

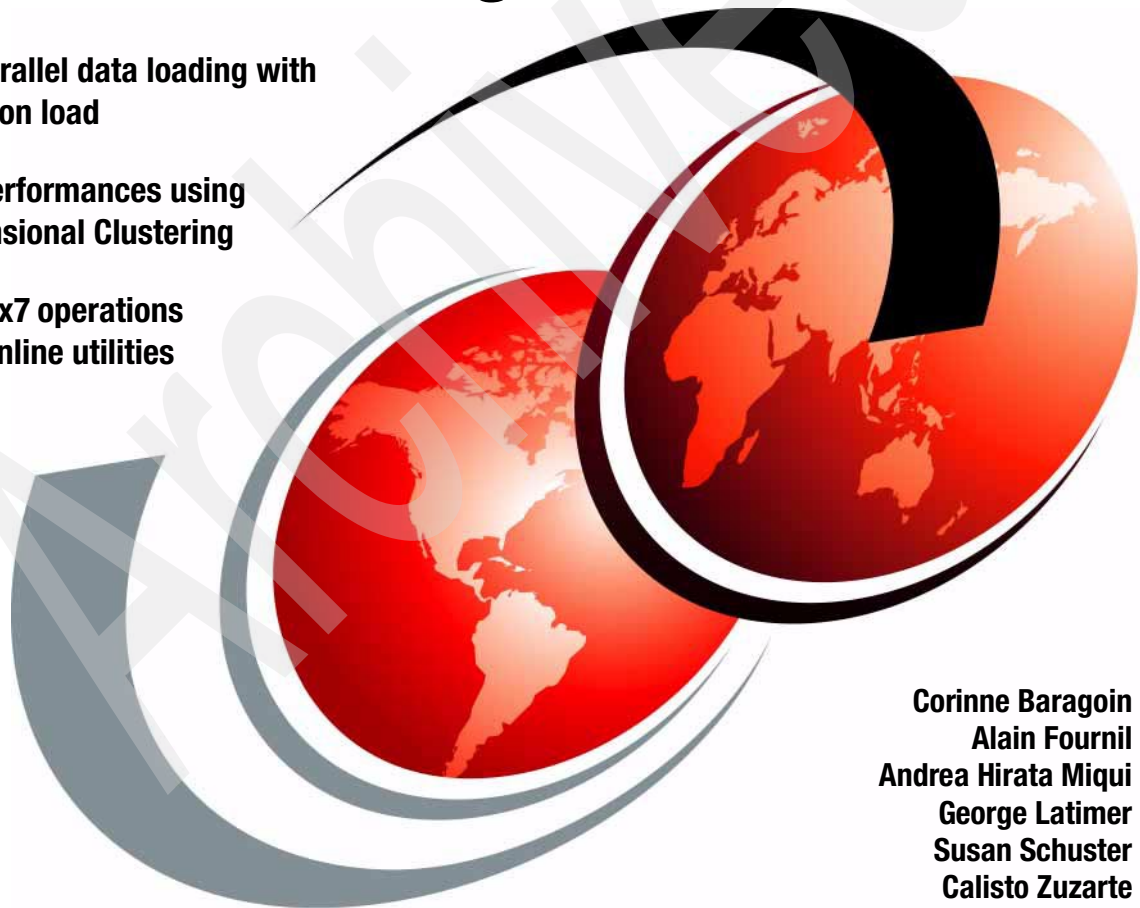
Up and Running with DB2 UDB ESE

Partitioning for Performance in an e-Business Intelligence World

Simplify parallel data loading with
multipartition load

Enhance performances using
MultiDimensional Clustering

Improve 24x7 operations
with new online utilities



Corinne Baragoin
Alain Fournil
Andrea Hirata Miqui
George Latimer
Susan Schuster
Calisto Zuzarte



International Technical Support Organization

**Up and Running with DB2 UDB ESE Partitioning for
Performance in an e-Business Intelligence World**

February 2003

Archived

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

First Edition (February 2003)

This edition applies to IBM DB2 Universal Database Version 8, Release 1.

Note: We recommend that you consult the product documentation 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	xx
Become a published author	xxii
Comments welcome	xxiii
Chapter 1. e-Business Intelligence and DB2 UDB ESE	1
1.1 Business issues and challenges	2
1.1.1 Build larger and larger data warehouses	3
1.1.2 Load and populate the data warehouse as fast as possible	3
1.1.3 Perform queries with performances	3
1.1.4 Operate in a 24 hours/7 days mode	4
1.1.5 Manage the data warehouse	4
1.1.6 Upgrade the data warehouse	4
1.1.7 Summary	4
1.2 DB2 UDB V8.1 functions for e-Business Intelligence	5
1.2.1 Performance enhancements	8
1.2.2 Availability enhancements	16
1.2.3 Manageability enhancements	18
1.2.4 UNION ALL views	19
1.3 New functionalities of IBM e-server p690	20
1.3.1 Hardware overview	21
1.3.2 Logical partitioning for OLTP-aware e-BI	22
1.3.3 Cluster 1600	23
1.3.4 Enhancements with AIX V5.2	24
1.3.5 Workload management	26
1.4 Case study and technical environment	31
1.4.1 The case study based on TPC-H	31
1.4.2 Technical environment	33
Chapter 2. Build a large data warehouse	37

2.1 Steps for building a large data warehouse	38
2.2 Disk considerations	39
2.2.1 Summary of the most popular RAID levels	39
2.2.2 Data placement	41
2.2.3 Log placement	42
2.2.4 Data availability and performance	43
2.2.5 General storage performance recommendations	43
2.3 Database design considerations	44
2.3.1 Understand data partitioning	44
2.3.2 Define the number of database partitions	46
2.3.3 Define database partition groups	49
2.3.4 Design the table spaces	52
2.3.5 Understand partitioning map	67
2.3.6 Choose the partitioning key	68
2.3.7 Size the tables	74
2.3.8 Size for MDC utilization	75
2.3.9 Size for MQT utilization	75
2.3.10 Configure DB2 UDB	75
2.3.11 Recommended parameters for performance	82
2.4 Installing and configuring DB2 UDB ESE V8.1	95
2.4.1 Generic checklist to install and configure DB2 UDB ESE V8.1	95
Chapter 3. LOAD and populate the data warehouse in parallel	111
3.1 Initial mass load	112
3.1.1 Online LOAD	112
3.1.2 Loading the case study data	113
3.1.3 Some considerations when using LOAD	117
3.1.4 Multipartition LOAD support	119
3.1.5 Partitioning LOAD subagents in DB2 UDB V8.1	126
3.1.6 LOAD QUERY command	129
3.2 Incremental updates and LOADs	129
3.2.1 Deleting data	130
3.3 LOAD and real-time notion	131
Chapter 4. Speed up performance with MultiDimensional Clustering	135
4.1 What is MultiDimensional Clustering?	136
4.1.1 Overview of the clustering index	136
4.1.2 Introducing MultiDimensional Clustering (MDC)	136
4.2 Design guidelines for MDC tables	142
4.2.1 Step 1: Identify dimension candidates	142
4.2.2 Step 2: Evaluate storage requirements	143
4.3 Creating a MDC table	147
4.4 Using a MDC table	149

4.4.1 Insert processing	149
4.4.2 Delete processing	150
4.5 MDC and multipartition database	150
4.6 Performance	155
4.7 Benefits of MDC	164
4.8 Considerations and recommendations	164
4.9 MDC design prototype.	165
Chapter 5. Speed up performance with Materialized Query Tables	167
5.1 MQTs overview	168
5.2 When to consider a MQT?	172
5.3 When will the MQT be used?	172
5.4 Intra database replicated tables and partitioning.	173
5.5 MQT and MDC?	176
Chapter 6. Enhance query performance in a partitioned environment .	177
6.1 Query performance and the DB2 compiler	178
6.2 Communication between partitions	178
6.2.1 Table queues	179
6.2.2 Table queue concepts.	179
6.2.3 Local bypass	180
6.3 Join partitioning strategies.	180
6.3.1 Collocated joins.	181
6.3.2 Directed joins	182
6.3.3 Broadcast joins	182
6.3.4 Repartitioned joins	183
6.4 Join planning using the join partitioning strategies	183
6.4.1 Replicated table joins	184
6.5 Sort and aggregation parallelization	185
6.5.1 Sort parallelization.	185
6.5.2 Aggregation parallelization	186
6.6 Statistics in a partitioned environment.	186
6.7 Analyzing query plans in a partitioned environment	187
Chapter 7. Improve 24x7 operations with new online utilities	191
7.1 Availability	192
7.2 Online table load	193
7.2.1 Improved methods of loading	193
7.2.2 Improved table space and table access while loading	194
7.2.3 Why, how, and when these features can be used	195
7.2.4 Load wizard utility	195
7.3 Online and classic reorganization of tables.	208
7.3.1 Why reorganize the data in a table?	208
7.3.2 How does an online reorganization benefit my environment?	209

7.3.3	What benefits will REORG by database partition provide?	210
7.3.4	When should the indexes on a table be reorganized?	211
7.3.5	Is there a way to monitor the REORG?	211
7.3.6	Is there a way to control the REORG?	212
7.3.7	Are there availability improvements for the classic REORG?	215
7.4	Index enhancements	216
7.4.1	How do the changes to the create index and rename help?	216
7.4.2	What are type-1 and type-2 indexes, and what do they mean?	218
7.4.3	Conversion of type-1 indexes to type-2 indexes	219
7.4.4	Online index reorganization	219
7.5	Online configuration parameters updates	220
7.5.1	Online bufferpool activities	223
7.5.2	Online changes to configuration parameters	223
7.5.3	How does online configuration improve availability?	224
7.6	Improve availability through backup/recovery plan	229
7.6.1	Backup and restore options	229
7.6.2	DB2 backup utility	231
7.6.3	Incremental backups	232
7.6.4	Backup using split mirror in conjunction with DB2 and ESS	233
7.6.5	DB2 full and incremental restore	236
7.6.6	Restore using split mirror in conjunction with DB2 and ESS	238
7.6.7	Parallel recovery	239
7.7	Additional ways to improve availability	241
7.7.1	DMS container operations	241
7.7.2	Online inspect command	244
Chapter 8.	Manage the data warehouse easily	247
8.1	Self Managing And Resource Tuning	248
8.2	Health Monitor	248
8.3	New wizards	252
8.3.1	Memory Visualizer	252
8.3.2	Storage Management	253
8.3.3	Configuration Advisor	254
8.3.4	Design Advisor	260
8.4	AUTOCONFIGURE command	261
Chapter 9.	Upgrade to DB2 UDB ESE environment	263
9.1	Upgrade considerations	264
9.1.1	Upgrade recommendations	264
9.1.2	Upgrade restrictions	265
9.1.3	Backing up databases before DB2 UDB upgrade	267
9.1.4	Space considerations for DB2 UDB upgrade	267
9.1.5	Recording system configuration for DB2 UDB upgrade	268

