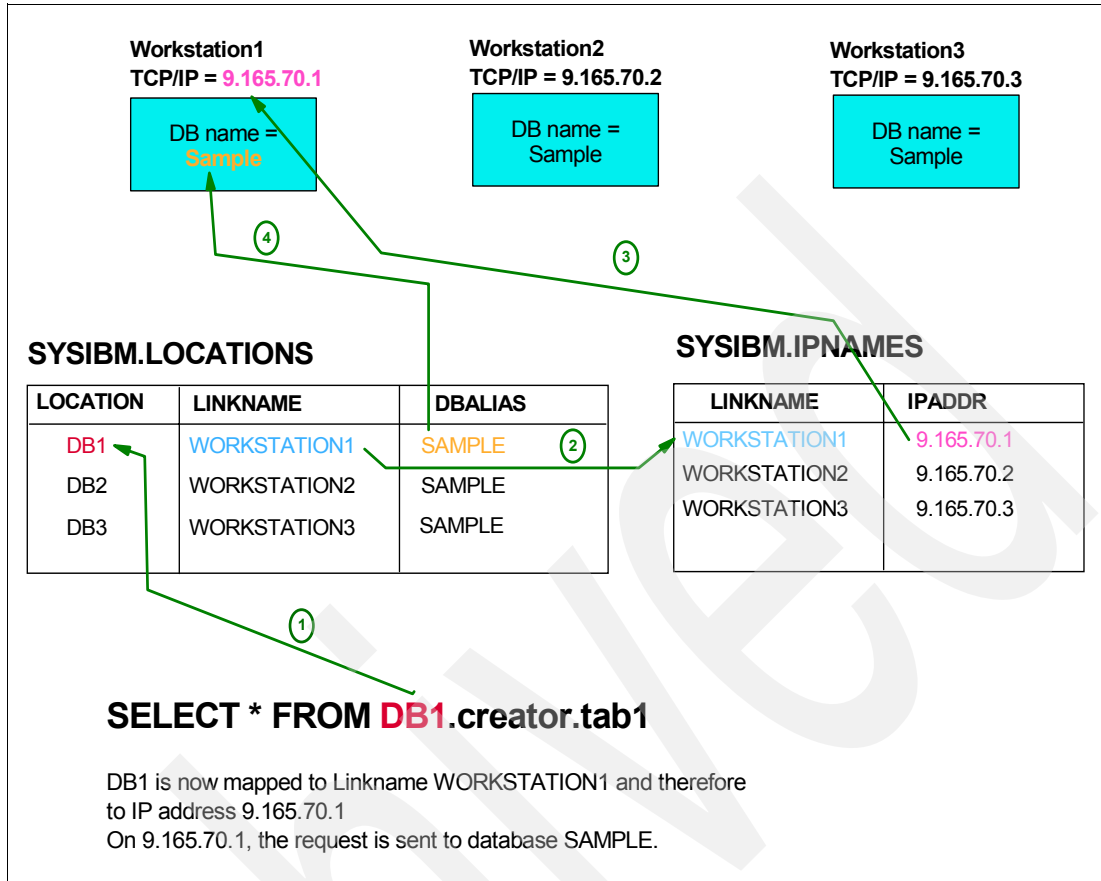


Figure 6-13 Access LUW database with DBALIAS

6.6.2 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 an entire DB2 data sharing group. When two or more DB2 subsystems are consolidated to 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.

To ease this type of migration, DB2 V8 allows you to define up to eight alias names in addition to the location name for a DB2 data sharing group. A *location alias* is an alternative name that a requester can use to access a DB2 subsystem. You can define those location alias names with the CHANGE LOG INVENTORY utility (DSNJU003). As shown in Figure 6-14, 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 additional location alias names to your BSDS data sets on each member of the data sharing group.

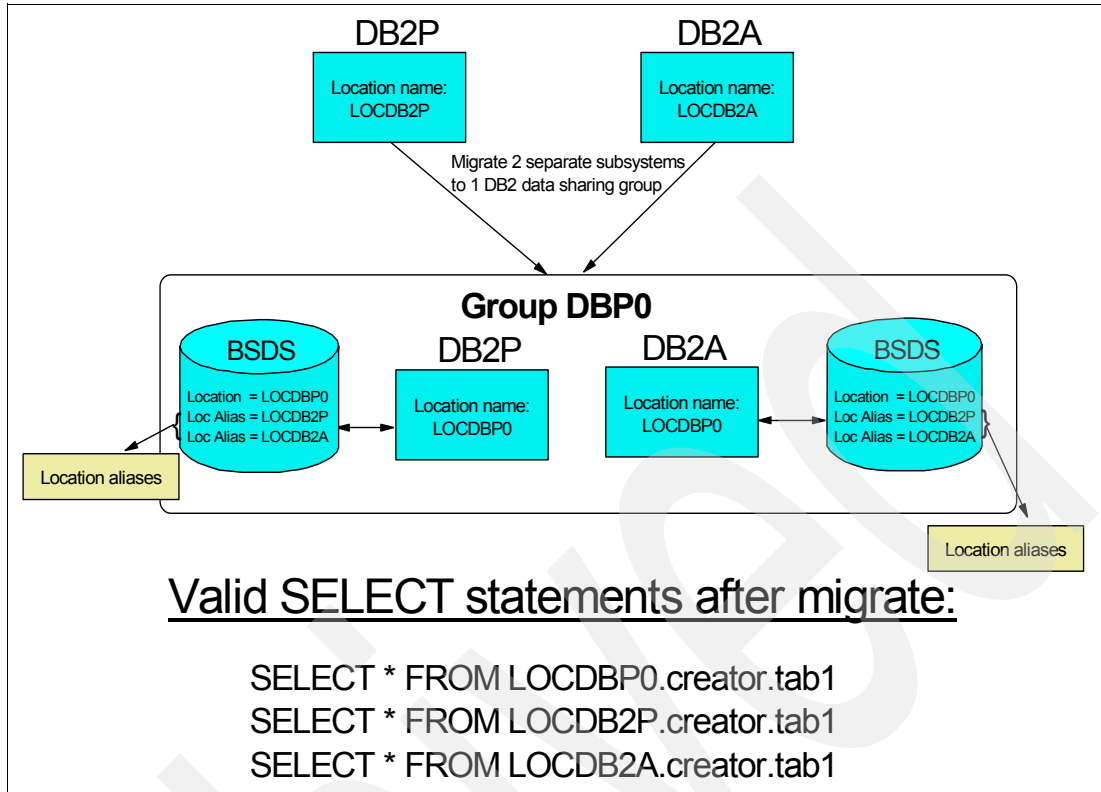


Figure 6-14 Location alias name

The syntax for the CHANGE LOG INVENTORY utility is:

DDF ALIAS = *aliasname*

The distributed data facility communication record in the BSDS data sets has been changed to store the location alias names you have specified for your subsystem. Figure 6-15 shows the output of the PRINT LOG MAP utility (DSNJU004). you can see the location alias name DB70GRP we added for our subsystem.

```

**** DISTRIBUTED DATA FACILITY ****
      COMMUNICATION RECORD
      14:26:17 SEPTEMBER 30, 2003
LOCATION=DB70 ALIAS=DB70GRP
LUNAME=LU1 PASSWORD=(NULL) GENERICLU=(NULL) PORT=33744 RPORT=33745

```

Figure 6-15 DDF communication record

6.6.3 Member routing in a TCP/IP network

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. Sometimes the default TCP/IP workload balancing does not meet your needs and you might therefore want to be able to route requests from certain DB2 UDB for z/OS 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, described in 6.6.2, “Server location alias” on page 156, with the use of a new table in the CDB, namely SYSIBM.IPLIST. Refer to Figure 6-16 to understand the process.

A location alias, LOCDBP1, has been defined for data sharing members DBP1 and DBP2. Note however that this alias does not exist in the BSDS for DBP3. In SYSIBM.LOCATIONS, you can find two entries. One for the “normal” location name of the data sharing group (LOCDBP0), and one for the location alias (LOCDBP1). If you now enter rows into the new communication database table SYSIBM.IPLIST, any requests for location name LOCDBP1 are routed to either of the mapped TCP/IP addresses in SYSIBM.IPLIST (for instance 9.165.70.1 or 9.165.70.2 in Figure 6-16) [1].

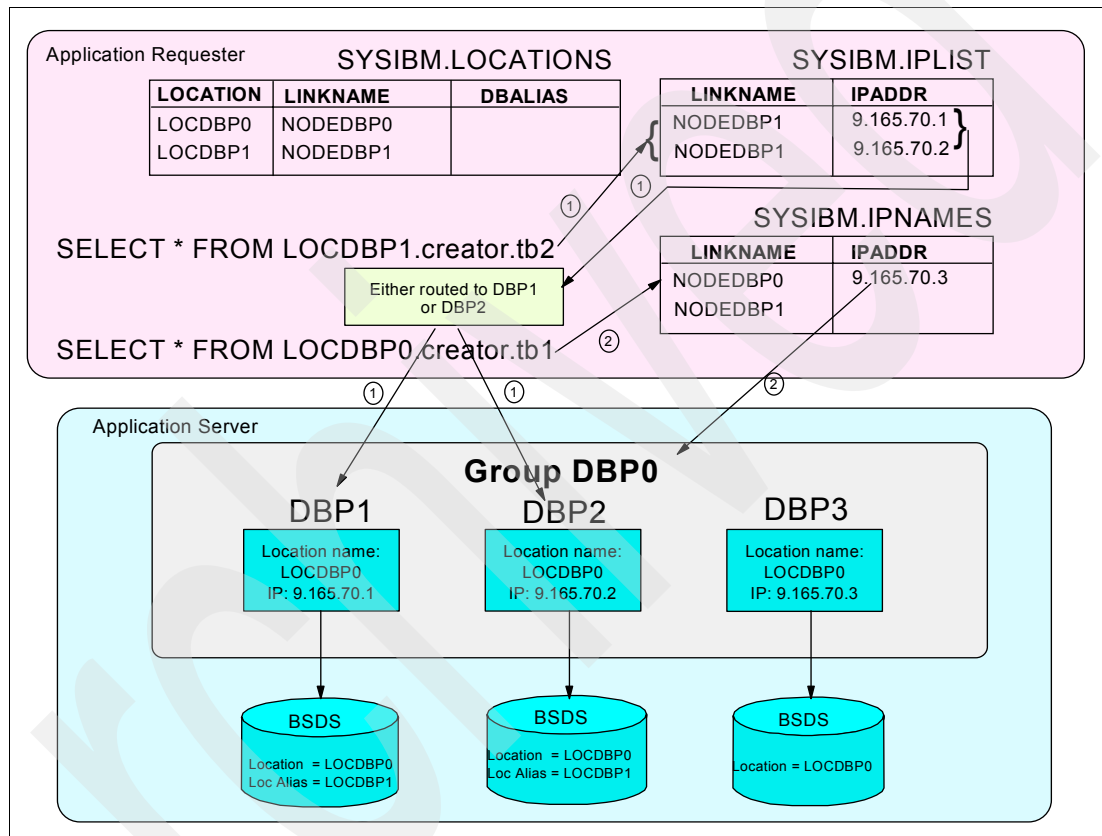


Figure 6-16 Member routing in a TCP/IP network

If a request comes in for location name LOCDBP0 [2], the entry in SYSIBM.SYSIPNAMES routes it to all available members in group DBP0.

When using WLM domain name server to perform workload balancing, the SYSIBM.IPLIST must contain the member specific domain name for each DB2 subsystem that requests are to be routed. When using dynamic VIPA to perform workload balancing, the IPLIST must contain the member specific dynamic VIPA for each DB2 subsystem whose requests are to be routed. Domain names and DVIPA cannot be defined in SYSIBM.IPLIST for the same DB2 data sharing group.

DB2 first checks the IPLIST table, and then the IPNAMES table. For member specific routing to NODEDBP1, if there is no entry in IPNAMES, or the entry for NODEDBP1 in IPNAMES is specified with an IPADDRESS, then SQLCODE -904 is issued.

6.7 Enhancements for stored procedures and UDFs

DB2 V8 offers some improvements for stored procedures and user-defined functions (UDF). Read the next few topics to learn more about those changes and how they might affect your daily work.

6.7.1 Maximum failures

With DB2 UDB for OS/390 and z/OS Version 7 you can specify a value for the maximum abend count on installation panel DSNTIPX (DSNZPARM parameter STORMXAB). With this value you can specify the number of times a stored procedure or UDF is allowed to terminate abnormally before it is stopped. This parameter is subsystem wide, which means that you have to treat all stored procedures and UDF equally.

With DB2 V8, you can specify a value for each stored procedure or UDF. This means that you can now specify an appropriate value at the procedure level (instead of the subsystem level), for example, depending upon whether it is already an established application, or a new procedure being tested.

How to invoke this new functionality

The new functionality (syntax shown in Figure 6-17) is valid for external scalar and table functions, as well as external and SQL stored procedures.

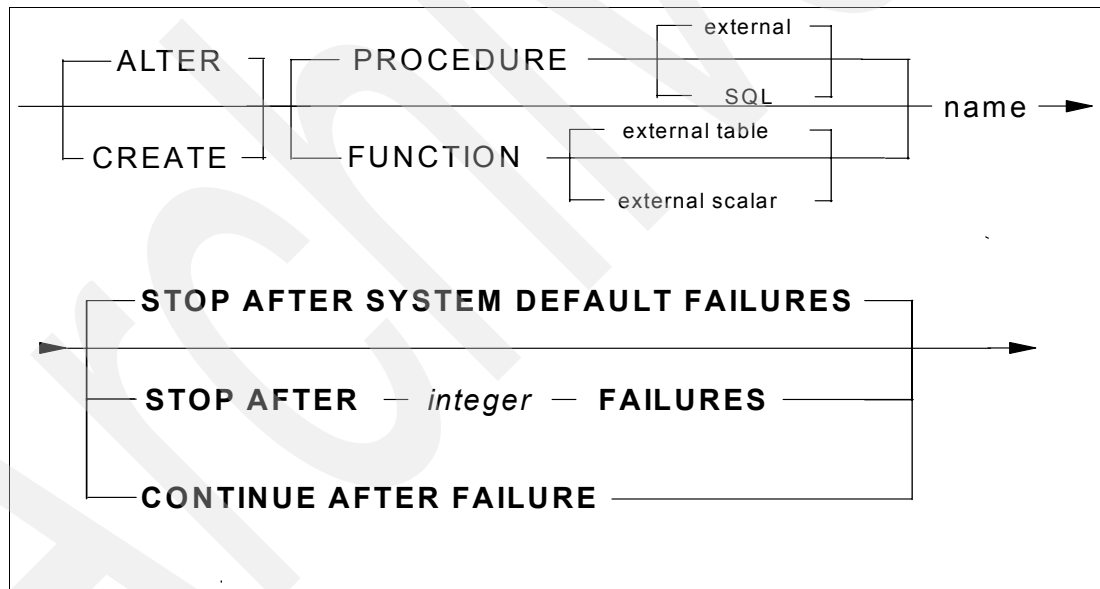


Figure 6-17 Stored procedure and UDF enhanced failure handling syntax

These parameters cannot be used for sourced functions and SQL scalar functions. These functions does not really have a “program” associated with them that runs in another address space. Therefore the options have no relevance for them.

We now describe the new options in more detail.

► *STOP AFTER nn FAILURES:*

This option puts the routine in a stopped state after nn failures. nn Can be any value between 1 and 32767. That means it can either be higher or lower than the value specified for the DB2 subsystem (STORMXAB).

► *STOP AFTER SYSTEM DEFAULT FAILURES:*

This option puts the routine in a stopped state when it reaches the number of abnormal terminations specified in the ZPARM value STORMXAB. This option is the default. Therefore, if you accept the default for STORMXAB (zero), and if you do not specify anything specific in your CREATE and ALTER PROCEDURE SQL statement, your stored procedure or UDF is stopped after every abnormal termination.

► *CONTINUE AFTER FAILURE:*

If you decide to use this option, your stored procedures or UDFs are never put in stopped state, unless you explicitly use the -STOP PROCEDURE command.

To store this information, a new column *MAX_FAILURE* has been added to the *SYSIBM.SYSROUTINES* catalog table.

Attention: To activate these new settings, you must stop and start the corresponding stored procedure or UDF.

Monitoring the current status of your SP or UDF

If you either specify STOP AFTER nn FAILURES or STOP AFTER SYSTEM DEFAULT FAILURES, it might be interesting to monitor the number of abnormal terminations that have already occurred. To satisfy your needs, the output of the DISPLAY PROCEDURE and DISPLAY FUNCTION SPECIFIC command has been enhanced to show you the failure count for the procedure or function.

6.7.2 Exploit WLM server task thread management

The stored procedure management has been changed to exploit z/OS workload manager functions that allow z/OS's System Resource Manager and Workload Manager to determine the appropriate resource utilization, and recommend changes in the number of tasks operating inside a single WLM managed stored procedure address space. Up to DB2 V7, specifying NUMTCB meant that a new WLM address space is started, if the number of TCBs running in one WLM managed stored procedure address space exceeded the value of NUMTCB. You can find more information regarding WLM application environments, in conjunction with stored procedures in redbook *Cross-Platform DB2 Stored Procedures: Building and Debugging*, SG24-5485.

This behavior is changed in DB2 V8. With V8, the value specified in NUMTCB is sent to WLM as a maximum task limit. WLM determines the actual number of TCBs that will run inside a WLM managed stored procedure address space. We recommend that you use higher numbers in NUMTCB than you use in V7. This way WLM has more flexibility to decide how many tasks should run in one address space, and when to start an additional started task. NUMTCB can easily be around 60 or 30 if using Java stored procedures.

This recommendation only applies for stored procedures which are able to share the resources in one address space. There are still some stored procedures, for example DSNUTILS, which requires a value of 1 in NUMTCB.

6.7.3 Enhancements to SQL stored procedure language

The SQL stored procedure language and SQL stored procedures were introduced in DB2 UDB for OS/390 V5. The SQL procedure language is a procedural language that is designed only for writing stored procedures. It is available across the entire DB2 family. The SQL procedure language is based on SQL extensions, as defined by the SQL/PSM (Persistent

Stored Modules) standard. The current implementation of the SQL procedures language on DB2 UDB for OS/390 and z/OS does not cover everything which is described in this standard. In addition to that, there are also some differences regarding the available language elements within the entire DB2 family. In order to achieve a better compatibility between the different DB2 platforms, some new language elements have been added to DB2 V8.

RETURN

Currently (up to V7) there are only two methods available for returning status information from an SQL procedure:

- ▶ Leave a condition unhandled

Each condition that is not handled in a SQL procedure is returned to the caller in the SQLCA.

Note: This behavior came in with APAR PQ56323. Prior to this change, SQL procedures for DB2 UDB for OS/390 did not return unhandled conditions to the calling application. This behavior is consistent with the rest of the DB2 family, and with the SQL standard.

- ▶ Define an extra parameter for status information

In this case, you would have to include an additional parameter (OUT or INOUT) for the status information as part of the parameter list.

Because both methods are not very convenient, support for the RETURN statement was added to SQL procedures. The RETURN statement can be used to return an integer value to the invoking application. Refer to Example 6-12 to see how this can be implemented:

Example 6-12 Using the RETURN statement

```
BEGIN
...
IF <failing condition> THEN
GOTO FAIL;
END IF;
...
SUCCESS: RETURN 0;
FAIL: RETURN -200;
END
```

SIGNAL

As described above, the RETURN statement allows you to pass back status information to the calling program of an SQL stored procedure. However, without the additional functionality of the SIGNAL SQL statement, the stored procedure cannot dictate what information is to be returned to the caller. To remove this restriction, the SIGNAL SQL statement can now be used for SQL stored procedures. The SIGNAL statement was already available for triggered actions of a trigger in DB2 V7.

The SIGNAL statement can be used to:

- ▶ Allow a procedure to set the SQLSTATE to a specific value
 - SQLCODE is set to +438 for an SQLSTATE class of '01' or '02'
 - SQLCODE is set to -438 otherwise
- ▶ Specify an optional MESSAGE_TEXT (1K max. size)
 - The first 70 bytes of the message text is stored in the SQLERRMC field of the SQLCA

- The full message text can be obtained from the MESSAGE_TEXT and MESSAGE_LENGTH fields of GET DIAGNOSTICS

The syntax for the SIGNAL SQL statement has changed from V7 to V8, as shown in Figure 6-18.

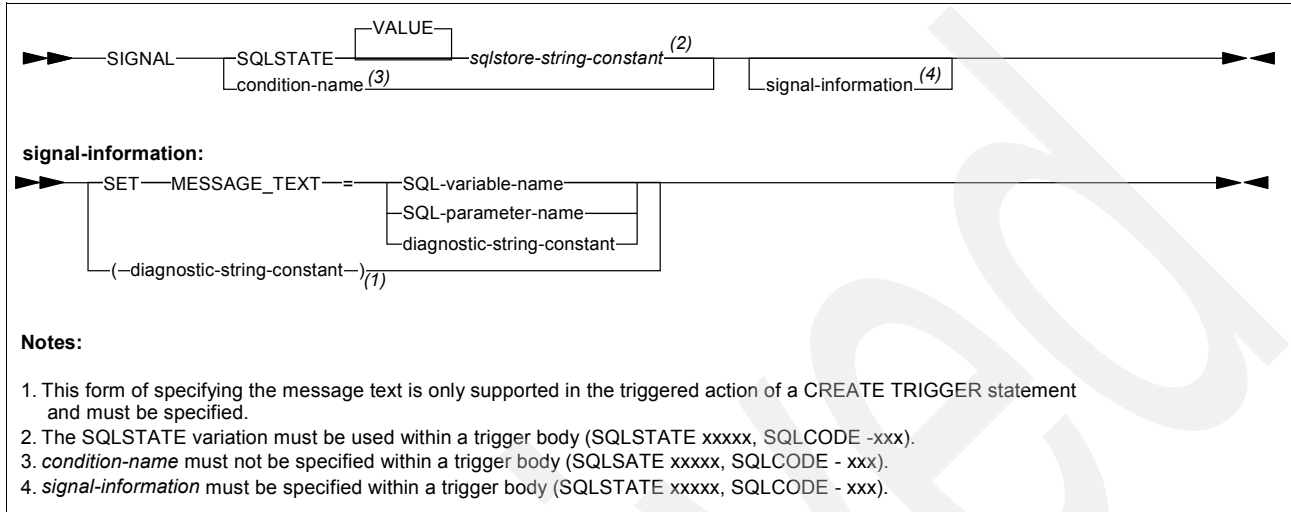


Figure 6-18 SIGNAL statement syntax diagram

You can use the SIGNAL keyword within a condition handler or anywhere else in the stored procedure body. Refer to Figure 6-19 to find out the differences in behavior, depending upon where the SIGNAL statement is specified.

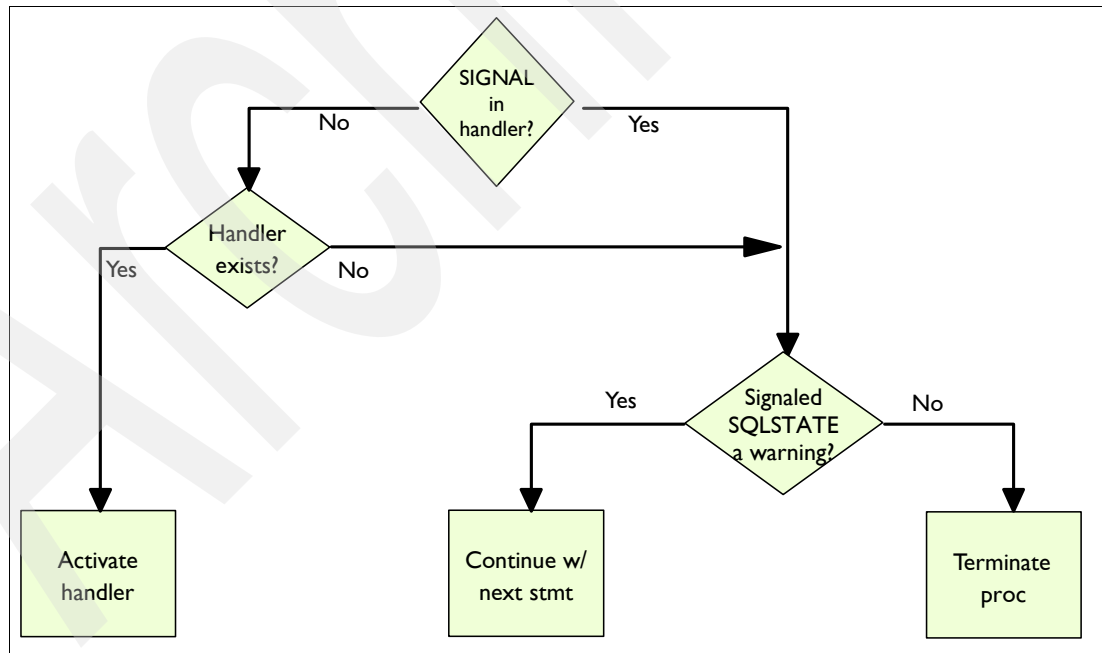


Figure 6-19 SIGNAL used in condition handler?

- 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:
 - Continue if warning is signaled.
 - Exit if exception is signaled.
- ▶ If SIGNAL is part of a handler, then:
 - Continue if a warning is signaled.
 - Exit if exception is signaled.

The CREATE PROCEDURE statements Example 6-13 shows the use of a SIGNAL statement which is part of a handler.

Example 6-13 Using the SIGNAL statement

```

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

```

If the stored procedure, during execution, encounters an SQLSTATE '23503' (which indicates that the insert or update value of a foreign key is invalid), the EXIT handler is invoked. As part of the EXIT handler, the SIGNAL statement signals SQLSTATE '75002' with message text 'Customer number is not known' instead. SQLSTATE '75002' is no predefined SQLSTATE used by DB2.

RESIGNAL

In contrast to the SIGNAL statement, the RESIGNAL statement is only used within a handler to return an error or warning condition. It causes an error or warning to be returned with the specified SQLSTATE, along with optional message text. As for SIGNAL, the RESIGNAL statement sets SQLCODE to:

- ▶ +438 if SQLSTATE class is '01' or '02'
- ▶ - 438 otherwise

The syntax diagram for the RESIGNAL statement is provided for your reference in Figure 6-20.

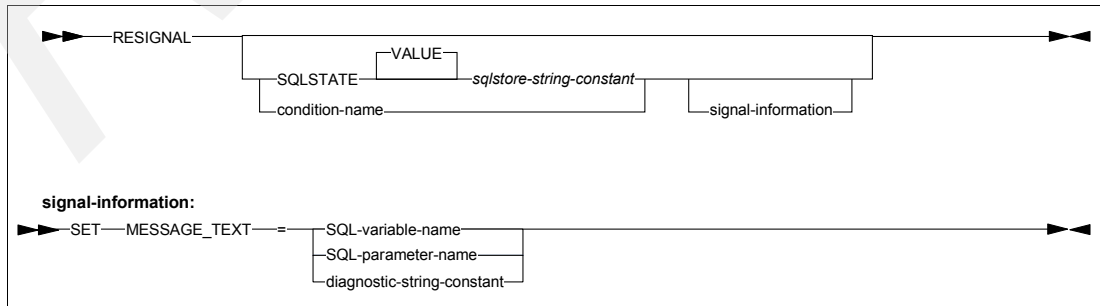


Figure 6-20 RESIGNAL statement syntax diagram

Example 6-14 shows how the RESIGNAL statement can be used.

Example 6-14 Using the RESIGNAL statement

```
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
```

Referring to the example above, you can see that if the denominator equals to 0, the SIGNAL statement is used to signal a condition name, that is overflow in this example. Since we defined an EXIT handler for this overflow condition, the handler is fired now, that is the RESIGNAL statement assigns '22375' to SQLSTATE. SQLSTATE '22375' is not predefined and used by DB2. As you can see in Figure 6-20, you could have also added a message text as additional information.

GET DIAGNOSTICS

The GET DIAGNOSTICS statement has also been enhanced. you can now retrieve much more information than just the ROW_COUNT. The GET DIAGNOSTICS statement can not also be used by any other programming language, not just by an SQL procedure as is the case in DB2 V7. More information can be found in 5.6, "Get diagnostics" on page 90.

6.7.4 COMPJAVA stored procedures no longer supported

Visual Age for JAVA does no longer support compiled Java link library files. For this reason, DB2 V8 no longer supports stored procedures written in compiled JAVA. Therefore, you can not use the keyword LANGUAGE COMPJAVA any more. Take the following steps to migrate a LANGUAGE COMPJAVA stored procedure to LANGUAGE JAVA:

1. Use ALTER PROCEDURE to change the LANGUAGE and the WLM ENVIRONMENT. The EXTERNAL NAME clause must also be specified, even if it is not changed. DB2 needs to verify it.
2. Make sure the WLM environment has been set up and the required JVM installed.
3. Make sure the .class file identified in the EXTERNAL NAME clause of the ALTER PROCEDURE is either contained in a JAR that has been installed to DB2 with an invocation of the INSTALL_JAR stored procedure, or that the .class file is in a directory name in the CLASSPATH ENVAR of the data set named on the JAVAENV DD card of the WLM stored procedures address space JCL.

6.7.5 DB2 established stored procedures

In DB2 V7 you have the choice between so called, DB2 managed, and WLM managed stored procedures. All DB2 managed stored procedures are executed in the same address space called *ssidSPAS*. The way to create a DB2 managed stored procedure is to specify NO WLM ENVIRONMENT in your CREATE or ALTER PROCEDURE statement.

WLM managed stored procedures provide many advantages, and because running WLM in goal mode is required in z/OS 1.3, and z/OS 1.3 is a prerequisite for DB2 V8, there are no reasons to continue using DB2 managed stored procedures. Therefore, DB2 V8 no longer allows the creation of DB2 managed stored procedures. This means that the keyword NO WLM ENVIRONMENT is no longer valid in the CREATE or ALTER PROCEDURE SQL statement.

For compatibility reasons, you can continue to use your existing DB2 managed stored procedures, but you should consider to convert them to WLM managed stored procedures, to take advantage of the enhanced functionality described above.

6.8 Miscellaneous enhancements

This section described a number of miscellaneous enhancements, but still mean a great deal to a large numbers of customers. The following topics are discussed:

- ▶ RRSAF compatibility for CAF applications
- ▶ Roll up accounting data for DDF and RRSAF threads
- ▶ Improved query and result set processing
- ▶ Time out for SNA allocate conversation requests
- ▶ Data stream encryption
- ▶ DISPLAY LOCATION command

6.8.1 RRSAF compatibility for CAF applications

The DB2 Call Attach Facility (CAF) is used by many applications today. Recoverable Resources Manager Services Attachment Facility (RRSAF) provides roughly the same functionality as CAF. However, RRSAF has a major advantage over CAF, and that is its ability to support two phase commit processing. Therefore, if your current CAF applications need two phase commit support, you have to migrate them to RRSAF.

To use RRSAF, you must link your programs with the RRSAF language interface module, DSNRLI instead of the CAF language interface DSNALI. For information on loading or linking this module, you can refer to *DB2 UDB for z/OS and OS/390 Application Programming and SQL Guide, SC26-9933-02*.

Explicit DB2 connections are established in almost the same way by CAF and RRSAF. For CAF applications, you must issue a CONNECT and OPEN. For RRSAF, an IDENTIFY and CREATE THREAD is necessary. You can then run your SQL statements. To disconnect, CAF uses a CLOSE and DISCONNECT, whereas RRSAF invokes a TERMINATE THREAD and TERMINATE IDENTIFY.

In contrast to this, CAF applications also have the possibility to connect implicitly to a DB2 subsystem; that is just by issuing SQL statements or IFI calls (without first doing a CONNECT and an OPEN). An implicit CAF connection derives the DB2 subsystem name it will connect to from the DSNHDECP module and will allocate a plan that has the same name as the program (DBRM) that issues the first SQL call.

With DB2 V7, implicit connections are not supported when you use RRSAF.

However, with DB2 V8, you can also connect to DB2 via RRSAF but just issuing SQL statements or IFI calls. When you request an implicit connection, RRSAF issues an IDENTIFY and CREATE THREAD under the covers, using the following values:

- ▶ **Subsystem name:** The default DB2 subsystem name specified in module DSNHDECP is used. (The usual search order is used to find the DSNHDECP module; that is STEPLIB, JOBLIB, linklist. In a data sharing group, the default subsystem name is the group attachment name.
- ▶ **Plan name:** The member name of the database request module (DBRM) that DB2 produced when you precompiled the source program that contains 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.
- ▶ **Authorization ID:** The authorization ID is set from the seven byte user ID associated with the address space, unless an authorized function has built an ACEE (an Accessor Environment Element is a RACF control block) for the address space. If an authorized function has built an ACEE, DB2 passes the eight byte user ID from the ACEE.

6.8.2 Roll up accounting data for DDF and RRSF threads

If you establish a connection to DB2 V7 via DDF, you normally want to use DB2's *type 2 inactive threads*. This function is also known as *thread pooling*. This feature is enabled by specifying CMSTAT=INACTIVE in DSNZPARM. Using the inactive thread support allows you to connect up to 150.000 users to a single DB2 subsystem. However, if you are running high volume OLTP workloads in this environment, you might encounter a performance bottleneck, because DB2 cuts an accounting record on every COMMIT or ROLLBACK when using thread pooling, and SMF might have a hard time to keep up with the massive number of DB2 accounting records that are produced.

You may encounter a similar problem when using the RRS attach, in combination with WebSphere. WebSphere drives the RRS signon interface on each new transaction, and DB2 cuts an accounting record when this happens. An accounting record is cut, even though some of these transactions contain just one SELECT statement followed by a COMMIT.

DB2 V8 adds a new installation option to activate the rollup of accounting information for DDF threads that become inactive, and RRS threads. The new option *DDF/RRSAF ACCUM* has been added to installation panel DSNTIPN. The *default* is *NO*. The values accepted for this option range from 2 to 65535. The corresponding DSNZPARM is ACCUMACC.

When *NO* is specified, DB2 writes an accounting record when a DDF thread is made inactive, or when signon occurs for an RRSF thread. If any number between 2 and 65535 is specified, DB2 writes an accounting record after every *n* occurrences of end user on any RRS or DDF thread, where *n* is the number specified for this parameter. An *end user* is identified by a combination of the following three values:

- ▶ End user userid
- ▶ End user transaction name
- ▶ End user workstation name

Even when you specify a value between two and 65535, DB2 may choose to write an accounting record prior to the *n*th occurrence of the end user in the following cases:

- ▶ A storage threshold is reached for the accounting *rollup* blocks.
- ▶ You have specified accounting interval = 'COMMIT' on the RRSF signon call.
- ▶ When no updates have been made to the rollup block for 10 minutes, that is the user has not performed any activity for over 10 minutes that can be detected in accounting.

6.8.3 Improved query and result set processing

DB2 provides more flexibility for requestors such as DB2 Connect to specify larger query block sizes. This helps requestors to optimize their use of network resources. The DRDA protocol has been enhanced to allow for a maximum query block size of 10 MB, instead of the current 32 KB.

6.8.4 Time out for SNA allocate conversation requests

With DB2 V8, DDF searches every three minutes for threads waiting for a VTAM allocate conversation request to complete. 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 reason code of 00D31033. This is an indicator that the network is down and the network administrator should be notified of the communication failure.

6.8.5 Data stream encryption

A new connection encryption security mechanism is introduced in The Open Group Version 3 DRDA Technical Standard. In DB2 V, if connection encryption security mechanism is selected, then the userid, password and user data will be encrypted. This function:

- ▶ Provides the ability to encrypt and decrypt data as it is sent and received on the remote connection
- ▶ Is based on the technology used for password encryption
- ▶ Uses z/OS ICSF facilities with hardware assist

During connect processing, requester and server connection keys are exchanged and a shared connection key is generated. The connection keys are generated using the standard Diffie-Hellman distribution algorithm. Diffie-Hellman is the first standard public key algorithm ever invented. It gets its security from the difficulty of calculating discrete logarithms in a finite field. Diffie-Hellman requires three agreed upon values n , g such that g is a primitive of large prime n and the size of the exponent. The values for n and g are fixed. First, the application requester chooses a random large integer x and generates the value X where $X = g^x \text{ mod } n$. X is the requester connection key. Second, the application server chooses another random large integer y and generates the value Y where $Y = g^y \text{ mod } n$. Y is the server connection key. The application requester computes the shared private key, $k = Y^x \text{ mod } n$. The application server computes the shared private key, $k_1 = X^y \text{ mod } n$.

The 56-bit DES encryption key is generated from the shared private key. The Data Encryption Standard (DES), known as the Data Encryption Algorithm (DEA) by ANSI and the DEA-1 by ISO, is the worldwide standard for encrypting data. DES is a block cipher; it encrypts data in 64-bit blocks. DRDA encryption uses DES CBC mode as defined by the FIPS standard (FIPS PUB 81). DES CBC requires a 56-bit encryption key and an 8 byte token to encrypt the data. The Diffie-Hellman shared private key is 256 bits. To reduce the number of bits, 64 bits are selected from the connection key by selecting the middle 8 bytes and parity is added to the lower order bit of each byte producing a 56-bit key with 8 bits of parity. The middle 8 bytes of the server's connection key is used as the token.

The userid and password are encrypted at the requester. Once the userid and password are decrypted and the connection is authenticated, data streams exchanged between the requester and the server are encrypted using the same DES encryption key. The connection key is uniquely generated for each connection preventing the replay of data streams on different connection.

Connection encryption to a remote location

The communications database needs to be configured to enable connection encryption to a remote location. The SECURITY_OUT column in the IPNAMES that relates to the remote location must be updated with an “E” for connection encryption required. This column defines the security option that is used when local SQL applications connect to any remote server associated with this TCP/IP host.

Connection encryption from a remote location

No configuration is required to enable connection encryption from a remote location. During connect processing, the client negotiates the security mechanism for the connection. DB2 as a server will accept any connections requesting connection encryption.

Open Cryptographic Services Facility

Prior to V8, DB2 provided software support for the Diffie-Hellman algorithm and the DES decryption by loading in the required BSAFE services into the distributed address space. The encryption, decryption and D-H services were invoked directly by DDF. In V8, connection encryption will use The Open Cryptographic Services Facility (OCSF) to exploit any cryptographic hardware installed on the processor. If properly configured, DB2 will utilize the IBM CCA Cryptographic Module and the Cryptographic Hardware feature instead of invoking the BSAFE services directly using the software-only support. Additionally, the OS/390 Integrated Cryptographic Service Facility (ICSF) must be installed, configured to run with the Cryptographic Hardware feature, and must be active. See the *OS/390 ICSF Administrator's Guide*, SC23- 3975, for more information.

6.8.6 DISPLAY LOCATION command

Prior to DB2 V3, the **DISPLAY LOCATION** command allowed no PARMs, and displayed all locations. Beginning with DB2 V3 and with SNA two-phase-commit support, the **DISPLAY LOCATION** command was enhanced to accept PARMs and also a *DETAIL* keyword. In order to provide simplified transition to the new option of this command, DB2 allowed a blank PARM to provide the same output as it did before the PARM was introduced. That is, **DISPLAY LOCATION()** would behave as if **DISPLAY LOCATION** was used and displayed all locations. Adding a PARM displayed only matching locations.

This behavior of the **DISPLAY LOCATION** with an empty PARM is different from the behavior of **DISPLAY DATABASE** with an empty PARM. **DISPLAY DATABASE** command parsing requires a value in the PARM. If you issue a **DISPLAY DB() SPACENAME()**, the command parser will fail the command with message DSN9010I.

In an effort to make all commands behave in a more predictable and consistent manner, beginning with DB2 V8, the semantics of **DISPLAY LOCATION** command with an empty PARM are changed. **DISPLAY LOCATION()** will now behave the same way as a **DISPLAY DATABASE() SPACE()** command, the command will fail with the message DSN9010I.

Utilities

The utilities have all been enhanced to support the other important changes that DB2 V8 has introduced, which we have described in the other chapters. For instance, extensive updates have been required to provide support for long names, Unicode, 64-bit addressing, the new data partitioning techniques, and online schema evolution.

In this chapter, we concentrate on the following topics:

- ▶ Online schema changes support:
 - We examine the changes implemented across the various SQL statements, DB2 commands, and utilities in order to fully support the online schema evolution functions. We discuss the details with examples of the following:
 - Adding, rotating, and changing partitions
 - Utility support for schema evolution
 - Point-in-time recovery restrictions
- ▶ Delimited LOAD and UNLOAD
- ▶ Utility Unicode parser
- ▶ Distribution statistics
- ▶ Backing up and restoring the system
- ▶ Other changes, which include:
 - New default: RESTART
 - New defaults: SORTKEYS and SORTDATA
 - COPY and RECOVER tape parallelism

7.1 Online schema changes support

Online schema evolution allows for table, index, and table space attribute changes while maximizing application availability. This is described in 4.3, “Online schema changes” on page 50. In this section we discuss in more detail how you can add and rotate partitions in partitioned table spaces, and then show the changes in the utilities to accommodate for the new functionalities.

7.1.1 More flexibility with partitions

As we have seen in 4.3, “Online schema changes” on page 50, DB2 V8 allows the unbundling of partitioning and clustering, the data of table-controlled partitioning rather than index partitioning, and the data partitioned secondary indexes. Online schema evolution allows adding a partition to a partitioned table space through the ALTER TABLE statement. For example, you want to add a partition to the table space shown in Figure 7-1. The table space has data for each month in a separate partition starting from January 1998 and currently has 59 partitions.

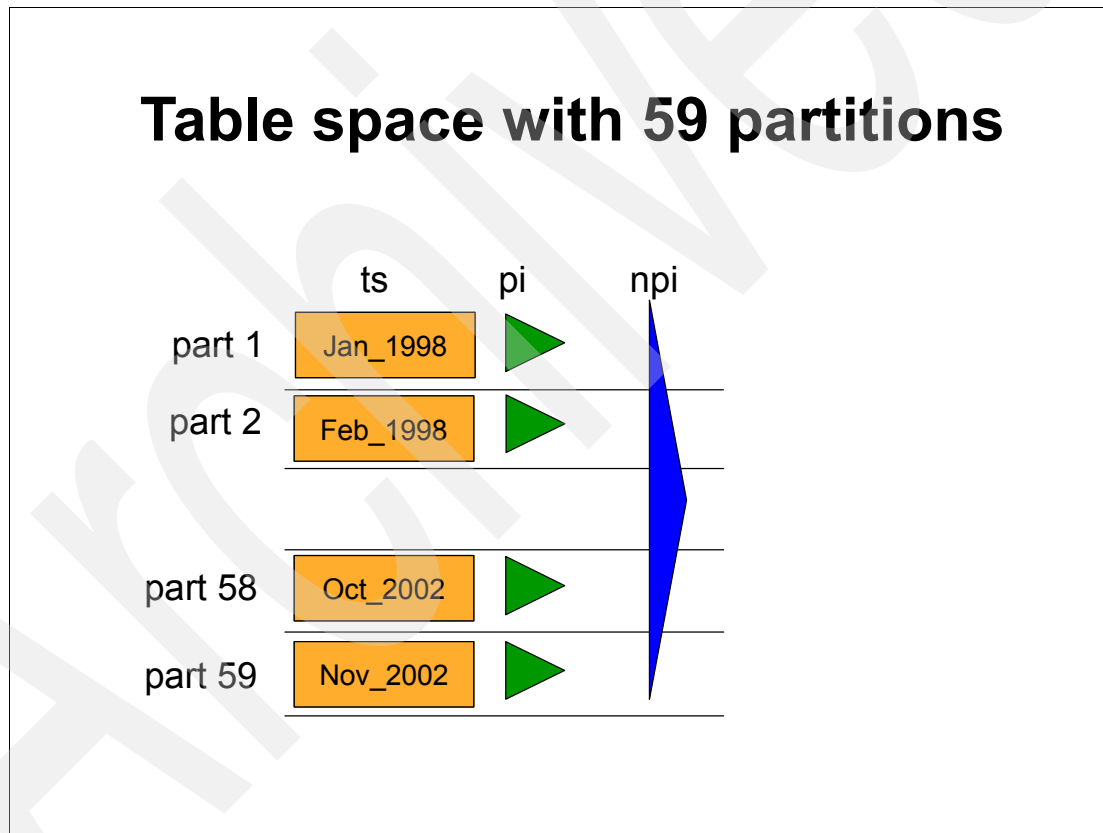


Figure 7-1 Online schema — Table space with 59 partitions

Adding partitions

Suppose that there is the requirement to define one more partition to store the data for December 2002. In V7, the only way this can be done is by dropping the table space and recreating the table space with 60 partitions. This would of course involve saving the existing data before dropping the table space, recreating the partitioning and non-partitioning indexes, reloading the data, recreating other objects which might have been dropped in the cascade process, reestablishing the privileges to users, rebinding all affected packages, etc.

In V8, you use the ALTER TABLE statement using the ADD PARTITION clause to add the partition. Figure 7-2 shows the new partition-definition block in the ALTER TABLE syntax. ADD PARTITION specifies that a partition is to be added to the table and each partitioned index of the table. The new partition is the next physical partition not being used, until the maximum for the table space has been reached.

The existing table space PRIQTY and SECQTY attributes are used for the new partition. For the partitioning index, the existing PRIQTY and SECQTY attributes of the previous index partition are used. You should execute ALTER TABLESPACE and ALTER INDEX statements after the new partition has been added to change the space attributes for the new partition. ENDING AT (constant,...) specifies the high key limit for the new partition, which should be higher than any other partition in the table.

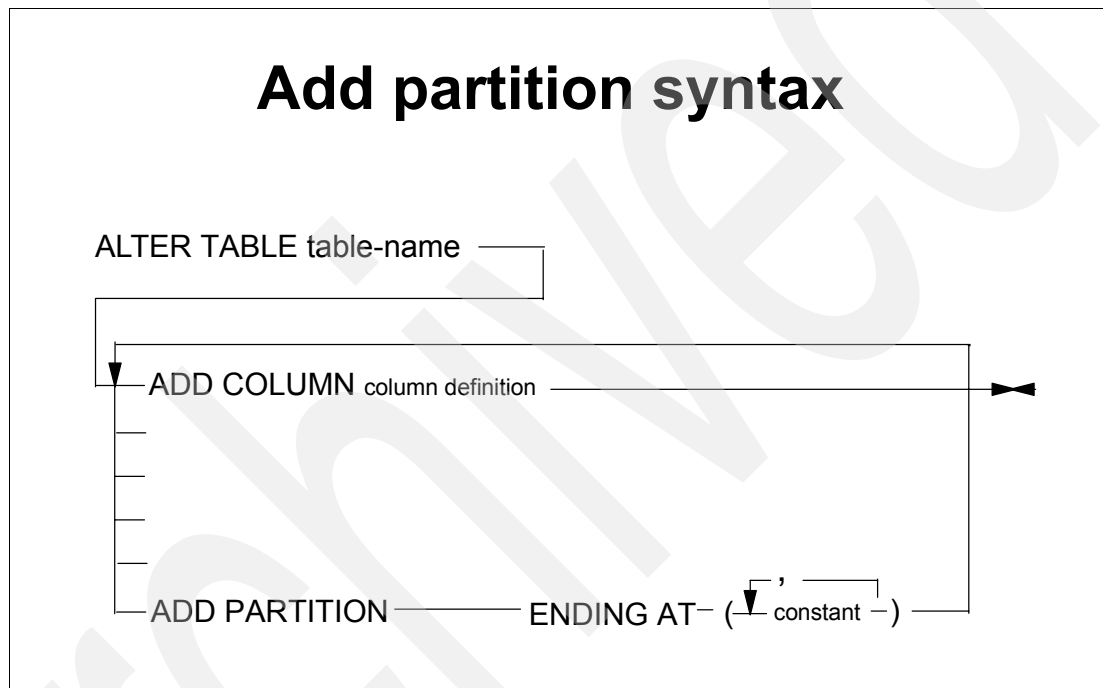


Figure 7-2 Online schema — ALTER TABLE ADD PARTITION syntax

You should be aware of the following considerations when using the ALTER TABLE statement with the ADD PARTITION clause.

- ▶ Stop the table space and the partitioned indexes before executing the ALTER statement.
- ▶ You can add only one partition within a single unit of work. If you want to add multiple partitions, add a COMMIT statement after each ALTER TABLE statement with the ADD PARTITION clause.
- ▶ When a partition is added, all the packages, plans, and dynamic cached statements referring to the table are invalidated.
- ▶ If the boundary for the last partition was not previously enforced, it is enforced after adding a partition, and the last two logical partitions are left in a Reorg Pending (REORP) state.
- ▶ If the last partition before adding the new one was in a REORP state, then the added partition is also placed in REORP state.
- ▶ You cannot add a partition if the table is a materialized query table or a materialized query table is defined on this table.

Figure 7-3 shows the table space with 60 partitions after the ALTER TABLE statement with the ADD PARTITION clause is executed.

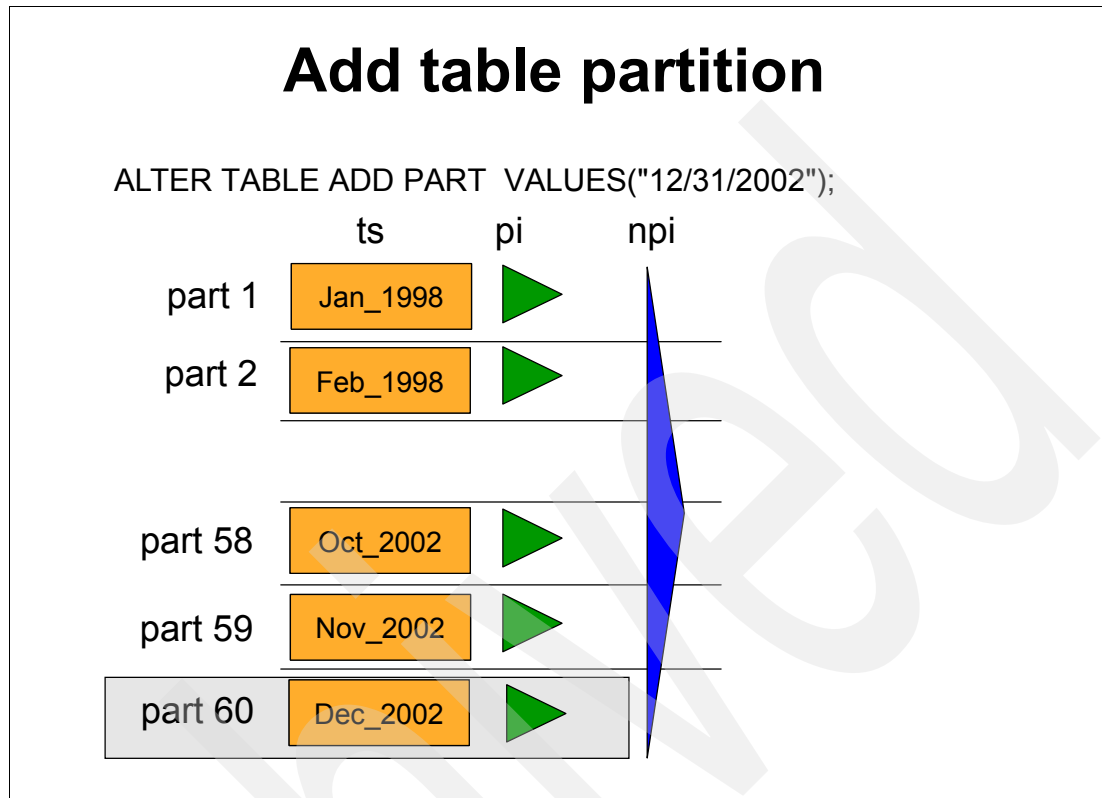


Figure 7-3 Online schema — Table space after adding a partition

The physical partition number, logical partition number, and time that the partition was added are recorded in columns PART, LOGICAL_PART, and CREATEDTS of the SYSTABLEPART catalog table.

In the above example, we started with the table created in V7. After the ALTER statement with the ADD PARTITION clause is executed, the table that used index-controlling partitioning is automatically converted to a table that uses table-controlled partitioning so that all the new features introduced by online schema enhancement can be exploited (see table controlled partitioning in 4.1.1, “Table-controlled partitioning” on page 40).

This conversion consists of the following steps:

- ▶ SYSTABLES.PARTKEYCOLNUM (new column introduced in V8) gets populated with the number of partitioning key columns.
- ▶ SYSCOLUMNS.PARTKEY_COLSEQ (new column introduced in V8) gets populated with the numeric sequence in the partitioning key.
- ▶ SYSTABLEPART.PARTKEY_ORDERING (new column introduced in V8) gets populated with the limit key values from SYSINDEXPART.LIMITKEY.
- ▶ SYSINDEXES.INDEXTYPE for the partitioning index is changed to ‘P’.
- ▶ SYSTABLESPACE.IBMREQD is marked with V8 release dependency.
- ▶ The last two logical partitions in the table space are placed in REORP state if the highest limit key was not previously enforced. In this example, partitions 59 and 60 are placed in REORP state.

Rotating partition

Online schema evolution allows rotating partitions in a partitioned table space through the ALTER TABLE statement. For example, you want to rotate a partition in the table space shown in Example 7-3. The table space has data for each month in a separate partition starting from January 1998 and currently has 60 partitions. Assume that at any time you want to have data for a maximum of 60 months in the table. Therefore, you want to rotate a partition so that the data in the oldest (or logically the first) partition is deleted and the partition essentially becomes the last logical partition in sequence ready to hold the new data which will be added. After the rotation, the table space should look as shown in Figure 7-4.

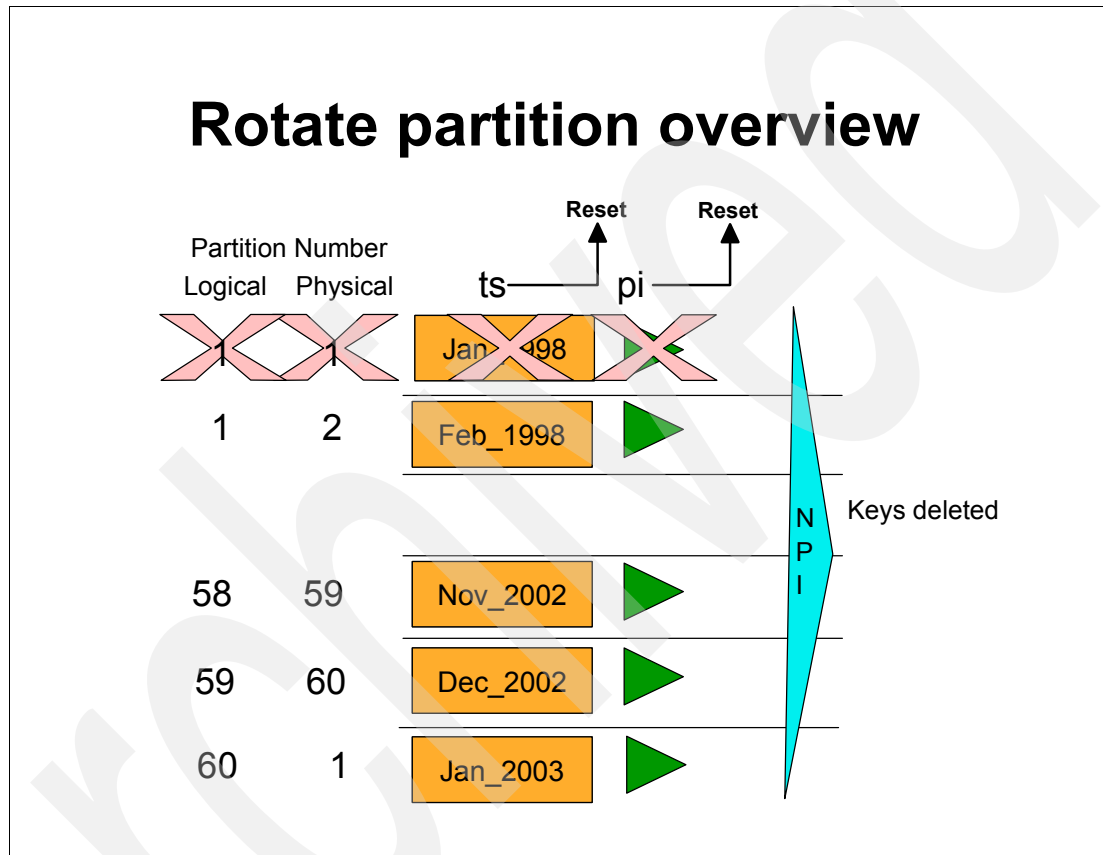


Figure 7-4 Online schema — Rotate partition overview

Figure 7-5 shows the ALTER TABLE syntax to rotate a partition. You can only rotate the oldest (or logically the first) partition and so the syntax does not allow you to specify the partition number. ROTATE FIRST TO LAST specifies that the first logical partition should be rotated to become the last logical partition. You can specify ROTATE and omit FIRST TO LAST. You supply the new partitioning key value with the ENDING AT clause. For an ascending partitioning key, this should be higher than any other partition in the table and for a descending partitioning key, this value should be lower than any other partition in the table.

If there is a referential constraint with DELETE RESTRICT on the table, the ROTATE will fail.

Rotate partition syntax

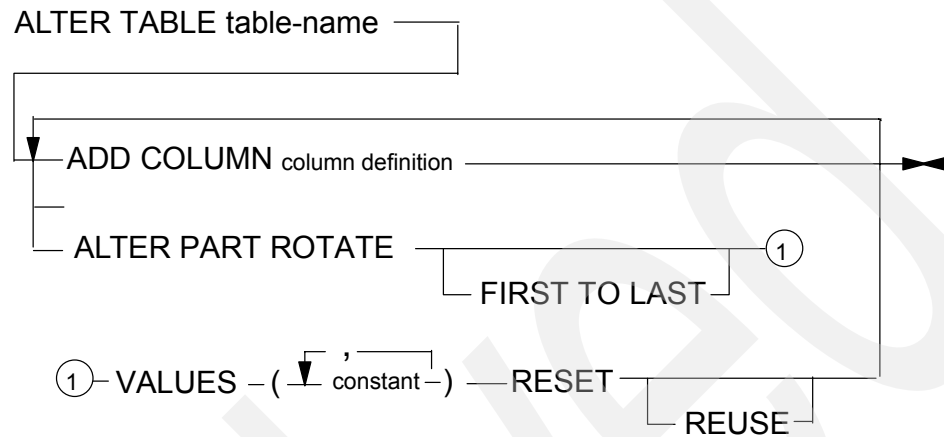


Figure 7-5 Online schema — Rotate partition syntax

Rotating a partition occurs immediately. Activity on the table is quiesced during the operation, and all the packages, plans, and dynamic cached statements referring to the table are invalidated.

If the boundary for the last partition was not previously enforced, it is enforced after issuing the ROTATE and the last two logical partitions are left in a REORP state.

If the last partition before issuing the ROTATE was in REORP state, then the last two logical partitions are left in REORP state.

A SYSCOPY record with an ICTYPE of 'A' and STYPE of 'R' is inserted for the partition rotated.

The keyword RESET specifies that the partition holding the "oldest" data should be rotated to the end and become a partition which will hold the "newest" data, or the active data being added to the table. The existing data in the "oldest" partition is deleted. SYSCOPY and SYSLGRNX rows associated with the partition being reset are deleted.

The optional keyword REUSE specifies that existing extents for the partition should be kept instead of deleting and redefining the underlying data sets. Logical reset of the partition is done instead of deleting and redefining data sets.

Figure 7-6 shows how the table space looks like after the rotate partition is achieved.

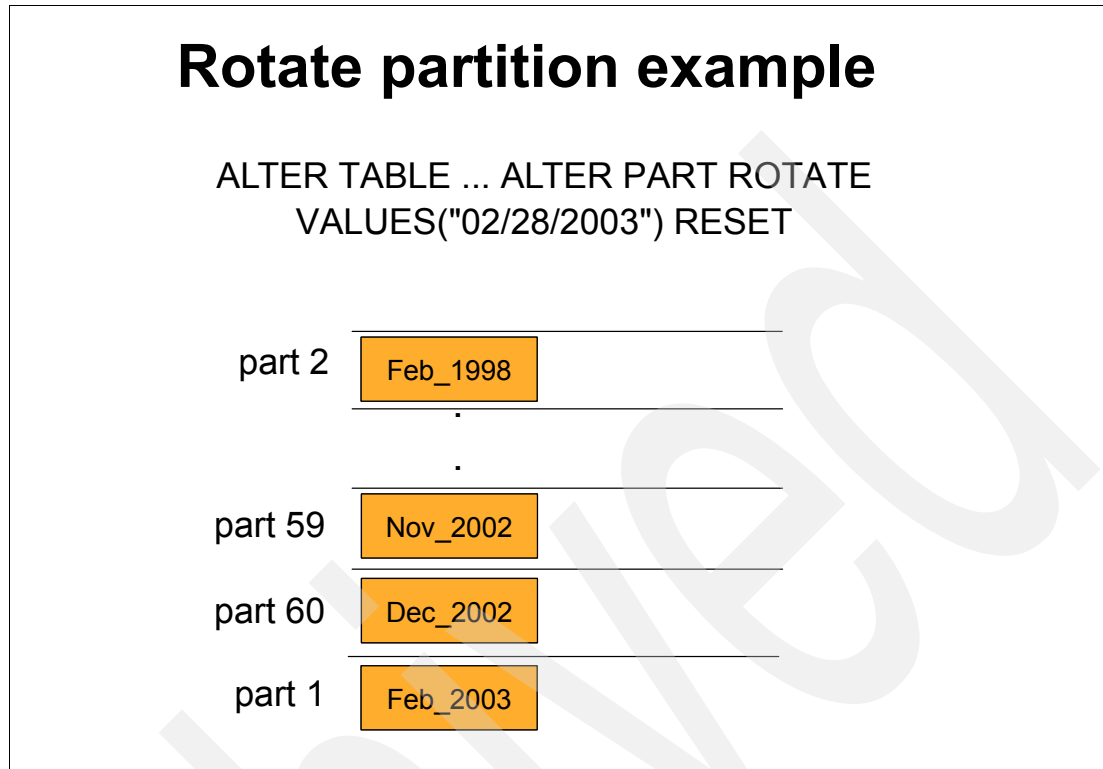


Figure 7-6 Online schema — Rotate partition example

You must have noticed though, the new key value specified is February whereas in fact it should have been January. You can alter the partition boundary by executing the ALTER TABLE statement.

Figure 7-7 shows the ALTER TABLE syntax to achieve this. The syntax allows you to specify the partition number and the key value.

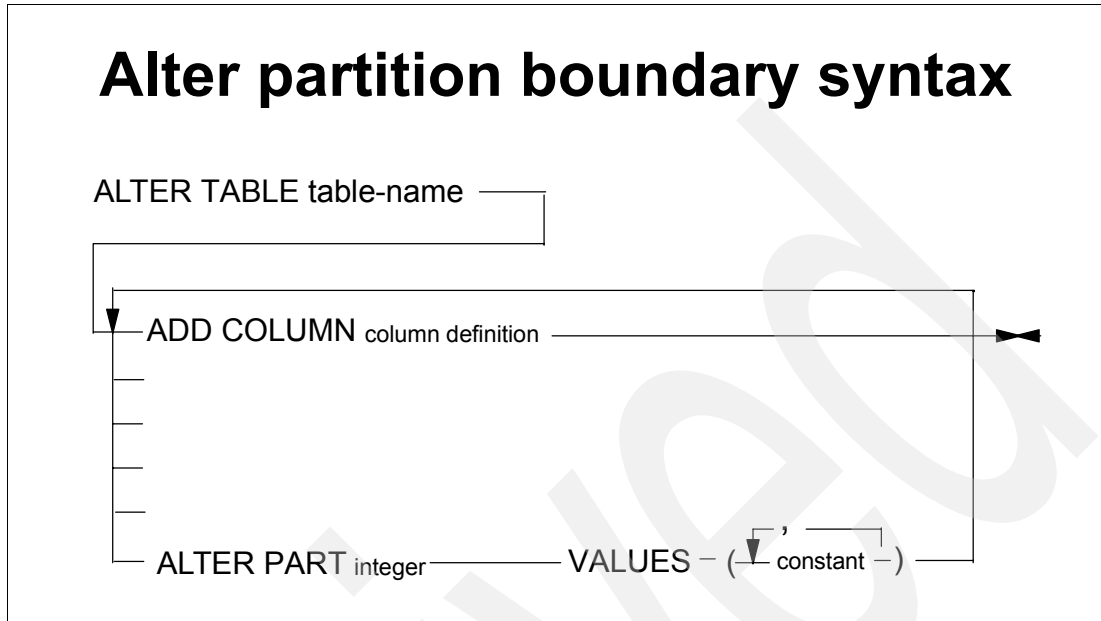


Figure 7-7 Online schema — Alter partition boundary syntax

Figure 7-8 shows the revised key value for the rotated partition after the ALTER TABLE statement is executed.

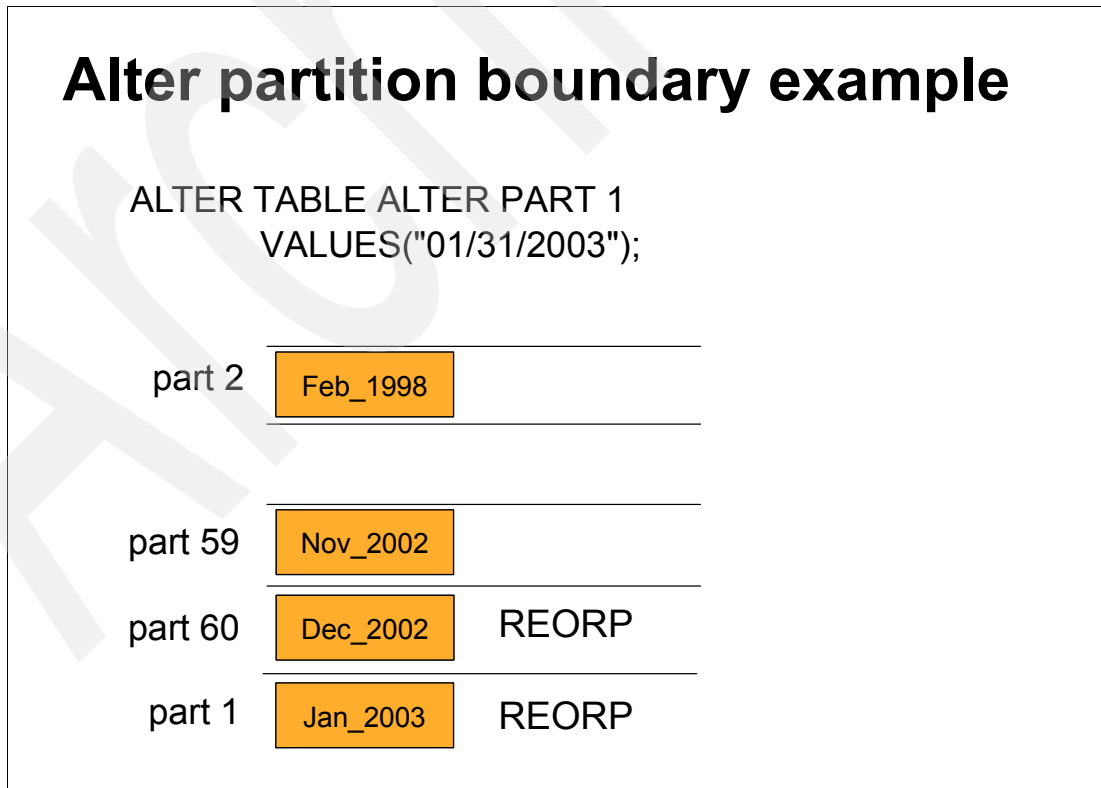


Figure 7-8 Online schema — Alter partition boundary example

DISPLAY DATABASE command changes

With the ability to add or rotate partitions at any time, the logical ordering can be quite different than the physical order. The DISPLAY DATABASE command in Example 7-1 shows the status of a table space with four partitions after a single part rotation:

```
-DISPLAY DATABASE(DB) SPACENAME(TS1) PART(1,2,3,4)
```

or

```
-DISPLAY DATABASE(DB) SPACENAME(TS1) PART(1:4)
```

Example 7-1 Displaying a four-partition table space

```
DSNT362I =          DATABASE = DB   STATUS = RW
          DBD LENGTH = 16142
DSNT397I =
NAME      TYPE PART STATUS              PHYERRLO PHYERRHI CATALOG  PIECE
-----
TS1       TS  0002 RW
TS1       TS  0003 RW
TS1       TS  0004 RW
TS1       TS  0001 RW
***** DISPLAY OF DATABASE DB ENDED *****
```

The DISPLAY DATABASE command in Example 7-2 is valid if 4 partitions were created within the table space, a single rotate operation was done and then 8 more partitions were added. The DISPLAY support shown in is included with the V8 enhancement that supports 4096 partitions:

```
-DIS DB(DB) SP(TS1) PART(1,2,4:6,9,10:12)
```

Example 7-2 Displaying partitions added with DB2 V8

```
DSNT362I =          DATABASE = DB   STATUS = RW
          DBD LENGTH = 16142
DSNT397I =
NAME      TYPE PART STATUS              PHYERRLO PHYERRHI CATALOG  PIECE
-----
TS1       TS  0002 RW
TS1       TS  0004 RW
TS1       TS  0001 RW
TS1       TS  0005 RW
TS1       TS  0006 RW
TS1       TS  0009 RW
TS1       TS  0010 RW
TS1       TS  0011 RW
TS1       TS  0012 RW
***** DISPLAY OF DATABASE DB ENDED *****
```

The above example also demonstrates that you can choose any partition or a range of partitions in the DISPLAY DATABASE command.