9.1.6	Changing the diagnostic error level before DB2 UDB upgrade.	269
9.1.7	Verifying that your databases are ready for migration	269
9.1.8	Understanding DB2 V8.1 objects terminology	269
9.2	64-bit considerations	270
9.2.1	Why 64-bit support?	270
9.2.2	64-bit restrictions on DB2 UDB V8.1.	271
9.2.3	64-bit client/server compatibilities	271
9.2.4	64-bit configuration on DB2 UDB V8.1	272
9.2.5	How to check if the DB2 UDB is 32-bit or 64-bit	274
9.3	Upgrade procedure	275
9.3.1	Environment used in the examples	275
9.3.2	From V7.2 32-bit to v8.1 32-bit on the same machine	277
9.3.3	From V7.2 32-bit to V8.1 32-bit on different machine	285
9.3.4	From V8.1 32-bit to v8.1 64-bit on the same machine	290
9.3.5	Upgrade strategies examples	293
Appendix A. Using DB2 LOAD for the case study		299
	Planning the load	300
	Load the multipartition tables	300
	Load the single partition tables, region, and nation	311
	Run SET INTEGRITY.	312
Appendix B. MDC dimension analyzer		315
	Overview of the dimension analyzer	316
	The script evaluate_dimensions.ksh	319
	How average row size is obtained	322
	The skeleton SQL file, evaluate_dimensions.sql	322
	A real example of use.	325
	Create a duplicate of the lineitem table.	325
	First cut	325
	Second cut, generated columns	327
	Creating lineitem as MDC table	330
	Summary of results	331
	Listing of evaluate_dimensions.ksh	332
Appendix C. Additional 64-bit migration considerations		351
	Environments that should not be migrated to Version 8.1	352
	Discontinued and deprecated functions	352
Appendix D. DB2 UDB configuration parameters		355
	Database manager configuration parameters summary	355
	Database configuration parameters summary	359
Appendix E. Additional material		365

Locating the Web material	365
Using the Web material	365
System requirements for downloading the Web material	366
How to use the Web material	366
Related publications	367
IBM Redbooks	367
Other resources	367
Referenced Web sites	368
How to get IBM Redbooks	368
IBM Redbooks collections	368
Index	369

Figures

0-1	Montpellier team: Andrea, Susan, George, Isabelle, Alain, Corinne . . .	xxi
1-1	DB2 UDB scalability	6
1-2	Multidimensional clustering example	9
1-3	AST and MQT	10
1-4	DB2 catalog caching on a cluster	12
1-5	Connection concentrator overview	13
1-6	DB2 UDB common client connectivity	15
1-7	INSERT through UNION ALL view example	20
1-8	Logical partitioning	22
1-9	Cluster 1600	24
1-10	CPU sparing	25
1-11	WLM concept	27
1-12	Logical partitioning or WLM	28
1-13	TPC-H tables design	32
1-14	p690 and SSA drives configuration	33
1-15	SSAs physical layout	35
1-16	p690 and ESS configuration	36
2-1	Physical partitions and logical database partitions	46
2-2	Number of logical partitions example	47
2-3	Database partition groups	50
2-4	Table space containers layout example	54
2-5	DMS table space structure	60
2-6	Partitioning map	68
2-7	Table collocation	73
2-8	Table sizes	74
2-9	DB2 Setup welcome	98
2-10	DB2 Setup wizard features	99
2-11	DB2 Setup install features	100
2-12	DB2 Setup select features	101
2-13	DB2 Setup instance setup	102
2-14	DB2 Setup instance use	103
2-15	DB2 Setup instance owner	104
2-16	DB2 Setup create new instance	105
2-17	DB2 Setup number of database partitions	107
3-1	PARTITION_AND_LOAD mode architecture overview	128
3-2	Ports and sockets used by multipartition LOAD	129
3-3	Real time components	132
3-4	Separating load on a large cluster of p690	133

4-1	MDC: dimensions.	137
4-2	MDC is using block indexes.	138
4-3	The slice for year = 2002.	139
4-4	The slice for country = Canada.	140
4-5	The slice for color = yellow.	140
4-6	The cell for dimension values (2002, Canada, yellow).	141
4-7	Star schema database for POPS.	156
4-8	Query 1: point query on block versus row ID index.	157
4-9	Query 2: range query on block versus row ID index.	158
4-10	Query 3: range query on two dimensions.	158
4-11	Query 4: query on a cell.	159
4-12	Query 5: full table scan.	160
4-13	Query 6: table scan with block predicate.	161
4-14	Query 7: Index ANDing of block and row ID indexes.	162
4-15	Query 8: Index ORing of block and row ID indexes.	163
4-16	Query 9: nested loop join.	163
4-17	Query 10: joining multiple tables.	164
5-1	MQTs overview.	169
5-2	Create and populate the MQT.	171
5-3	Replicated Materialized Query Table.	175
7-1	DB2 UDB high availability features.	193
7-2	Selection of tables to be loaded.	196
7-3	Select the type of load operation.	197
7-4	Type of load to be performed.	198
7-5	Input and output file specifications.	199
7-6	An example of the SQL Assist wizard.	200
7-7	Partitions to participate in the load.	201
7-8	Definition of the mapping of input columns to output columns.	202
7-9	Selection of performance and statistics options.	203
7-10	Failure options and recovery strategy.	204
7-11	Advanced options window.	205
7-12	Scheduling the execution of the load.	206
7-13	Task creation summary.	207
7-14	Task Center showing load task just created.	208
7-15	Online reorganization flow.	210
7-16	REORG status from a table snapshot.	212
7-17	Table snapshot showing that the REORG status is Paused.	213
7-18	Table snapshot showing that the REORG status is again started.	214
7-19	REORG entries from the history file for partition 1.	215
7-20	REORG INDEXES ALL FOR TABLE.	220
7-21	Configuration parameter files.	221
7-22	Partial output received for partition 0 from db2 get db cfg for pfp.	224
7-23	Partial output for partition 0 from db2 get db cfg forpfp show detail.	227

7-24	Types of incremental backups	232
7-25	Using split mirror for backing up the database	234
7-26	Using a split mirror copy to restore the primary database	238
7-27	Parallel recovery	239
7-28	The data was ALWAYS redistributed prior to Version 7	242
7-29	Redistribution was no longer a necessity in Version 7	242
7-30	Adding additional stripe sets	243
7-31	Dropping a container from a DMS table space	244
7-32	INSPECT command syntax	245
8-1	Configuring the Health Indicator settings	250
8-2	Excerpt from the database manager configuration	250
8-3	Health Indicator alert from the administration notification log	251
8-4	Recommendations for spilled_sorts health alerts	251
8-5	Storage Management view	254
8-6	Configuration Advisor: specify the database	255
8-7	Configuration Advisor: specify server specifications	256
8-8	Configuration Advisor: specify the workload	257
8-9	Configuration Advisor: Specify the typical database transaction	257
8-10	Configuration Advisor: Specify the database administration priority	258
8-11	Configuration Advisor: Specify database population	258
8-12	Configuration Advisor: Specify the number of connections	259
8-13	Configuration Advisor: Specify isolation	259
8-14	Configuration Advisor: specify scheduling	260
9-1	SP database partitions definition	276
9-2	p690 database partitions definition	277
9-3	From V7.2 32-bit to V8.1 32-bit on the same machine	277
9-4	From V7.2 32-bit to V8.1 32-bit on different machine	285
9-5	From V8.1 32-bit to V8.1 64-bit on the same machine	290
9-6	Example 1 upgrade environment	295
9-7	Example 2 upgrade environment	297

Tables

1-1	Software environment	33
1-2	Database partition group configuration	34
1-3	Bufferpool configuration	34
1-4	Table space configuration	34
2-1	Steps for building a large data warehouse.	38
2-2	Basic considerations between SMS and DMS table spaces	56
2-3	Page size relative to table space size	59
2-4	Suggested overhead and transfer rates for ESS	64
2-5	Required users and groups	97
2-6	db2nodes.cfg file	108
4-1	Size comparison	156
7-1	ESS highest performance	236
9-1	Catalog table space recommendations	267
9-2	Terminology differences.	269
9-3	64-bit compatibilities table	271
9-4	Memory-related database manager configuration parameters	272
9-5	Memory-related database configuration parameters	273
9-6	DB2 UDB level information	274
B-1	Description of buckets	317
B-2	Options for evaluate_dimensions.ksh	320
B-3	Comparison of MDC prediction versus MDC actual	332
D-1	Configurable database manager configuration parameters.	356
D-2	Database manager informational parameters	359
D-3	Database configuration parameters.	360
D-4	Database configuration informational parameters	362