Assume there are 4096 partitions for table space TS1. Example 7-3 is one more example of output using DISPLAY support included with the 4096 partition enhancement:

```
-DIS DB(DB) SP(TS1)
```

Example 7-3 Displaying ranges of partitions with DB2 V8

```

DSNT362I =          DATABASE = DB   STATUS = RW
              DBD LENGTH = 16142

DSNT397I =
NAME        TYPE PART  STATUS                PHYERRLO PHYERRHI CATALOG  PIECE
-----
TS1         TS  0001  STOP
- THRU      0100
TS1         TS  0101  RW
- THRU      0999
TS1         TS  1000  RW,REORP
- THRU      1002
TS1         TS  2002  RW,REORP
- THRU      2003
TS1         TS  1005  RW,REORP
- THRU      1006
TS1         TS  1007  RW
- THRU      1100
TS1         TS  1101  RW,REORP
TS1         TS  1102  RW,REORP
TS1         TS  1103  RW
- THRU 4096
***** DISPLAY OF DATABASE DB ENDED *****

```

The DBET state,RBDP, is set for indexes after altering from PADDED to NOT PADDED. Example 7-4 shows an example.

-DIS DB(DB) SP(PIX)

Example 7-4 Displaying indexes after ALTER

```

DSNT362I =          DATABASE = DB   STATUS = RW
              DBD LENGTH = 16142

DSNT397I =
NAME        TYPE PART STATUS                PHYERRLO PHYERRHI CATALOG  PIECE
-----
PIX         IX  0001  RW,RBDP
PIX         IX  0002  RW,RBDP
PIX         IX  0003  RW,RBDP
PIX         IX  0004  RW,RBDP
***** DISPLAY OF DATABASE DB ENDED *****

```

7.1.2 Utility support for schema evolution

In this section we examine the changes implemented across the various utilities in order to fully support the schema evolution functions.

REORG TABLESPACE

We discuss the various enhancements introduced in the REORG TABLESPACE utility.

SCOPE PENDING

REORG TABLESPACE is extended with the new keyword SCOPE to indicate the scope of the reorganization for the table space or partition range. You specify SCOPE ALL to reorganize the entire table space or the partition range. This is the default. You specify SCOPE PENDING to 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 reorganize a range of partitions and specify SCOPE PENDING, make sure that the adjacent partition outside the specified range is not in a REORP state. The REORG terminates with an error otherwise.

You cannot specify SCOPE PENDING with REBALANCE, OFFPOSLIMIT, INDREFLIMIT, REPORTONLY, UNLOAD ONLY, and UNLOAD EXTERNAL keyword specifications.

Rebalance

The REORG 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. Figure 7-9 shows an example of reorganizing a table space to rebalance the partitions.

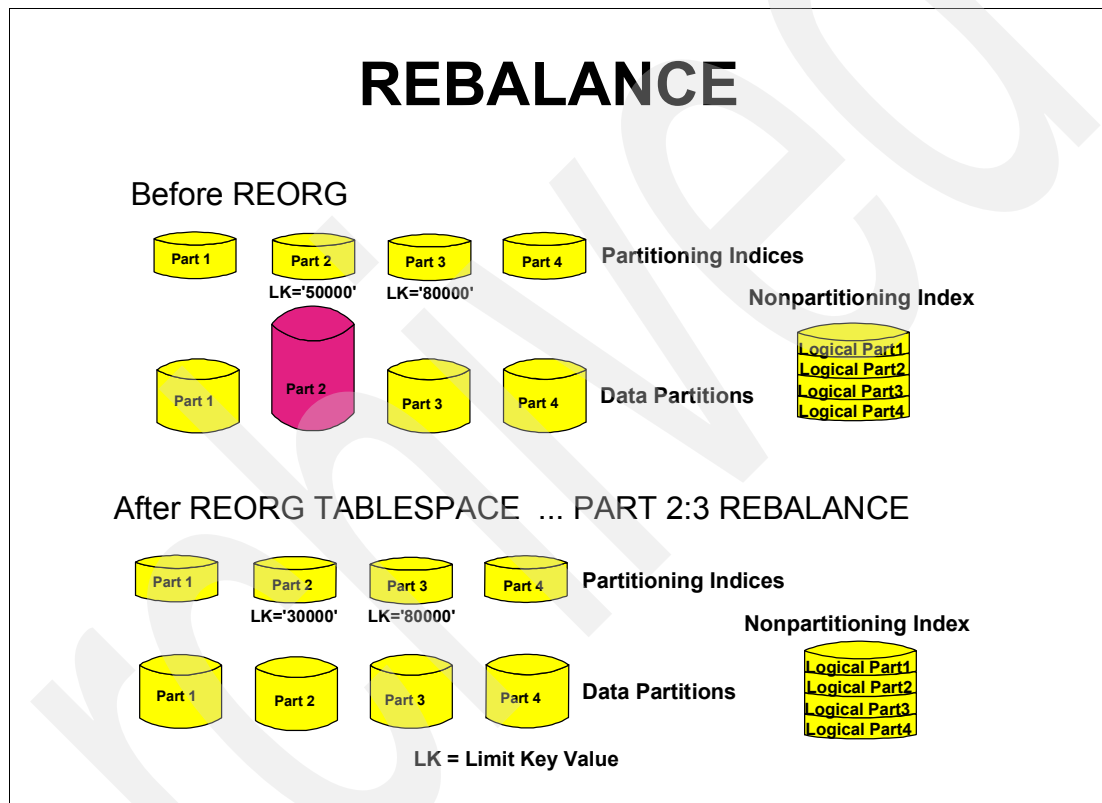


Figure 7-9 Example of REORG TABLESPACE to rebalance partitions

REBALANCE specifies that REORG 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 values and limit keys.

Perfect rebalancing is not always possible if the columns used in defining the partition boundaries have many duplicate values within the data rows. You cannot specify the keyword REBALANCE with SHRLEVEL CHANGE, nor with SCOPE PENDING, OFFPOSLIMIT, INDREFLIMIT, REPORTONLY, UNLOAD ONLY, and UNLOAD EXTERNAL keyword specifications.

The default setting for REORG TABLESPACE is now SORTDATA when either UNLOAD CONTINUE or UNLOAD PAUSE is in effect. Sorting uses the implicit clustering index if there is no explicit CLUSTER index. If there is a table in the table space without an index, SORTDATA operates as in previous releases.

The default setting for REORG TABLESPACE is SORTKEYS.

When using REBALANCE on a table where the clustering index does not match the partitioning key, REORG must be run on the partition range twice to ensure that the rows are in optimal clustering order. The first reorganization moves data rows to the appropriate partition, and the second reorganization orders each data row in clustering order within the appropriate partition.

When reorganizing a table space which has indexes defined as COPY NO and IOFACTOR is -1 (no longer using pre-V8 versioning), REORG updates the new catalog column SYSINDEXPART.OLDEST_VERSION to the value of CURRENT_VERSION for those indexes (this is not done for non-partitioned indexes unless reorganizing the whole table space). For non-partitioned table spaces, REORG also updates SYSINDEXES.OLDEST_VERSION to the same value. If the index is partitioned, REORG also updates SYSINDEXES.OLDEST_VERSION if the lowest OLDEST_VERSION from all SYSINDEXPART entries has changed.

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 REORG is reset. SYSCOPY records include the current version number, or CURRENT_VERSION, at the time of reorganization. The REORG TABLESPACE utility inserts SYSCOPY records for each index that is built as part of the reorganization.

Discard with SHRLEVEL CHANGE

You can specify the keyword DISCARD with SHRLEVEL CHANGE. However, modifications to data rows are not allowed during the window when they are being discarded in the unload phase by reorganization. If this is detected, REORG terminates with an error.

Online REORG has no BUILD2 phase for DPSI

You can use the new data partitioned secondary indexes to your advantage when using online REORG. We illustrate this with the following example:

A partitioned table space with 13 partitions is created in V7, with each partition holding the data for a month. The partitioning index is on the date. The non-partitioning index is on the account number. The DDL is as shown in Example 7-5.

Example 7-5 Sample DDL for index partitioned table space

```
CREATE TABLE TRANSACTIONS (
  ACCTNO  INTEGER          NOT NULL,
  POSTED  DATE             NOT NULL,
  AMOUNT  DECIMAL(12,2)   NOT NULL)
IN DB.TS ;

CREATE INDEX IX1 ON TRANSACTIONS(POSTED)
CLUSTER
(PART 1  VALUES('01/31/2002'),
 PART 2  VALUES('02/28/2002'),
 PART 3  VALUES('03/31/2002'),
 PART 4  VALUES('04/30/2002'),
 PART 5  VALUES('05/31/2002'),
 PART 6  VALUES('06/30/2002'),
 PART 7  VALUES('07/31/2002'),
 PART 8  VALUES('08/31/2002'),
 PART 9  VALUES('09/30/2002'),
 PART 10 VALUES('10/31/2002'),
 PART 11 VALUES('11/30/2002'),
```

```
PART 12 VALUES('12/31/2002'),
PART 13 VALUES('01/31/2003');
```

```
CREATE INDEX IX2 ON TRANSACTIONS(ACCTNO) ;
```

```
REORG TABLESPACE DB.TS PART 11 SHRLEVEL CHANGE
```

Figure 7-10 shows the index-based partitioning with a non-partitioning index on ACCTNO. Access to the data is predominantly through the index based on ACCTNO. The partitioning index is a mandatory requirement in index-based partitioning and so has to be created. If you run online REORG at a partition level, data cannot be accessed during the BUILD2 phase when DB2 corrects the logical partitions of the non-partitioning index on ACCTNO. This may not always be acceptable in a 24x7 environment.

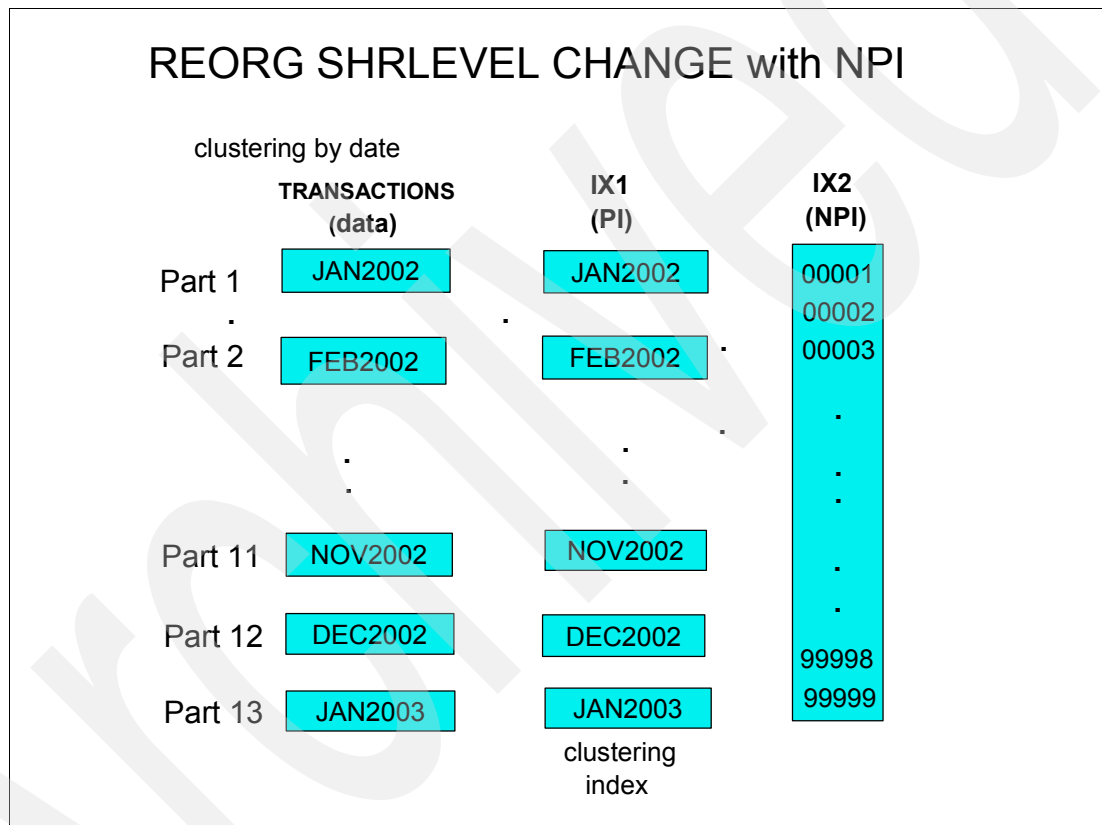


Figure 7-10 Example of index-based partitioning

In V8, data-based partitioning and use of data-partitioned secondary indexes helps to overcome the non-availability of data in the BUILD2 phase as shown in Example 7-6.

Example 7-6 Avoiding BUILD2 phase

```
DROP INDEX IX1 ;
CREATE INDEX IX3 ON TRANSACTIONS(ACCTNO)
PARTITIONED CLUSTER DEFER YES;
COMMIT ;
DROP INDEX IX2 ;
COMMIT ;
REBUILD INDEX IX3;
```

```
REORG TABLESPACE DB.TS PART 11 SHRLEVEL CHANGE
```

This solution retains a single index based on ACCTNO. However, since this index is a DPSI, there is no BUILD2 phase during online REORG of partition 11.

Figure 7-11 shows a proposed solution to eliminate BUILD2 phase with online REORG.

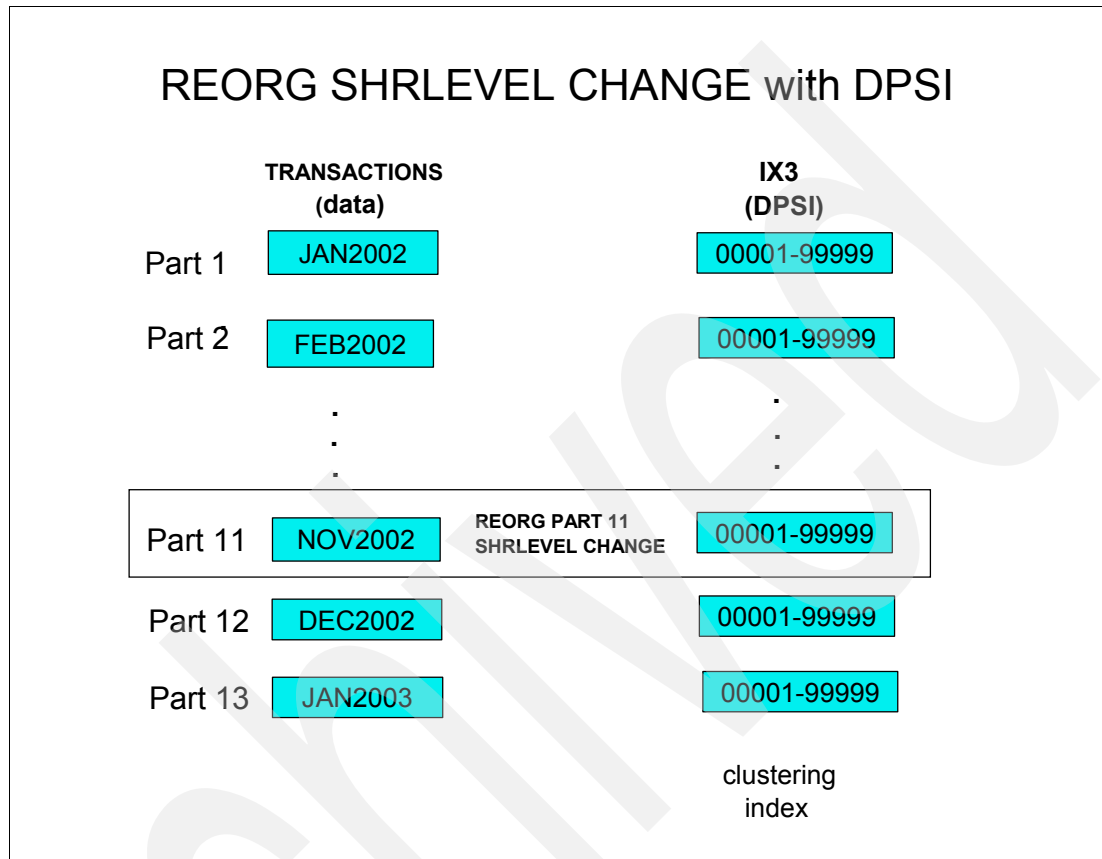


Figure 7-11 A proposed solution to eliminate BUILD2 phase with online REORG

When you drop the partitioning index IX1, DB2 converts the index-based partitioning to table-based partitioning. At this point, DB2 marks the table space as release dependent. The DPSI IX3 is a partitioned index with 13 partitions matching the number of data partitions. There is no need to retain the non-partitioning index and therefore you can drop index IX2.

REORG INDEX

REORG INDEX accepts an index space name instead of just an index name.

When reorganizing an entire index defined as COPY NO and IOFACTOR is -1 (no longer using pre-V8 versioning), REORG INDEX updates the new catalog columns SYSINDEXES.OLDEST_VERSION and SYSINDEXPART.OLDEST_VERSION to the value of CURRENT_VERSION.

For non-partitioned indexes, REORG also updates SYSINDEXES.OLDEST_VERSION to the same value. If the index is partitioned, REORG INDEX also updates SYSINDEXES.OLDEST_VERSION if the lowest OLDEST_VERSION from all SYSINDEXPART entries has changed.

The REORG INDEX utility resets indexes which are in AREO* state.

SYSCOPY records have the OLDEST_VERSION column filled in from the CURRENT_VERSION value at the time of reorganization.

REBUILD INDEX

REBUILD INDEX accepts index space names instead of just index names.

REBUILD INDEX is extended with the new keyword SCOPE to indicate the scope of the rebuild. You specify SCOPE ALL to reorganize all the specified indexes. This is the default. You specify SCOPE PENDING to indicate that the specified indexes should be built 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 RBDP.

When rebuilding an entire index defined as COPY NO and IOFACTOR is -1 (no longer using pre-V8 versioning), REBUILD updates the new catalog column SYSINDEXES.OLDEST_VERSION and SYSINDEXPART.OLDEST_VERSION to the value of CURRENT_VERSION.

This is not done for non-partitioned indexes unless rebuilding the entire index. For non-partitioned indexes, REBUILD INDEX also updates SYSINDEXES.OLDEST_VERSION to the same value. If the index is partitioned, REBUILD INDEX also updates SYSINDEXES.OLDEST_VERSION if the lowest OLDEST_VERSION from all SYSINDEXPART entries has changed.

SYSCOPY records with an ICTYPE value of 'B' and an STYPE value of 'A' (when RBDP is reset) are inserted for each index partition.

LOAD

LOAD is able to process a partitioned table space which does not have a partitioning index. If a single index exists and the data is ordered, the sort phase is skipped if SORTKEYS is not specified.

LOAD PART REPLACE is still restricted from running against partitions in REORP state, even if all partitions in the pending state have been specified with the PART REPLACE option.

When doing LOAD REPLACE of an entire table space LOAD updates the catalog for all indexes defined as COPY NO so that the new catalog column SYSINDEXES.OLDEST_VERSION is equal to the value of CURRENT_VERSION.

The LOAD REPLACE utility resets indexes which are in AREO* or RDBP state.

SYSCOPY records with an ICTYPE value of 'R' or 'S' and an STYPE of 'A' are inserted for each index partition where RDBP is reset.

SYSCOPY records include the current version number, or CURRENT_VERSION, at the time of reorganization. LOAD REPLACE inserts SYSCOPY records for each index that is built as part of LOAD.

The default setting for LOAD is SORTKEYS.

UNLOAD

UNLOAD supports image copies with data from different versions. If the system pages containing the version describing the format of a row is not available within a image copy, an error message is issued up to the maximum allowed by the MAXERR specification.

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 is changed to recognize the table space system pages. 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.

DSN1COPY

DSN1COPY is changed to tolerate the existence of table space system pages. When the PRINT option is specified, they are printed in hexadecimal format.

When the OBIDXLAT option is specified all OBIDs within the system pages are translated. It is recommended that the object be reorganized before using DSN1COPY with the OBIDXLAT option. Before using the copied object on the target system, the version numbers should be updated using the REPAIR utility.

The CHECK option now also validates system pages.

COPY

COPY is extended with the new keyword SYSTEMPAGES. You can specify either YES (this is the default) or NO. SYSTEMPAGES YES ensures the dictionary and version system pages are located at the beginning of the object being copied. This ensures that an unload from the image copy has the necessary system pages to correctly format and unload all data rows. SYSTEMPAGES NO does not ensure this and the behavior of COPY utility is the same as in the versions prior to V8.

COPY ensures that each image copy includes a header page for each partition, page set, or piece that is copied. This simplifies processing by other utilities which can exploit embedded information to interpret the contents and original environment of the image copy.

The SYSCOPY records inserted by COPY have the OLDEST_VERSION column filled in with the lowest version of data within the copied object.

The CHECKPAGE option also validates system pages.

MODIFY

MODIFY reclaims obsolete versions for reuse by updating the new catalog column OLDEST_VERSION values for an objects within the SYSTABLEPART, SYSINDEXPART, SYSTABLESPACE, and SYSINDEXES tables in the catalog. The lowest version for an object is determined by the lowest OLDEST_VERSION value among all SYSCOPY rows left for the object or partition.

MODIFY updates SYSTABLEPART.OLDEST_VERSION for table spaces and SYSINDEXPART.OLDEST_VERSION for all indexes with the COPY YES attribute.

For non-partitioned table spaces, MODIFY also updates the appropriate SYSTABLESPACE.OLDEST_VERSION or SYSINDEXES.OLDEST_VERSION to the same value. If the object is partitioned, MODIFY also updates OLDEST_VERSION of the appropriate SYSTABLESPACE or SYSINDEXES table if the lowest OLDEST_VERSION from all partitions has changed.

RUNSTATS

The RUNSTATS utility continues to handle unusable statistics in the same manner it does when a column is changed to extend the length of a varying length character column.

When ALTER TABLE ALTER COLUMN SET DATA TYPE updates the STATSTIME column in SYSCOLUMNS and SYSCOLSTATS tables to default + 1 day to indicate that certain statistics are unusable, RUNSTATS treats these statistics as not existing when doing aggregation of partition level statistics. A DSNU623 message is issued during aggregation if all parts in SYSCOLSTATS do not have legitimate STATSTIME values.

REPAIR

REPAIR DBD rebuilds new OBD structures from the catalog. The catalog contains information about added partitions, rotated partitions, and limit keys for the partition boundaries, and partitioning columns.

The REPAIR utility is extended with new syntax, REPAIR VERSIONS. Figure 7-12 shows the new syntax.

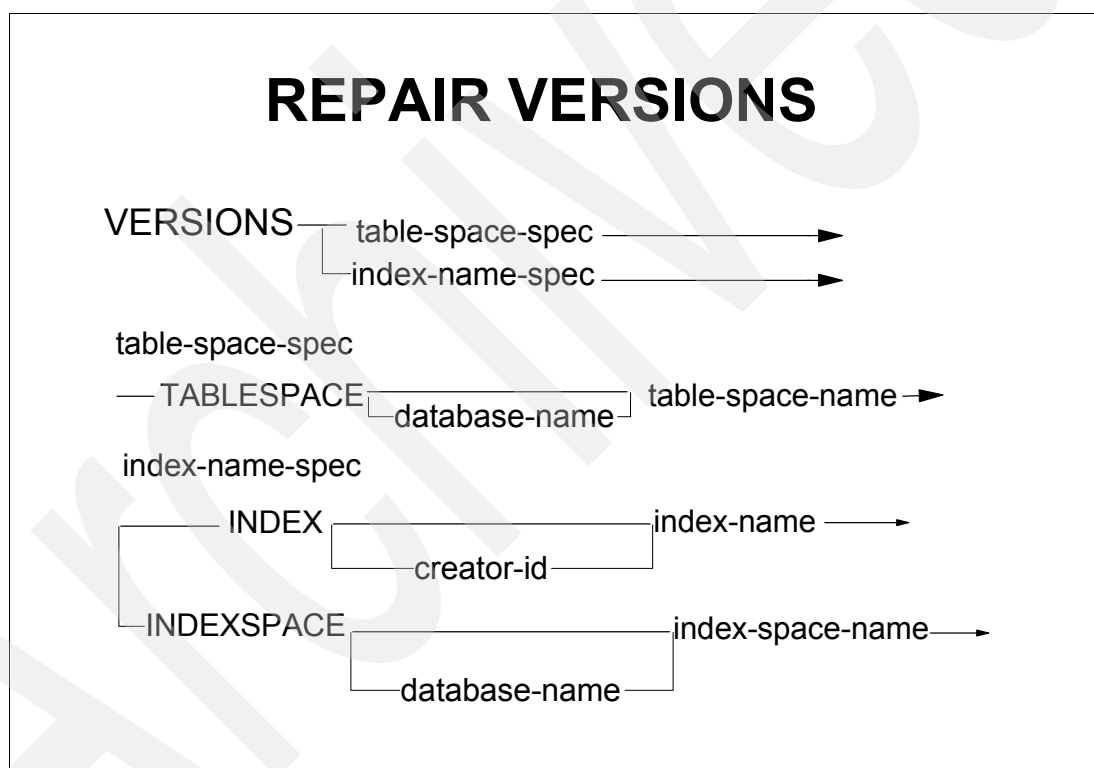


Figure 7-12 REPAIR VERSIONS syntax

REPAIR VERSIONS allows fixing version information for table spaces and indexes when moving them between systems using DSN1COPY with the OBIDLAT option. This also facilitates version recycling procedure for objects that do not use the IBM REORG utility. REPAIR VERSIONS also cuts a SYSCOPY record with an ICTYPE of 'V' that MODIFY can use for reclaiming version numbers. It fills in the OLDEST_VERSION column with the lowest version found within the active object.

When objects are moved from one system to another and contain system pages, the version information on the target system must match the source versions 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.

Indexes that are copied may or may not have been altered in V8 so that they have a OLDEST_VERSION and CURRENT_VERSION. If not altered in V8, then version information is tracked in IOFACTOR. In this case ensure that CURRENT_VERSION and OLDEST_VERSION contain zeros for both the source and target systems. REPAIR VERSIONS updates IOFACTOR appropriately on the target system.

2. When IOFACTOR is -1, OLDEST_VERSION and CURRENT_VERSION are being used. Ensure that the source object does not have a OLDEST_VERSION of 0 with a CURRENT_VERSION greater than 0. If this is the case, first reorganize the object so that the OLDEST_VERSION matches the high version.
3. Ensure there are enough versions available. 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;
#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 the DSN1COPY with the OBIDLAT 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 will update the OLDEST_VERSION and CURRENT_VERSION in SYSTABLEPART. It will also update VERSION in SYSTABLES. For indexes, this will update the OLDEST_VERSION and CURRENT_VERSION in SYSINDEXES.

The formula for updating the SYSTABLEPART and SYSINDEXES version numbers follows:

```
CURRENT_VERSION = MAX(target.CURRENT_VERSION,source.CURRENT_VERSION)
OLDEST_VERSION = MIN(target.OLDEST_VERSION,source.OLDEST_VERSION)
```

REPAIR can also be used to reset the RBDP status for the specified index and the AREO* status for the specified table space or index. The keywords provided for these are NORBDPEND and NOAREORPENDSTAR respectively.

7.1.3 Point-in-time recovery restrictions

With the enhancements introduced by online schema, it is not always possible for you to recover a table space to a point-in-time. In some instances such a recovery is blocked.

Figure 7-13 discusses the following scenario of recovering a table space to a point-in-time:

- ▶ At time T1, you do an ALTER TABLESPACE to rotate a partition. When the rotation is completed, DB2 writes a record in SYSCOPY.
- ▶ At time T2, you do an ALTER TABLE ALTER COLUMN to change, for example, the data type of a column. This causes DB2 to place the table space in AREO* state.
- ▶ At time T4, you do a REORG TABLESPACE and this resets the AREO* state.

Recovery of table space to point-in-time

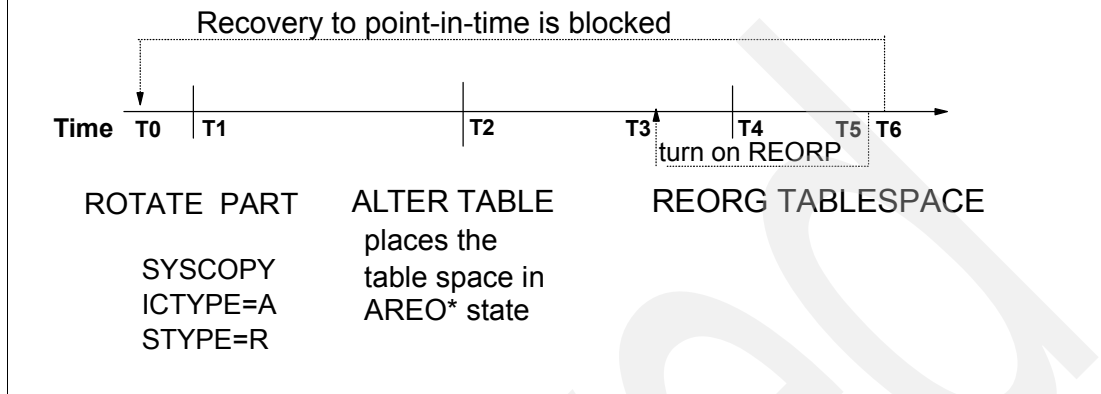


Figure 7-13 Online schema — Recovery of table space to a point-in-time

- ▶ At time T5, if you recover the table space to point-in-time T3 which is before the time you did the REORG TABLESPACE, then after the recovery is done, DB2 places the table space in REORP state.
- ▶ At time T6, if you recover the table space to a point-in-time T0 which is before you did the rotate partition, DB2 displays a message to the effect that the recovery is blocked and cannot be done.

We discuss some specific details regarding recovery of a table space to a point-in-time.

RECOVER TABLESPACE

RECOVER TABLESPACE to current assumes that the current catalog and directory definitions remain intact. There is no special handling required.

RECOVER TABLESPACE to a previous point-in-time is changed when crossing new SYSCOPY records inserted to support new functions in this enhancement.

New SYSCOPY records are listed in Table 7-1.

Table 7-1 RECOVER TABLESPACE PIT actions

Action	SYSCOPY ICTYPE	SYSCOPY STYPE	PART RANGE	RECOVER to PIT ACTION
SET DATA TYPE	N/A	N/A	N/A	Completes
ADD PARTITION	N/A	N/A	N/A	Completes
ROTATE PARTITION	A	R	For physical part reset	Blocked
ALTER PARTITION BOUNDARY	W,X,R or S	A	For each REORP	Turn on REORP

Since all version definitions within the object being recovered are kept, there is no problem recovering data from a previous version. When recovering to a point before the first version was saved in system pages, the version definition is contained within the new SYSOBDS catalog table.

With the enhancements introduced by online schema, it is not always possible for you to recover a COPY YES index to a point-in-time. In some instances such a recovery is blocked.

Example 7-14 discusses the following scenario of recovering an index to a point-in-time:

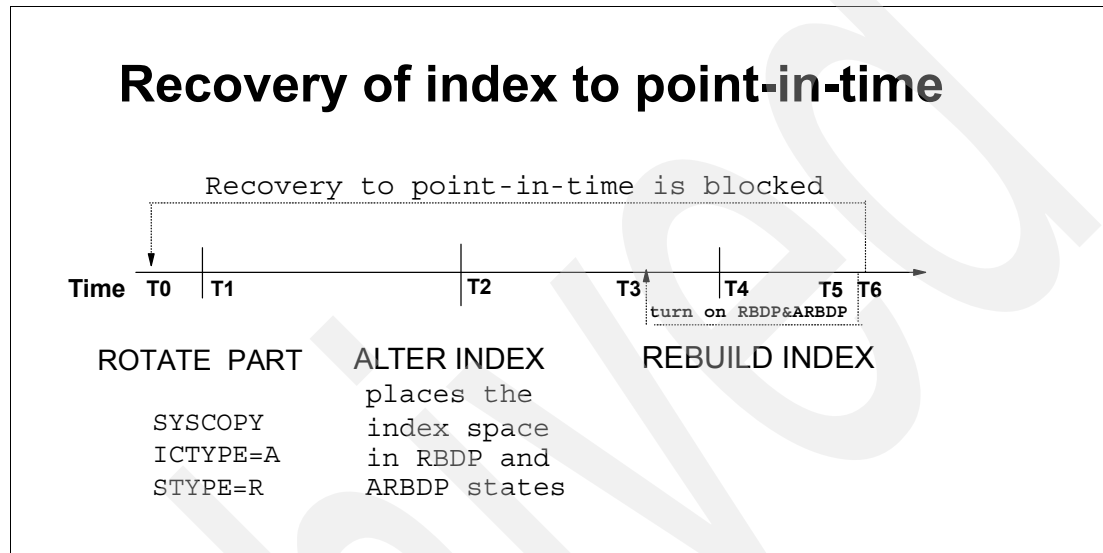


Figure 7-14 Online schema — Recovery of index to a point-in-time

- ▶ At time T1, you do an ALTER TABLESPACE to rotate a partition. When the rotation is completed, DB2 writes a record in SYSCOPY.
- ▶ At time T2, you do an ALTER INDEX to change, for example, from NOT PADDED to PADDED. This causes DB2 to place the index space in RBDP state.
- ▶ At time T4, you do a REBUILD INDEX and this resets the RBDP state.
- ▶ At time T5, if you recover the index to point-in-time T3 which is before the time you did the REBUILD INDEX, then after the recovery is done, DB2 places the index in RBDP state.
- ▶ At time T6, if you recover the index to a point-in-time T0 which is before you did the rotate partition, DB2 displays a message to the effect that the recovery is blocked and cannot be done.

We discuss some specific details regarding recovery of an index to a point-in-time.

RECOVER INDEXSPACE

Indexes for point-in-time recovery are handled the same as without versions. If copies of the indexes taken at a consistent point with the table space are also part of the recovery, then the indexes are immediately available. If the table space is recovered but not the indexes, then the indexes are set to RBDP state.

New SYSCOPY records are listed in Table 7-2.

Table 7-2 RECOVER INDEXSPACE PIT actions

Action	SYSCOPY ICTYPE	SYSCOPY STYPE	PART RANGE	RECOVER ACTION
SET DATA TYPE numeric	N/A	N/A	N/A	Blocked because SYSCOPY rows were deleted
SET DATA TYPE numeric	B	A	All parts	Set RBDP
ALTER NOT PADDED	X,W,R,S, or B	V	All parts	Sets RBDP
ALTER PADDED	N/A	N/A	N/A	Completes
ALTER PARTITION BOUNDARY	A	R	For each REORP reset	Turn on REORP
ALTER PARTITION BOUNDARY	W,X,R, or S	A	For each REORP reset	Turn on REORP
ROTATE PARTITION	A	R	For physical or logical part reset	Blocked

7.2 Delimited LOAD and UNLOAD

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.

Most relational database management systems including DB2 on the UNIX and Windows platforms, are capable of unloading data in a delimited format, where each record is a row, with columns separated by commas, and optionally delimited with, for example, double quote marks. The LOAD utility in DB2 V7 and prior releases expects data in the positional format and most other DBMS systems cannot unload data in the positional format. If you want to move data to DB2 for z/OS and OS/390, you have to therefore either write a program to convert the data into the positional format, or use insert processing, thereby without exploiting the performance advantages of the LOAD utility.