Examples

2-1	Create database partition group command	51
2-2	Table space example for PREFETCHSIZE with RAID	61
2-3	ORDERS table creation command	69
2-4	Snapshot for database output	89
2-5	Snapshot output for database manager.	90
2-6	/etc/services entries for DB2 instance	106
2-7	/etc/services based on case study	107
2-8	db2nodes.cfg based on case study	108
2-9	db2start command	110
3-1	SET INTEGRITY prior to DB2 UDB V8.1.	113
3-2	Output from gen_and_load_one.ksh on lineitem table	114
3-3	Create exception schema and table	118
3-4	Setting DB2_PARTITIONEDLOAD_DEFAULT	119
3-5	PARTITIONED DB CONFIG clause of LOAD utility	120
3-6	Syntax for LOAD invocation of FILE_TRANSFER_CMD.	120
3-7	DB2ATLD_PORTS registry variable	124
4-1	Counting the number of cells for MDC.	144
4-2	SQL to examine rows per cell	145
4-3	Create a MDC table with single-column dimensions	148
4-4	Create MDC table with one multicolumn dimension	149
4-5	Rows per cell in lineitem table	151
4-6	Twenty lineitem table cells with lowest row counts	151
4-7	Calculating the number of extents	152
4-8	Accounting for the impact of partitioning	153
4-9	Compare dimension values in two partitions	154
5-1	Set CURRENT QUERY OPTIMIZATION level	173
5-2	Set CURRENT MAINTAINED TABLE TYPES register	173
5-3	Create a replicated Materialized Query Table	175
5-4	Create a MQT table multidimensionally clustered	176
6-1	Analyzing query access plan	187
7-1	REORG with nottruncate option	209
7-2	Sample of starting an online reorganization.	211
7-3	Sample of pausing an online reorganization	212
7-4	Sample of resuming an online reorganization	213
7-5	Steps create, rename, and drop an index	218
9-1	db2level output from DB2 UDB v7 instance before migration	280
9-2	db2 list database directory output before migration	280
9-3	db2imigr utility execution	281

9-4	db2level output from DB2 UDB v7 instance after migration.	281
9-5	db2 list db directory output after DB2 UDB v7 instance migration	281
9-6	Remote DB2 UDB V7 database connection	282
9-7	Local database connection before database migration	282
9-8	Migrate database command execution	284
9-9	Connect command.	284
9-10	Restore database example	287
9-11	db2level output before migration	292
9-12	Connect output before migration	292
9-13	db2iupdt command execution	292
9-14	db2level output after migration.	292
9-15	Connect output after instance migration	293
A-1	The load script for multipartition tables, gen_and_load_one.ksh	300
A-2	Loading the multipartition tables using gen_and_load_one.ksh	303
A-3	Number of rows loaded for each table	304
A-4	Output from gen_and_load_one.ksh for lineitem table	304
A-5	Example of LOAD QUERY during load of lineitem table	307
A-6	Example of LOAD QUERY after load of lineitem table completes.	310
A-7	Loading nation and region tables with IMPORT	311
A-8	Using SET INTEGRITY to return tables to normal state	312
B-1	Output showing good utilization of extents	318
B-2	Output showing poor utilization of extents	318
B-3	Syntax for evaluate_dimensions.ksh	319
B-4	The skeleton SQL file, evaluate_dimensions.sql	323
B-5	Example of a generated SQL file	324
B-6	Evaluate l_shipdate and l_receiptdate as dimensions	326
B-7	Evaluate two new generated columns for MDC table	328
B-8	DDL and SQL to create tpced.lineitem_mdc as MDC table.	330
B-9	Script evaluate_dimensions.ksh	332

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:

AIX 5L™	FlashCopy®	RETAIN®
AIX®	IBM®	RS/6000®
Balance®	Informix®	S/390®
IBM eServer™	iSeries™	Sequent®
Database 2™	MVS™	SPT™
DataJoiner®	NUMA-Q®	Tivoli®
DataPropagator™	OS/2®	Viewpoint™
DB2 Connect™	OS/390®	z/OS™
DB2 Extenders™	Perform™	zSeries™
DB2 Universal Database™	pSeries™	Lotus®
DB2®	PTX®	Word Pro®
DRDA®	Redbooks (logo)™ 	
Enterprise Storage Server™	Redbooks™	

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

Data warehouses in the 1990s were for the privileged few business analysts. Now, Business Intelligence is being democratized by being shared with the rank and file employee who demands higher levels of Relational Database Management System (RDBMS) scalability and ease of use. Ease of use is delivered through Web portals that act as single point of access for all information demands, and provide small snapshots of analyzed data to non Business Intelligence skilled professionals.

To support this emerging e-Business Intelligence world, the challenges that face the enterprises for their centralized data warehouse RDBMS technology are:

- ▶ Scalability and performance
- ▶ Reliability, Availability and Serviceability (RAS)
- ▶ Smart manageability

DB2 Universal Database Enterprise Edition and DB2 Universal Database Enterprise-Extended Edition have been merged with Version 8.1 into a single product: DB2 Universal Database Enterprise Server Edition (DB2 UDB ESE throughout this book).

This IBM Redbook focuses on the innovative technical functionalities of DB2 UDB ESE V8.1 in the areas of scalability, performance, availability, and manageability. This redbook explains how DB2, coupled with p690 systems, benefits e-Business Intelligence solutions for building the enterprise data warehouse.

This redbook positions the new functionalities, so you can understand and evaluate their applicability in your own data warehouse environment. It provides information and examples to help you to get started prioritizing and implementing the new functionalities.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the PSSC Benchmark Center in Montpellier (France).

Corinne Baragoin is a Business Intelligence Project Leader at the International Technical Support Organization, San Jose Center. She has over 17 years of experience as an IT Specialist in data management and Business Intelligence solutions. Before joining the ITSO in 2000, she worked as an IT Specialist for IBM France, supporting Business Intelligence technical presales activities, and assisting customers on DB2 UDB, data warehouse, and OLAP solutions.

Alain Fournil is an IT Specialist at the Montpellier Benchmarking Center. He has 8 years of experience in UDB EEE and AIX presales activities. His areas of expertise include performance and tuning of AIX and DB2 UDB for large configurations. He coauthored the redbook *Backup, Recovery, and Availability with DB2 Parallel Edition on RISC/6000 SP*, SG24-4695.

Andrea Hirata Miqui is a Data Management IT Specialist at ITS at IBM Brazil providing post-sales technical support since she joined IBM in 1996. She has 4 years of experience in DB2 UDB on UNIX and Intel platforms, supporting and implementing BI solutions in a wide range of companies in Brazil. Her areas of expertise include DB2 UDB partitioned environment implementation and support, as well as performance and tuning.

George Latimer is a Senior Database Consultant with Fourth Millennium Technologies, Inc., which is an IBM business partner in the USA. He has 28 years of experience in data processing. His areas of expertise include DB2 on MVS and UNIX, RS6000 SP system administration, and Tivoli Storage Manager implementation. He coauthored the *DB2 Cluster Certification Guide* and the redbook *Managing VLDB Using DB2 UDB EEE*, SG24-5105. George can be contacted at glatimer@fmtusa.com

Susan Schuster is a Senior DB2 Integration Professional with IBM USA, currently assigned to the Viewpointe Archive Services account. She has 15 years of experience in information systems development, processing, and support. She has worked with DB2 on distributed systems since its first release, and for the last 7 years has provided DB2 UDB technical support for production databases for a large variety of IBM and commercial customers. Her area of expertise is in implementation, support, and the performance tuning of DB2 UDB partitioned databases in very dynamic environments.



Figure 0-1 Montpellier team: Andrea, Susan, George, Isabelle, Alain, Corinne

Special thanks to **Calisto Zuzarte** for his input and writing. Calisto is the manager of the DB2 Query Rewrite Development Group at the IBM Toronto Lab. His key interest is in the SQL query performance area. His expertise spans the query, rewrite, and the cost-based optimizer components, both key components within the SQL compiler, which impact query performance. Other development projects include features in DB2 involving data integrity constraints. He also manages the DB2 interests within the Centre for Advanced Studies (CAS) based in Toronto.

Thanks to the PSSC pSeries Benchmark Center in Montpellier (France) for their support and more specifically to:

Isabelle Andary-Yriarte
Herve Sabrie
IBM PSSC Benchmark Center

Thanks to the following people for their specific contributions to this project and their input on MultiDimensional Clustering:

Keith Blanar
Gail Mueller
Aetna

Thanks to the following people for their involvement and input during the project:

Dwaine Snow
Glen Sheffield
IBM Toronto Lab

Dawn Seymour
Gus Branish
Randall Holmes
Ronald Sherman
IBM Customer Benchmark Group

Tim Malkemus
Sriram Padmanabhan
IBM Watson Research Center

Daniel Graham
IBM WW Sales

Thanks to the following people for their reviews and contribution on this redbook:

Leslie Cranston
Haider Rizvi
Kwai Wong
IBM Toronto Lab

Bishwaranjan Bhattacharjee
IBM Watson Research Center

Bruce Baumbush
IBM WW Marketing

Simon Woodcock
IBM EMEA Technical Support

Jacques Milman
IBM France

Diana Gfroerer
Angel Gonzales
IBM Germany

Nagraj Alur
International Technical Support Organization, San Jose 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

e-Business Intelligence and DB2 UDB ESE

This chapter gives an e-Business Intelligence (e-BI) overview that covers the following topics:

- ▶ The business issues that organizations have to face nowadays to stay competitive in an e-Business Intelligence world
- ▶ The new DB2 UDB ESE V8.1 functions to use when building and using a large data warehouse
- ▶ The scalability considerations when using a p690 machine
- ▶ Our technical environment

1.1 Business issues and challenges

The emergence of the Internet and e-business as e-commerce have created new opportunities and challenges for enterprises to adapt, evolve, and innovate at an even more rapid pace, so as to stay competitive. Business Intelligence is no longer a background process.

e-analytics is the next wave of Business Intelligence (BI). e-analytics combine traditional data warehouse infrastructure with BI inside transaction applications, which deliver BI to every employee's desktop, and include unstructured data in a comprehensive information supply chain.

Beyond the initial Business Intelligence scope of gathering data together into a single integrated and consolidated view, and storing historical data in a data warehouse to be able (through data analysis) to make strategic business decisions, new business requirements are emerging as:

- ▶ Tight integration of operational and information systems for data and business processes for building real time or near real time solutions
- ▶ Immediate access by customers to a fully integrated view of business and analytics through the Web

These requirements need the technology infrastructure to be designed and built so that these types of operations can be supported and maintained:

- ▶ To simplify and automate information flow through the enterprise for the interoperation of processes between operational data and data warehouse
- ▶ To enable rapid accommodation to any change
- ▶ To facilitate all customers contacts with the organization in a more customer-centered infrastructure

When building their data warehouses today, enterprises will have to face the following challenges:

- ▶ Build and manage larger and larger data warehouses
- ▶ Load and populate the data warehouse as fast as possible, based on their business cycle
- ▶ Perform queries as fast as possible and control them
- ▶ Operate in 24 hours, 7 days a week (or 24x7) mode and be able at the same time to perform queries and data warehouse operations without stopping the database as the Web does not know any time frames
- ▶ Manage the data warehouse with the lowest administrative overhead

- Upgrade their existing environment to a more scalable one, taking in account the data warehouse growth and performance

1.1.1 Build larger and larger data warehouses

Data warehouses are becoming larger and larger, and organizations have to handle high volumes of data with multiple terabytes of stored raw data. To take in account data warehouses growth, Relational Database Management Systems (RDBMS) have to demonstrate near linear scalable performance as additional computing resources are applied. The administrative overhead should be as low as possible.

1.1.2 Load and populate the data warehouse as fast as possible

To improve business value when providing analysis through the Web; to make some cost savings when maintaining data warehouses; and to minimize the latency between data sources and data availability in the data warehouse, organizations are looking more and more to load and populate the data warehouse as fast as possible, using parallelism, and taking into account the business cycle and the data availability requirements. This requires fast and parallel bulk loaders as integrated tools that can refresh the data warehouse.

1.1.3 Perform queries with performances

To improve business value when providing analysis through the Web, data warehouses require superior performance and flexibility in the data access for e-analytics.

E-analytics require support of queries against data stored in multiple dimensions, as typically found within cubes and star schemas, but not only.

Workloads in a data warehouse vary from customer to customer. They can be:

- Large, concurrent reports, meaning either a single query workload or the population application workload running independently
- Mixed workloads, meaning running at the same time:
 - Short running queries
 - Long running complex queries
 - Well-known queries
 - Random queries
 - Occasional updates to the data warehouse
 - Online utilities

The challenge is always to drive the workload as fast as possible with little prior setup.

Running workload with high performance requirements need the ability to monitor the queries and measure their performances to be able to adjust them when performance drops. It also requires being able to prioritize the different queries, updates, and utilities in a given workload, and to handle high user concurrency.

1.1.4 Operate in a 24 hours/7 days mode

Since the Web e-analytics application may be accessed in any time zone, the availability must be close to 24 hours a day and 7 days a week. This requires running database operations online and keeping both read and write access optimal during data load, data reorganization, and any database maintenance operations without performance repercussion on queries performances.

1.1.5 Manage the data warehouse

To reduce the complexity of data administration tasks specifically in a large enterprise data warehouse environment, enterprises require more automation and simplification of maintenance tasks, which are needed by this environment.

1.1.6 Upgrade the data warehouse

Customers already set up with DB2 UDB Extended Enterprise Edition on a SP machine may require (for growth and performances reasons) to upgrade to DB2 UDB ESE V8.1 on a p690 machine, so as to take advantages of both the software's and hardware's best performance combination.

1.1.7 Summary

To satisfy the above customers requirements, the data warehouse solution must provide the following characteristics:

- ▶ **Scalability**

Building large data warehouses means that both hardware and software components should be scalable up and down to achieve the desired performance and throughput rates of your enterprise.

- ▶ **Performance**

Performance issues address both the data warehouse population (or load) and the workloads.

- A data warehouse should be populated (loaded) and incrementally updated as fast as possible.
 - A data warehouse environment should handle any question at any time without degradation of the mission critical/time sensitive workload. Complex queries should handle aggregations, full-table scans, and multiple table joins with little or no performance degradation. Multidimensional queries should be satisfied without any joining performance degradation.
- **Availability**
- A data warehouse should operate 24 hours, 7 days a week.
- **Manageability**
- A data warehouse environment should be flexible and extensible, and minimize the administrative costs of a high volume databases to provide:
- Automatic processing parallelization
 - Advanced queries optimizations
 - Administration tool ease-of-use

1.2 DB2 UDB V8.1 functions for e-Business Intelligence

The main considerations in an e-Business Intelligence environment are related to scalability, performance, availability, and manageability.

DB2 UDB Enterprise Edition and DB2 UDB Enterprise-Extended Edition have been merged with V8.1 into a single product: DB2 UDB Enterprise Server Edition (ESE) bringing the best of the two worlds on both SMP and MPP environment and deliver the right data management solutions for e-Business Intelligence solutions.

DB2 UDB ESE combined with the pSeries, IBM eServer, and storage has the ability to keep up with this growth, providing rapid response for complex query workloads on a very large scale.

According to Winter Corporation, large data warehouses are projected to grow 169 percent by 2004. According to the firm, based on a recent survey of companies with large databases, the number of data warehouses containing at least 10 terabytes of data is expected to quintuple over the next two years. Other analysts believe that by 2004, enterprises will need to process 30 times as much data as they handled three years ago.

DB2 UDB ESE V8.1 with Database Partitioning Feature (DPF) allows to spread your data warehouse. Partitioning or the ability to create and manage multiple

database partitions requires DPF feature either on a single server or multiple servers. If you want to create multiple database partitions on a single SMP server, or you want to create multiple database partitions across more than one physical server (that is, a clustered hardware configuration), you must acquire DPF in addition to DB2 UDB ESE. The choice of partitioning can be done a posteriori, with DB2 UDB ESE already running. New physical servers can be added as need so you do not need to make all of your decisions today about your future growth. Not only can new servers be added, but partitions can be created within the existing server as well.

Figure 1-1 shows the DB2 UDB scalability.

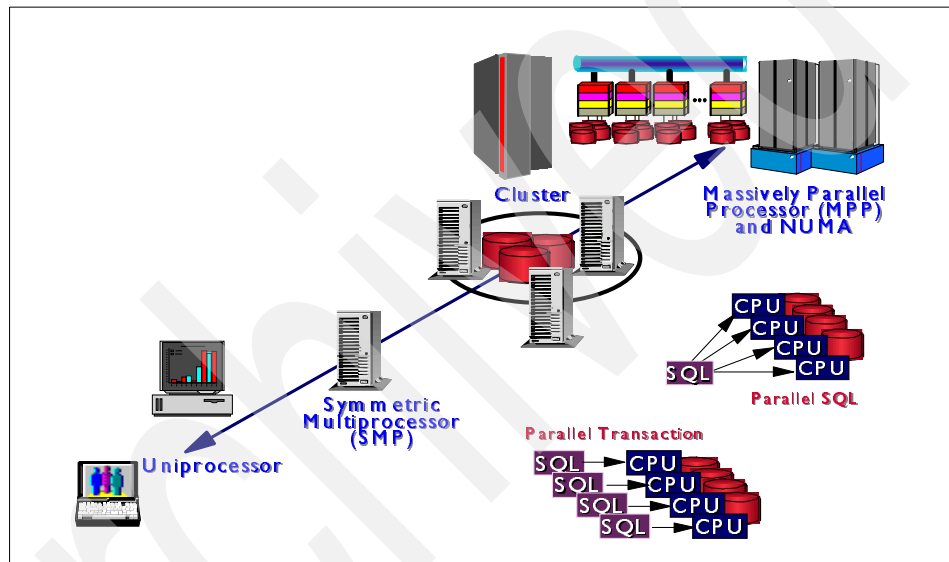


Figure 1-1 DB2 UDB scalability

DB2 UDB V8.1 not only delivers a wealth of manageability and availability features, but also delivers a set of easy-to-use performance enhancements. This means you can spend more time analyzing business information, and less time on performance improvement and tuning issues.

The new v8.1 major to benefit large data warehouses are:

- Performance enhancements
 - MultiDimensional Clustering (MDC)

MultiDimensional Clustering of tables reduces the indexing workload for your DBAs, while providing guaranteed data clustering for fast data query. This means you get better decision-making information sooner, with less system management overhead, and improved data warehouse

applications. In an e-BI environment, it is very useful for OLAP style hierarchical analysis, such as executive reports and summaries.

For example, simply clustering the line item and order tables (TPC-H) by date caused query performance to improve significantly. In addition, the simplicity of MDC is evident when looking at the **CREATE TABLE** statement with MDC.

- **Materialized Query Table (MQT)**

MQTs allow you to pre-compute the results of queries, which are often issued repetitively against large volumes of data, avoiding the repeated scanning that might have been needed to answer these queries.

This allows you to spend more time analyzing business information, and less time on performance improvement and tuning issues.

- **Full 64-bit support**

DB2 UDB V8.1 provides new 64-bit support enabling increased performance of database servers and their database applications. You can utilize much larger bufferpools, SORTHEAP, package cache, and similar other resources that consume large amounts of memory, providing better scalability.

For example, a p690 configured with 256 GB of memory and the 64-bit version of DB2 UDB ESE V8.1 were able to fully utilize all available real memory running on a 64-bit kernel with AIX V5.2.

- **New **LOAD** utility enhancements**

The first tests with DB2 UDB ESE V8.1 loader were done against 10 TB of data on a p690 server, and DB2 UDB V8.1 demonstrated that it was able to provide a truly scalable load utility.

- **New optimizer and query rewrite enhancements**

DB2's innovative query rewrite and optimization technologies, and performance configuration capabilities, lead the industry. In every release, the DB2 optimizer is enhanced to deliver more query rewrite transformations and improved optimizations. V8 has delivered a number of new optimizations and query rewrite enhancements that can take complex queries and optimize them to fully utilize the power of both small and very large clusters.