The LOAD utility in DB2 V8 is enhanced to accept data from a delimited file. The UNLOAD utility in DB2 V8 is also enhanced to produce a delimited file when unloading the data. This enhancement helps to simplify the process of migrating data into and out of DB2 for z/OS. This function is implemented in total compatibility with the other members of the DB2 family.

LOAD delimited input functional description

The delimited file on z/OS is a sequential file consisting of one or more fixed or variable records. Since the end of the record is inherent in the file structure, record delimiters, such as CRLF, are not used. The LOAD utility syntax has been changed and additional options are added for the keyword FORMAT. Figure 7-15 shows the syntax to support the enhancement.

LOAD delimited input syntax

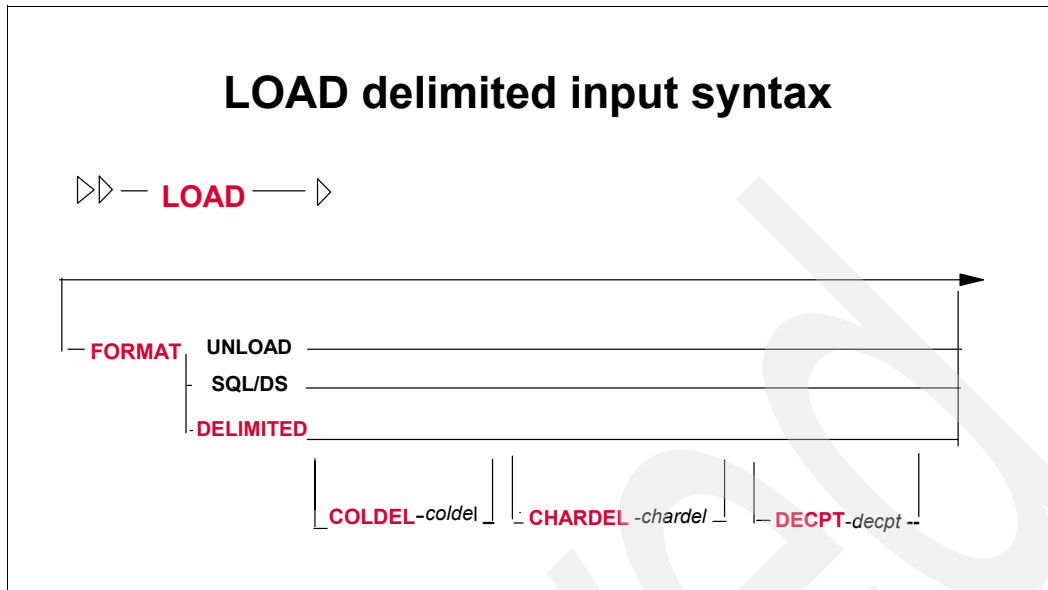


Figure 7-15 LOAD delimited input syntax

`DELIMITED` 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 in character string or numeric external form, each column value is separated from the next by a column delimiter character (the default is a comma), and character strings which contain column delimiters are demarcated by character string delimiter characters (the default is a double quote).

`COLDEL coldel` specifies the column delimiter character (the default is a comma) that is used in the input file when `FORMAT DELIMITED` is specified.

`CHARDEL chardel` specifies the character data string delimiter (the default is double quote) that is used in the input file when `FORMAT 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:

"what a ""nice day"" day" is loaded as what a "nice day" day

`DECPT decpt` specifies the decimal point character (the default is a period) that is used in the input file when `FORMAT DELIMITED` is specified.

Example 7-7 on page 190 shows a sample `LOAD` job with delimited input.

In the example the column delimiter is a comma, the character string delimiter is a double quote, the decimal point is a period; these are also the defaults. Note that a comma is not allowed in the data since it is defined as column delimiter. Two successive commas would mean that no value is specified for the skipped column. A comma, or any column delimiter, would be allowed within any cell value as long as the values was enclosed in character delimiters.

Example 7-7 Sample LOAD job with delimited input

```

/**
//STEP3 EXEC DSNUPROC,UID='JUQBU101.LOAD2 ',TIME=1440,
//      UTPROC=' ',
//      SYSTEM='SSTR ',DB2LEV=DB2A

```

```

//SYSERR DD DSN=JUQBU101.LOAD2.STEP3.SYSERR,
//      DISP=(MOD,DELETE,CATLG),UNIT=SYSDA,
//      SPACE=(4096,(20,20),,,ROUND)
//SYSDISC DD DSN=JUQBU101.LOAD2.STEP3.SYSDISC,
//      DISP=(MOD,DELETE,CATLG),UNIT=SYSDA,
//      SPACE=(4096,(20,20),,,ROUND)
//SYSMAP DD DSN=JUQBU101.LOAD2.STEP3.SYSMAP,
//      DISP=(MOD,DELETE,CATLG),UNIT=SYSDA,
//      SPACE=(4096,(20,20),,,ROUND)
//SYSUT1 DD DSN=JUQBU101.LOAD2.STEP3.SYSUT1,
//      DISP=(MOD,DELETE,CATLG),UNIT=SYSDA,
//      SPACE=(4096,(20,20),,,ROUND)
//UTPRINT DD SYSOUT=*
//SORTOUT DD DSN=JUQBU101.LOAD2.STEP3.SORTOUT,
//      DISP=(MOD,DELETE,CATLG),UNIT=SYSDA,
//      SPACE=(4096,(20,20),,,ROUND)
//SYSIN DD *
LOAD DATA
FORMAT DELIMITED COLDEL ','CHARDEL '"'DECPT '.'
INTO TABLE TBQB0103
(FILENO CHAR,
DATE1 DATE EXTERNAL,
TIME1 TIME EXTERNAL,
TIMESTAMP TIMESTAMP EXTERNAL)
/*
//SYSREC DD *
"001",2000-02-16,00.00.00,2000-02-16-00.00.00.0000
"002",2001-04-17,06.30.00,2001-04-17-06.30.00.2000
"003",2002-06-18,12.30.59,2002-06-18-12.30.59.4000
"004",1991-08-19,18.59.30,1991-08-19-18.59.30.8000
"005",2000-12-20,24.00.00,2000-12-20-24.00.00.0000
/*

```

UNLOAD delimited output functional description

The UNLOAD utility syntax is changed to add the DELIMITED, COLDEL, CHARDEL, and DECPT keywords.

Figure 7-16 on page 192 shows the changes to the UNLOAD syntax.

DELIMITED specifies that the output data file is a delimited file format. In this format, all fields in the output data set are in character string or numeric external form, each column value is separated from the next by a column delimiter character (the default is a comma), and character strings which contain column delimiters are demarcated by character string delimiter characters (the default is a double quote).

COLDEL specifies a single column delimiter character (the default is a comma) that is used in the output file when DELIMITED is specified.

CHARDEL specifies a single character string delimiter character (the default is a double quote) that is used in the output file when DELIMITED is specified. The character string delimiter character is permitted within character string output fields; to delimit character strings that contain the character string delimiter, UNLOAD repeats it. For example:

what a "nice day" day is unloaded as "what a ""nice day"" day"

DECPT specifies a single decimal point character (the default is a period) that is used in the output file when DELIMITED is specified.

UNLOAD delimited output syntax

▷▷ — UNLOAD —▷

UNLOAD-SPEC

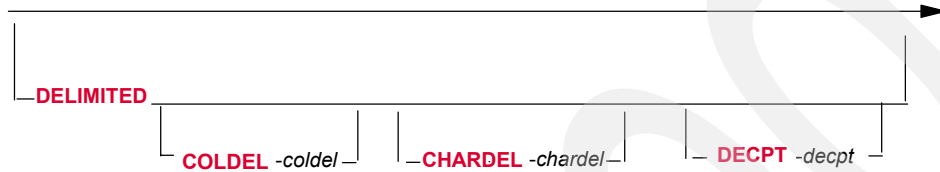


Figure 7-16 UNLOAD delimited output syntax

Example 7-8 shows a sample UNLOAD job with delimited output. In the example the column delimiter is a semicolon, the character string delimiter is a pound sign, the decimal point character is an exclamation point.

Example 7-8 Sample UNLOAD job with delimited output

```

/*
//STEP3      EXEC DSNUPROC,UID='JUQBU105.UNLD1 ',
//              UTPROC=' ',
//              SYSTEM='SSTR'
//UTPRINT    DD SYSOUT=*
//SYSREC     DD DSN=JUQBU105.UNLD1.STEP3.TBQB0501,DISP=(MOD,DELETE,CATLG),
//              UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSPUNCH  DD DSN=JUQBU105.UNLD1.STEP3.SYSPUNCH
//              DISP=(MOD,CATLG,CATLG)
//              UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSUT1     DD DSN=JUQBU105.UNLD1.STEP3.SYSUT1,
//              DISP=(MOD,DELETE,CATLG)
//              UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SORTOUT   DD DSN=JUQBU105.UNLD1.STEP3.SORTOUT,
//              DISP=(MOD,DELETE,CATLG)
//              UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSIN DD *
            UNLOAD TABLESPACE DBQB0501.TSQB0501
            DELIMITED CHARDEL '#'COLDEL ';'DECPT '!'
            PUNCHDDN SYSPUNCH
            UNLDDN SYSREC EBCDIC
            FROM TABLE ADMF001.TBQB0501
            (RECID      POSITION(*) CHAR,
             CHAR7SBCS  POSITION(*) CHAR,
             CHAR7SBIT  POSITION(*) CHAR(7),
             VCHAR20    POSITION(*) VARCHAR,
             VCHAR20SBCS POSITION(*) VARCHAR,
             VCHAR20BIT  POSITION(*) VARCHAR)
/*

```


Delimiter considerations

You should be aware of the following considerations when using the delimited file format:

- ▶ **LOAD:**
 - When the delimited input format is used, field position specifications, if supplied, are ignored. 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 taken from the delimited lengths of each field in the input data set, and all numeric types are assumed to be in external format.
 - For either EBCDIC or ASCII input, if any of the field data type specifications include the MIXED keyword, then the input data is assumed to be in the mixed CCSID specified in the LOAD statement; or if CCSID is omitted, in the mixed CCSID specified at install time. If no field data type specifications include the MIXED keyword then the input data is assumed to be in the single byte CCSID specified in the LOAD statement, or defaults to the system SBCS CCSID.
 - 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.
 - Multiple INTO TABLE statements are not allowed with FORMAT DELIMITED. FORMAT DELIMITED may be used to load only a single table at a time.
- ▶ **UNLOAD:**
 - When DELIMITED is specified for UNLOAD, the NOPAD option is in effect for variable length columns output even if this keyword is not specified by the user. However, if the length is specified for a fixed length (for example, CHAR) column output field, and is longer than the length of the table column, the normal padding rule applies.
 - The default for HEADER is HEADER NONE.
 - HEADER OBID and ROWID are not valid output fields for 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 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 7-3 shows the acceptable data type forms for the delimited file format.

Table 7-3 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	Representation of number in the range of -7.2E+75 7.2E+75 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: that 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.

7.3 Unicode

This enhancement modifies the utility control statement parser to allow Unicode input to utilities, specifically UTF-8 CCSID 1208. You are allowed to provide control statements in the control statement input data sets either entirely in EBCDIC characters or entirely in Unicode characters. This function could be used to execute utilities with stored procedures or via a front end tool.

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 (i.e. a utility comment delimiter)
- ▶ '41'x through '5A'x inclusive — upper case ASCII alphabets

Utility control statement input data sets are those provided to the DSNUTILB program with DD name SYSIN, SYSLISTD or SYSTEMPL or the contents of the UTSTMT field passed to the DSNUTILU stored procedure discussed below.

All output to the SYSPRINT data set and the MVS console continue to be in EBCDIC with translation taking place as required.

The DSNUTILU definition is identical to DSNUTILS with two exceptions:

1. The inputs 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 processed in Unicode as outlined above.
2. The dynamic allocation of data sets is removed. As of V7 this function is performed by the TEMPLATE control statement. In order to eliminate dynamic allocation, the following DSNUTILS keywords are not supported by DSNUTILU for all values of xxxx:
 - UTILITY
 - xxxxDSN
 - xxxxDEVT
 - xxxxSPACE

Figure 7-17 shows the DSNUTILU definition. A new sample C-language caller program DSN8ED8 and a sample job DSNTEJ6R to prepare and execute the caller demonstrate the use of DSNUTILU.

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
NO COLLID
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
EXTERNAL SECURITY USER;
```

Figure 7-17 DSNUTILU definition

7.4 Distribution statistics

With DB2 V7, Runstats does not collect distribution statistics on non-leading indexed columns. Skewed distributions within the data can cause performance problems with DB2 queries, especially in ad hoc query applications. The symptom includes apparently illogical join sequences and excessive use of synchronous I/O, as well as long response times.

When there is asymmetrical distribution of data, not having distribution statistics on non-leading indexed columns and/or non-indexed columns can cause DB2 to make sub-optimal table join order and table join method decisions. This ultimately results in queries which perform inefficiently or do not complete.

Collection of distribution statistics for non-leading indexed columns and/or non-indexed columns ensures that DB2 can use these statistics for better access path selection. Better index selections can be made, when there are screening predicates or there are matching in-list/in-subquery predicates which break up matching equals predicates.

DSTATS (Distribution Statistics for DB2 for OS/390), a downloadable tool, was made available for use with Versions 5, 6, and 7. 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. DSTATS builds the SYSCOLDIST catalog table entries.

With the growth of data warehousing, data mining, and ad hoc query applications, Runstats utility is enhanced in V8 to provide more information to DB2 by collecting additional distribution statistics on columns that would likely be used in a predicate. This enhancement to the Runstats utility eliminates the need to use the DSTATS tool for DB2 V8.

This enhancement is implemented only in Runstats and not in inline statistics. It greatly improves the accuracy of the filter factors determined by DB2. 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.

To summarize, Runstats enhancements provide the following functionalities:

- ▶ 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.

Two subtasks, one to sort the records based on the column group specified and the other to sort the frequency records generated, are used in the execution of RUNSTATS utility.

Figure 7-18 and Figure 7-19 show the changes to the RUNSTATS syntax to collect distribution statistics on non-indexed columns and column correlation statistics on indexed columns.

RUNSTATS - syntax changes

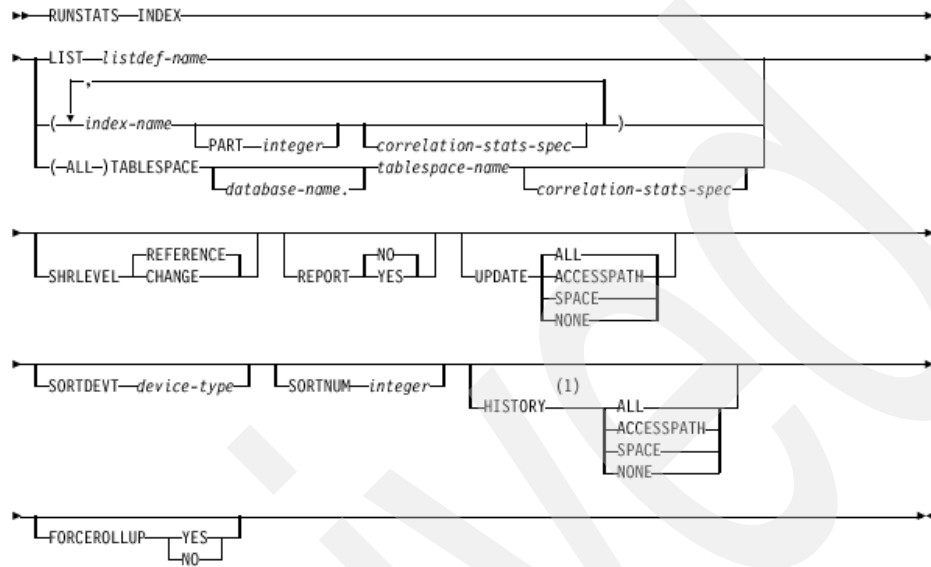


Figure 7-18 RUNSTATS syntax changes

RUNSTATS - Distribution statistics and key correlation statistics blocks

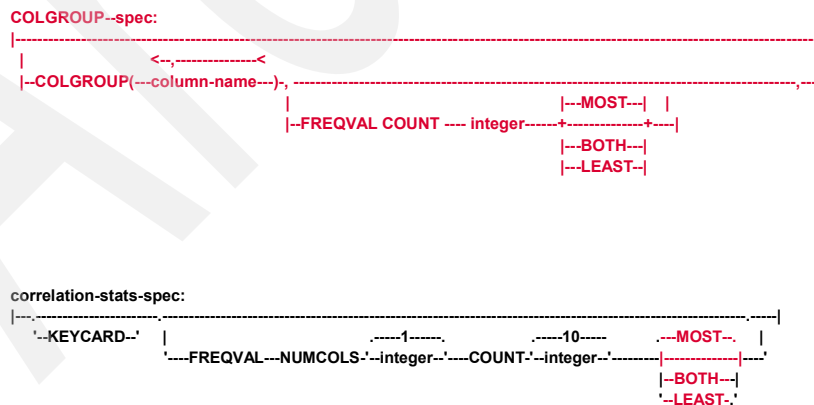


Figure 7-19 RUNSTATS — Distribution statistics and key correlation statistics blocks

7.4.1 Collecting cardinality and distribution statistics

To enable the collection of cardinality and distribution statistics on non-indexed columns, a new dist-spec block is introduced. New keywords COLGROUP, MOST, LEAST, and BOTH are introduced in this block. In addition the existing keywords FREQVAL and COUNT are also 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.

The distribution statistics are collected in SYSCOLDIST and if the table space is partitioned also in SYSCOLDISTSTAT catalog tables. If the COLGROUP keyword is specified, the cardinality statistics are also collected in these tables. Otherwise, the cardinality statistics are collected in SYSCOLUMNS catalog table.

COLGROUP

When the keyword COLGROUP is specified, the set of columns specified with the COLUMN keyword is treated as a group. The cardinality values are collected on the column group. When COLGROUP is not specified, the columns are treated as individual columns and cardinality values are collected on the columns specified in the list.

FREQVAL

Controls the collection of frequent value statistics. These are collected either on the column group or on individual columns depending on whether COLGROUP is specified or not. If FREQVAL is specified, then it must be followed by the keyword COUNT.

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. Although the syntax might suggest the default value is 10, you have to explicitly specify COUNT 10. This can be optionally followed by the keyword MOST (which is the default), 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.

7.4.2 Collecting column correlation statistics

In V7, the keywords FREQVAL, NUMCOLS, and COUNT in the correlation-stats-spec block can be used to collect the frequent value statistics for the specified index. In V8, this block is enhanced to include specification of the keyword MOST ((which is the default), LEAST, or BOTH.

MOST

The most frequent values on the indexed columns are collected when the keyword MOST is specified.

LEAST

The least frequent values on the indexed columns are collected when the keyword LEAST is specified.

BOTH

The most frequent values and the least frequent values on the indexed columns are collected when the keyword BOTH is specified.

7.4.3 Use of work data sets

When RUNSTATS is executed to collect distribution statistics on non-indexed columns, only one SORT subtask is created. The SORT subtask invokes DFSORT (or equivalent) and each instance of sort needs its own sort work data sets and sort message data set.

The DDNAMEs STATWKnn define the sort work data sets used by the utility subtask, where nn identifies one or more data sets to be used by that subtask's invocation of DFSORT (or equivalent). The sort work data sets may be allocated by dynamic allocation or may be allocated by the user with DD statements in the job JCL.

SORTDEVT device-type

SORTDEVT specifies the device type for sort work data sets to be dynamically allocated by SORT. It can be any device type acceptable to the DYNALLOC parameter of the SORT or OPTION options for DFSORT.

You cannot use a TEMPLATE specification to dynamically allocate sort work data sets. Dynamic allocation of these data sets is controlled by the SORTDEVT keyword.

SORTNUM integer

SORTNUM specifies the number of sort work data sets to be dynamically allocated by the sort program. If you omit SORTDEVT, SORTNUM is ignored. If you use SORTDEVT and SORTNUM, no value is passed to DFSORT. If you omit SORTDEVT, SORT is not requested to do dynamic allocation, and whether it does so is governed by its installation options.

7.4.4 Examples

We provide some examples to collect cardinality and distribution statistics for non-indexed columns.

In Example 7-9, by specifying the COLGROUP keyword the cardinality of the column group is collected for the specified group of columns (EMPLEVEL, EMPGRADE, EMPSALARY).

Example 7-9 Distribution statistics — Example 1

```
RUNSTATS TABLESPACE DSN8D71A.DSN8S71E
TABLE(DSN8710.DEPT)
COLGROUP(EMPLEVEL, EMPGRADE, EMPSALARY)
```

In Example 7-10, the 10 most frequent values, and the group column cardinalities are collected for the specified columns (EMPLEVEL, EMPGRADE, EMPSALARY).

Example 7-10 Distribution statistics — Example 2

```
RUNSTATS TABLESPACE DSN8D71A.DSN8S71E
TABLE(DSN8710.DEPT)
COLGROUP(EMPLEVEL, EMPGRADE, EMPSALARY) FREQUAL COUNT 10
```

In Example 7-11, the 15 least frequent values are collected for the specified columns (EMPLEVEL, EMPGRADE, EMPSALARY) as well as the cardinalities for the column group specified in the list.

Example 7-11 Distribution statistics — Example 3

```
RUNSTATS TABLESPACE DSN8D71A.DSN8S71E
TABLE(DSN8710.DEPT)
COLGROUP(EMPLEVEL, EMPGRADE, EMPSALARY) FREQVAL COUNT 15 LEAST
```

In Example 7-12, the 15 most frequent values and the 15 least frequent values are collected for the specified columns (EMPLEVEL, EMPGRADE, EMPSALARY) as well as the group cardinalities for the column group specified in the list.

Example 7-12 Distribution statistics — Example 4

```
RUNSTATS TABLESPACE DSN8D71A.DSN8S71E
TABLE(DSN8710.DEPT)
COLGROUP(EMPLEVEL, EMPGRADE, EMPSALARY) FREQVAL COUNT 15 BOTH
```

In Example 7-13, by specifying the COLUMN keyword the individual cardinalities of the columns are collected. By specifying the COLGROUP keyword the cardinality of the column group is collected, and by specifying the FREQVAL keyword the frequent values are collected. The cardinality is collected for the specified column group (EMPLEVEL, EMPGRADE, EMPSALARY). Also both the 10 most frequent values and the 10 least frequent values are collected for the column group (EMPLEVEL, EMPGRADE, EMPSALARY).

Example 7-13 Distribution statistics — Example 5

```
RUNSTATS TABLESPACE DSN8D71A.DSN8S71E
TABLE(DSN8710.DEPT)
COLUMN(EMPLEVEL, EMPGRADE, EMPSALARY)
COLGROUP(EMPLEVEL, EMPGRADE, EMPSALARY) FREQVAL COUNT 10 BOTH
```

7.5 Backing up and restoring the system

Two new utilities have been introduced with DB2 V8 with the intent to provide a system level, point-in-time level of recovery. This architecture is described at 4.4, “System level point-in-time recovery” on page 62. Please also refer to the figures in that section when reading about these utilities:

- ▶ BACKUP SYSTEM
- ▶ RESTORE SYSTEM

Certain activities are disabled during the backup stage, but by activating through these new DB2 utilities the new functionalities provided within the new z/OS V1R5 by DFSMSHsm, you now have a much easier and less disruptive way for fast volume-level backup and recovery to be used for disaster recovery and system cloning. This function is of great interest for ERP solutions where recovery copies of large number of disk volumes for data and indexes need to be synchronized for application related consistency.

BACKUP SYSTEM

This utility statement invokes DHSMSHsm services to take fast, and (few minutes) minimally disruptive volume copies of all the DB2 data and/or logs. No DB2 QUIESCE point is required, nothing gets stopped as it did during the previous solution which required the SET LOG SUSPEND command.

Two options are available:

- ▶ **BACKUP SYSTEM FULL:** This is the default. It is a full backup which allows recovery of the entire system in later stage, both the *database* and *log* volumes need to be defined. DB2 will backup both database and then all log related data sets (system logs and BSDS).
- ▶ **BACKUP SYSTEM DATA:** This is a data-only system backup; only the *database* volumes needs to be defined for database backup.

A new SMS construct representing a set of SMS volumes is used. Each DB2 system uses two pools of volumes with prescribed DB2 naming convention (30 bytes in length) for:

- ▶ Database data
- ▶ LOG data

A new storage group will hold the volume copies.

When the BACKUP SYSTEM command is issued the 32 KB page writes are disabled, if necessary, as well as data set creation, deletion, rename and extension operations. The PITR locks are acquired in X mode in order to ensure that no restore is taking place on any other member. The BACKUP is serialized with the RESTORE SYSTEM process. The Recover Based Log Point (RBLP) is recorded in DBD01 for use to speed up the logscan process in the RESTORE log apply phase. In data sharing the RBLP LRSN is determined by taking the minimum of all member level RBLP values. Backup of the volumes are done in parallel.

Once the “logical” copies have completed (should typically be within a few seconds), each data sharing member updates the BSDS with the system backup information, while only the submitting member logs BSDS information. Full volume copies of the LOG volumes are taken in case of Backup System Full option. Once terminated, the quiesced activities will resume.

RESTORE SYSTEM

RESTORE SYSTEM, in order to Recover the system to the PIT at which the backup copy was taken, uses the copies from BACKUP SYSTEM FULL and the normal restart/recovery process.

The RESTORE SYSTEM utility is needed to recover the system to an arbitrary PIT. It uses copies from BACKUP SYSTEM FULL or DATAONLY. RESTORE SYSTEM does not restore LOG backup copies, therefore copies from DATAONLY are sufficient.

It consists on two phases:

- ▶ **RESTORE phase:** Recover the database volumes from the latest BACKUP version prior to the arbitrary PIT
- ▶ **LOG APPLY phase:** Apply log records to recover database object to that arbitrary PIT

If the optional parameter LOGONLY is specified, the RESTORE phase is skipped. The assumption here is that the database volumes have been resored outside of DB2.

Prerequisites

In terms of prerequisites, the following is necessary for PITR:

- ▶ z/OS V1R5 or above
- ▶ DASD control units which support ESS FlashCopy APIs
- ▶ DB2 data sets must reside on SMS-managed volumes
- ▶ DB2 V8 must be in 'new function' mode

7.6 Other changes

In this section we mention other miscellaneous utility changes.

7.6.1 New default RESTART

In DB2 V8 you no longer need to add the RESTART or RESTART(PHASE) parameters to the utility jobs. The RESTART is assumed to be the default.

A restartable online utility can now be restarted with or without the RESTART keyword. DB2 determines whether PHASE or CURRENT is used depending on the utility and phase where it failed, unless the RESTART PARM has been explicitly coded.

You should ensure that the data sets have the correct size and disposition to enable restart.

This function is going to be available for DB2 V7 via the PTF for APAR PQ72337, currently open.

7.6.2 New defaults SORTDATA and SORTKEYS

With DB2 V8 two REORG parameters SORTDATA and SORTKEYS (this one also applies to LOAD) are assumed as default.

SORTDATA

The SORTDATA parameter of REORG invokes the execution of a sort on the unclustered data that are columns of the clustering index. This allows consistent execution time, and better performance than the method of accessing the data through the clustering index. And the more data is disorganized, the better it will perform, since the previous default of going through the clustering index will take progressively longer with more disorganized rows.

SORTKEYS

The SORTKEYS keyword is a performance related option which 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

During the index key sort, with SORTKEYS, 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 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 SORTKEYS 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 redbook *DB2 for OS/390 Version 5 Performance Topics*, SG24-2213.

You can reduce the elapsed time of a LOAD job for a table space or partition with more than one defined index by invoking a 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, the DB2 V6, with SORTKEYS specified, also 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. You can therefore 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.

7.6.3 COPY and RECOVER tape parallelism

Parallelism has been enhanced in COPY and RECOVER when tapes are utilized. This enhancement is added to DB2 Utilities Suite V7 via Program Temporary Fixes (PTFs). This new function removes the parallelism support restriction in COPY and RECOVERY when backup copies are on tape. This improves backup and recovery elapsed time performance by permitting parallel processing of backup copies going to or from different tape devices.

Furthermore, this enhancement improves usability and performance by implicitly RETAINing mounted volumes for input data sets used by RECOVER and COPYTOCOPY. This allows dynamic allocation access for data sets stacked onto a tape volume, without causing an unnecessary unload and remount of the tape volume between data set accesses. To obtain this new function, please refer to the following APAR Numbers when ordering from your IBM marketing representative:

- ▶ PQ56295 for DB2 Operational Utilities (5655-E63)
- ▶ PQ56296 for DB2 Diagnostic and Recovery Utilities (5655-E62)
- ▶ PQ56295 and PQ56296 for DB2 Utilities Suite, V7 (5697-E98)
- ▶ PQ56293 for DB2 V7 (5675-DB2)

You can refer to *DB2 UDB for OS/390 and z/OS Version 7 Utility Guide and Reference*, SC26-9945-02 for details.

Archived



Performance

In this chapter we provide a description of the following topics:

- ▶ Comparing unlike data types
- ▶ Materialized query tables
- ▶ Multi-row INSERT and FETCH
- ▶ Cost based parallel sort
- ▶ Data caching and sparse index usage for star join
- ▶ Long and variable length keys
- ▶ Trigger enhancements
- ▶ Reduced lock contention on volatile tables
- ▶ Support for backward index scan
- ▶ Table UDF cardinality option and block fetch

8.1 Comparing unlike 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). 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 like Example 8-1 in a Java application, DB2 cannot use an index on RESOURCE_GROUP because of the mismatch in data type. The data type of the RESOURCE_GROUP column is CHAR, and that of the :hv_res_gr host variable has to be 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.

Example 8-1 Sample SELECT statement

```
SELECT RESOURCE_GROUP,RESOURCE_OPTION,INTVAL,CHARVAL
FROM Q.RESOURCE_TABLE
WHERE RESOURCE_GROUP = :hv_res_gr
```

In DB2 V8, we improve the 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*

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 '<>' cannot be indexable, but can be processed at stage 1).

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.col1 > 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 the following conditions:

Numeric comparisons

All numeric comparisons are stage 1 and indexable except the following:

- ▶ REAL -> DEC(p,s) where p > 15
- ▶ FLOAT -> DEC(p,s) where p > 15

Note that in the comparison notation above (and throughout the rest of this section), 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_colum 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.

String comparisons

In the following sections we 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:

- ▶ 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.

Unlike CCSID string comparisons

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 will be stage 1 but *not* indexable. The reason is all predicates comparing unlike CCSID are evaluated in Unicode encoding scheme.

In all cases involving “column = column” comparisons (same or unlike CCSID) with columns from different tables, the optimizer will consider a merge scan join as a potential access path.

String and date/time/timestamp comparisons

Let us consider predicates comparing date/time/timestamp to string columns, such as:

```
date/time/timestamp -> string column
```

These are stage 2 (residual) if the string column is on the inner table or the “col side” of the predicate.

Note that if the comparison is the other way around, the predicate is stage 1 and indexable:

```
string -> date/time/timestamp
```

Examples

Let us now look at a few examples to illustrate these enhancements.

In the first example, assume that we have a table defined as shown in Example 8-2:

Example 8-2 Employee table definition

```
EMPLOYEE ( NAME CHAR(20),  
           SALARY DECIMAL(12,2),  
           DEPTID CHAR(3) );
```

Figure 8-1 shows how a decimal column type (see SALARY’s definition in Example 8-2) 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, can be indexable. Note that salary has a precision less than 16.

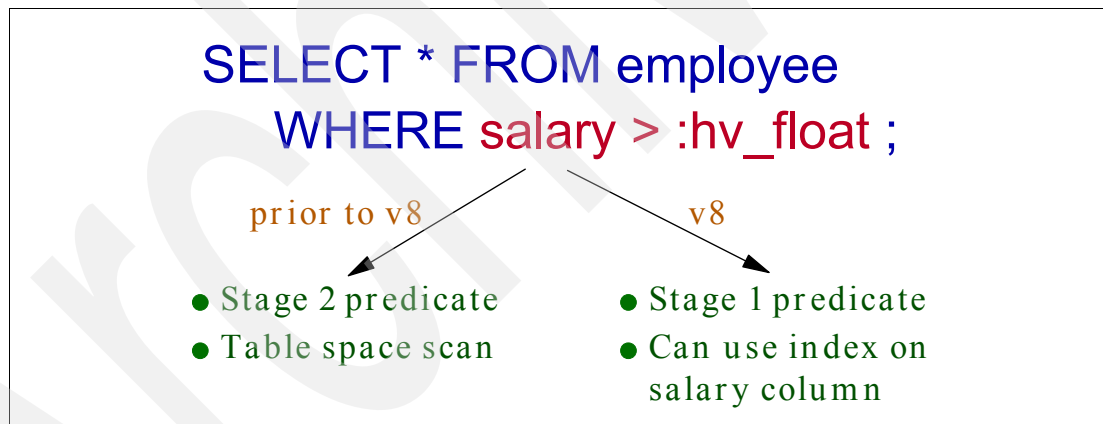


Figure 8-1 Mismatched operands — Numeric type

In Figure 8-2, assume that emp.deptID is defined as CHAR(4). Here we can see that predicate transitive closure is done even though emp.deptID and dept.id are of different lengths. The new predicate is stage 1 and can be indexable. The generated parameter marker has the same size as the parameter marker in the emp.deptID predicate.

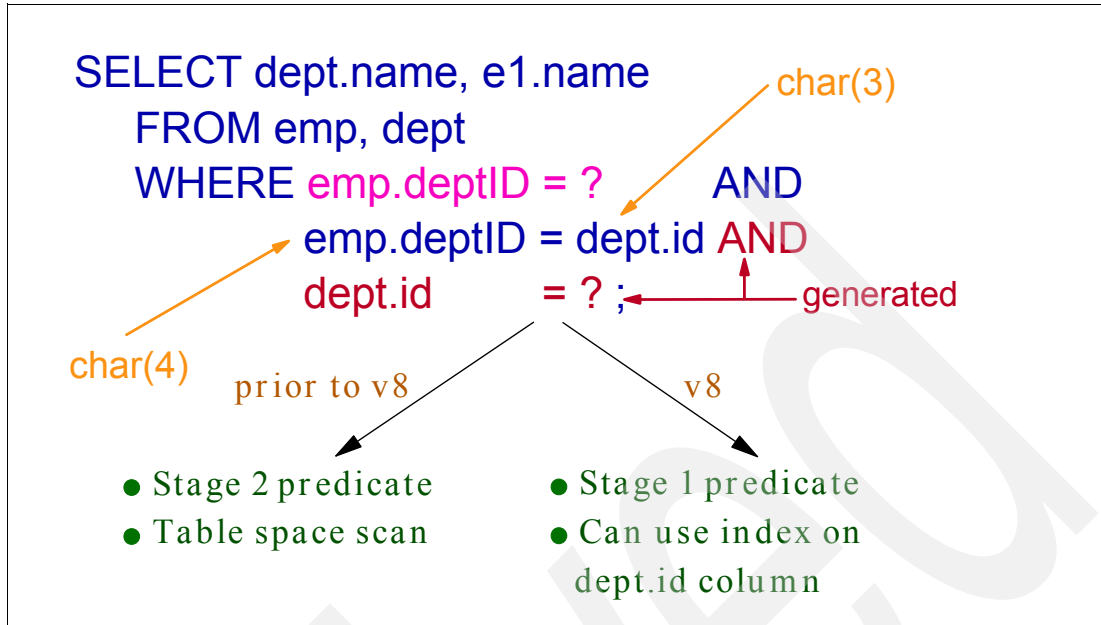


Figure 8-2 Mismatched operands — Transitive closure

8.2 Materialized query tables

The nature of queries in a data warehouse is to access a significant amount of rows of very large fact tables joined with one to several dimension tables, sometimes billions of rows. A typical query selects based on dimensions, aggregates on a few dimension columns, and applies column functions on the records of interest. A common access path for these types of queries is via a star join access path. Due to the large amount of data to be processed, these queries can take up to many hours of elapsed time to process. In order to improve the performance and reduce the elapsed time of these queries, we can either use parallelism or somehow save (precompute and materialize) the results of prior queries and reuse these common query results for subsequent queries. With DB2 V8 we can now materialize and save these results for later use and avoid recomputation of the same (complex) result thus reducing the elapsed time from hours down to minutes or seconds. This method we call *Materialized Query Tables (MQTs)*.

You should expect to see a reduction in elapsed time for queries which can make use of materialized query tables. Queries which reference tables on which materialized query tables are defined may see increased BIND time due to the catalog accesses and processing during the automatic query rewrite phase.

The design and use of materialized query tables involves trade-off between conflicting design objectives. On one hand, MQTs that are specialized to a particular query or 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 MQTs can be expensive to define and keep current, this approach can be expensive.

On the other hand, MQTs that are more general purpose, and that support a large number of submitted queries will often tend to provide less performance improvement. Since there are fewer of these, the maintenance of the MQTs will be reduced.

In order to exploit MQTs, you have to:

- ▶ Create the MQT
- ▶ Populate the MQT
- ▶ Enable the MQT for query optimization

8.2.1 Creating an MQT

As we mentioned previously, a materialized query table contains pre-computed data. The pre-computed data is the result of a query, that is a fullselect associated with the table.

You can either:

- ▶ Create an MQT from scratch using the CREATE TABLE statement
- ▶ Change an existing table into an MQT using the ALTER TABLE statement

Creating an MQT from scratch

The CREATE TABLE statement syntax has been enhanced to allow you to create an MQT. The new syntax is shown in Figure 8-3.

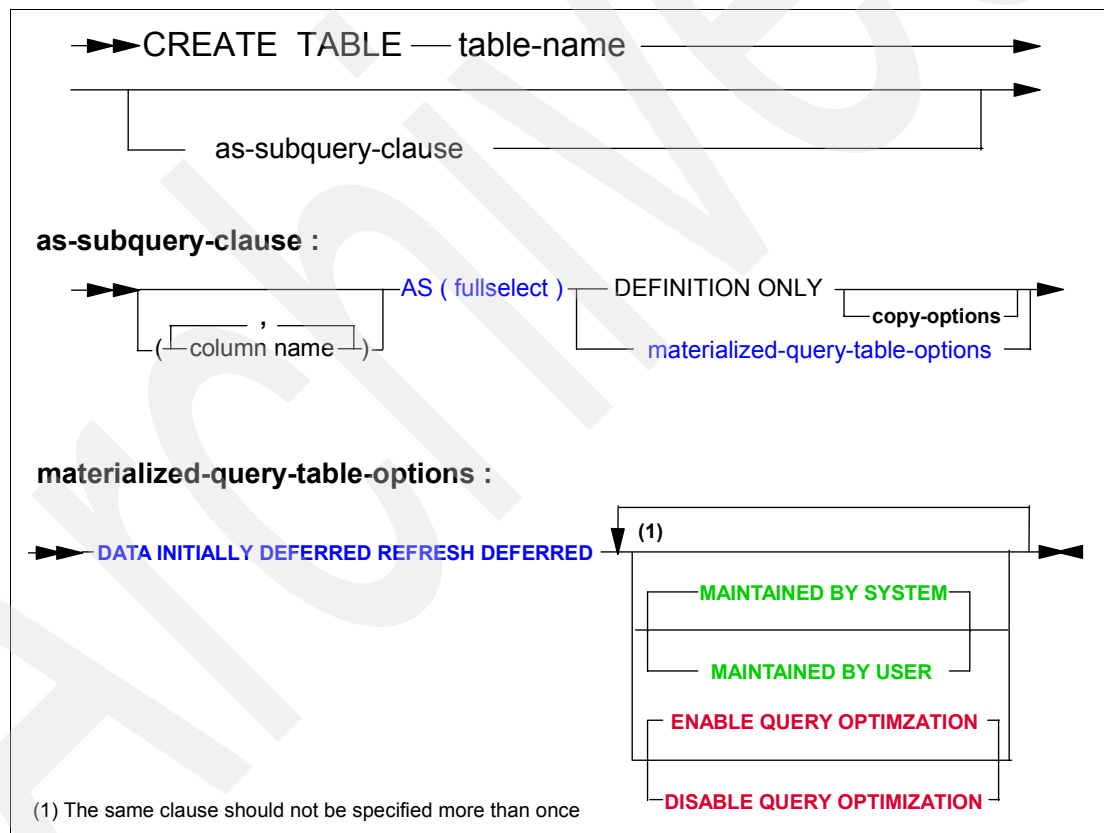


Figure 8-3 CREATE MQT syntax

Creating a materialized query table is similar to creating a view. In both cases you specify a fullselect to define its contents. The difference is that a view is only a logical definition, while a materialized query table contains materialized data of the query result on disk. For this reason, a materialized query table is also called a *Materialized View*. In Example 8-3 we see a CREATE TABLE statement to create a materialized query table.

Example 8-3 Sample create of a materialized query table

```
CREATE TABLE TRANSCNT (ACCTID, LOCID, YEAR, CNT)
  AS ( SELECT ACCTID, LOCID, YEAR, COUNT(*)
        FROM TRANS
        GROUP BY ACCTID, LOCID, YEAR )
  DATA INITIALLY DEFERRED
  REFRESH DEFERRED
  MAINTAINED BY SYSTEM
  ENABLE QUERY OPTIMIZATION;
```

The fullselect, together with the DATA INITIALLY DEFERRED and REFRESH DEFERRED clauses, defines the table as a materialized query table.

The column names of a materialized query table can be explicitly specified or be derived from the fullselect associated with the table.

You can specify most fullselects for a materialized query table if the materialized query table is disabled for the automatic query rewrite (the DISABLE QUERY OPTIMIZATION clause is specified). However, if a materialized query table is enabled for automatic query rewrite (by default or the ENABLE QUERY OPTIMIZATION clause is specified), there are more restrictions on what kind of fullselects can be specified.

Registering existing tables as MQT

Many customers have already use base tables as a form of materialized query tables in their systems. With DB2 V8, they may want to take advantages of the automatic query rewrite for these tables by registering these existing tables as materialized query tables. This can be achieved by using the extension of the ALTER TABLE statement for materialized query tables.

The statement in Example 8-4, registers a table TRANSCOUNT as a materialized query table with the associated subselect to DB2. The data in the table will remain the same as indicated by DATA INITIALLY DEFERRED, and will still be maintained by the user, as specified by the MAINTAINED BY USER clause. The user can continue to LOAD, INSERT, UPDATE, or DELETE data in the table TRANSCOUNT. ALTER TABLE can also change a materialized query table into a base table.

Example 8-4 Converting a base table into an MQT

```
ALTER TABLE TRANSCOUNT ADD MATERIALIZED QUERY
  (SELECT ACCTID, LOCID, YEAR, COUNT(*) as cnt
   FROM TRANS
   GROUP BY ACCTID, LOCID, YEAR)
  DATA INITIALLY DEFERRED
  REFRESH DEFERRED
  MAINTAINED BY USER;
```

The ALTER TABLE statement can be used to enable (the default) or disable an existing materialized query table for consideration by automatic query rewrite. Altering a table to change it to a materialized query table with query optimization enabled makes the table eligible for use in query rewrite immediately. Therefore, pay attention to the accuracy of the data in the table. If necessary, the table should be altered to a materialized query table with query optimization disabled, and then the table should be refreshed and enabled with query optimization.

You can also switch materialized query table types between system-maintained and user-maintained with the ALTER TABLE statement.

The isolation level at the time when a base table is first altered to become a materialized query table by the ALTER TABLE statement is the isolation level for the materialized query table.

8.2.2 Populating and maintaining an MQT

The time when a materialized query table is populated with the pre-computed data depends on the definition of DATA INITIALLY DEFERRED or REFRESH DEFERRED.

DATA INITIALLY DEFERRED means that when a materialized query table is created, the materialized query table will not be populated by the result of the query.

REFRESH DEFERRED means the data in the materialized query table is not refreshed immediately when its base tables are updated. However the data can be (manually) refreshed at any time, for example by using the REFRESH TABLE statement.

Important: With DB2 UDB for z/OS Version 8, all MQTs have to be defined as DATA INITIALLY DEFERRED or REFRESH DEFERRED. This means the user has to ensure that the data currency meets the user requirements to avoid using outdated data and that the user is responsible to keep the data in the MQT up to date.

The MAINTAINED BY option indicates how the data in the MQT is to be refreshed:

- ▶ **MAINTAINED BY SYSTEM**, which is the default, indicates that the materialized query table is system-maintained. The only way to refresh the data in a system-maintained MQT is by using the **REFRESH TABLE** statement. A system-maintained materialized query table cannot be updated by using the LOAD utility, INSERT, UPDATE or DELETE SQL statements. Therefore, a system-maintained materialized query table is read-only. If a view or a cursor is defined on a system-maintained materialized query table, it becomes read-only. Since the REFRESH TABLE statement is implemented by using DELETE, INSERT, and UPDATE, materialized query tables are transparent to the DATA CAPTURE and AUDIT options.
- ▶ Alternatively, **MAINTAINED BY USER** can be specified to define a user-maintained materialized query table. A user-maintained materialized query table can be updated by the LOAD utility, INSERT, UPDATE or DELETE SQL statements, as well as the REFRESH TABLE statement. Therefore, a user-maintained materialized query table is updatable.

Note: With DB2 UDB for Linux, UNIX, and Windows the REFRESH TABLE is only allowed on system-maintained tables.

REFRESH TABLE

The REFRESH TABLE statement can be used to populate a materialized query table. See Example 8-5 for a sample REFRESH TABLE statement.

Example 8-5 Sample REFRESH TABLE statement

```
REFRESH TABLE mq_table;
```

The REFRESH TABLE statement:

1. Deletes all rows in the materialized query table.

2. Executes the fullselect in the materialized query table definition to recalculate the data from the tables specified in the fullselect with the isolation level for the materialized query table (as recorded in the catalog).
3. Inserts the calculated result into the materialized query table.
4. Updates the catalog for the refresh timestamp and cardinality of the materialized query table.

Even though the REFRESH TABLE statement involves delete and insert, it is a single commit scope. All changes made by the REFRESH TABLE statement are logged. Note that because the REFRESH TABLE statement uses a mass delete (DELETE FROM table-name), the performance of REFRESH TABLE will be better if the materialized query table is stored in a segmented table space.

The REFRESH TABLE statement is an explainable statement. The explain output contains rows for INSERT with the fullselect in the materialized query table definition.

8.2.3 Automatic query rewrite using Materialized Query Tables

The good thing about MQTs is that the optimizer understands them. In your queries, you always reference the base table. During access path selection, the optimizer will take a look at your query, and determine whether or not your table can be replaced by an MQT to reduce the query cost.

The process of recognizing when a materialized query table can be used in answering a query, deciding whether one or more materialized query tables should actually be used in answering a query, and rewriting the query accordingly, is done by a DB2 function called *automatic query rewrite (AQR)*.

Automatic query rewrite is based on the fact that the submitted query may share a number of common sub-operations specified in the fullselect of a materialized query table definition. Therefore, the result of the submitted query can be derived from or can directly use the result of one or more materialized query tables. In other words, the automatic query rewrite process analyzes the user query to see if it can take advantage of any of the existing materialized query tables, by “proving” that the contents of a materialized query table overlaps with the content of a query, and compensating for the non-overlapping parts. When such an overlap exists, the query and the materialized query table are said to match. After discovering a match, the query is rewritten to access the matched materialized query table instead of one or more source tables, originally specified in the query.

The automatic query rewrite process searches for matched materialized query tables that result in an access path with the lowest cost after rewrite. The costs of the rewritten query and the original query are compared and the one with the lowest cost is chosen. If the final query plan comes from a rewritten query, the PLAN_TABLE will show the name of the matched materialized query table(s) and the access path using the materialized query table(s). For more information on the information about MQTs in the PLAN_TABLE, see 8.2.4, “Determining if query rewrite occurred” on page 220. No authorization on a materialized query table is required for it to be used in automatic query rewrite.

There are a number of options and settings that affect whether or not an MQT will be considered by AQR. They can be grouped into the following categories:

- ▶ DDL options
- ▶ Special registers
- ▶ Query properties
- ▶ Database design properties

DDL options

You can specify the following options on the CREATE or ALTER TABLE statement that affect whether or not DB2 will consider an MQT during automatic query rewrite.

- ▶ *ENABLE QUERY OPTIMIZATION*, which is the default, specifies that this materialized query table can be exploited by automatic query rewrite. The other choice is *DISABLE QUERY OPTIMIZATION*.
- ▶ When the *DISABLE QUERY OPTIMIZATION* clause is specified, the materialized query table will not be considered by the automatic query rewrite process.

In addition, the optimizer (automatic query rewrite) will only use a system-maintained MQT if a REFRESH TABLE has occurred. When using user-maintained MQTs, you may wish to create the MQT with the *DISABLE QUERY OPTIMIZATION* option, and ALTER it later to *ENABLE QUERY OPTIMIZATION*, once the table has been (re)populated.

Special registers

The new special registers *CURRENT REFRESH AGE* and *CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION* also control whether or not a materialized query table can be considered by automatic query rewrite for a dynamically prepared query.

- ▶ *CURRENT REFRESH AGE*. The value in this special register represents a refresh age. The refresh age of a materialized query table is the time between the current timestamp and the time that the MQT was last refreshed using the REFRESH TABLE statement. (The latter information is recorded in the REFRESH_TIME column of the SYSVIEWS system catalog table.)

In DB2 V8, only *CURRENT REFRESH AGE* of 0 or ANY is supported:

- 0 means only materialized query tables that are kept current with the source tables are considered by automatic query rewrite. Since DB2 V8 does not support immediately refreshed MQTs, specifying 0 means that AQR will not consider any MQTs.
- ANY represents the maximum duration, meaning all materialized query tables are considered by automatic query rewrite.

A subsystem default value for *CURRENT REFRESH AGE* can be specified in the *CURRENT REFRESH AGE* field on panel DSNTIP4 at installation time, DSNZPARM REFSHAGE.

- ▶ Besides the REFRESH TABLE statement, user-maintained materialized query tables can be updated using INSERT, UPDATE, or DELETE SQL statements, or via the LOAD utility. Therefore, the refresh age of a user-maintained materialized query table can no longer truly represent the freshness of data in the MQT. Hence, the new special register *CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION* is used to determine which type of materialized query tables, system-maintained materialized query tables or user-maintained materialized query tables, are considered by automatic query rewrite.
 - **ALL** Indicates that all MQTs will be considered by automatic query rewrite.
 - **NONE** Indicates that no materialized query tables will be considered.
 - **SYSTEM** Indicates that only system-maintained materialized query tables (that are refresh deferred) will be considered.
 - **USER** Indicates that only user-maintained materialized query tables that are refresh deferred will be considered.

A subsystem default value for *CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION* can be specified in the *CURRENT MAINT TYPES* field on panel DSNTIP4 at installation time, DSNZPARM MAINTYPE.

Materialized query tables created or altered with `DISABLE QUERY OPTIMIZATION` specified are not eligible for automatic query rewrite, thus, are not affected by the above special registers. If a system-maintained materialized query table has not been populated with data, then the materialized query table 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.

Query properties

Automatic query rewrite is only supported for *read-only dynamically prepared queries*. AQR is not supported for statically bound queries.

However, a materialized query table can be “used manually”, that is selecting directly from the MQT instead of the base table, if appropriate, in either a statically bound query or a dynamically prepared query to improve the response time. The users must be aware of the freshness of data in the materialized query table and refresh or update it when necessary.

In addition, only materialized query tables with an *isolation level* equal to or higher than the query isolation level will be considered in query optimization.

The user query may contain multiple query blocks, for example, subselect of UNION or UNION ALL, temporarily materialized views, materialized table expressions, and subquery predicates. In general, automatic query rewrite is considered at the query block level. When a query block is considered for automatic materialized query table rewrite, it has to adhere to a set of rules. It *must not be or contain any of the following*:

- ▶ A fullselect in the UPDATE SET statement
- ▶ A fullselect in the INSERT statement
- ▶ A fullselect in the materialized query table definition in the REFRESH TABLE statement
- ▶ An outer join
- ▶ The predicate contains user-defined scalar or table functions with the EXTERNAL ACTION or NON-DETERMINISTIC attribute, including built-in function RAND

Otherwise, automatic query rewrite will not be attempted and the query block is processed using the normal optimization options provided today.

Database design properties

Referential constraints between the base tables specified in materialized query table definitions are important in automatic query rewrite in determining whether or not a materialized query table can be used in answering a query when the definition of the MQT contains extra tables that are not referenced by the query. For this reason, *informational referential constraints* are introduced. For these constraints, DB2 relies on the user application to enforce the constraints. DB2 ignores informational referential constraints during INSERT, UPDATE and DELETE processing. This way you avoid the overhead of enforcing the referential constraints by DB2. The DB2 LOAD and CHECK DATA utilities also ignore informational referential constraints. However, DB2 will employ both types of referential constraints in query optimization using materialized query tables.

Informational referential constraints can be specified in both the CREATE TABLE and ALTER TABLE statements for base tables, by using the *NOT ENFORCED* keyword, as shown in Example 8-6.

Example 8-6 Creating a informational RI constraint

```
CREATE TABLE 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 ACCT NOT ENFORCED,
        CONSTRAINT LOC_ACCT FOREIGN KEY (LOCID)
            REFERENCES LOC NOT ENFORCED
    )
IN DBND0101.TLND0101;

```

Informational constraints are not enforced by DB2, as mentioned before, and are also ignored by most DB2 utilities. However, they are exploited by the QUIESCE and REPORT TABLESPACESET utility, as well as by the LISTDEF RI utility control statement.

Since informational RI constraint are not enforced by the DBMS, the applications have to take care of enforcing them.

You can still take advantage of using informational RI, even when not using MQTs. You can use informational RI to document the relationships between tables. When application RI exists, you normally still want to make a backup or take a quiesce point for all application related RI tables, just as would for “normal” DB2 RI related tables. Now that the system allows you to define informational RI constraints, you can use these for that purpose, since QUIESCE TABLESPACESET and LISTDEF RI will take informational RI into account and quiesce or generate a list of informational RI related tables.

Examples

The examples below demonstrate how automatic query rewrite is applied to queries to use materialized query tables. Figure 8-4 shows the sample database schema. The schema consists of seven tables that represent the data of a simplified credit card application. The CUST table describes the credit card holders (customers). The ACCT table stores the corresponding credit card accounts.

It is assumed that each account has a single customer (but each customer can have many credit card accounts). This is indicated in Figure 8-4 by the N:1 arrow between ACCT.CUSTID and CUST.ID, which designates an informational referential integrity constraint between those two columns. The TRANS table records the set of credit card transactions. It is assumed that each transaction consists of a set of products that are purchased together, and this information is stored in the TRANSITEM table. TRANS and TRANSITEM are the fact tables of the schema, as they are both large and are continuously updated as new transactions are performed.

In addition to these two fact tables, the schema contains three hierarchical dimensions as well that further describe a transaction. The product dimension is recorded in two normalized tables, PGROUP and PLINE, representing the product group and product line respectively. The location dimension contains CITY, STATE, and COUNTRY and is represented by a single, denormalized table (LOC). The time dimension contains DAY, MONTH, and YEAR and is located in the TRANS table.

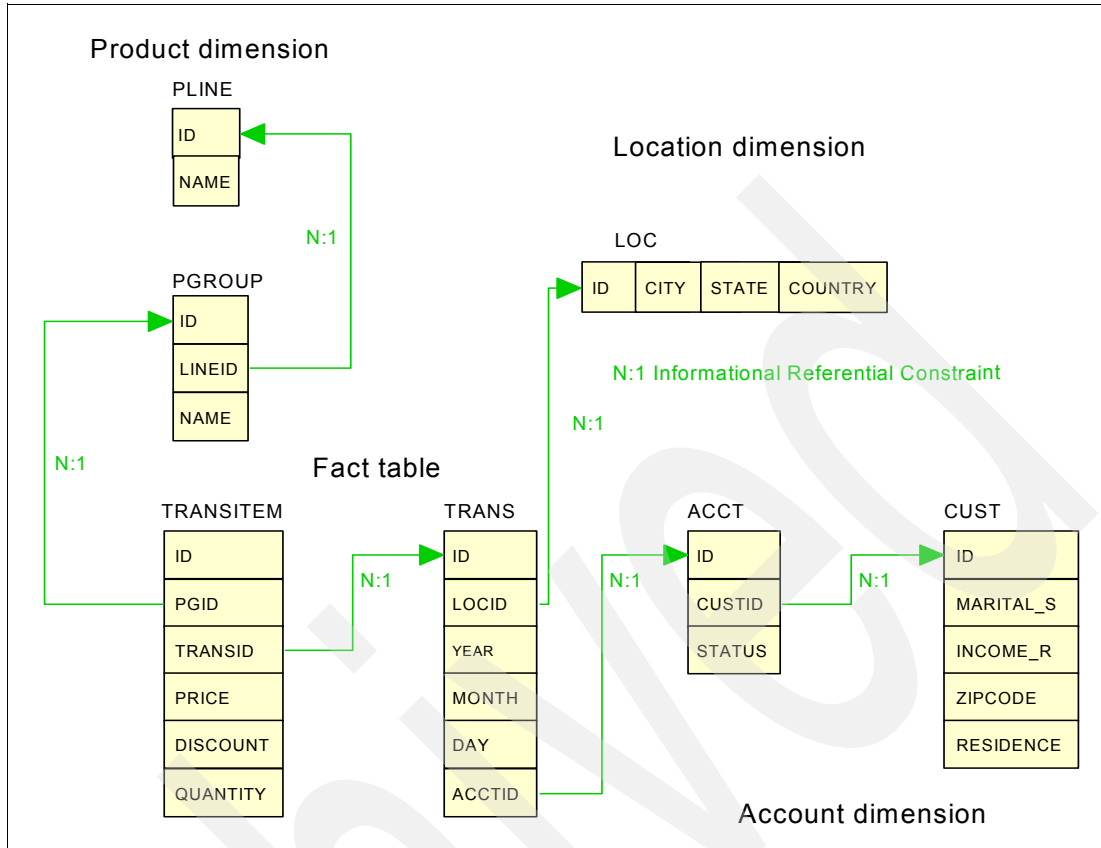


Figure 8-4 Credit card application schema

Example 1

An analyst of such a credit card application may be interested in the aggregation of the sales data for the different dimensions and at different levels of the hierarchy for each dimension. Queries may typically perform joins of one or more dimension tables with fact tables according to the referential integrity relationships shown in the figure. For example, the query UserQ1 in Example 8-7, counts the number of transactions performed in the USA for each credit card account, state, and year.

Example 8-7 UserQ1

```
SELECT T.ACCTID, L.STATE, T.YEAR, COUNT(*) AS CNT
FROM TRANS T, LOC L
WHERE T.LOCID = L.ID AND
      L.COUNTRY = 'USA'
GROUP BY T.ACCTID, L.STATE, T.YEAR;
```

Assume we define an MQT TRANSCNT as shown in Example 8-8.

Example 8-8 MQT TRANSCNT

```
CREATE TABLE TRANSCNT
AS (SELECT ACCTID, LOCID, YEAR, COUNT(*) AS CNT
    FROM TRANS
    GROUP BY ACCTID, LOCID, YEAR
   )
DATA INITIALLY DEFERRED
REFRESH DEFERRED;
```

Then, assuming the MQT is populated and automatic query rewrite is enabled, DB2 can rewrite UserQ1 as NewQ1, shown in Example 8-9, which accesses the TRANSCNT MQT instead of the TRANS fact table.

Example 8-9 NewQ1

```
SELECT A.ACCTID, L.STATE, A.YEAR, SUM(A.CNT) AS CNT
   FROM TRANSCNT A, LOC L
  WHERE A.LOCID = L.ID      AND
        L.COUNTRY = 'USA'
  GROUP BY A.ACCTID, L.STATE, A.YEAR;
```

Given that customers typically perform a few hundred transactions per year and most of them are within the same city, TRANSCNT is about a hundred times smaller than the TRANS table. Therefore, using TRANSCNT instead of TRANS in can improve the response time of the UserQ1 query significantly.

Example 2

Example 8-10 shows how the columns in the select list, and predicates can be matched between the query and the fullselect in the materialized query table definition. Query UserQ2 selects some important transactions, such as TV items with a price over 100 and a discount greater than 0.1 that were purchased by a credit card account.

Example 8-10 UserQ2

```
SELECT T.ID, TI.QUANTITY * TI.PRICE * (1 - TI.DISCOUNT) AS AMT
   FROM TRANSITEM TI, TRANS T, PGROUP PG
  WHERE TI.TRANSID = T.ID      AND
        TI.PGID = PG.ID      AND
        TI.PRICE > 100      AND
        TI.DISCOUNT > 0.1 AND
        PG.NAME = 'TV';
```

Assuming the following MQT, called TRANSIAB, is defined (see Example 8-11), DB2 can rewrite UserQ2 as NewQ2 (shown in Example 8-12).

Example 8-11 TRANSIAB MQT

```
CREATE TABLE TRANSIAB
   AS (SELECT TI.TRANSID, TI.PRICE, TI.DISCOUNT, TI.PGID,
            L.COUNTRY, TI.PRICE * TI.QUANTITY as VALUE
      FROM TRANSITEM TI, TRANS T, LOC L
     WHERE TI.TRANSID = T.ID AND
           T.LOCID = L.ID      AND
           TI.PRICE > 1      AND
           TI.DISCOUNT > 0.1
    )
  DATA INITIALLY DEFERRED
  REFRESH DEFERRED;
```

Example 8-12 NewQ2

```
SELECT A.TRANSID, A.VALUE * (1 - A.DISCOUNT) as AMT
   FROM TRANSIAB A, PGROUP PG
  WHERE A.PGID = PG.ID AND
        A.PRICE > 100 AND
        PG.NAME = 'TV';
```

DB2 can rewrite UserQ2 as the NewQ2 query that uses the TRANSIAB MQT because of the following reasons:

- ▶ Although the predicate T.LOCID = L.ID only appears in the MQT, DB2 still considers the MQT for query rewrite because the predicate does NOT result in rows being discarded. The referential constraint that exists between the TRANS.LOCID and LOC.ID columns makes the join between TRANS and LOC in the subselect “lossless”, provided that the foreign key in the constraint is defined as NOT NULL.
- ▶ The predicates TI.TRANSID = T.ID and TI.DISCOUNT > 0.1 appear in both the query and the TRANSIAB fullselect.
- ▶ The query predicate TI.PRICE > 100 references a column that is also produced by the TRANSIAB fullselect. The TRANSIAB fullselect specifies TI.PRICE > 1. Therefore, all rows where TI.PRICE > 100 are also included in the MQT. The predicate PG.NAME='TV' references a table which is not in the TRANSIAB fullselect. Both predicates (TI.PRICE > 100 and PG.NAME='TV') must be part of the rewritten query NewQ2, and both must be computable from either TRANSIAB columns, or from the residual table which is not in the TRANSIAB fullselect.
- ▶ The columns in the select list of the query result can be derived from the materialized query table definition, although that may not be readily apparent:
 - T.ID in the query is derived from the TI.TRANSID column in the fullselect. Although these two columns originate from different tables (TRANS and TRANSITEM respectively), they are in fact equivalent because of the TI.TRANSID = T.ID predicate. Such column equivalency is recognized through join predicates and thus T.ID can be derived from TI.TRANSID.
 - AMT in the query is derived from the DISCOUNT and VALUE in the TRANSIAB fullselect.

Example 3

Example 8-13 shows how the GROUP BY items and column functions can be matched between the query and fullselect in the materialized query table definition. Query UserQ3 selects the average value for transaction items for each year.

Example 8-13 UserQ3

```
SELECT YEAR, AVG(QUANTITY * PRICE) AS AVGVAL
FROM TRANSITEM TI, TRANS T
WHERE TI.TRANSID = T.ID
GROUP BY YEAR;
```

Assuming that the following materialized query table TRANSAVG is defined (see Example 8-14), DB2 can rewrite UserQ3 as NewQ3 (shown in Example 8-15).

Example 8-14 TRANSAVG MQT

```
CREATE TABLE TRANSAVG
AS (SELECT T.YEAR, T.MONTH,
        SUM(QUANTITY * PRICE) AS TOTVAL,
        COUNT(QUANTITY * PRICE) AS CNT
FROM TRANSITEM TI, TRANS T
WHERE TI.TRANSID = T.ID
GROUP BY T.YEAR, T.MONTH
)
DATA INITIALLY DEFERRED
REFRESH DEFERRED;
```

Example 8-15 NewQ3

```
SELECT YEAR, CASE WHEN SUM(CNT) = 0 THEN NULL
                ELSE SUM(TOTVAL)/SUM(CNT)
                END AS AVGVAL
FROM TRANSAVG
GROUP BY YEAR;
```

DB2 can rewrite UserQ3 as NewQ3 that uses the TRANSAVG MQT for the following reasons:

- ▶ YEAR in the two SELECT lists are considered to be matched exactly.
- ▶ The AVG function in NewQ3 is derivable from two column functions SUM and COUNT in the TRANSAVG MQT, even when the MQT is at the monthly level and the query requests information at the yearly level. However, the monthly SUM and COUNT information is enough to derive a yearly average.
- ▶ The two GROUP BY lists are considered matches as well because the GROUP BY in the query NewQ3, requests data at a higher level than that in the definition of TRANSAVG.
- ▶ In the NewQ3 query, AVGVAL is calculated differently compared with UserQ3's AVG(QUANTITY * PRICE) calculation. It is now derived from the TRANSAVG's two columns CNT and TOTVAL, using a case expression.

8.2.4 Determining if query rewrite occurred

You can use the SQL EXPLAIN statement to determine if DB2 has rewritten a user query to use a materialized query table. When DB2 rewrites the query, the PLAN TABLE shows the name of the materialized query being used in the TNAME column, instead of the table you specified in the query, and the value of the TABLE_TYPE column is set to "M", to indicate that the table in the TNAME column is a materialized query table.

This information is also available in mini-plan performance trace record (IFCID 0022).

8.3 Multi-row INSERT and FETCH

Prior to DB2 V8, a user has to execute a separate SQL FETCH statement for each row of data that the application requires from the database. Likewise, if an application needs to insert several rows, that application has to execute a separate SQL INSERT statement for each row being stored into the database.

For local processing, the execution cost consists of the multiple trips between the application and the database engine. For distributed applications, the performance cost consists of multiple trips into the database engine plus the network cost to send each request. In some cases, block fetching mitigated the network costs for the FETCH statement but not for INSERT statements.

Now, in DB2 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 set. For example, you may fetch the next rowset, or update the current rowset. The program or application controls how many rows are returned on a single FETCH statement. Fetching multiple rows of data can be done with both normal and scrollable cursors. New syntax on the FETCH statement allows specification of the number of rows to be returned in the rowset.

The INSERT statement can now also insert one or more rows into a table or view with one SQL statement. There are two forms of multiple-row INSERT: a static, and a dynamic form.

This reduces the number of trips between the application and the database engine, as well as reduces the number of network trips required for multiple fetch or insert operations for distributed requests. For some applications this can have a dramatic performance improvement. For details on how to use this new function, see the section 5.5, “Multi-row fetch and insert” on page 84.

8.4 Cost based parallel sort

In data warehousing environments, it is often good practise 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 was not possible to fully utilize the CPU when parallel sort was involved. This was mainly due to the fact that *sort-composite* was not pushed down for parallelism if the composite involves more than one 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 exploited and elapsed time can be reduced. 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.

Let us assume that the access path for the query in Example 8-16 (a 3 table join) uses two merge scan joins, also known as sort merge join (SMJ):

Example 8-16 Sample SQL statement