- **Significant improvements in hash join processing**

The use of hash join within DB2 UDB has been significantly enhanced in V8.1 and is now available by default to the optimizer (prior to V8.1, a registry variable was required to turn hash joins on).

- Improved prefetching performance

With processors increasing in performance faster than disks, it is vital for data warehouses to have high performing, intelligent prefetching. Forcing an agent to wait on I/O or to have an agent perform its own I/O (rather than working on the data in memory) results in slower query performance. The effectiveness of the DB2 prefetchers may turn many of large IO bound queries into CPU bound queries. As an example, DB2 UDB was able to push the peak total throughput of all the raid devices across five p690 servers up to 29.7GB/sec with 223,000 I/Os per second.

- ▶ Availability enhancements

- DB2 UDB V8.1 provides online utilities like index rebuild, index create, table load and **RUNSTATS**, as well as allowing many configuration parameters to be changed without stopping the database. This means improved overall performance and high availability.

- ▶ Data maintenance enhancement

As an example, one useful SQL statement enhancement allows the INSERT through UNION ALL view that supports SELECT, UPDATE, DELETE and INSERT. In an e-BI environment, it is often useful to have distributed partitioned views by date to facilitate data maintenance.

- ▶ Manageability enhancements

There are many; as an example, you can set the DB2 UDB variable (BLK_LOG_DSK_FUL) that specifies that the DB2 UDB should not fail in a log disk full condition. Another enhancement is that you can have larger logs (256 GB) allowing your e-BI environment to have larger workloads and more availability.

1.2.1 Performance enhancements

The new performance enhancements provide many new possibilities to increase performance from your database servers, so getting better decision-making information sooner and with less system management and tuning overhead.

The main performance enhancements to consider are:

- ▶ MultiDimensional Clustering (MDC)
- ▶ Materialized Query Table (MQT)
- ▶ Catalog caching
- ▶ Connection concentrator
- ▶ Enhanced RUNSTATS command
- ▶ 64-bit support

MultiDimensional Clustering (MDC)

To enforce automatic clustering of data physically in multiple dimensions, MultiDimensional Clustering (MDC) provides a new kind of index called block index. It directs data related to specific multidimensional keys to specific logical grouping of data stored altogether. This results in significant improvement in the performance of queries, as well as significant reduction in the overhead of data maintenance operations, such as reorganization, and index maintenance operations that occur during insert, update, and delete operations. All these improvements are because MDC clustering is automatically and dynamically maintained over time; that is, reorganization is not necessary. Indexes on dimensions are block based and are much smaller than row indexes. It is perfect for OLAP-style hierarchical analysis such as executive reports and summaries.

DB2 UDB maintains the physical order of data pages in the key order of the index, as records are inserted and updated in the table. Clustering indexes greatly improve the performance of range queries that have predicates containing one or more keys of the clustering index. With good clustering, only a portion of the table needs to be accessed, and when the pages are sequential, more efficient prefetching can be performed.

One example of MDC on two dimensions:

- Region (east, north, south, west)
- Year (97, 98, 99, 00)

If you create a multidimensional clustering on both dimensions, DB2 UDB will create one block for each dimension combination as shown in Figure 1-2.

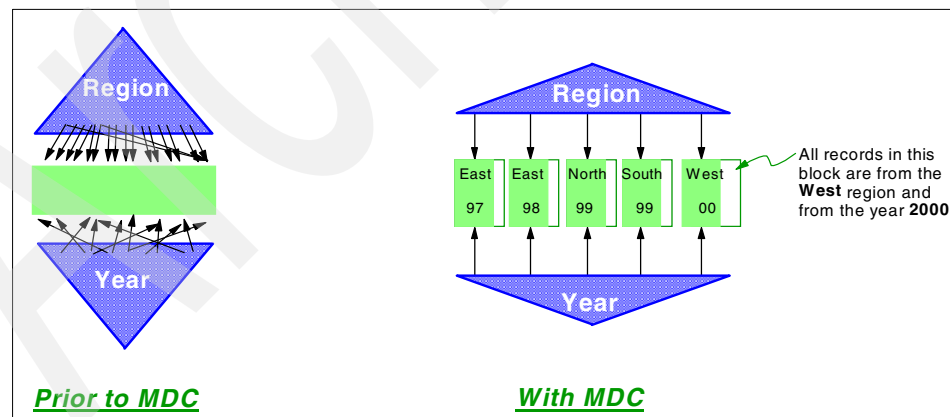


Figure 1-2 Multidimensional clustering example

Materialized query table

In an e-BI environment, many applications have pattern of analysis using more or less similar queries with only minor variations in a query's predicates, and they are often issued repetitively against large volumes of data.

For example:

- ▶ Query X might request monthly information for printers for the eastern region of France over the past four months.
- ▶ Query Y may request that the number of items sold to a specific electronic store group in each month of the current year.
- ▶ Query Z may request the same kind of information for only the month of January for all regions in France.

The result of these queries are almost always expressed as summaries or aggregates, which easily involve looking at millions of rows stored in one or more tables, which would need to be scanned repeatedly to answer these queries. In such cases, the query performance can be poor.

Materialized Query Table (MQT) was introduced to address this kind of performance problem. MQT is a new term used by DB2 UDB, and covers tables that contain summary data, Automatic Summary Tables (ASTs), replicated tables, and some tables that are not aggregated.

Figure 1-3 shows the AST with aggregation, and MQT without aggregation.

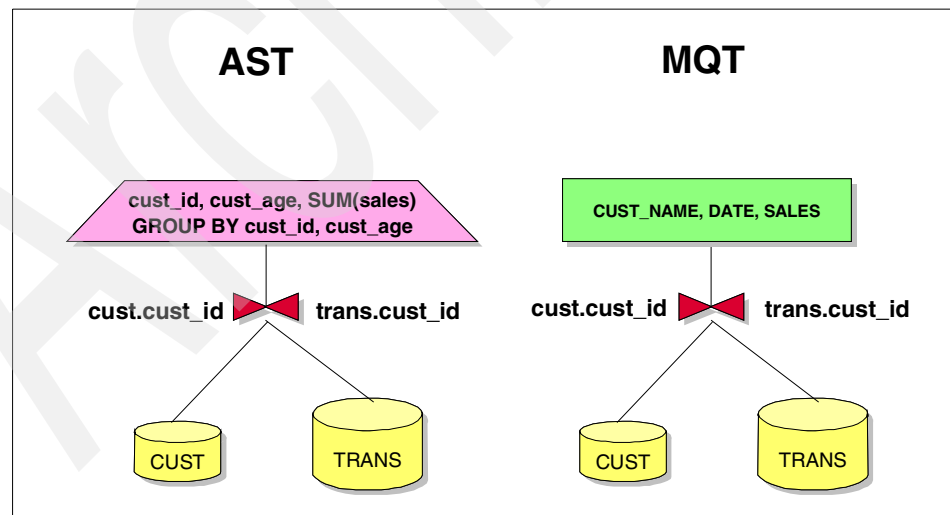


Figure 1-3 AST and MQT

MQT is a powerful feature in DB2 UDB that allows you to precompute and materialize a query result into a table, and it includes:

- ▶ Automatic Summary Tables (ASTs) where we have aggregation.
- ▶ Single table queries or join queries with no aggregation. Single table MQT is good for isolating hot data where more indexes can be defined on less data. Join queries are good for speeding up queries where hot data is pre-joined.
- ▶ Replicated MQTs where a replica of a table or a subset of it is placed on each database partition of the partition group in a partitioned environment. These tables are created to encourage collocated joins.
- ▶ Re-partitioned MQTs which are a refresh deferred table that allows you to copy a table or subset with different partitioning.

Since the materialized table often contains precomputed summaries and/or a filtered subset of data, it would tend to be much smaller in size than the base tables. When a user query accessing the base tables is automatically rewritten by the DB2 optimizer to access the materialized table instead, then significant performance gains can be achieved.

Catalog caching

This feature extends the existing system catalog and authorization data cache to provide a cache at each partition of the partitioned database.

In an e-BI environment, workloads often have many dynamic SQL statements and/or operations that require authorizations to be verified for database access, user-defined functions, and stored procedures; these will show improved performance from this feature. Applications that currently connect to non-catalog partitions will benefit the most by this enhancement by enabling SQL to execute on the current DB2 database partition, without having to go back to the coordinator partition to do binds or authorization lookups.

An example of DB2 catalog cache on a cluster environment is shown in Figure 1-4.

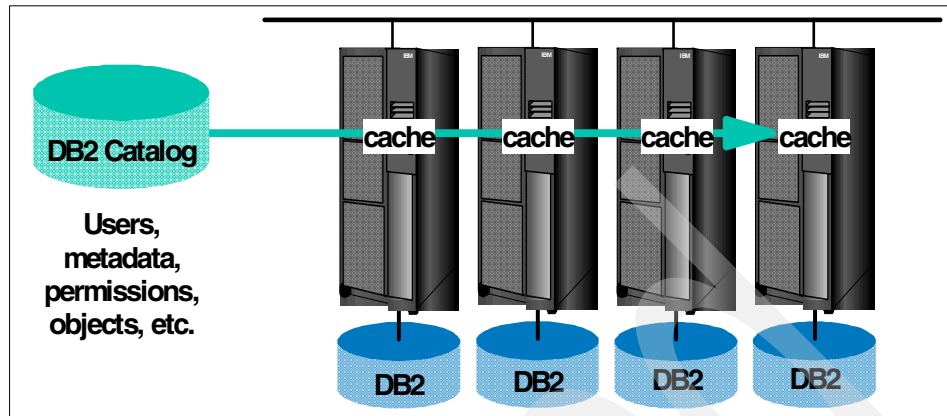


Figure 1-4 DB2 catalog caching on a cluster

Connection concentrator

Web requests with many relatively transient connections or similar kinds of Operational Data Store, or OLTP-like requests can be found in a data warehouse environment for specific applications. The connection concentrator improves performance by allowing many more client connections to be processed efficiently. It also reduces memory use for each connection, and decreases the number of context switches.

For example, the connection concentrator (see the overview in Figure 1-5) will provide performance benefits in an e-BI environment that has many Web client connections, due to more efficient connection manageability.

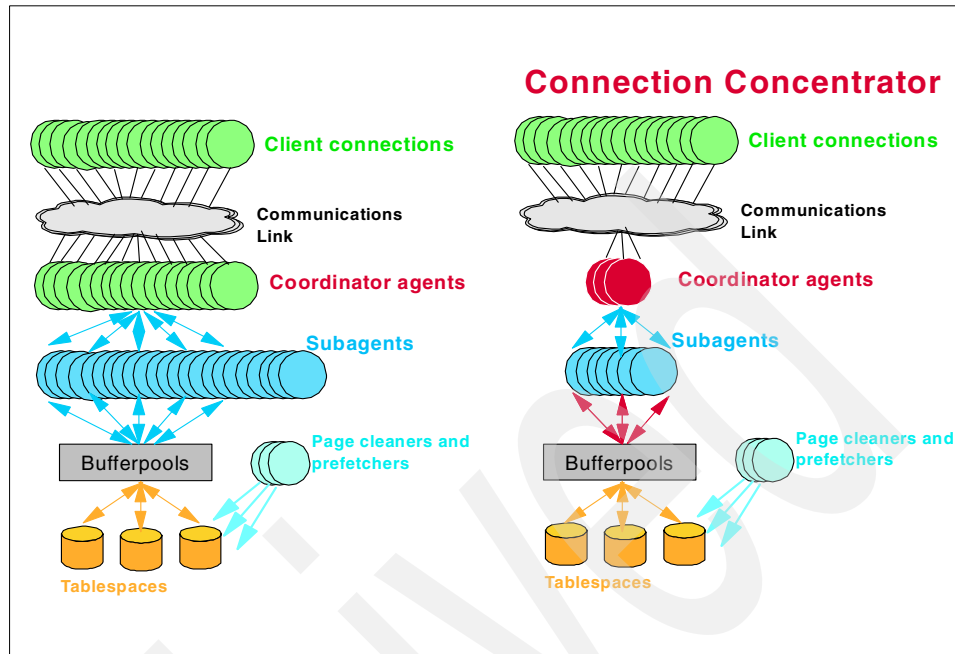


Figure 1-5 Connection concentrator overview

Note: Connection concentrator is enabled by setting `MAX_CONNECTIONS > MAX_COORDAGENTS`, for example:

```
update dbm cfg using max_connections 10000 max_coordagents 100
```

Enhanced RUNSTATS command

Updating statistics about the physical characteristics of a table and the associated indexes such as: the number of records; the number of pages; and the average record length is very important since the DB2 optimizer uses these statistics when determining access paths to the data.

The **RUNSTATS** command is invoked when a table has many updates, or is reorganized and in the e-BI environment, and has important benefits like:

- ▶ Faster performance from better statistics for Optimizer
- ▶ Less DBA labor and more control over statistics collection

In DB2 UDB V8.1, the **RUNSTATS** utility has been enhanced to improve the performance of statistics collection, and to gather statistics during **Create Index** processing:

New options are:

- ▶ Accept a list of index names
- ▶ Accept a list of columns for which we want to collect statistics
- ▶ Accept distribution statistics limits: *NUM_FREQVALUES* and *NUM_QUANTILES* values at the table level as well as the column level.
- ▶ Collect statistics on column combinations
- ▶ “Like” statistics are now controlled with a column option.

When using **RUNSTATS**, some of the different possibilities are:

- ▶ Collect statistics on the table only, on all columns without distribution statistics:

```
RUNSTATS ON TABLE db2user.employee
```

- ▶ Collect statistics on the table only, and on columns empid and empname with distribution statistics:

```
RUNSTATS ON TABLE db2user.employee WITH DISTRIBUTION ON COLUMNS (empid, empname)
```

- ▶ Collect statistics on the table only, on all columns with distribution statistics using a specified number of frequency limit for the table while picking the *NUM_QUANTILES* from the configuration setting:

```
RUNSTATS ON TABLE db2user.employee WITH DISTRIBUTION DEFAULT NUM_FREQVALUES 50
```

- ▶ Collect basic statistics on all indexes only:

```
RUNSTATS ON TABLE db2user.employee FOR INDEXES ALL
```

- ▶ Collect basic statistics on the table, and all indexes using sampling for the detailed index statistics collection:

```
RUNSTATS ON TABLE db2user.employee AND SAMPLED DETAILED INDEXES ALL
```

- ▶ Collect statistics on table, with distribution statistics on columns empid, empname, and empdept and the two indexes Xempid and Xempname. Distribution statistics limits are set individually for empdept, while the other two columns use a common default:

```
RUNSTATS ON TABLE db2user.employee WITH DISTRIBUTION ON COLUMNS (empid, empname, empdept NUM_FREQVALUES 50 NUM_QUANTILES 100) DEFAULT NUM_FREQVALUES 5 NUM_QUANTILES 10 AND INDEXES Xempid, Xempname
```

- ▶ Collect statistics on critical columns only (index and foreign key columns).

```
RUNSTATS ON TABLE db2user.employee ON KEY COLUMNS AND INDEXES ALL
```

- ▶ Collect statistics on all columns without distribution except for one column. Consider T1 containing columns c1, c2, ..., c8:


```

RUNSTATS ON TABLE db2user.T1
  WITH DISTRIBUTION ON COLUMNS (c1, c2, c3 NUM_FREQVALUES 20
    NUM_QUANTILES 40, c4, c5, c6, c7, c8) DEFAULT NUM_FREQVALUES 0,
    NUM_QUANTILES 0 AND INDEXES ALL
or
RUNSTATS ON TABLE db2user.T1 ON COLUMNS (c1, c2, c4, c5, c6, c7, c8)
  WITH DISTRIBUTION ON COLUMNS (c3 NUM_FREQVALUES 20 NUM_QUANTILES 40)
  AND INDEXES ALL

```

64-bit support

The advent of 64-bit computing platforms presents new possibilities for increased performance of database servers as well as database applications. The 32-bit platforms have an inherent address space limitation of 4 gigabytes (GB) for the kernel, plus user text and data. Removal of this 4 GB limit on the address space of database servers allows for the creation of larger bufferpools, SORTHEAP, package caches, and other resources that can consume large amounts of memory. This, in turn, leads to much improved performance, especially for sort and input/output (I/O) operations.

The DB2 UDB ESE V8.1 common client supports also both 32-bit application and 64-bit application, and accesses both 32-bit and 64-bit DB2 UDB servers.

Large e-BI applications can gain significant performance advantages since the 64-bit database can keep many of its tables, and possibly file data in memory, and utilize a larger number of buffers for its operation. This will minimize the amount of I/O, and thus increase performance; however, you must have sufficient real memory on the system to reap the benefits.

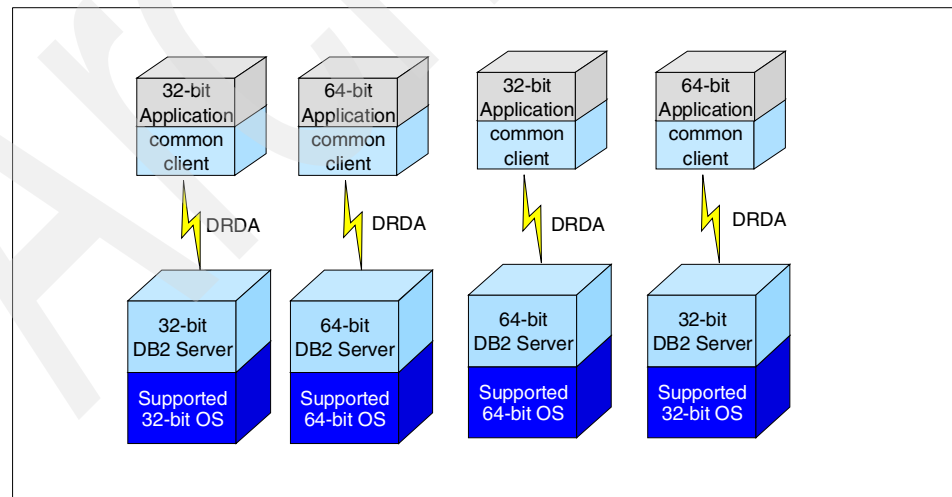


Figure 1-6 DB2 UDB common client connectivity

For more details, see section 9.2, “64-bit considerations” on page 270.

1.2.2 Availability enhancements

Availability is one of the main issues in an e-BI environment. The DB2 UDB V8.1 now provides numerous online enhancements to reduce outages.

The main availability enhancements to consider are:

- ▶ Online table load
- ▶ Incremental maintenance of MQTs during load append
- ▶ Online table reorganization
- ▶ Online index maintenance
- ▶ Online database checking tool

Online table load

In DB2 UDB V8.1, when loading data into a table, the table space in which table resides will no longer be locked. Users have full read and write access to all the tables in the table space, except for the table being loaded. For the table being loaded, the existing data in the table will be available for read access if the load is appending data to the table.

These new load features significantly improve the availability of the data, and help customers deal with the maintenance of large data volumes and shrinking maintenance windows.

Incremental maintenance of MQTs during load append

The materialized query table (MQT) can remain available during the load append operation on the underlying table. When the load of the appended rows in the underlying is complete, the materialized query table may be refreshed incrementally using only the appended data, which significantly reduces the time to update it.

For example, if the MQT is an aggregate (automatic summary table) for rows appended to the underlying table that corresponds to new groups in the aggregate, new summary rows will be inserted. For appended rows that correspond to existing groups in the aggregate, existing rows will be updated. Although the aggregate table remains unavailable during this maintenance phase, when a small number of rows are appended to the underlying table (when compared to the size of the table), the time that the aggregate is unavailable is reduced.

The ability to incrementally maintain a MQT is not restricted to aggregates. Many MQTs can be incrementally maintained. These changes significantly improve the availability of MQTs to your users.