```
SELECT * from T1 , T2 , T3 where a2 = b2 and b3 = c3;
```

Prior to DB2 V8, the “sort composite” for SMJ2 (output of SMJ1 involving T1 and T2 and input to SMJ2), 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.

8.5 Data caching and sparse index usage for star join

For background information on star join refer to the redbook *DB2 UDB Server for OS/390 Version 6 Technical Update*, SG24-6108 and the whitepaper *The Evolution of Star Join Optimization* available from the Web site:

<http://www.ibm.com/software/data/db2/os390/techdocs/starjoin.pdf>

The performance of star join is critical to data warehousing applications where the star schema is one of the main database designs. The star join implementation in DB2 for z/OS has to deal with potentially a large number of work files because:

- ▶ Snowflakes that appear in a star join query are materialized as workfiles.
- ▶ When the pushdown star join method is used in the access path ('S' in the JOIN_TYPE column of PLAN_TABLE), the dimension tables joined before the fact table are sorted and stored as workfiles.

If one or more snowflake workfiles are joined after the fact table, the sort merge join tends to be selected as the join method, because the workfiles do not have indexes. In this case, the cost of the sorting can be large, both in time and space.

For the pushdown star join method, even with the index key feedback and repositioning mechanism (JOIN_TYPE = 'S' in the PLAN_TABLE), the dimension workfiles joined before the fact table may be scanned many times. This can cause a high getpage count on the workfile buffer pool and, possibly, many physical page reads.

In DB2 V8, an in-memory virtual index may be created on each workfile created for a star join query. By default, the records of the entire workfile will be cached along with the join keys in a dedicated storage pool (created above the 2 GB bar). The maximum size of the pool is determined by a new installation parameter SJMXPOOL. The SJMXPOOL parameter is in effect only when the STARJOIN parameter is set to enable. This global pool is shared by multiple DB2 threads that execute star join queries.

When a star join query is executed, DB2 tries to allocate the space required to cache a work file in the pool. If the allocation of a block is successful, the sorted records for the workfile are cached in-memory and the physical workfile will not be created. This technique is called *data caching*. The key field is prepended to the record and serves as the (virtual) index. A binary search is used to find a match. When data caching is used, the keys are always dense, not sparse (see next paragraph), that is, the entire work file content is always cached.

If the pool is not created, or a block allocation failed because no more space is available in the pool, only the key data is saved in-memory as a *sparse index*, and the data records are stored in a physical workfile. A sparse index is a dynamically built index, pointing to a single value or a range of values, depending on the number of keys that can be stored in a pre-allocated space. (Sparse indexes were first introduced in DB2 V4, to improve non-correlated IN subquery performance.)

The decision to use this enhancement (both sparse indexing as well as data caching) is made by the optimizer, based on the estimation of the costs of the available access paths.

Data caching is applicable to any work file created for a star join query. Figure 8-5 illustrates the process.

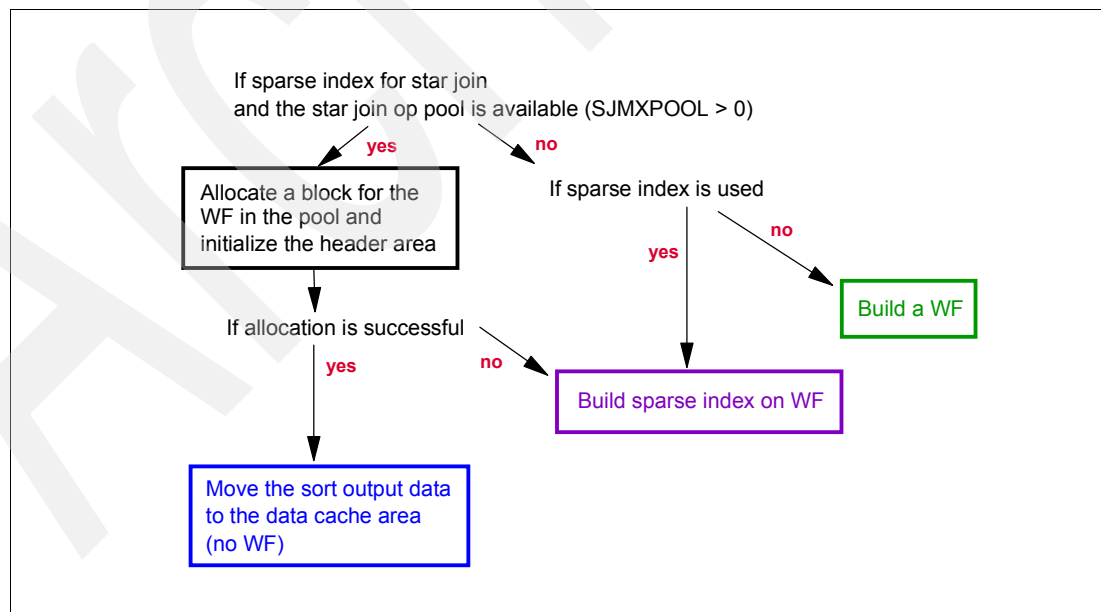


Figure 8-5 Decision between workfile caching or sparse index

Data caching is most effective in the outside-in phase of a pushdown star join plan (JOIN_TYPE = 'S' in the PLAN_TABLE), shown in Figure 8-6, particularly when a large number of rows of the fact table is retrieved. Some queries on a star schema, typically aggregate queries that populate summary tables or generate summary reports, can run hours and take large amounts of resources.

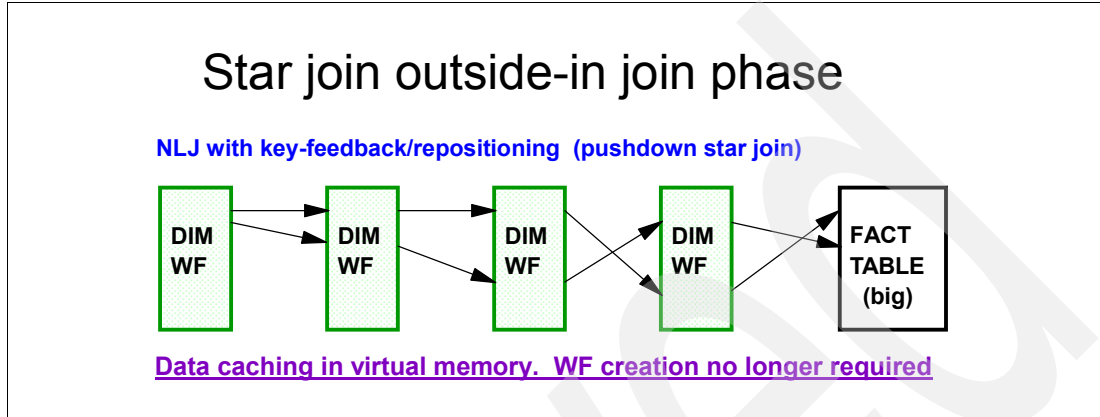


Figure 8-6 Data caching in outside-in join phase

In general, the dimension workfiles joined before the fact table are scanned many times. Inner dimension tables tend to be accessed more randomly and sparsely because of the key-feedback feature. This can cause a high getpage count on the workfile buffer pool, and possibly, many physical page reads. By caching the workfile data in memory, getpage and I/O activities on the workfile buffer pool can be reduced.

Let us now look at an example where data caching is not used (for example because the SJMXPOOL DSNZPARM is set to 0 (zero)). Figure 8-7 shows the access path of a star join query before any of the enhancements described in this section. We see that a sort is performed on the composite table (the result of outside-in processing) at the start of the merge scan join processing. This can use many resources and cause additional parallelism overhead (merge and repartition).

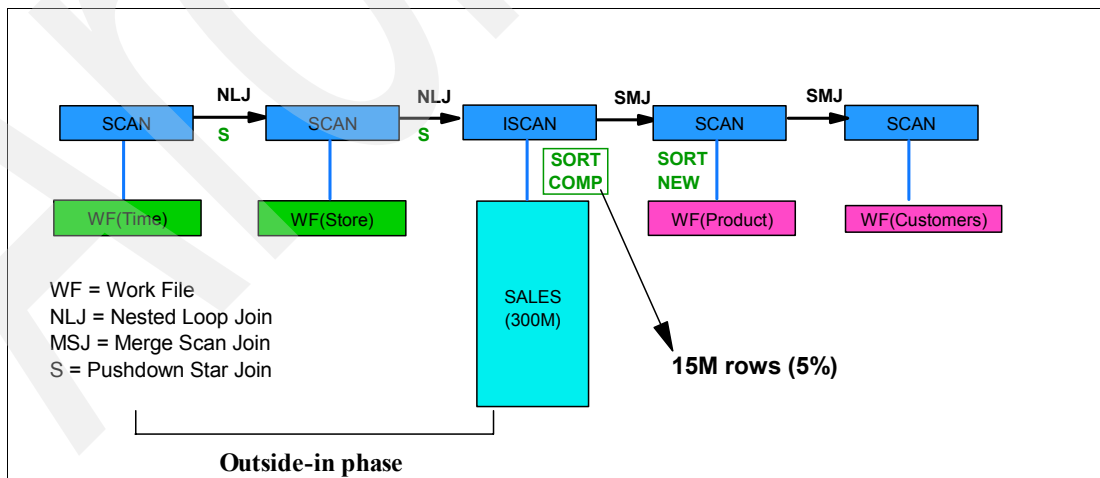


Figure 8-7 Work file sorts prior to sparse index enhancement

In Figure 8-8 we show the benefits of using a sparse index on workfiles. The benefits are:

- ▶ Avoid merge scan join during the inside-out joining phase. A nested loop join is used instead:
 - 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).
- ▶ Speed up the skipping of unqualified keys in the inside-out joining phase:
 - Significant I/O cost reduction.
 - Slight CPU cost reduction. More CPU reduction can be expected when large sorts or multiple sorts are involved.
- ▶ Greater exploitation of parallelism.

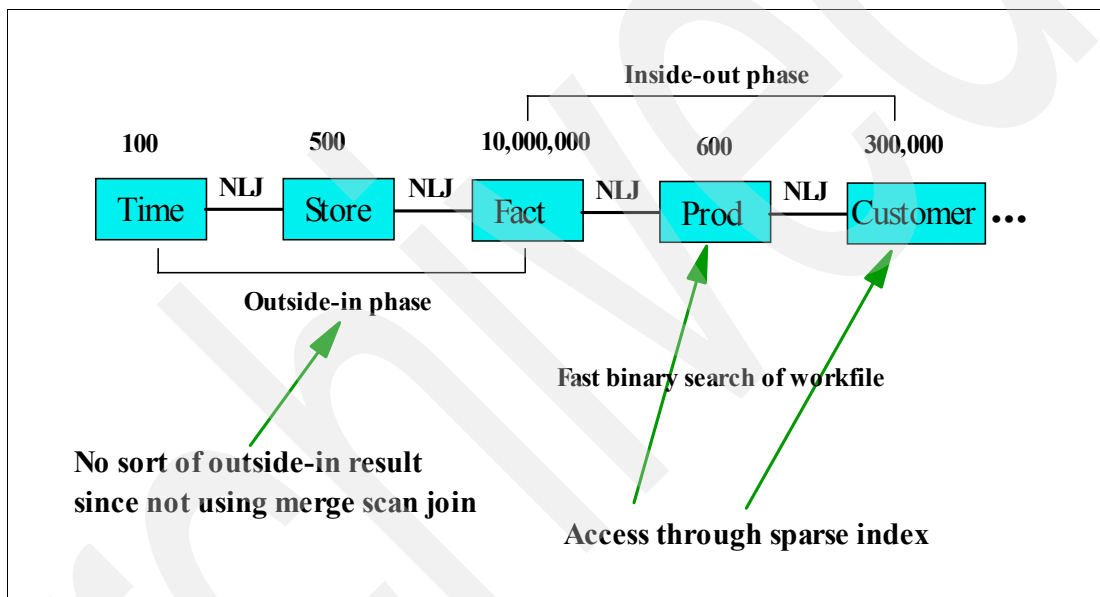


Figure 8-8 Benefits of using a sparse index on workfiles used in a star join plan

The biggest performance gain can be expected by replacing a sort merge join by a nested loop join with a sparse index on the snowflake workfile, thus avoiding the sort operation, especially if the outer composite size is large. Note that in Figure 8-8 the query can also use data caching instead. However, we assumed that data caching is disabled (SJMXPOL=0).

You can determine if this enhancement is used by looking at the PLAN_TABLE. Figure 8-9 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 both features is actually used when running the query.

Example of a push-down star join plan

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 done at runtime and cannot be shown by EXPLAIN

Figure 8-9 Example of star join plan

The possibility of caching workfiles not only can help star join performance. Other concurrently running jobs which normally use workfiles, such as sort, merge join, view materialization, nested table expression materialization, trigger, created temp table, non-correlated subquery, table UDF, etc. can also benefit. Since the star join is now capable of caching the data in memory, instead of using workfiles, these jobs will not have to compete for workfile storage in the buffer pool and table spaces used for workfile processing, resulting in fewer workfile access contention.

Note: Sparse index support is planned to be available with DB2 V7 via maintenance. Check PTF UQ67433 for APAR PQ61458 for details. At the time of writing, there are no plans to make data caching available in V7.

8.6 Long and variable length keys

In this section we discuss three new functions:

- ▶ Varying-length index keys
- ▶ Long index
- ▶ Keys and long predicate support

Varying-length index keys

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 allowed us to have fast key comparisons since these comparisons are between equal length columns. The disadvantage to this approach is that index only access was not allowed when retrieving a varying length key column.

Prior to V8, you can use the RETVLCFK=YES DSNZPARM. 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 column. 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 will no longer need to be padded to their maximum lengths. This will reduce 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, since the length of the variable length column is now stored in the index, and can potentially improve performance.

Indexes can now be created or altered to contain true varying-length columns in the keys. The padding of both VARCHAR and VARGRAPHIC data to their maximum length can now be controlled.

The new keywords NOT PADDED or PADDED on a CREATE and ALTER INDEX statement specify how varying-length columns will be stored in the index.

- ▶ **NOT PADDED** specifies that varying-length columns will not be padded to their maximum length in the index. If there exists at least one varying-length column within the key, length information will be stored with the key. For indexes comprised only of fixed length columns, there is no length information added to the key. The default on the CREATE INDEX statement, NOT PADDED for new V8 installation, can be controlled via the new PADIX ZPARM. A sample create of a non-padded index is shown in Example 8-17.

Example 8-17 Create a NOT PADDED index

```
CREATE UNIQUE INDEX DSN8710.XDEPT1
ON DSN8710.DEPT (DEPTNO ASC)
NOT PADDED
USING STOGROUP DSN8G710
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 NFM are padded by default.

Example 8-18 Create a PADDED index

```
CREATE UNIQUE INDEX DSN8710.XDEPT1
ON DSN8710.DEPT (DEPTNO ASC)
PADDED
USING STOGROUP DSN8G710
PRIQTY 512
SECQTY 64
ERASE NO
BUFFERPOOL BP1
CLOSE YES
PIECESIZE 1 M;
```

- ▶ An index can be created or altered to NOT PADDED, even though the index key may not contain any varying-length columns. The index will be marked as “not padded”, which will allow for varying-length columns to be added in the future without the index being set to pending state.

When comparisons are made between keys with varying-length columns, the keys have to match in length. This will require 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. Example 8-19 illustrates this using a single column index.

Example 8-19 Comparing non-padded index entries

```
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
```

You can continue to use existing indexes that contain padded varying-length columns. However, with DB2 V8, you will now have the ability to convert padded indexes to varying-length indexes and also to convert varying-length indexes back to padded indexes.

Indexes from a prior release will 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 will be marked as being in pending state, if there exists at least one varying-length column in the index. A REBUILD of the index will be necessary to realize the full benefit of a not-padded index.

How to dynamically change an index from padded to not padded and vice versa, is also at “Varying length index keys” on page 56.

Altering a padded index to a not padded index can be done as shown in Example 8-20

Example 8-20 Alter an index to NOT PADDED

```
ALTER INDEX DSN8710.XDEPT1 NOT PADDED;
```

When altering from NOT PADDED to PADDED, the index is placed into 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 in Example 8-21

Example 8-21 Alter an index to PADDED

```
ALTER INDEX DSN8710.XDEPT1 PADDED;
```

Long index keys

Prior to DB2 V8, the maximum key length on a DB2 for z/OS and OS/390 is 255 bytes. This presented 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 greater than the prior limit of 255 bytes.

Furthermore, long index keys are already supported by DB2 for UNIX, Linux and Windows. Applications using keys greater 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.

Long predicates

Prior to DB2 V8, the maximum column length for predicate operands is 255 bytes, or 254 bytes for graphic strings. This is incompatible with the rest of the DB2 family. In DB2 V8 the maximum column 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.

8.7 Support for 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.

For 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 an ORDER BY on STATUS_DATE DESC, STATUS_TIME ASC, and a backward index scan for ORDER BY STATUS_DATE ASC, STATUS_TIME DESC. For ORDER BY STATUS_DATE DESC, STATUS_TIME DESC and STATUS_DATE ASC and STATUS_TIME ASC, as in the above SQL statements, DB2 has to perform a sort.

8.8 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.

For example, let's say that you want to insert 1000 rows into a table that has a trigger. Assume that only 3 of these rows will fire 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 will be created 1000 times and only used by trigger manager 3 times. So 997 times the workfile is created and deleted needlessly.

In Example 8-22, the insert of the first row will cause the row to be put into a transition table, but the trigger will not fire, because the NAME = 'TASHA' and POUNDS = 10, does not match the WHEN clause for TRIGGER NEWCAT. The transition table will be deleted after the statement is completed. The same happens for the second row (it does not match the WHEN clause). The third row however does, so the row that is inserted into the transition table will actually be used to process the after trigger.

Example 8-22 Conditional after trigger

```
CREATE TRIGGER NEWCAT
  AFTER INSERT ON CATS
  REFERENCING NEW AS NROW
  FOR EACH ROW MODE DB2SQL
  WHEN (NROW.NAME = 'SUNSHINE' AND NROW.POUNDS = 12)
  BEGIN ATOMIC
    INSERT INTO PETS
      VALUES (0, 1, NROW.NAME, 'INSERTED SUNSHINE')
  END?

INSERT INTO CATS
  VALUES (1, 'TASHA', 10, '001', 4, 2, 2, 4342, 'PURINA CAT CHOW', 'ANN')
INSERT INTO CATS
  VALUES (2, 'BLACKIE', 9, '001', 4, 2, 2, 3023, 'KAL KAN', 'BETH')
INSERT INTO CATS
  VALUES (3, 'SUNSHINE', 12, '001', 4, 2, 2, 1000, 'FRISKIES BUFFET', 'BETH')
```

An enhancement has been made to the trigger processing to save the changes in memory if there are only few, instead of creating and deleting the workfile each time.

8.9 Reduced lock contention on volatile tables

This 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 physical rows are intended to be accessed in this order (primary key + sequence number). This reduces the chance of deadlocks occurring when two applications attempt to lock the same logical row but touch the underlying physical rows in a different order. This means that certain types of access paths are disabled for these types of tables. They are: list prefetch, hybrid join and multi-index access.

Another case in which “forcing index access” may be desired is for tables whose size can vary greatly. If statistics are taken when the table is empty or has only a few rows, those statistics might not be appropriate when the table has many rows. The optimizer might decide to use a table space scan, which is fine for an almost empty table, but can be disastrous when the table contains a million rows at the end of a business day.

DB2 V8 adds two new keywords to the CREATE TABLE and ALTER TABLE statements; VOLATILE (to force index access whenever possible) and NOT VOLATILE (to allow any type of access to be used). Note that DB2 uses the CLUSTER keyword for other purposes.

- ▶ **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 will 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.

Note: Users are warned that for VOLATILE tables, index access will always be chosen whenever possible, regardless of the efficiency of the available index. Therefore, it is highly recommended that the available index be constructed such that index access would perform satisfactorily on both single table and join scenarios.

8.10 Table UDF cardinality option and block fetch

Today, when you create a table UDF, 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. This is a non-standard SQL feature, specific to IBM DB2 for z/OS implementation. 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 will be used by DB2 at bind time to evaluate the table function access cost.

8.10.1 Table UDF cardinality clause

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 Figure 8-10.

Table-function-reference / table-spec

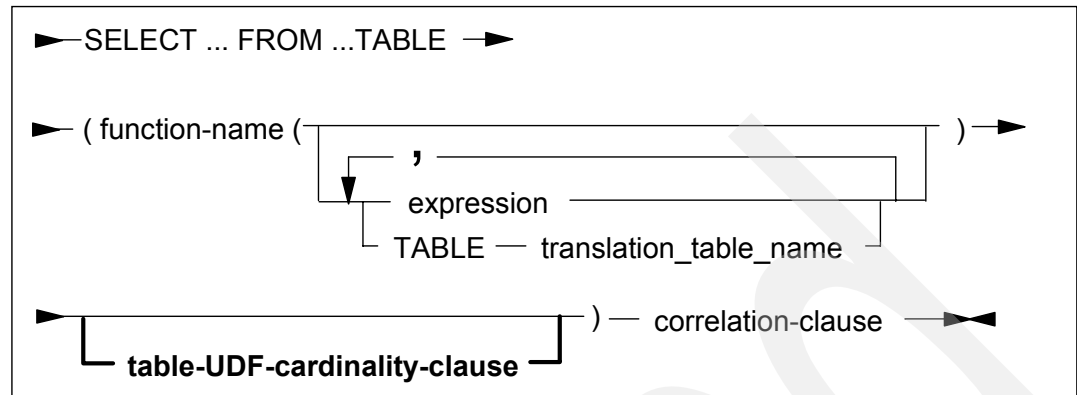


Table-UDF-cardinality-clause:



Figure 8-10 Table UDF cardinality clause

- ▶ The **CARDINALITY** keyword, followed by an integer that represents the expected number of rows returned by the table UDF.
- ▶ The **CARDINALITY MULTIPLIER** keyword, followed by a numeric constant. The expected number of rows returned by the table function will be computed by multiplying the given number to the reference cardinality value that is retrieved from the **CARDINALITY** field of **SYSIBM.SYSROUTINES** for the corresponding table function name, that was specified when the table UDF was created.

Specifying the cardinality option when referencing a table UDF in a **SELECT** statement does not change the corresponding **CARDINALITY** field in **SYSIBM.SYSROUTINES**. When you specify the cardinality clause when referencing a table UDF, the value only applies for that particular query, and the value in the **CARDINALITY** column in **SYSIBM.SYSROUTINES** is ignored for that particular query (unless of course when you use the **CARDINALITY MULTIPLIER** keyword when referencing the table UDF).

The **CARDINALITY** field value in **SYSIBM.SYSROUTINES** can 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.

Example 8-23 illustrates a case where the cardinality option for a table UDF can influence the query optimization process of DB2.

Example 8-23 Using the **CARDINALITY MULTIPLIER** clause in a query

```

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).

Example 8-24 shows that, instead of using the CARDINALITY MULTIPLIER clause, the same query can be written using the CARDINALITY keyword.

Example 8-24 Using the CARDINALITY clause instead

```
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.

8.10.2 Table UDF block fetch

The performance improvement of queries like Example 8-24 are achieved in conjunction with another new features introduced in DB2 V8, called “table UDF block fetch”.

Before this enhancement, each returned row from a table UDF needs a UDF fetch call, and each call needs a context switch since a UDF runs in an 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 can use 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, we save #rows_in_table_UDF_result - 1 context switches, at the cost of a one-time workfile creation, the cost to insert the rows into 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:

- ▶ Estimated #rows returned by the table UDF
- ▶ 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 will use the table UDF block fetch feature by using EXPLAIN. When used, the ACCESSTYPE field in the PLAN_TABLE will contain "RW." Equivalent information is available in the mini-plan trace record IFCID 22.

Another performance improvement that impacts table UDFs was introduced in DB2 V7 with the APAR PQ54042, and extended in V8 to unlike data types. An example is shown in Figure 8-11. With this enhancement, the book.ID = tf.id predicate is stage 1 and indexable provided that the table UDF is accessed first (outer table).

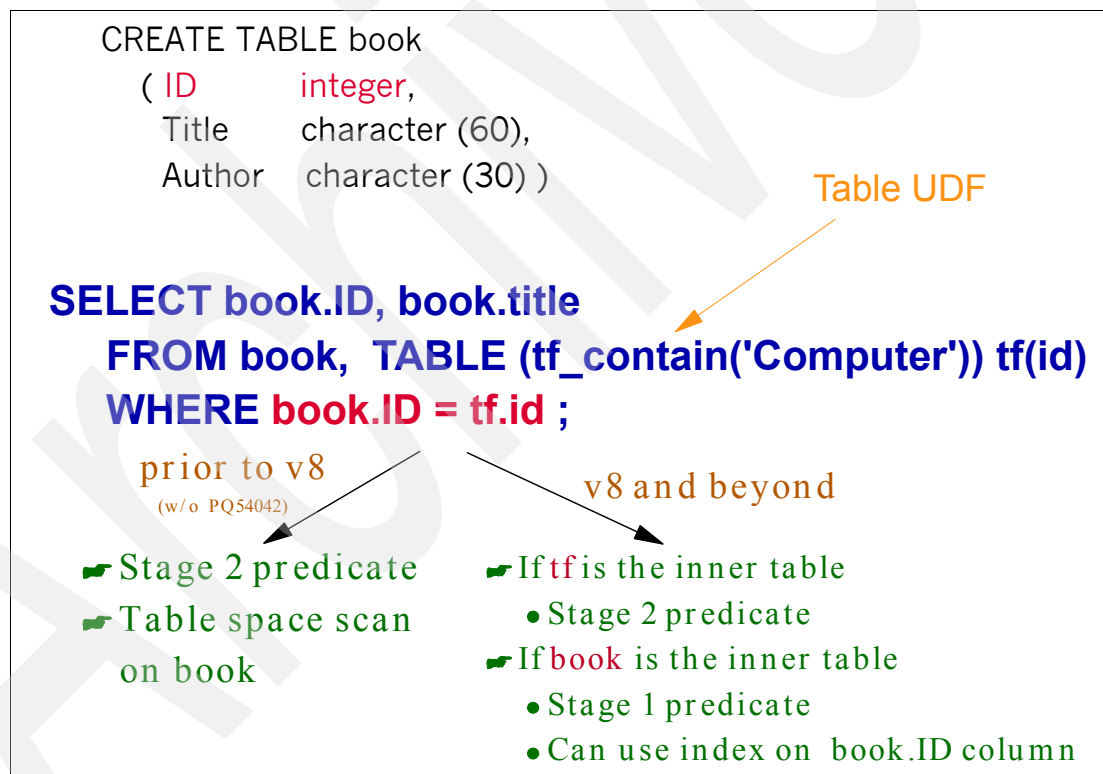


Figure 8-11 Predicate using table UDF indexable and stage 1

More information about the performance enhancements when comparing unlike data types in general, can be found in 8.1, "Comparing unlike data types" on page 206.

Archived

Data sharing

In this chapter we introduce the following data sharing topics:

- ▶ CF lock propagation reduction
- ▶ Reduction of overhead costs for data sharing workloads
- ▶ Batched updates for index page splits
- ▶ Improved LPL recovery
- ▶ Resolution of indoubt units of recovery in restart light
- ▶ Change to IMMEDIATE BIND option default
- ▶ Change to -DISPLAY GROUPBUFFERPOOL output

9.1 CF lock propagation reduction

The purpose of this enhancement is to avoid the cost of global contention processing whenever possible, and improve availability due to a reduction in retained locks following a subsystem failure.

This enhancement will remap IX parent L-locks from XES-X to XES-S. Data sharing locking performance will benefit because IX and IS parent L-locks are now both mapped to XES-S, and are therefore compatible and can now be granted locally. Invoking global lock contention processing to determine that the new IX or IS lock is compatible with existing IX or IS locks is no longer required.

However, to ensure that an IX-lock remains incompatible with an S-lock, *S*-table and table space locks are remapped to XES-X locks. This means that global contention processing will now be required to verify that a page set S L-lock is compatible with another page set S L-lock. This is normally not a problem as this is a relatively rare case. The majority of cases are comparing IS-IS, IS-IX and IX-IX, all of which can now be done locally. Because DB2 can now avoid having to go through global lock contention processing for the majority of cases, the performance of applications that previously had a great deal of XES false contention will improve significantly.

This enhancement has some repercussions on other areas as well.

With this enhancement, explicit hierarchical locking can no longer propagate child L-locks based on the parent L-lock. Instead, child L-locks will be propagated based on the held state of the pageset P-lock. If the page set P-lock is negotiated from X to SIX or IX, then child L-locks will be propagated.

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.

Another consequence of this enhancement is that it is now important that L-locks and P-locks are maintained at the same level of granularity. For partitioned table spaces defined LOCKPART NO DB2 versions prior to V8 lock the last partition to indicate a lock on the whole table space. This behavior will be changed so that even when you specify LOCKPART NO, table spaces will now obtain locks at the part level.

As before, LOCKPART YES is not compatible with LOCKSIZE TABLESPACE. However, if LOCKPART NO and LOCKSIZE TABLESPACE are specified then DB2 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 acquired on individual partitions even though LOCKPART NO is specified.

This change applies to both data sharing and non-data sharing environments.

With this enhancement, the recommendation for RELEASE(DEALLOCATE) and thread reuse to reduce XES messaging for page set L-locks in a data sharing environment 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.

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.

9.2 Reduction of overhead costs for data sharing workloads

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 bufferpool for castout processing with a single CF read request. The actual command is called Read For Castout Multiple (RFCOM).

This enhancement will 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. The most benefit is expected for workloads which update large numbers of pages belonging to GBP-dependent objects, for example batch workloads.

DB2 instrumentation (statistics IFCID 2 and Accounting 3 and 148) records are enhanced to reflect the usage of these new commands.

9.3 Batched updates for index page splits

Another enhancement to boost performance by reducing the cost of page set GBP dependency, but even more by reducing the log I/O wait time, is a more efficient mechanism to process index page splits.

In previous versions of DB2, index page splits require up to five separate writes the group buffer pool and flushes of the DB2 log buffers. This may cause significant overhead for GBP-dependent indexes, resulting in additional synchronous GBP writes, and even more important, in high wait times for synchronous log I/O.

With this enhancement, DB2 accumulates the index page split updates and process them as a single entity, thereby reducing log writes and CF traffic. This will improve performance for high-volume INSERT OLTP workloads and other operations.

The implementation of these improvements is transparent to the end-user.

9.4 Improved LPL recovery

Prior to Version 8, you had to manually recover pages that DB2 put into the logical page list (LPL). But DB2 V8 adds automatic recovery of LPL pages. When pages are added to the LPL, DB2 issues message DSNB250E, which is enhanced to indicate the reason the pages are added to the LPL. DB2 then attempts automatic recovery, except in the following situations:

- ▶ Disk I/O errors
- ▶ During DB2 restart or end_restart times
- ▶ Group buffer pool structure failures
- ▶ 100% loss of connection to the group buffer pool

DB2 does not attempt automatic LPL recovery in the situations mentioned above, because DB2 understands that it will not benefit from trying to do so in those cases.

If automatic-LPL recovery completes successfully, DB2 deletes the pages from the LPL and issues message DSN1021I, which indicates completion.

In addition, and probably more significant than the previous enhancement, DB2 V8, only locks the LPL pages during the recovery process, leaving the remaining pages in the page set or partition accessible to DB2 applications while LPL recovery is in progress. This improves system performance and enhances data availability. Before Version 8, even though a few pages may have been in the LPL list, the entire pageset or partition was unavailable while DB2 was doing LPL recovery.

9.5 Resolution of indoubt units of recovery in restart light

Prior to DB2 V8, starting a DB2 member in LIGHT(YES) mode (restart light) will remove retained locks with minimal disruption in the event of a system failure. Restart light is improved in DB2 V8. If indoubt units of recovery (UR) exist at the end of restart recovery, DB2 will 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.

9.6 Change to IMMEDIATE BIND option default

Currently, changed pages in a data sharing environment are written during phase 2 of the commit process, unless otherwise specified by the IMMEDIATE BIND parameter, or IMMEDIWRI DSNZPARM. This enhancement changes the default processing to write changed pages during phase 1 of commit processing. This change will aid those customers that have transactional data dependencies across data sharing members.

The options you can specify for the IMMEDIATE BIND parameter remain unchanged. However, whether you specify "NO" or "PH1", the behavior will be the identical. The "PH1" option remains for compatibility reasons, but its usage should be deprecated. The DSNZPARM IMMEDIWRI parameter will no longer accept a value of "PH1".

9.7 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. 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 microcode level as displayed by the **D CF** command.

Archived

Archived



Installation and migration

In this chapter we provide you with enough information to start evaluating the changes and planning for a successful installation and/or migration to DB2 UDB for z/OS Version 8 (program number 5625-DB2). The following topics are covered:

- ▶ Currency of versions and migration paths
- ▶ Major changes to installation and migration
- ▶ Installation
- ▶ Migration
- ▶ Catalog changes
- ▶ msys for Setup DB2 Customization Center
- ▶ Samples

10.1 Currency of versions and migration paths

Figure 10-1 summarizes the versions of DB2 which are currently available or announced.

Version	PID	Generally Available	Minimum MVS level	Marketing Withdrawal	End of Service
V3	5685-DB2	December 1993		February 2000	January 2001
V4	5695-DB2	November 1995		December 2000	December 2001
V5	5655-DB2	June 1997	MVS V4R3	December 2001	December 2002
V6	5645-DB2	June 1999	OS/390 V1R3	June 2002	
V7	5675-DB2	March 2001	OS/390 V2R7		
V8	5625-DB2	QPP announced January 2003	z/OS V1R3		

Figure 10-1 Currency of DB2 versions

Customers still running DB2 Version 3, 4, and 5 have a maintenance issue since the end of service is past. End of marketing has occurred also for Version 6.

The customers that are currently still running Version 5 must move to DB2 V7 taking advantage of the skip-migration from V5 to V7 supported as one-time effort only for the Y2K issues.

Updated information is available from the Web site:

<http://www.ibm.com/software/data/db2/os390/availsum.html>

In Figure 10-2 we show the possible migration paths. DB2 V7 is the only release from which you can migrate to DB2 V8. If your DB2 is still at V5 level, the only way to migrate to V7 is to skip V6 and migrate to V7 directly. After you have completed this, you can then migrate to V8. DB2 V6 has been withdrawn from market on June 30, 2002 and it cannot be acquired any more.

Attention:

The only current OS/390 release with DB2 V6 and V7 is V2R10, the other levels mentioned in Figure 10-1 were correct at the time of announcement, but now they are obsolete.

Before migrating to V8 you need to have z/OS V1R3 or higher (depending on the requirements of the functions that will be utilized) active on your zSeries processor.

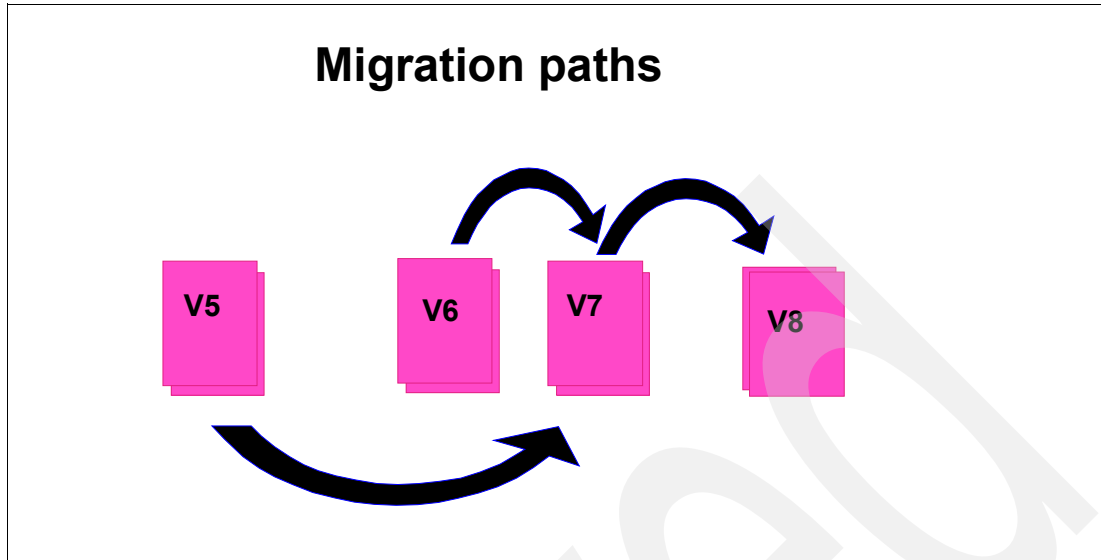


Figure 10-2 Possible migration paths

10.2 Major changes to installation and migration

We define *installation* as the process of installing a *new* DB2 subsystem. In this case there is no worry of compatibility and regression or conversion of pre-existing data. With a newly installed DB2 V8 subsystem, the user can immediately take advantage of all new functions provided by the re-engineered product.

We describe the major changes in the install procedure (also valid for migration) in 10.3, "Installation" on page 246.

Migration is the process of converting an *existing* DB2 V7 subsystem, user data and catalog data, to V8. This process is changed with V8 in order to minimize the possible impact of regression and fall-back incompatibilities.

The key changes to the installation and migration processes are:

- ▶ A CCSID must be defined.
- ▶ DECP *user* library must be defined.
- ▶ Buffer pools of sizes 4 KB, 8 KB, 18 KB, 32 KB must be defined.
- ▶ Only WLM-supported stored procedures can be defined.
- ▶ DB2 system parameters can have short and long components.
- ▶ DB2 can now use VSAM control intervals larger than 4 KB.
- ▶ The migration process now consists of three distinct phases:
 - **CM: Compatibility Mode:** During this phase, which can last as long as deemed necessary, the user will make all the tests needed to ensure that no regression is taking place, and is always allowed to fall back to V7 in case of problems
 - **ENFM: Enable New Function Mode:** During this phase the user will convert the DB2 subsystem to the format ready to support the new functions by using on-line Reorg executions. No fallback to DB2 V7 is allowed once the ENFM is entered.