Online table reorganization

The DB2 UDB reorganization table method has been enhanced to improve support for multi-partition databases, since it allows you to reorganize a single partition, a set of partitions, or all partitions.

Note: By default, the **REORG** utility affects all database partitions in the database partitions group defined for the table space where the table or/and index is/are located. But there is an option (**DBPARTITIONNUM**) that lets you specify the partitions you want or do not want executing the **REORG** utility.

DB2 UDB now provides two methods of reorganizing tables:

► Online

Online method allows applications to access the table during the reorganization. In addition, online table reorganization can be paused and resumed later by anyone with the appropriate authority by using the schema and table name.

Online table reorganization is allowed only on tables with type-2 indexes and without extended indexes.

► Offline

The offline method provides faster table reorganization, especially if you do not need to reorganize LOB or LONG data, because these data types are no longer reorganized unless specifically requested. Indexes are rebuilt in order after the table is reorganized. Read-only applications can access the original copy of the table except during the last phases of the reorganization, in which the shadow copy replaces the original copy and the indexes are rebuilt.

Online index reorganization

Now you can read and update a table and its existing indexes during an index reorganization using the **REORG INDEXES** command.

All indexes are rebuilt during an online index reorganization. A *shadow copy* of the index object is made, leaving the original indexes and the table available for read and write access. Any concurrent transactions that update the table are logged. Once the logged table changes have been forward-fitted, and the new index (the shadow copy) is ready, the new index is made available. While the new index is being made available, all access to the table is prohibited.

The default behavior of the **REORG INDEXES** command is **ALLOW NO ACCESS** which places an exclusive lock on the table during the reorganization process, but you can also specify **ALLOW READ ACCESS** or **ALLOW WRITE ACCESS** to permit other requests or transactions to read from or update the table.

Since you need to have sufficient temporary space for the building and reorganization process, indexes can now be created in large table spaces (long table spaces). In situations where the existing indexes consume more than 32 GB, this will allow you to allocate sufficient space to accommodate the two sets of indexes that will exist.

Online database checking tool (INSPECT)

The new online database checking tool or **INSPECT** command replaces **db2dart** command. It checks database for architectural integrity. This tool is now available online and checks the structures of tables and table spaces. In a multi partitioned environment, the scope is the collection of all database partitions defined in the db2nodes.cfg file.

1.2.3 Manageability enhancements

The manageability enhancements provide support for larger workloads, and also more availability for the e-BI environment.

The main enhancements to consider are:

- ▶ New features to help monitoring the health of the DB2 UDB systems: Health Monitor and Health Center
- ▶ New wizards and GUI tools
- ▶ Log enhancements

Health monitoring

As its first steps to Self Managing And Resource Tuning, and towards starting autonomic computing, DB2 UDB integrates a new management by exception capability through two new features: Health Monitor and Health Center, which alert you to potential system health issues, and provide advice about resolving them. This enables you to address health issues before they become real problems that affect the DB2 UDB performance.

New wizards and GUI tools

Several new wizards have been added to provide step-by-step guidance when creating objects, manipulating data, or configuring your environment:

- ▶ Memory Visualizer to display and understand the memory usage on DB2 UDB instance components
- ▶ Add Partitions, Alter Database Partition Group and Redistribute Data wizards for partitioned database objects
- ▶ Backup and Restore wizards

- ▶ Storage Management view to monitor the storage state of a partitioned database
- ▶ Configuration Advisor wizard to recommend starting configuration parameters
- ▶ Design Advisor wizard to recommend indexes on a specific workload sample

Log enhancements

DB2 UDB V8.1 logging now provides the following enhancements:

- ▶ A maximum log space of up to 256 GB. For a partitioned database, this limit is per partition. This will provide support for larger workloads.
- ▶ An infinite active logging which allows a single transaction to span both the primary logs and the archive logs, effectively allowing a transaction to use an infinite number of log files.
- ▶ The MIRRORLOGPATH parameter is a database configuration parameter allows you to enable dual logging. It should be set with a fully qualified path name.
- ▶ The BLK_LOG_DSK_FUL is a database configuration parameter that allows to specify that DB2 UDB should not fail when running applications on disk full condition from the active log path. When you enable this option, DB2 UDB will retry every five minutes allowing you to resolve the full disk situation and allowing the applications to complete.

1.2.4 UNION ALL views

The treatment of the UNION ALL views in the DB2 query rewrite component of the DB2 SQL compiler has been sufficiently enhanced to make it worth considering when there is a requirement to manage data that is large, but needs to be viewed as a single solution.

There might be a situation when a single-partition DB2 user has not anticipated the growth of a table, or does not want to move to a partitioned database in the near time frame. One approach that might be worth considering is instead of storing the data in a single table, use a UNION ALL view over multiple tables.

Application queries can refer to this view to look at the data in all the component tables as a single entity.

The following article under the DB2 Developer Domain Web site details how UNION ALL views can be useful in such a situation, and discusses the advantages and disadvantages of this approach:

<http://www7b.software.ibm.com/dmdd/library/techarticle/0202zuzarte/0202zuzarte.pdf>

UNION ALL views are being greatly improved to make this a viable solution for dealing with large amounts of data. In an e-BI environment, **UNION ALL** views are very useful for many reasons:

- ▶ Use of database partitioned views to facilitate data maintenance. For example: drop Q1-table / back it up to tape / create new Q1-table/
- ▶ These views collect data from remote sources and represent it in a transparent fashion. An application that uses partitioning to spread the workload and data over multiple independent tables.
- ▶ In HSM environments where the ranges over a partitioned table hierarchy of which parts are stored on tertiary storage; for example, data partitions can roll out to tape, and roll back in on demand.
- ▶ In case of overcoming table size limits or facilitating data maintenance.

The new DB2 UDB V8.1 capability to **INSERT** through the **UNION ALL** view expands the possibilities of UNION ALL views. The use of INSERT through UNION ALL view example is shown in Figure 1-7.

```
CREATE TABLE Q1(order DATE, item VARCHAR(10), CHECK (MONTH(order)
BETWEEN 1 AND 3));
CREATE TABLE Q2(order DATE, item VARCHAR(10), CHECK (MONTH(order)
BETWEEN 4 AND 6));
CREATE TABLE Q3(order DATE, item VARCHAR(10), CHECK (MONTH(order)
BETWEEN 7 AND 9));
CREATE TABLE Q4(order DATE, item VARCHAR(10), CHECK (MONTH(order)
BETWEEN 10 AND 12));

CREATE VIEW V(order, item)
AS SELECT * FROM Q1 UNION ALL SELECT * FROM Q2 UNION ALL
SELECT * FROM Q3 UNION ALL SELECT * FROM Q4;

INSERT INTO V VALUES('2002-01-06','Shoes'),('2002-06-17','Socks');
```

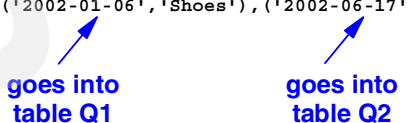


Figure 1-7 INSERT through UNION ALL view example

1.3 New functionalities of IBM e-server p690

Looking for more and more performances also requires tight integration between software and hardware. This chapter only focuses on advanced p690 features as well as the AIX 5L, which may benefit DB2 UDB ESE V8.1, because of their tight product architecture integration.

1.3.1 Hardware overview

IBM pSeries systems run on AIX 5L or Linux operating systems.

The IBM p690 systems use the following system technologies that provide the following: (For detailed information refer to the redbook *IBM eServer pSeries 670 and pSeries p690*, SG24-7040.)

- ▶ High performance technology
 - The p690 uses POWER processors and is powered by one to four multi-chip processor modules. Each Multi-Chip Module (MCM) contains either four or eight processors (up to 32 processors).
 - Three levels of cache memory: L1, L2, and L3. The L1 memory resides inside the processors, and the L2 and L3 are shared between the processors.

Each processor contains 32 KB of data cache, and 64 KB of instruction cache memory.

Each processor MCM contains 5.6 MB of Level 2 cache, which is shared between the processors. This provides 1.44 MB of L2 cache per processor for 4-processor MCMs and 0.72MB of L2 cache per processor for 8-processor MCMs. Each installed processor MCM is supported with 128 MB of Level 3 cache.

This provides 32 MB of L3 cache per processor for 4-processor MCMs, and 16 MB of L3 cache per processor for 8-processor MCMs.

- Up to 8 system I/O drawers of 20 PCI I/O slots supporting hot plug
Each system I/O drawer hosts up to 16 SCSI3 internal disks.
- ▶ Excellent reliability, availability, and serviceability (RAS) providing:
 - Error checking and correcting (ECC) system memory and cache
 - I/O link failure recovery
 - Disk drives that can be hot-swapped
 - Environmental sensing
 - Chip kill memory
 - N+1 Power and N+1 cooling units
 - Service processor for integrated system monitoring
 - Concurrent diagnostics

Note: p690 can host terabyte databases with a high level of availability without the need of MPP configuration.

Tip: Sizing guidances. You may consider allocating 1 CPU and from 60-150 GBs raw data per DB partition in a DSS environment. In a mixed environment, DSS/OLTP allocating 2 CPUs per DB partition may be more suitable. Memory sizing is highly related to the workload.

1.3.2 Logical partitioning for OLTP-aware e-BI

This concept, borrowed from mainframe world, gives the flexibility to build multiple hardware configurations on the same p690 family system. Each *logical partition* (LPAR) is independent of other ones running on the same system, either in terms of operating system, or in terms of resources.

Logical partitioning relies on the ability to allocate any processor, any I/O adapter, or memory amount (in 256 MB chunk) to any partition.

Future generation will support allocation in percentage of CPU (minimum of 10% of the power of one single CPU).

Figure 1-8 provides an overview on logical partitioning.