- **NFM: New Function Mode:** This is the target phase, triggered by the user activated execution of a job which confirms the conclusion and synchronization of the preparatory enabling steps for one or more DB2 subsystems.

We examine the major migration changes in 10.4, “Migration” on page 248.

10.2.1 Before migrating

Before you start with the migration of your DB2 subsystem to V8, make sure that you are aware of some changes that will affect your DB2 operations.

Fall back SPE on DB2 V7

Make sure the fall back SPE PQ48486 is installed and started on all DB2 V7 members that are being migrated. DB2 code will not allow the migration unless the SPE is active.

CCSIDs and Unicode

You now must specify valid CCSIDs, first the one that identifies the coded character set supported by the I/O devices at your site (otherwise character conversion may produce incorrect results), but also CCSIDs for EBCDIC, ASCII, and Unicode, even though you do not use these encoding schemes to store application data. You also need to define and customize z/OS Unicode support as described in the *z/OS V1R3.0 Support for Unicode Using Conversion Services*, SA22-7649-01.

Define a user DSNHDECP module

DB2 is no longer defining a default DECP module library. Just use job DSNTIJUZ to define it in SDSNLOAD and SDSNEXIT. DSNTIJUZ also supports short and long ZPARMs, Unicode definitions, and new and updated DB2 system and DSNHDECP parameters.

Verify BP8K0, BP16K0, and BP32K buffer pools definition

In DB2 V7, all catalog and directory table spaces have been allocated in buffer pool BP0. In DB2 V8, some table spaces are allocated in buffer pools BP8K0, BP16K0 and BP32K. Verify Ensure that BP8K0, BP16K0, and BP32K buffer pools definitions (notice the exception in not having a buffer pool called BP32K0) are corresponding to your needs. If these buffer pools do not exist before you begin ENFM, the processing will not fail, DB2 will allocate them in any case with the default values. For data sharing groups, you must define GBP8K0, GBP16K0, and GBP32K buffer pools.

Install IRLM 2.2

DB2 V8 requires IRLM 2.2, delivered with DB2 as FMID HIR220. IRLM has both the 64-bit and the 31-bit versions delivered, so the size of the libraries is about double. With DB2 running IRLM in 64-bit mode you can have approximately 100 million locks. IMS V8 uses the 31-bit version.

Use CI size > 4 KB

The *VARY DS CONTROL INTERVAL* in DSNTIP7 specifies whether you want DB2-managed data sets have variable VSAM CI size possibly providing better availability and performance. This parameter has also effect on the DB2 catalog (user) allocation by defining the new values and then implementing them during the ENFM phase.

New utility for converting BSDSs

In V8, you can convert BSDSs with a new utility (DSNJCNVB) to support more data sets, up to 10000 for archive logs and 93 for each copy of the active log. You can run the conversion utility once DB2 is in ENFM or NFM. Converting the BSDS is required for a larger number of active log data sets and archive log data sets. For information about converting BSDSs, see Section , “Enlarge current BSDS data sets” on page 257.

Consider increasing the external sort size

The new default SORTDATA and SORTKEYS for Load and Reorg utilities, active in CM, will provide better and consistent performance by increasing parallelism of execution. This might require more active tasks concurrently active and more sort work definitions. You might want to verify DB2's settings for IBACK and CTHREAD, and check out the SORTWORK and current DFSORT (or equivalent product) definitions with your MVS system programmer.

Scrollable cursors

Temporary result tables for static scrollable cursors are stored in declared temporary tables. You must allow sufficient storage space for the new TEMP database and segmented table spaces. The updated installation documentation provides guidance in estimating the storage needs.

Modify RUNSTATS jobs

After you migrate to V8, some existing RUNSTATS jobs might fail if the tables on which they run have data-partitioned secondary indexes. RUNSTATS jobs on data-partitioned secondary indexes require sort operations; if the sort package you use does not dynamically allocate sort work data sets, these RUNSTATS jobs need to be modified to allocate the sort work data sets. You can modify the RUNSTATS jobs with the SORTDEVT and SORTNUM keywords, or by adding STATWKnn DD statements to the JCL.

More space for SYSSTATS table space

You may need to allocate more space for the SYSSTATS table space. In V8, the SYSCOLDISTSTATS and SYSCOLDIST catalog tables contain more data than in previous versions.

Consider increasing the size for catalog table spaces and index spaces

After finishing ENFM, your catalog and directory accept long names for most objects being stored in your subsystem. This might cause the underlying VSAM cluster to grow over time. In addition to that, the fact that the catalog data is stored in Unicode might also require a bit more disk space. To avoid running out of space in your catalog and directory table spaces, you should consider to enlarge the underlying clusters either before entering the ENFM or monitor very carefully how the extents develop over time after migration. You can also enlarge catalog and directory allocations by increasing the shadow data sets used during the ENFM conversion.

Consider increasing EDM pool size

With the additional partitioned objects in V8, the database descriptors for the database containing the objects will grow, which might create a need to increase the size of the EDM pool. The EDM pool size can be increased by modifying the EDMPOOL STORAGE SIZE field on installation panel DSNTIPC and stopping and restarting DB2. You can also modify the EDM pool size without stopping and restarting DB2 by using the SET SYSPARM command. However, using the SET SYSPARM command may result in a non-contiguous pool.

LANGUAGE COMPJAVA no longer supported for stored procedures

After migrating to V8, you will no longer be able to define or run COMPJAVA stored procedures. Convert LANGUAGE COMPJAVA stored procedures to LANGUAGE JAVA using the three following steps:

1. Use ALTER PROCEDURE to change the LANGUAGE and the WLM ENVIRONMENT. The EXTERNAL NAME clause must also be specified. Use the following example as a model:

```
ALTER PROCEDURE SYSPROC.JAVADV  
LANGUAGE JAVA EXTERNAL  
NAME 'display.display.main '  
WLM ENVIRONMENT WLMENVJ;
```

2. Ensure that the WLM environment has been configured and that the required JVM is installed.
3. Ensure that the .class file identified in the EXTERNAL NAME clause of the ALTER PROCEDURE is either:
 - Contained in a JAR that has been installed to DB2 by an invocation of the INSTALL_JAR stored procedure, or
 - Located in a directory in the CLASSPATH ENVAR of the data set named on the JAVAENV DD card of the WLM stored procedures address space JCL.

JDBC/SQLJ driver no longer supported

The DB2 for OS/390 and z/OS type 2 JDBC/SQLJ driver is no longer supported. In V8, only the Java combined client JDBC/SQLJ drivers are supported.

Support for DB2-established address spaces is deprecated

In V8, the support for DB2-established address spaces is deprecated. You can no longer specify the NO WLM ENVIRONMENT option when you create or alter stored procedure definitions. Although pre-existing stored procedures can still run in a DB2-established stored procedure address space, you should move your stored procedures to WLM environments as soon as possible.

New precompiler option for string host variables

In previous releases of DB2, if you selected a value from a character column into a C or C++ host variable of the null-terminated character form, and the length of the host variable was longer than the length of the value, DB2 padded the string with blanks and inserted the null-terminator after the blanks. In DB2 V8, the default behavior is to not pad the string with blanks. If you want to produce blank-padded strings, as in previous releases, specify YES in field PAD NUL-TERMINATED in installation panel DSNTIP4, or precompile your program with the PADNTSTR option.

New column for encoding scheme

After you successfully migrate to V8, the encoding scheme used for system-generated parameters for procedures and functions will be stored in a new column in SYSIBM.SYSROUTINES. This information was previously stored in a special row in the SYSIBM.SYSPARMS table.

System level point-in-time recovery

If you plan on using the BACKUP SYSTEM online utility to take volume copies of the data and logs of your non-data sharing DB2 subsystem or your DB2 data sharing group, all of your DB2 data sets must reside on volumes managed by DFSMS. The BACKUP SYSTEM utility and its counterpart, the RESTORE SYSTEM utility, require:

- ▶ z/OS Version 1 Release 5 or above.
- ▶ Disk control units that support ESS FlashCopy.
- ▶ DFSMSHsm Copy constructs that are defined, following the DB2 naming convention.
- ▶ DFSMS Copy target storage pools that are defined. (The BACKUP SYSTEM utility enables volume-level backups of a DB2 system that use these target storage groups.)

10.3 Installation

You can install your DB2 V8 subsystem either as a host based installation or via msys for Setup as shown in Figure 10-3. Refer to 10.6, “msys for Setup DB2 Customization Center” on page 259 for more information on msys for Setup.

Since DB2 V5, the DB2 Installer was the workstation based alternative to the traditional way of installing DB2 through TSO command line. With DB2 V8, *msys for Setup* is the replacement for DB2 Installer. The replacement is referred to as the *msys for Setup DB2 Customization Center*.

The host based installation process must be executed from the TSO command line. The process itself does not differ from the previous releases. As usual, some panels have been changed to support those new, added, or changed functionalities whose behavior you are able to influence by specifying your own values for specific parameters. In this section we show the most important new or changed panel options and the changes to the installation jobs.

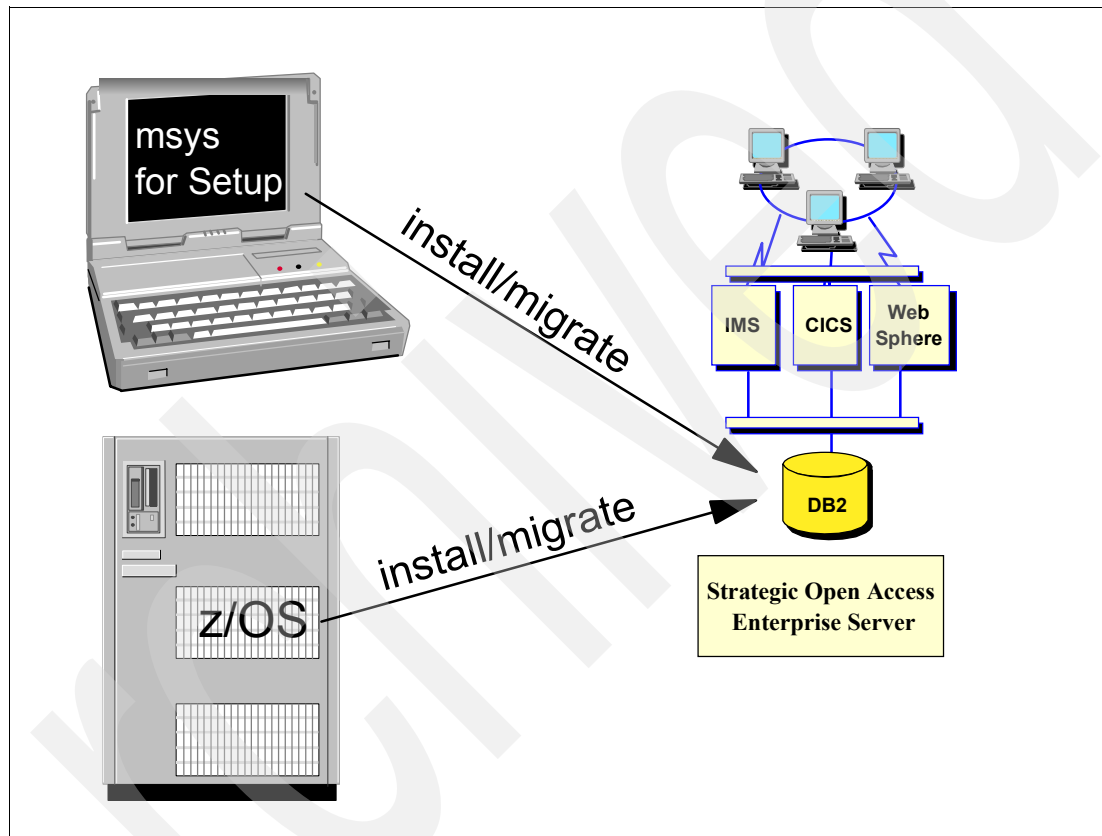


Figure 10-3 Install through TSO or *msys for Setup*

10.3.1 Major changes to install jobs

The installation jobs are as follows:

- ▶ DSNTIJPM: Identify unsupported objects:
This job now only checks for type 1 indexes.
- ▶ DSNTIJMV: Define DB2 to MVS:
This job now defines the DBM1 address space for 64-bit addressing, creates SPAS only during migration, and modifies the SQL procedure, *proc*, to use CCSID(1047.)
- ▶ DSNTIJIN: Define system data sets:
This job now defines several new table and index spaces for the catalog and, depending on the new VSAM control interval ZPARM, defines them with new CI values.

- ▶ **DSNTIJUZ: Define DB2 parameters:**
This job now defines DECP in SDSNLOAD and SDSNEXIT and runs before DSNTIJID. It also supports short and long ZPARMs, Unicode definitions, and changes in DSNHDECP.
- ▶ **DSNTIJID: Initialize system data sets:**
This job now initialize catalog new and old tables with the new CI size if requested.
- ▶ **DSNTIJTM: Define user exit routines:**
This job now defines the required buffer pool for each size, since the V8 catalog has table spaces with 4, 8, 16, and 32 KB.
- ▶ **DSNTIJSG: Define and BIND DB2 objects:**
This job now defines the RLF table for long names support, adds support for DSNUTILU, the Utilities stored procedure with Unicode parser, and defines the DSNWZP stored procedure for system parameters tracking in WLM-managed address space.
- ▶ **DSNTIJTC: Tailor catalog (CATMAINT):**
This job now has only one step.

10.4 Migration

The migration to DB2 V8 is only allowed from V7 and differs from the previous migration processes. As we have seen, a DB2 V8 subsystem can operate in three different modes:

- ▶ Compatibility Mode (CM)
- ▶ Enabling New Function Mode (ENFM)
- ▶ New Function Mode (NFM)

We describe the modes in detail on the following pages. During the migration process, you must go through the CM and ENFM, before reaching the NFM. As shown in Figure 10-4, in CM, your Catalog and Directory have successfully been converted to V8, that is new columns have been added to existing tables, new indexes and table spaces have been created and so on.

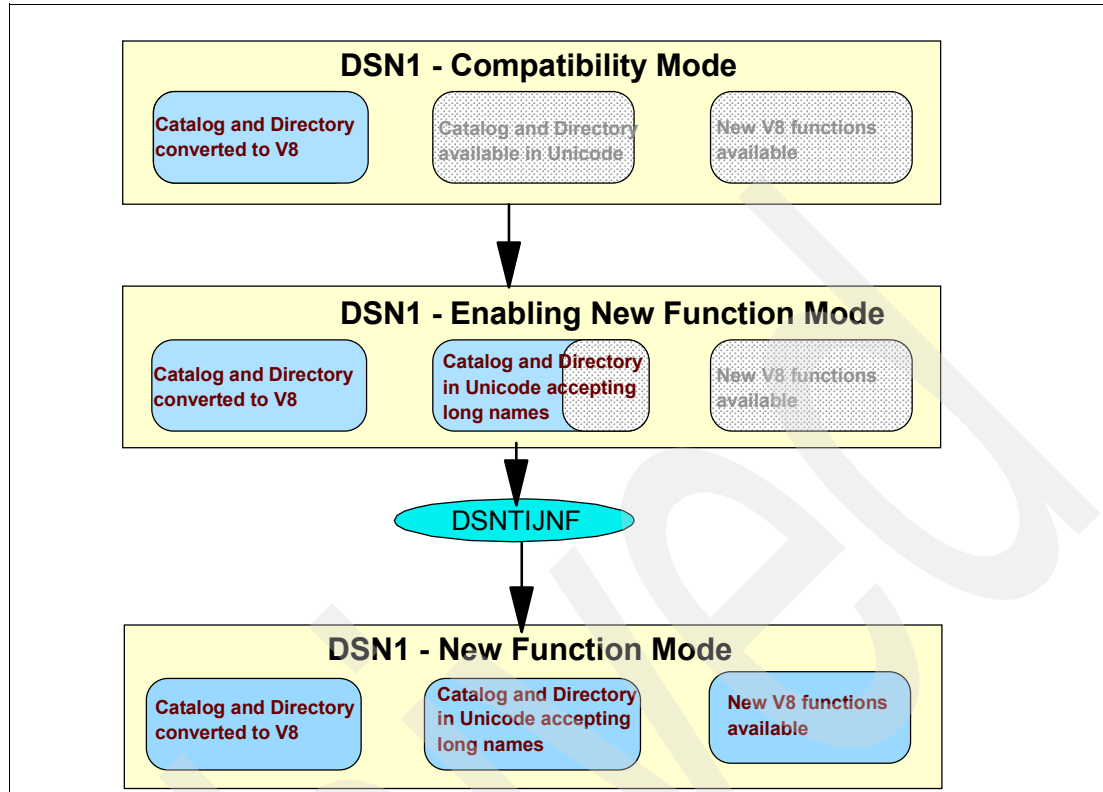


Figure 10-4 Modes of operation

There are two major changes to the DB2 catalog, which are necessary to make most of the new functions provided with DB2 V8 possible. These changes are:

- ▶ Change existing catalog columns so that they can store long names. Refer to 5.1, “Long names” on page 72.
- ▶ Convert the catalog to Unicode.

In CM, none of these changes have been done. Furthermore, very few of the new functions are available for use in this mode of operation. Refer to 10.4.1, “Compatibility Mode” on page 250 for more detailed information.

In ENFM, you are in the process of converting your Catalog table spaces so that after completion of the conversion, they now accept long names and exist in Unicode encoding scheme. Catalog conversion is performed by running job DSNTIJNE. You can spread conversion of your catalog table across several maintenance intervals. Use job DSNTIJNH to direct DSNTIJNE to halt after conversion of the current table space has completed. To resume, simply rerun DSNTIJNE from the top. It will automatically locate the next table space to be converted. The process of conversion is not disruptive, that is, you can continue operation while being in ENFM. Refer to 10.4.2, “Enabling New Function Mode” on page 251 for more detailed information.

Once you have migrated all table spaces of your DB2 catalog to Unicode, your DB2 subsystem will operate in NFM once you run the DSNTIJNF to indicate that everything is ready. This step can be helpful for instance in making sure that all your member of a data sharing group, or all subsystems DRDA related, have been migrated. Now all new functions which have been introduced with DB2 V8 are available.

10.4.1 Compatibility Mode

The first step in the process of migrating from V7 to V8 is a traditional release migration implemented by DSNTIJTC (CATMAINT) execution as in previous migrations. The completion of this migration phase places the DB2 subsystem into V8 CM.

This migration is a very quick process. The catalog and directory migration is one part of this migration step. Table 10-1 shows how the catalog continues to grow with every DB2 release.

Table 10-1 Evolution of the DB2 catalog

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	84	117	1212	105
V8	22	87	128	1286	105

Job DSNTIJTC only contains one step. The result of this process is a current V8 catalog and directory. Refer to 10.5, “Catalog changes” on page 259 for a detailed description of all catalog changes for V8.

Note: Note that you can only migrate to V8 if your V7 subsystem has the fallback small program enhancement (SPE) PQ48486 applied. The existence of the fallback SPE is enforced for both data sharing and non-data sharing. The information is kept in the BSDS/SCA, which is checked at startup time to make sure that all group members of a data sharing group have the SPE applied.

Note: Note that DB2 V7 must be started at least once after the fallback SPE is applied before V8 catalog maintenance is allowed to run successfully.

Installation Verification Process in CM

After you have successfully migrated your subsystem to V8 CM, you should run the installation verification process (IVP) samples. As stated previously, since you cannot perform any of the new functions provided by V8 at this point of time, the IVP differs very little from DB2 V7. You can run selected V7 sample jobs if the DB2 objects from V7 sample jobs still exist on your subsystem. If you do not have the V7 objects available any more, you can either decide to verify the migration only using your own application samples or use the old V7 sample jobs to recreate the objects and run the IVP job afterwards.

Fallback

If you successfully finished the migration of your catalog and directory to DB2 V8 CM, fallback to DB2 V7 is fully supported. In case you encounter any severe error while operating your DB2 V8 subsystem, follow the fallback procedure.

Release coexistence

If your DB2 V8 subsystem is a member of a data sharing group, all other members of this group can only either be V7 or V8. DB2 V8 subsystems operating in CM, cannot coexist within one data sharing group with any DB2 subsystems prior to V7. Before you can enter the next mode of DB2 V8 operation, which is the ENFM, you must successfully migrate all members to V8 and make sure that they run without any problems.

10.4.2 Enabling New Function Mode

The next mode you have to go through is the ENFM, an interim phase until you actually reach the NFM. During ENFM, you do two things to your DB2 V8 catalog:

- ▶ Convert the Catalog from EBCDIC to Unicode UTF-8 encoding scheme
- ▶ Convert all affected columns so that your catalog can handle long names as described in 5.1, “Long names” on page 72.

In contrast to the first migration step, that is the migration from DB2 V7 to DB2 V8 CM, the steps which need to be performed during ENFM only have a minor impact on DB2’s operations. The migration of your catalog table spaces is done via online REORG SHRLEVEL REFERENCE, which means that DB2 only needs exclusive access to the underlying table spaces during a very short period of time.

To start the conversion, you must first go through the installation CLIST again to get new jobs created for this task. Once you called the installation CLIST, you must enter ENFM for option 1 ‘INSTALL TYPE’ as shown in Figure 10-5. Make sure that option 2 ‘DATA SHARING’ is set to ‘blank’ now.

```
DSNTIPA1 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     ====> DSNTIDcm Default parameter values
7  OUTPUT MEMBER NAME    ====> DSNTIDnf Save new values entered on panels

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

Figure 10-5 DSNTIPA1 — Main panel for ENFM

The next panel which is displayed is panel DSNTIPT.

DSNTIPT — Data set names panel 1

You should consider choosing a new name for your customized SDSNSAMP output data set, which you can specify for option 2 as shown in Figure 10-6. This data set will then contain, based on the values you used for input member (option 6 in DSNTIPA1), three new migration and all the IVP jobs.

```
DSNTIPT                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.ENFM.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
```

Figure 10-6 DSNTIPT — Choose a new name for SDSNSAMP library

DSNTIP00 — Shadow data set allocation

The migration process during ENFM is done via online REORG. Since the table spaces of the DB2 catalog are user-defined table spaces, you must manually allocate shadow data sets for the REORG SHRLEVEL REFERENCE. Panel DSNTIP00, shown in Figure 10-7, lists all 18 DB2 catalog and directory table spaces that are transformed during ENFM processing. The values are based on PERMANENT UNIT NAME and VOLUME SERIAL 3 from panel DSNTIPA2, and PRIMARY RECS and SECONDARY RECS from panel DSNTCALC in CM migration. They can be overridden in this panel.

```

DSNTIP00      ENABLE NEW FUNCTION MODE FOR DB2 - SHADOW DATA SET ALLOCATION
===>
      OBJECT      DASD DEVICE      VOL/SERIAL      PRIMARY RECS      SECONDARY RECS
1  SPT01      ==> SYSDA      ==> DSNV02      ==> 636      ==> 636
2  SYSDBASE  ==> SYSDA      ==> DSNV02      ==> 7049     ==> 7049
3  SYSDBAUT  ==> SYSDA      ==> DSNV02      ==> 478      ==> 478
4  SYSDDF    ==> SYSDA      ==> DSNV02      ==> 144      ==> 144
5  SYSGPAUT  ==> SYSDA      ==> DSNV02      ==> 3060     ==> 3060
6  SYSGROUP  ==> SYSDA      ==> DSNV02      ==> 48       ==> 48
7  SYSGRTNS  ==> SYSDA      ==> DSNV02      ==> 144      ==> 144
8  SYSHIST   ==> SYSDA      ==> DSNV02      ==> 144      ==> 144
9  SYSJAVA   ==> SYSDA      ==> DSNV02      ==> 144      ==> 144
10 SYSOBJ    ==> SYSDA      ==> DSNV02      ==> 616      ==> 616
11 SYSPKAGE  ==> SYSDA      ==> DSNV02      ==> 9673     ==> 9673
12 SYSPLAN   ==> SYSDA      ==> DSNV02      ==> 13373    ==> 13373
13 SYSSEQ    ==> SYSDA      ==> DSNV02      ==> 144      ==> 144
14 SYSSEQ2   ==> SYSDA      ==> DSNV02      ==> 144      ==> 144
15 SYSSTATS  ==> SYSDA      ==> DSNV02      ==> 53355    ==> 53355
16 SYSSTR    ==> SYSDA      ==> DSNV02      ==> 661      ==> 661
17 SYSUSER   ==> SYSDA      ==> DSNV02      ==> 1675     ==> 1675
18 SYSVIEWS  ==> SYSDA      ==> DSNV02      ==> 7093     ==> 7093
19 INDEXES   ==> SYSDA      ==> DSNV02      Catalog and directory index shadows
PRESS:  ENTER to continue  RETURN to exit  HELP for more information

```

Figure 10-7 DSNTIP00 — Shadow data set allocation

DSNTIP01 — Image copy data set allocations

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. During online REORG the image copies are taken. You can use panel DSNTIP01 as shown in Figure 10-8 to enter image copy names that follow your installation's naming standards and which output device you want for the copies.

Tape devices are supported, but not stacking.

```

DSNTIP01  ENABLE NEW FUNCTION MODE FOR DB2 - IMAGE COPY DATA SET ALLOCATIONS
===>

      OBJECT      IMAGE COPY DATA SET NAME      DEVICE TYPE
1  SPT01      ==> DSN810.IMAGCOPY.SPT01      ==> SYSDA
2  SYSDBASE   ==> DSN810.IMAGCOPY.SYSDBASE     ==> SYSDA
3  SYSDBAUT   ==> DSN810.IMAGCOPY.SYSDBAUT     ==> SYSDA
4  SYSDDF     ==> DSN810.IMAGCOPY.SYSDDF       ==> SYSDA
5  SYSGPAUT   ==> DSN810.IMAGCOPY.SYSGPAUT     ==> SYSDA
6  SYSGROUP   ==> DSN810.IMAGCOPY.SYSGROUP     ==> SYSDA
7  SYSGRTNS   ==> DSN810.IMAGCOPY.SYSGRTNS     ==> SYSDA
8  SYSHIST    ==> DSN810.IMAGCOPY.SYSHIST      ==> SYSDA
9  SYSJAUXB   ==> DSN810.IMAGCOPY.SYSJAUXB     ==> SYSDA
10 SYSJAVA    ==> DSN810.IMAGCOPY.SYSJAVA      ==> SYSDA
11 SYSOBJ     ==> DSN810.IMAGCOPY.SYSOBJ       ==> SYSDA
12 SYSPKAGE   ==> DSN810.IMAGCOPY.SYSPKAGE     ==> SYSDA
13 SYSPLAN    ==> DSN810.IMAGCOPY.SYSPLAN      ==> SYSDA
14 SYSSEQ     ==> DSN810.IMAGCOPY.SYSSEQ       ==> SYSDA
15 SYSSEQ2    ==> DSN810.IMAGCOPY.SYSSEQ2      ==> SYSDA
16 SYSSTATS   ==> DSN810.IMAGCOPY.SYSSTATS     ==> SYSDA
17 SYSSTR     ==> DSN810.IMAGCOPY.SYSSTR       ==> SYSDA
18 SYSUSER    ==> DSN810.IMAGCOPY.SYSUSER      ==> SYSDA
19 SYSVIEWS   ==> DSN810.IMAGCOPY.SYSVIEWS     ==> SYSDA
PRESS:  ENTER to continue  RETURN to exit  HELP for more information

```

Figure 10-8 DSNTIP01 — Image copy data set allocations

After the necessary jobs have been generated, you can start the ENFM processing. The first job you must submit is job DSNTIJNF.

10.4.3 ENMF jobs

The ENMF jobs are:

- ▶ DSNTIJNE: ENFM processing:
 - ▶ This new job calls online Reorg and executes 18 groups of steps for each catalog table space to be converted. It can be stopped with job DSNTIJNH and it will restart from the beginning of the interrupted group.
- ▶ DSNTIJNH: Halt DSNTIJNE:
 - ▶ This new job stops the execution of DSNTIJNE at the end of the active group.
- ▶ DSNTIJNF: Turn NFM on:
 - ▶ This new job positions the catalog for NFM.
- ▶ DSNTIJNG: Update DECP for NFM:
 - ▶ This job updates DECP with NEWFUN=YES in SDSNEXIT and SDSNLOAD.
- ▶ DSNTIJEN: Return to ENFM status:
 - ▶ This job returns from NFM to ENFM status at user request.
- ▶ DSNTIJNR: Convert the RLST for long name support:
 - ▶ This job ALTERS the columns to support long names.
- ▶ V8 IVPs:
 - ▶ This job performs a health check on the V8 new functions. See 10.7, “Samples” on page 261.

DSNTIJNE

This job performs the actual conversion of the table spaces listed in Figure 10-8 starting with SYSVIEWS and then in alphabetical order and last the DB2 directory. There is no need for you to run the conversion of all affected table spaces without interruption. You can interrupt the execution. You can start the conversion whenever you think your subsystem is available for a short interrupt during the last log apply and switch phase of online reorg. There is another new migration job, DSNTIJNH, which can be used any time to stop the processing of DSNTIJNF. Refer to “DSNTIJNH” on page 255 for more information regarding DSNTIJNH, but it basically stop the execution at the end of the current table space being reorganized. You must then run DSNTIJNF from the top when you resume the conversion process for your table spaces, and the job will go through the already executed phases and restart from the next one to be run.

Note: It is absolutely necessary to keep the sequence of steps provided in job DSNTIJNE.

The job contains the following steps:

- ▶ ENFM0001 — Signal start of ENFM processing
- ▶ ENFM0nn0 — Check NFM status of table space in sequence
- ▶ ENFM0nn1 — Clean up workfiles for converting table space in sequence
- ▶ ENFM0nn3 — Allocate shadow data sets for table space in sequence
- ▶ ENFM0nn7 — REORG SHRLEVEL REFERENCE table space in sequence
- ▶ ENFM0nn9 — Delete workfiles used converting the table space in sequence

Steps ENFM0nn1 — ENFM0nn9 are repeated for all table spaces which are converted during ENFM.

DSNTIJNF

This job seals the completion of ENFM and allows the user to synchronize NFM across multiple DB2 subsystems.

DSNTIJNH

The only step in this DSNTIJNH job contains the following job card for Utility CATENFM:

```
CATENFM HALTENFM
```

If you execute this job while DSNTIJNF is running, DSNTIJNF will continue to run until the conversion of the current catalog table space is completed. DSNTIJNF will then end with return code six.

Note: Use job DSNTIJNH when you plan to interrupt the conversion processing of DSNTIJNF. Never use any other means except this job to halt conversion processing.

DSNTIJNG

The purpose of this job is to rebuild the DSNHDECP module so that the NEWFUN parameter is set from NO to YES. From then on, the precompiler will accept by default all SQL that uses V8 NFM.

Verifying your migration mode

You can check the mode in which your DB2 subsystem is currently operating using the following command:

```
-DISPLAY GROUP DETAIL
```

As you can see highlighted in Figure 10-9, the output of the DISPLAY GROUP command has been enhanced so that you can figure out the current mode of operation for your DB2 subsystem. This command is valid for data sharing and for non-data sharing members.

```

DSN7100I  -DB80 DSN7GCMD
*** BEGIN DISPLAY OF GROUP(.....) GROUP LEVEL(...) MODE(N)
                                GROUP ATTACH NAME(....)
-----
DB2                                DB2 SYSTEM  IRLM
MEMBER  ID  SUBSYS  CMDPREF  STATUS  LVL  NAME  SUBSYS  IRLMPROC
-----
.....    0  DB80   -DB80   ACTIVE  810  SC48   ID80   DB80IRLM
-----
*** END DISPLAY OF GROUP(.....)
DSN9022I  -DB80 DSN7GCMD 'DISPLAY GROUP ' NORMAL COMPLETION
***

```

Figure 10-9 DISPLAY GROUP command

MODE can show *C* for CM, *E* for ENFM and *N* for NFM.

Fallback

Once you started the conversion of your DB2 catalog and directory using the new CATENFM utility, you are no longer able to fallback to:

- ▶ DB2 V8 CM or
- ▶ DB2 V7

Therefore, you should make sure that you do not begin with the migration to ENFM unless your DB2 subsystems, including all data sharing members, are stabilized in DB2 V8 CM.

Release Coexistence

The scope of the fact of whether your DB2 subsystem is operating in ENFM is group wide. This means that prior to start with ENFM, you must make sure that all members of you data sharing group are running in DB2 V8 CM.

10.4.4 New Function Mode

Once you have completed the ENFM, your DB2 subsystem or data sharing group enters the NFM. At this mode, all new DB2 V8 functions are enabled and available for use. The catalog is completely Unicode and long names can be used.

Increase number of active and archive log data sets in BSDS

Large DB2 systems are creating so many archive log data sets that the maximum number of archive log volumes of 1000, which existed up to DB2 V7, only allows you to restore a few days of log data in your BSDSs.

In addition to that, you might feel like having more then the 31 active log data sets per log copy, which are allowed for DB2 V7, would be helpful for you. This is especially important if you write your archive logs to tape. In this case, having more than 31 active log data sets available for extended rollback or for media recovery could speed up these operations enormously.

DB2 V8 takes care of these needs and enables you to store up to the following amounts in your BSDS data sets:

- ▶ 10000 archive log data sets (20000 if recording in dual mode)
- ▶ 93 active log data sets (186 for dual copy)

The current layout of your BSDS data sets does not allow to store all these information. Therefore in order to be able to use these new limits for active and archive log data sets, you must:

- ▶ Enlarge your current BSDS VSAM KSDS clusters using the following procedure
- ▶ Convert your old BSDS copy1 and cop2 using the newly introduced utility DSNJCNVB

You can use stand-alone utility DSNJU004 to find out if your BSDS data sets have already been converted to the new format. Refer to the highlighted information in Figure 10-10. As you can see, the BSDS data set has not yet been converted.

```
DSNJCNVB CONVERSION PROGRAM HAS NOT RUN  DDNAME=SYSUT1
LOG MAP OF BSDS DATA SET COPY 1, DSN=DB70U.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=DB70U
HIGHEST RBA WRITTEN      00000428E1F4  0000.000  00:00:00.0
HIGHEST RBA OFFLOADED   0000041F3FFF
RBA WHEN CONVERTED TO V4 000000000000
```

Figure 10-10 Checking BSDS conversion

Enlarge current BSDS data sets

Follow the steps below to enlarge your current BSDS data sets while DB2 is shutdown:

- ▶ Rename your existing BSDS copies. Be sure and save the original versions.
- ▶ Allocate larger BSDS data sets. Remember that you can use DSNTIJIN as sample.
- ▶ Use VSAM REPRO to copy the original data set to the new, larger data set

Run DSNJCNVB conversion utility

DSNJCNVB conversion utility is a new DB2 V8 stand-alone utility. You can run it in the same manner as the DSNJU003 and DSNJU004 utilities. Refer to Figure 10-11 for a sample JCL.

```
//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=*
//SYSUDUMP DD SYSOUT=*
//SYSIN DD *
```

Figure 10-11 DSNJCNVB sample JCL

Execute this utility to convert your existing BSDS data sets. If the job completes successfully, you will receive the utility output shown in Figure 10-12.

```

CONVERSION OF BSDS DATA SET - COPY 1, DSN=DB70U.BSDS01
  SYSTEM TIMESTAMP - DATE= 2.252  LTIME=22:15:43.72
  UTILITY TIMESTAMP - DATE= 2.249  LTIME=18:23:22.80
  PREVIOUS HIKEY - 04000053
  NEW HIKEY - 040002F0
  RECORDS ADDED - 669
CONVERSION OF BSDS DATA SET - COPY 2, DSN=DB70U.BSDS02
  SYSTEM TIMESTAMP - DATE= 2.252  LTIME=22:15:43.72
  UTILITY TIMESTAMP - DATE= 2.249  LTIME=18:23:22.80
  PREVIOUS HIKEY - 04000053
  NEW HIKEY - 040002F0
  RECORDS ADDED - 669
DSNJ260I DSNJCNVB BSDS CONVERSION FOR DDNAME=SYSUT1 COMPLETED SUCCESSFULLY
DSNJ260I DSNJCNVB BSDS CONVERSION FOR DDNAME=SYSUT2 COMPLETED SUCCESSFULLY
DSNJ200I DSNJCNVB CONVERT BSDS UTILITY PROCESSING COMPLETED SUCCESSFULLY
***** BOTTOM OF DATA *****

```

Figure 10-12 DSNJCNVB SYSPRINT

If you print your BSDS again, you can see the information highlighted in Figure 10-13.

```

DSNJCNVB CONVERSION PROGRAM HAS RUN DDNAME=SYSUT1
LOG MAP OF BSDS DATA SET COPY 2, DSN=DB70U.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=DB70U
HIGHEST RBA WRITTEN 00000428E1F4 0000.000 00:00:00.0
HIGHEST RBA OFFLOADED 0000041F3FFF
RBA WHEN CONVERTED TO V4 000000000000
.....

```

Figure 10-13 DSNJU004 output indicating new BSDS structure

Note: If you install your DB2 V8 from scratch, that is if you do not migrate from Version 7, the installation already provides large BSDS data sets. But in order to be able to use more than 31 active log data sets and more than 1000 archive log data sets, you must manually convert your BSDS data sets using the stand-alone utility DSNJCNVB.

Fallback

Once your DB2 subsystem is operating at DB2 V8 NFM, you are no longer able to fallback to:

- ▶ DB2 V8 CM or
- ▶ DB2 V7

Release coexistence

The scope of the fact of whether your DB2 subsystem is operating in NFM is group wide. This means that prior to start with NFM, you must make sure that all members of your data sharing group are running in DB2 V8 CM.

Verification

After activating NFM with job DSNIJNF, and rebuilding the DSNHDECP with job DSNTIJNG, you might want to run the DB2 V8 IVB jobs. Once stabilized in NFM, you can run DSNTIJNR to convert the RLST tables for long names support.

10.5 Catalog changes

As for previous versions, DB2 V8 requires some changes to the DB2 catalog in order to support new functions. The only difference is that these changes are massive with V8.

The introduction of long names (varchar 128) for most of the objects defined in the DB2 V8 catalog has a cascading and pervasive effect on other definitions. For instance keys need more than 255 bytes, rows can be bigger than 4056, and therefore buffer pools of 8 KB and 16 KB pages have to be defined for use by the catalog, in addition to the 4 KB and 32 KB already used with DB2 V7.

Another major change is that the catalog is now defined as Unicode, so if you have existing queries on the catalog, you might receive results in a different ordering.

Besides the large numbers of changes to the existing objects, two new tables have been added in existing table spaces:

- ▶ **SYSIBM.IPLIST** has been added to DSNDB06.SYSDDF table space.
SYSIBM.IPLIST allows multiple IP addresses to be specified for a given LOCATION. Insert rows to this table when you want to define a remote DB2 data sharing group. The same value for IPADDR column cannot appear in both the SYSIBM.IPNAMES table and the SYSIBM.IPLIST table. Rows in this table can be inserted, updated, and deleted. Refer to Chapter 6, “e-business” on page 129 for additional usage information.
- ▶ **SYSIBM.SYSSEQUENCEAUTH** has been added to DSNDB06.SYSSEQ2 table space.
This table records the privileges that are held by users over sequences. Refer to 5.14, “Sequences and identity columns comparison” on page 117 for additional usage about sequences.

10.6 msys for Setup DB2 Customization Center

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 DB2 (see Figure 10-14.)

msys for Setup has been developed to automate all setup processes that do not require decisions by a system programmer and 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 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.

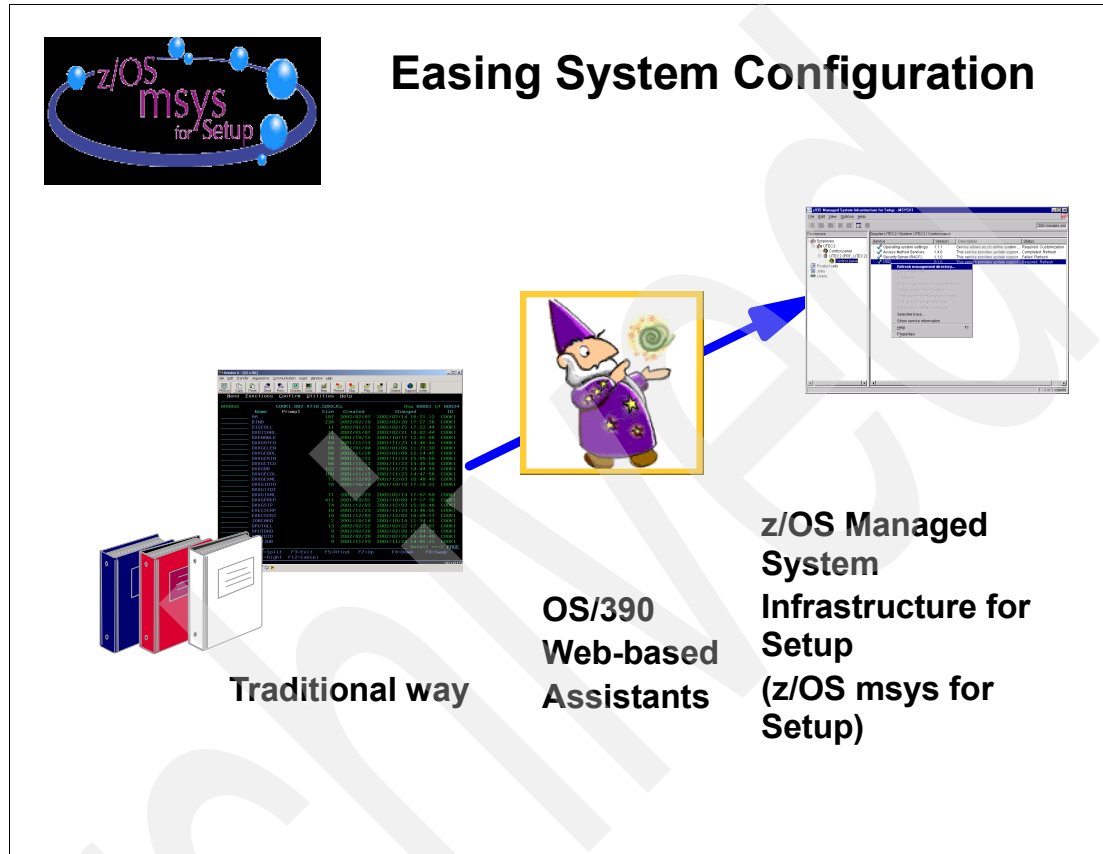


Figure 10-14 msys description

DB2 V8 base includes a DB2 for msys plug-in which provides installation support for DB2 for use within the z/OS msys framework. It requires msys for Setup, which is included with z/OS Version 1.3. The msys for Setup framework consists of 3 components:

- ▶ **msys workplace:** This runs on the workstation and provides the user with a Windows Explorer-style GUI to manage z/OS products
- ▶ **msys host program:** This resides and runs on a z/OS system and manages all installation/customization tasks
- ▶ **msys management directory:** This uses an LDAP Server and stores configuration data for all msys-enabled products.

More information about msys for Setup is available from the Web site:

<http://www.ibm.com/servers/eserver/zseries/msys/moreset.html>

10.7 Samples

New samples have been introduced to facilitate the use of new V8 functions. The new samples are related to:

- ▶ Multi-row fetch:

The existing sample program DSNTEP2 (sample dynamic SQL program in the PL/I language) now uses multi-row FETCH.

- ▶ Online schema:

The new sample is executed with four new steps in DSNTEJ1:

- The first step reduces the partitioning key on partition of table space DSN8S!!E
- The second step adds a fifth partition to the DSN8S!!E table space for the EMP table
- The third step reorganizes table space DSN8D!!A.DSN8S!!E
- The fourth step extends the length of a fixed char column in the PARTS table. It also convert a small integer field to a decimal type field in the EEMP table

Running the DSNTEJ1 job with DSN8S!!E table space started demonstrates that these steps can be done without stopping the table space.

- ▶ 2 MB SQL statement:

The sample programs DSNTEP2 and DSNTIAUL have been modified to handle SQL statements up to 2 MB in size.

- ▶ Materialized Query Tables (MQTs):

The new 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.

- ▶ > 18 char table/column names:

The sample program DSNTEP2 has been modified to handle greater than 18 character table or column names.

- ▶ Utilities Unicode Parser:

The new job DSNTEJ6R prepares and executes program DSN8ED8, a C language sample caller of the Utilities Unicode Parser (DSNUTILU) stored procedure. Since DB2 IVPs are provided in EBCDIC, DSN8ED8 uses z/OS Unicode Services to:

- Convert arguments for DSNUTILU statements from EBCDIC to Unicode
- Convert DSNUTILU results from Unicode to EBCDIC

Some enhancements have also been provided to existing functions:

- ▶ New MAXERRORS value:

A new MAXERRORS value has been added in DSNTEP2 during runtime. It allows the user to dynamically set the number of errors that DSNTEP2 will tolerate. The MAXERRORS value can be modified during runtime via the functional comment `--#SET MAXERRORS` being included in the SQL statements allowing the user to dynamically set the number of errors that DSNTEP2 will tolerate.

- ▶ SYSPRINT blocking in DSNTEP2:

A change in SYSPRINT blocking in DSNTEP2 will speed up the rate in which DSNTEP2 outputs results. The blocking size was very small before, thus impacting the performance when processing large result sets.

► New COBOL samples for the usage of LOBs:

There are four new COBOL samples that demonstrate the usage of LOBs:

- DSN8CLPL is a COBOL version of DSN8DLPL. It uses a LOB locator data type to populate BLOB columns greater than 32 KB in length. Job DSNTEJ76 prepares and runs the programs DSN8CLPL and DSN8CLTC.
- DSN8CLTC is a COBOL version of DSN8DLTC. It fetches the data back from DSN8CLPL and verifies that this was the same as the source data.
- DSN8CLRV is a COBOL version of DSN8DLRV. It demonstrates LOB locator functions, parsing a CLOB column to pull out resume data to be displayed on an ISPF panel. Job DSNTEJ77 prepares and runs the program DSN8CLRV.
- DSN8CLPV is a COBOL version of DSN8DLPV. It extracts the BLOB data that contains a photo image and displays it using GDDM. Job DSNTEJ78 prepares and runs the program DSN8CLPV.

► Support for Common Sample Tables:

A standard set of tables, indexes, and data are provided for usage across the DB2 family. The new job DSNTEJCS calls a new sample module, DSN8CST, which creates and populates the tables within a user-specified schema.

► Support for long tokens in formatted SQL messages:

DSNTIAR is being enhanced to support long name tokens in formatted SQL messages

► New sample Java stored procedure:

A new sample stored procedure, MRSPcli.java and MRSPsrv.java, returns multiple result sets back to the caller.

► WLM-managed stored procedures:

All the sample stored procedures have been converted to WLM managed 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.

Unicode definitions

DB2 V8 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, Korean, etc., you must also specify valid, non-zero CCSIDs for mixed-byte (MBCS) and double-byte (DBCS) character sets for both EBCDIC and ASCII.

DB2 uses the same CCSIDs for Unicode regardless of language. DB2's Unicode CCSIDs are: 367 for SBCS data, 1208 for MBCS data, and 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, consult the *z/OS V1R3.0 Support for Unicode Using Conversion Services*, SA22-7649-01, and go to the Web sites:

<http://www.ibm.com/downloads>

<http://www.s390.ibm.com/os390/bkserv/v2r10books.html>

A.1 Basic conversions

These z/OS Unicode Services conversion definitions are required by all DB2 V8 systems.

In planning your z/OS Unicode Services setup, you need to begin with a set of basic conversions between DB2's SBCS, DBCS, MBCS CCSIDs for Unicode, 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;
```

A.2 Additional conversions

These z/OS Unicode Services conversion definitions are required to use DB2 V8 samples:

- ▶ Some DB2-supplied sample programs require precompiler option CCSID(37) or CCSID(1047), 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;
```

These z/OS Unicode Services conversion definitions are required for your EBCDIC CCSIDs.

- ▶ If your DSNHDECP specifies an EBCDIC SBCS CCSID (SCCSID) other than 37 or 1047, you need 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;

- ▶ For 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;

These z/OS Unicode Services conversion definitions are required for your 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;

- ▶ For Far East languages, also add conversions between your ASCII MBCS CCSID (AMCCSID) and ASCII DBCS CCSID (AGCCSID) and each Unicode CCSID:

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;
```

These 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>

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 269.

- ▶ *A Practical Guide to DB2 UDB Data Replication V8*, SG24-6828
- ▶ *DB2 for z/OS and OS/390 Version 7 Using the Utilities Suite*, SG24-6289
- ▶ *DB2 for z/OS and OS/390 Version 7 Selected Performance Topics*, SG24-6884
- ▶ *DB2 for z/OS and OS/390 Version 7 Performance Topics*, SG24-6129
- ▶ *DB2 UDB Server for OS/390 and z/OS Version 7 Presentation Guide*, SG24-6121
- ▶ *DB2 UDB Server for OS/390 Version 6 Technical Update*, SG24-6108
- ▶ *DB2 UDB for OS/390 Version 6 Performance Topics*, SG24-5351
- ▶ *DB2 for z/OS Application Programming Topics*, SG24-6300
- ▶ *DB2 for OS/390 Version 5 Performance Topics*, SG24-2213

Other resources

These publications are also relevant as further information sources:

- ▶ *IBM DB2 UDB for z/OS V8 What's New?* manual available from the Web site:
<http://www.ibm.com/software/data/db2/os390/v8books.html>
- ▶ *The Evolution of Star Join Optimization*, whitepaper available from the Web site:
<http://www.ibm.com/software/data/db2/os390/techdocs/starjoin.pdf>
- ▶ *z/OS V1R3.0 Support for Unicode Using Conversion Services*, SA22-7649-01
- ▶ *OS/390 Integrated Cryptographic Service Facility Administrator's Guide*, SC23- 3975
- ▶ *DB2 UDB for OS/390 and z/OS Version 7 Installation Guide*, GC26-9936
- ▶ *DB2 UDB for OS/390 and z/OS Version 7 Command Reference*, SC26-9934
- ▶ *DB2 UDB for OS/390 and z/OS Version 7 Messages and Codes*, GC26-9940
- ▶ *DB2 UDB for OS/390 and z/OS Version 7 Utility Guide and Reference*, SC26-9945-02
- ▶ *DB2 UDB for OS/390 and z/OS Version 7 Programming Guide and Reference for Java*, SC26-9932
- ▶ *DB2 UDB for OS/390 and z/OS Version 7 Administration Guide*, SC26-9931
- ▶ *DB2 UDB for OS/390 and z/OS Version 7 Application Programming and SQL Guide*, SC26-9933-02
- ▶ *DB2 UDB for OS/390 and z/OS Version 7 Release Planning Guide*, SC26-9943
- ▶ *DB2 UDB for OS/390 and z/OS Version 7 SQL Reference*, SC26-9944
- ▶ *DB2 UDB for OS/390 and z/OS Version 7 Text Extender Administration and Programming*, SC26-9948

- ▶ *DB2 UDB for OS/390 and z/OS Version 7 Data Sharing: Planning and Administration*, SC26-9935
- ▶ *DB2 UDB for OS/390 and z/OS Version 7 Image, Audio, and Video Extenders*, SC26-9947
- ▶ *DB2 UDB for OS/390 and z/OS Version 7 ODBC Guide and Reference*, SC26-9941
- ▶ *DB2 UDB for OS/390 and z/OS Version 7 XML Extender Administration and Reference*, SC26-9949
- ▶ *DB2 UDB for OS/390 and z/OS Version 7 Diagnosis Guide and Reference*, LY37-3740
- ▶ *DB2 UDB Replication Guide and Reference Version 7*, SC26-9920
- ▶ *SQL Reference for Cross-platform Development Version 1.1*, available as PDF from:
<http://ww7b.boulder.ibm.com/dmdd/library/techarticle/0206sqlref/0206sqlref.html>
- ▶ *IBM eServer zSeries 900 z/OS 64-bit Virtual Storage Roadmap*, available as PDF from:
<http://www.ibm.com/servers/eserver/zseries/library/whitepapers/gm130076.html>

Referenced Web sites

These Web sites are also relevant as further information sources:

- ▶ DB2 for z/OS and OS/390
<http://ibm.com/software/data/db2/os390/ibm.com/software/db2zos>
- ▶ DB2 for z/OS and OS/390 Version 7 books
<http://www.ibm.com/software/data/db2/os390/v7books.html>
- ▶ Unicode
<http://ibm.com/servers/s390/os390/bkserv/latest/v2r10unicode.html>
- ▶ DB2 Image, Audio, and Video Formats
<http://www.ibm.com/software/data/db2/extenders/imgfmt.htm>
- ▶ Developing Java applications using DB2 Image and Audio Extenders
<http://ww7b.boulder.ibm.com/dmdd/library/techarticle/cox/0201cox.html>
- ▶ XML Extender WIZARD
<http://www.ibm.com/software/data/db2/extenders/xmlext/downloads.html>
- ▶ DAD examples
<http://www.ibm.com/software/data/pubs/papers/db2webservices/db2webservices.pdf>
- ▶ Building an XML application and writing a DTD
<http://www.ibm.com/developerworks/library/buildappl/writetd.html>
- ▶ z/OS UNIX System Services
<http://www.s390.ibm.com/products/oe>
- ▶ zSeries whitepapers
<http://www.ibm.com/servers/eserver/zseries/library/whitepapers/gm130076.html>
- ▶ DB2 Replication home page:
<http://www.ibm.com/software/data/dpropr>
- ▶ DB2 Cross-platform SQL Reference
<http://ww7b.boulder.ibm.com/dmdd/library/techarticle/0206sqlref/0206sqlref.html>

How to get IBM Redbooks

You can order hardcopy Redbooks, as well as view, download, or search for Redbooks at the following Web site:

ibm.com/redbooks

You can also download additional materials (code samples or diskette/CD-ROM images) from that site.

IBM Redbooks collections

Redbooks are also available on CD-ROMs. Click the CD-ROMs button on the Redbooks Web site for information about all the CD-ROMs offered, as well as updates and formats.

Archived

Archived

Abbreviations and acronyms

AIX	Advanced Interactive eXecutive from IBM	DDL	data definition language
APAR	authorized program analysis report	DLL	dynamic load library
AQR	automatic query re-write	DML	data manipulation language
AR	access register	DNS	domain name server
ARM	automatic restart manager	DPSI	data partitioned secondary index
ART	access register translation	DRDA	distributed relational database architecture
ASCII	American National Standard Code for Information Interchange	DSC	dynamic statement cache, local or global
BLOB	binary large object	DTT	declared temporary tables
CCA	client configuration assistant	EA	extended addressability
CCSID	coded character set identifier	EBCDIC	extended binary coded decimal interchange code
CD	compact disk	ECS	enhanced catalog sharing
CEC	central electronics complex	ECSA	extended common storage area
CF	coupling facility	EDM	environment descriptor manager
CFCC	coupling facility control code	ENFM	enabling new function mode
CFRM	coupling facility resource management	ERP	enterprise resource planning
CLI	call level interface	ESA	Enterprise Systems Architecture
CLOB	character large object	ESS	Enterprise Storage Server
CLP	command line processor	ETR	external throughput rate, an elapsed time measure, focuses on system capacity
CM	compatibility mode	FTP	File Transfer Program
CPU	central processing unit	GB	gigabyte (1,073,741,824 bytes)
CRLF	carriage return and line feed	GBP	group buffer pool
CSA	common storage area	GRS	global resource serialization
CTT	created temporary table	GUI	graphical user interface
DAD	document access definition	HA	Host adapter
DASD	direct access storage device	HFS	Hierarchical File System
DAT	dynamic address translation	HPJ	high performance Java
DB2 PM	DB2 performance monitor	I/O	input/output
DBAT	database access thread	IBM	International Business Machines Corporation
DBCLOB	double byte character large object	ICF	integrated catalog facility
DBET	database exception tables states	ICF	integrated coupling facility
DBD	database descriptor	ICMF	internal coupling migration facility
DBID	database identifier	IFCID	instrumentation facility component identifier
DBMS	database management system	IFI	instrumentation facility interface
DBRM	database request module	IPLA	IBM Program Licence Agreement
DCL	data control language	IRLM	internal resource lock manager
DDCS	distributed database connection services		
DDF	distributed data facility		

IRWW	IBM Relational Warehouse Workload	RBA	relative byte address
ISPF	interactive system productivity facility	RECFM	record format
ISV	independent software vendor	RID	record identifier
ITR	internal throughput rate, a processor time measure, focuses on processor capacity	ROT	rule of thumb
ITSO	International Technical Support Organization	RR	repeatable read
IVP	installation verification process	RRS	resource recovery services
JDBC	Java Database Connectivity	RRSAF	resource recovery services attach facility
JFS	journaled file systems	RS	read stability
JIT	Just in time (Java compiler)	RSM	Real Storage Manager
JNI	Java Native Interface	RTS	real time statistics
JVM	Java Virtual Machine	RVA	RAMAC Virtual Array
KB	kilobyte (1,024 bytes)	SDK	software developers kit
LCU	logical control unit	SMIT	System Management Interface Tool
LOB	large object	SPE	small program enhancement
LPAR	Logical Partition	SVL	IBM Silicon Valley Laboratory
LPL	logical page list	TCB	Task control block
LRECL	logical record length	USS	UNIX System Services
LRSN	log record sequence number	WAS	WebSphere Application Service
LVM	logical volume manager	WLM	Workload Manager
MB	megabyte (1,048,576 bytes)	WSAD	WebSphere Studio Application Developer
MQT	materialized query table	XML	eXtensible Markup Language
MSM	Multidimensional Storage Manager		
NFM	new function mode		
NPI	non-partitioning index		
NVS	Non-Volatile Storage		
ODB	object descriptor in DBD		
ODBC	Open Data Base Connectivity		
OLAP	Online Analytical Processing		
OS/390	Operating System/390		
PAV	Parallel Access Volume		
PDS	partitioned data set		
PIB	parallel index build		
PITR	point-in-time recovery		
PSID	pageset identifier		
PSP	preventive service planning		
PTF	program temporary fix		
PUNC	possibly uncommitted		
QBIC	query by image content		
QMF	Query Management Facility		
RACF	Resource Access Control Facility		

Index

Numerics

00D31033 167
0217 32
0225 32
-127 106
-173 79
2 GB bar 26
32 KB CI size 69
42801 79
5625-DB2 241
64-bit addressability 23
64-bit virtual storage 10
-805 153
-904 167
-952 136

A

ACCUMACC 166
ADD PARTITION 35, 171
adding index columns 55
adding partitions 60, 170
Advisory Reorg 59
Advisory Reorg Pending 53
ALTER BUFFERPOOL 29
ALTER INDEX 55
ALTER SEQUENCE 109, 118
ALTER TABLE ALTER COLUMN 116
AREO* 53, 59
ASCII precompiler option 126
ASENSITIVE 77
ATOMIC 89
automatic query rewrite 213

B

back up and restore system 16, 200
BACKUP SYSTEM 62, 200
backward index scan 18, 228
BOTH 198
BP32K 244
BSDS 36, 257
BUILD2 48, 180

C

CACHE 108, 113
CAF 165
Call Attach Facility 165
CASTOUT 29
CCSID 137
CF lock propagation 19
CF lock propagation reduction 236
CHARDEL 190
CI size 11, 69, 244

cluster tables 18
clustering 44, 56
clustering index 42
CM 250
COBOL samples 262
COLDEL 190
COLGROUP 198
common table expressions 80
compare unlike data types 17
compatibility mode 19, 243, 248
COMPJAVA 245
compression dictionaries 30
conditional restart control record 65
converting BSDS 244
COPY 184
COUNT 198
CRCR 65
CREATE SEQUENCE 107
CTHREAD 32
CURRENT PACKAGE PATH 15, 153
current versions 242
CURRENT_VERSION 57–58, 180
CYCLE 118

D

DATA ONLY 65
Data Partitioned Secondary Index 46
data set naming convention 35
data sharing 235
data space 27
database ALIAS 155
Database Exception Tables 59
DB2 Administration Server 5
DB2 Administration Tools 5
DB2 Control Center 5
DB2 features 3
DB2 for z/OS evolution 2
DB2 Installer 247
DB2 Management Clients Package 4
DB2 Replication Center 5
DB2 V8 requirements 33
DB2 Visual Explain 6
db2prof 133
db2sqljcustomize 134
DBD 30
DBD01 67
DBET 59, 70
DDF enhancements 15
DDL 51
DECPT 191
delimited Load and Unload 16
DESCRIBE 135
DESCSTAT 135
DISPLAY DATABASE 177

- DISPLAY GROUP DETAIL 255
- DISPLAY GROUPBUFFERPOOL 239
- DISPLAY LOCATION 168
- DISPLAY LOG 69
- distribution statistics 16
- DPSI 46
- DROP SEQUENCE 110, 118
- DSN1COMP 184
- DSN1COPY 184
- DSN1PRNT 184
- DSN9010I 168
- DSNB250E 70
- DSNB536I 28
- DSNB539I 29
- DSNB610I 28
- DSNDB06.SYSDDF 259
- DSNDB06.SYSSEQ2 259
- DSNHDECP 244
- DSNI005I 70
- DSNI021I 70
- DSNJCNVB 37, 257
- DSNJU004 257
- DSNR031I 69
- DSNTEJ6R 195
- DSNTEP2 75, 261
- DSNTIAD 75
- DSNTIAUL 75
- DSNTIJEN 254
- DSNTIJID 248
- DSNTIJIN 37, 247
- DSNTIJMV 247
- DSNTIJNE 254–255
- DSNTIJNF 254–255
- DSNTIJNG 254–255
- DSNTIJNH 254–255
- DSNTIJNR 254
- DSNTIJPM 247
- DSNTIJSG 248
- DSNTIJTC 248
- DSNTIJTM 248
- DSNTIJUZ 248
- DSNTIP00 252
- DSNTIP01 253
- DSNTIPT 252
- DSNUTILS 195
- DSNUTILU 195
- DSSIZE 34
- DSTATS 196

E

- EBCDIC 137
- e-business 14
- ECSA 33
- EDM pool 30
- EDM pool size 245
- element 146
- enable new function mode 19, 243, 248
- ENDING AT 173
- ENFM 251
- ENFM panels

- DSNTIP00 252
- DSNTIP01 253
- DSNTIPA1 247
- DSNTIPT 252
- enhanced scrollable cursors 12
- expressions in GROUP BY 13, 104
- external sort size 245

F

- fast log apply 67
- FLA 67
- FOR n ROWS 86
- forest 146
- FREQVAL 198
- FULL 65

G

- gaps 112
- GENERATED 118
- get diagnostics 12, 90
- GROUP BY 104
- groups 120

H

- hiperpools 26
- host variable array 89
- HPSIZE 28

I

- identity columns 117
- identity columns enhancement 13
- IFCID 69
- IFCID 0142 121
- IFCIDs 32
- IMMEDWRITE BIND option 238
- INCREMENT BY 118
- index classification 44
- index page splits 237
- index space name 182
- index-controlled partitioning 40
- indicator array 89
- indoubt units of recovery 238
- INSENSITIVE 75
- INSERT within SELECT
 - effect of updates and deletes 102
 - multi-row results 102
- install jobs changes 247
- IOFACTOR 182
- IRLM 2.2 244
- ISOLATION 79

J

- Java API 132
- JDBC 2.0 131
- JDBC 3.0 132

K

KSDS 257

L

LARGE 34
larger buffer pools 26
LEAST 198
linear data sets 69
L-locks 236
LOAD 183
LOAD delimited input 189
LOB 136
LOBs 31
LOBVALA 31
LOBVALS 31
log data sets 36
LOGONLY 66
long and variable length keys 18
long names 12, 72
long running UR backout 11
LPL 70
LPL recovery 11, 238

M

major changes to installation and migration 243
Managed System Infrastructure for Setup 259
Materialized Query Tables 209
materialized query tables 17
Materialized View 210
MAXERRORS 261
MEMLIMIT 33
memory management 23
migration paths 242
MINVALUE 118
MLS 119
MODIFY 184
more log data sets 10
more partitions 10
more tables in join 10
MOVE PAGE 22
MQSeries UDF 14
msys for Setup 247, 259
multilevel security 14, 119
multiple COUNT(DISTINCT) 13, 106
multiple SQL FETCH 84
multiple SQL INSERT 84
multi-row fetch and insert 12

N

nested elements 148
new function mode 19, 244, 248
NEWFUN 138
NFM 256
Non-Partitioned Secondary Index 47
NPSI 47
NUMCOLS 198
NUMPARTS 34, 41

O

ODBC SQL Connect user and password support 14
OLDEST_VERSION 57–58
online schema and utilities 16
Online schema changes 16
online schema changes 11, 50
online ZPARAMs 11, 67
OS/390 V2R10 22

P

PADDED 56, 178
PADIX 56, 226
page set control log record 64
parallel sort 18
PARTITION BY 41, 43
partitioned 44
partitioned index 42
partitioned secondary indexes 10
partitioning 44
partitioning index 42
partitions roll-on and roll-off 49
performance 31
piece 42
PIT 65
plans 54
PQ31326 36
PQ48126 36
PQ48486 250
PQ54042 233
PQ56293 203
PQ56295 203
PQ56296 203
PQ56323 161
PQ59549 123
PQ61458 225
PQ72337 202
private protocol 75
PSCR 64

Q

qualified column names in INSERT and UPDATE 13, 103

R

RACF 14
RBDP 42
rcount 92
REBALANCE 179
rebalancing partitions 61
REBUILD INDEX 183
Rebuild Pending 54
RECOVER INDEXSPACE 188
RECOVER TABLESPACE 187
Recoverable Resources Manager Services Attachment Facility 165
recovery 186
recursive SQL 80
Redbooks Web site 1, 269

- Contact us xxi
- referential constraint 173
- REORG 16
- REORG INDEX 182
- REORP 174
- REPAIR 185
- REPAIR VERSIONS 185
- RESET 174
- RESTART 202
- restart light 238
- RESTART(PHASE) 202
- RESTORE SYSTEM 62, 64, 200
- RETVLCFK=YES 226
- REUSE 174
- RID pool 29
- RIDLIST 30
- RIDMAP 30
- RIDs 30
- RMF 32
- rotating partition 173
- rotating partitions 61
- rowset 84, 86
- RRS signon 166
- RRSAF 165
- RUNSTATS 16, 185, 196, 245
- Runstats 54

S

- samples 261
- scalability 10, 21
- scalar fullselect 13, 94
 - CASE expression 98
 - nested 97
 - WHERE clause 96
- SCOPE PENDING 178
- scrollable cursors 75
- seclabel 120
- SecureWay Security Server 120
- SELECT from INSERT 98
- Select from insert 13
- SENSITIVE 75
- SENSITIVE DYNAMIC 77
- SENSITIVE STATIC 77
- sequences 13, 117
- sequences in applications 114
- SET DATATYPE 52
- SET LOG SUSPEND 62
- SET SYSPARM 67
- SIGNAL 15
- SMF 32
- sort storage 30
- SORTDATA 202
- SORTDEVT 199
- SORTKEYS 202
- SORTNUM 199
- sparse index for star join 18
- SPUFI 75
- SQL 12
- SQL Procedure 15
- SQL statement 2 MB long 12, 73

- SQL statements 2 MB long 12
- SQLCA 89
- SQLDA 89
- SQLJ 133
- STATWKnn 199
- storage monitoring and tuning 32
- stored procedures 15
- SYSCOLDIST 196
- SYSCOPY 174
- SYSIBM.IPLIST 259
- SYSIBM.LOCATION 155
- SYSIBM.SYSSEQUENCEAUTH 259
- SYSIBM.SYSSEQUENCES 117
- SYSLGRNX 174
- SYSSTATS 245
- system checkpoint and log offload activity 11
- system level point in time recovery 246
- system level point-in-time recovery 11, 62
- System Recover Pending 65

T

- table UDF block fetch 232
- table UDF cardinality 19, 230
- table-controlled partitioning 40
- tape parallelism 203
- thread pooling 166
- trigger 18, 229
- type 1 drivers 132
- type 2 drivers 132
- type 2 inactive threads 166
- type 3 drivers 132
- type 4 drivers 132
- type 4 JDBC driver 131

U

- UDFs 15
- Unicode 137, 194
- Unicode support 14
- UNIQUE 46
- Universal Driver 130
- Universal Driver for SQLJ and JDBC 14
- unlike data types 206
- UNLOAD 183
- UNLOAD delimited output 191
- UQ60475 144
- UQ60476 144
- UQ67433 225
- UQ67626 143

V

- VARCHAR 53
- variable length keys 225
- version information 57
- versioning 56
- views 54
- virtual storage expansion 25
- volatile tables 18, 229
- VPPSEQT 28

VPSEQT 28
VPSIZE 28
VPXSEQT 28

W

WebSphere Studio Application Developer 134
WITH 80
WSAD 134, 272

X

XLM2CLOB 146
XML 144, 153
XML built-in functions 146
XML data type 146
XML publishing 15
XML publishing functions 145
XMLAGG 151
XMLATTRIBUTES 148
XMLCONCAT 151
XMLELEMENT 147
XMLFOREST 149

Z

z/Architecture 21
z/OS architecture 22
z/OS recent versions 25
z800 23
z900 23
zSeries 22

Archived



DB2 UDB for z/OS Version 8 Technical Preview

(0.5" spine)
0.475" x 0.873"
250 x 459 pages



DB2 UDB for z/OS Version 8 Technical Preview



Browse the functional contents of the largest release ever

Understand the prerequisites and the setup for the new functions

Start planning for a smooth migration

IBM DATABASE 2 Universal Database Server for z/OS Version 8 (DB2 V8 throughout this IBM Redbook) is the twelfth and largest release of DB2 for MVS. It brings synergy with the zSeries hardware and exploits 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 major changes and enhancements made available with DB2 V8. It will help you understand the functions offered by DB2 V8, and provides enough information to start evaluating their applicability to your environment, as well as to start planning 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**

SG24-6871-00

ISBN 0738427462