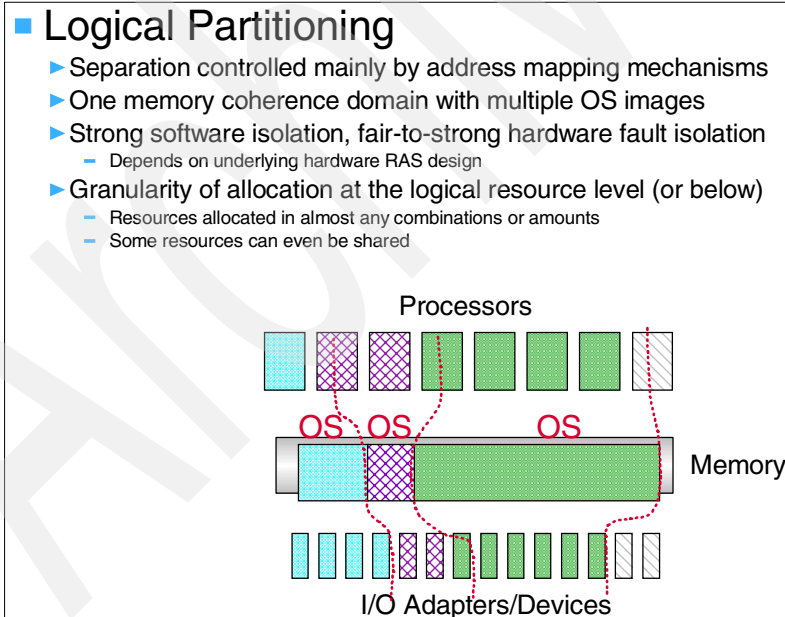


Figure 1-8 Logical partitioning

Physical partitioning also called PPAR that is used by other hardware vendors relies on hardware bricks such as quad (block of processors and memory). As a result, PPAR generally affords less flexibility when attempting to create partitions with just the right amount of resources required to meet the needs of the application.

Logical partitioning can also be a good solution when needing to host both applications and database servers on the same hardware without any interference between each other, or to separate the development, test, and production environments on different logical partitions.

It can also be a means of running online transaction processing (OLTP) and decision support applications inside the same system. For example, logical partitioning provides the flexibility of mixing on the same p690 machine different activities as your OLTP and data warehouse environments; or if you have many legacy systems and multiple operational data sources to build on the same machine your Operational Data Store and your data warehouse without any interference. Such solutions will shorten the data warehouse populating time.

Another benefit from LPAR in an e-BI environment may be to separate short running queries coming from an OLTP-like environment such as Web personalization; and requiring real-time data and long running queries coming from decision support.

1.3.3 Cluster 1600

The Cluster 1600 is a scalable system that can include clusters of pSeries along with a Control Workstation (CWS), running Parallel System Support Program (PSSP), or Cluster Systems Management (CSM) software. The Cluster 1600 system benefits from the single point of control and provides scalability when partitioning data on multiple clusters.

The p690 systems can be integrated in a cluster 1600. Each p690 is considered as a *frame*; each partition is considered a *node*. SP switch2 or colony switch attachments can be integrated in each partition for better performance.

You can also benefit and share all the functionalities of the RS/6000 SP systems as shown on Figure 1-9:

- ▶ File collections
- ▶ Distributed commands
- ▶ SP users
- ▶ Automounter
- ▶ Single point of control.
- ▶ SP switch

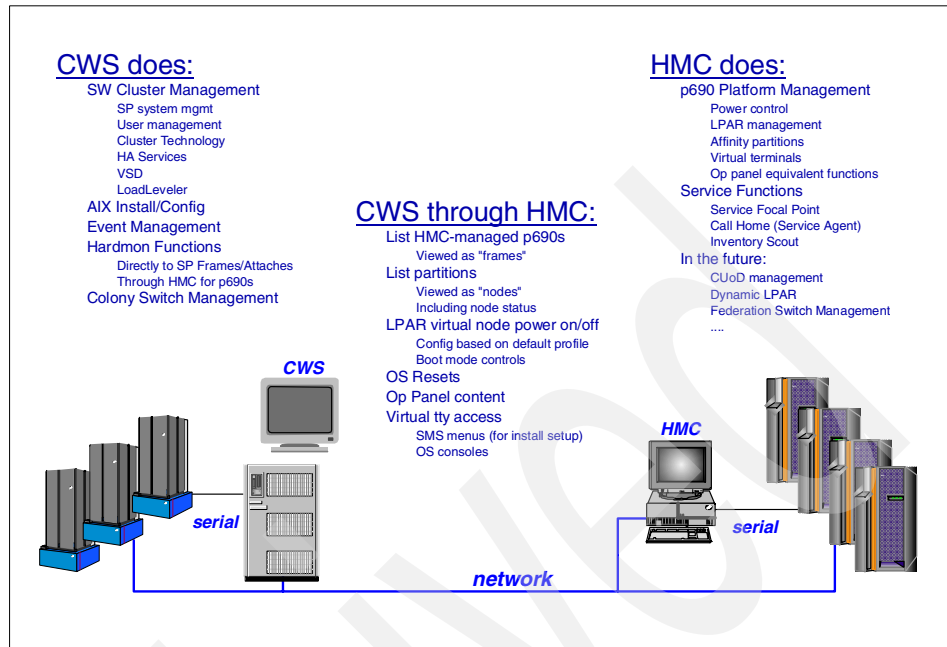


Figure 1-9 Cluster 1600

Note: Cluster 1600 is a nearly unlimited configuration for very large databases with a single point of control (CWS) and SP switch bandwidth.

1.3.4 Enhancements with AIX V5.2

DLPAR in conjunction with CuOD and DB2 UDB dynamic resources allocation is a good way to answer the need for extra power and 24/7 availability.

p690 also has the ability to add new (hot pluggable) I/O adapters and their devices concurrently with customer operations.

DLPAR

This AIX level is the entry point of DLPAR (Dynamic LPAR) which allows you to dynamically allocate/deallocate a piece of hardware without the need for rebooting the system. One piece of hardware can be released from one partition and integrated into another one. This feature benefits customers having different workloads (day/night or working days/week end) without interrupting the service. Power can be distributed on demand.

Since DB2 UDB V8.1 can change its configuration parameters (such as bufferpool) without needing to stop and restart the database, this could be of great benefit to the exploitation of physical resources on an LPAR and the performance of the database.

Our first test results showed that the dynamic allocation and deallocation of physical resources worked well. DB2 UDB also had no problems in changing the parameters on the fly.

CuOD

The *Capacity upgrade On Demand* (CuOD) feature with dynamic CPU sparing (as shown in Figure 1-10) answers the need of customers on short or long term extra CPU power, and allows for more continuous system operations.

Note: CPU sparing prevents predictable CPU defect from happening. This feature improves RAS level.

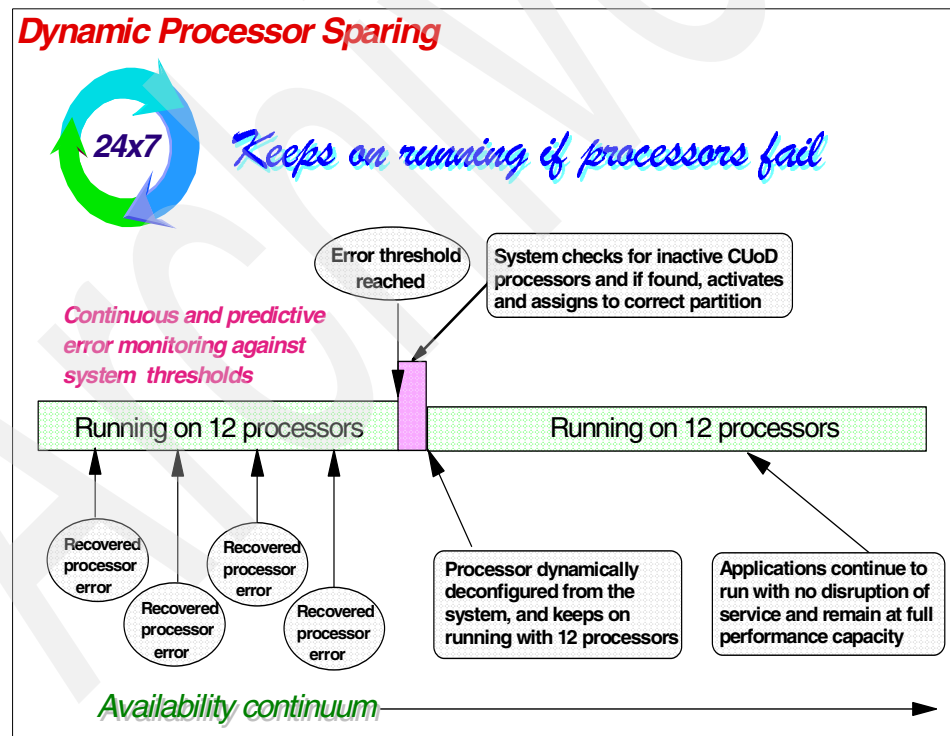


Figure 1-10 CPU sparing

1.3.5 Workload management

In this section, we just provide a quick overview of WorkLoad Manager (WLM) tool before providing some ideas on when to use logical partitioning, and when to use WLM; and also on how to control the data warehouse workload with WLM.

WLM in a few words

The AIX Workload Manager was developed to control the resource consumption of concurrent processes, so a system administrator can easily configure priorities and limits for CPU, memory, and disk bandwidth resources. This configuration intends to guarantee the availability of resources for critical jobs, and prevents them from being seriously influenced by others of less importance.

WLM is a resource management tool that specifies the relative importance of workload by tiers, classes, limits, shares, and rules. It is part of standard AIX operating system kernel at no additional charge.

Chapter 2 of the redbook *AIX 5L Workload Manager (WLM)*, SG24-5977, provides insights into the workload balancing possibilities and configurations you can use.

WLM is to be used to control workload of different applications running concurrently on the same system (see Figure 1-11):

- ▶ Tiers give class priority ranking (0 is the highest).
- ▶ Superclasses determine the amount of allocated resources (shares, limits).
- ▶ Subclass limits are based on parent superclass.

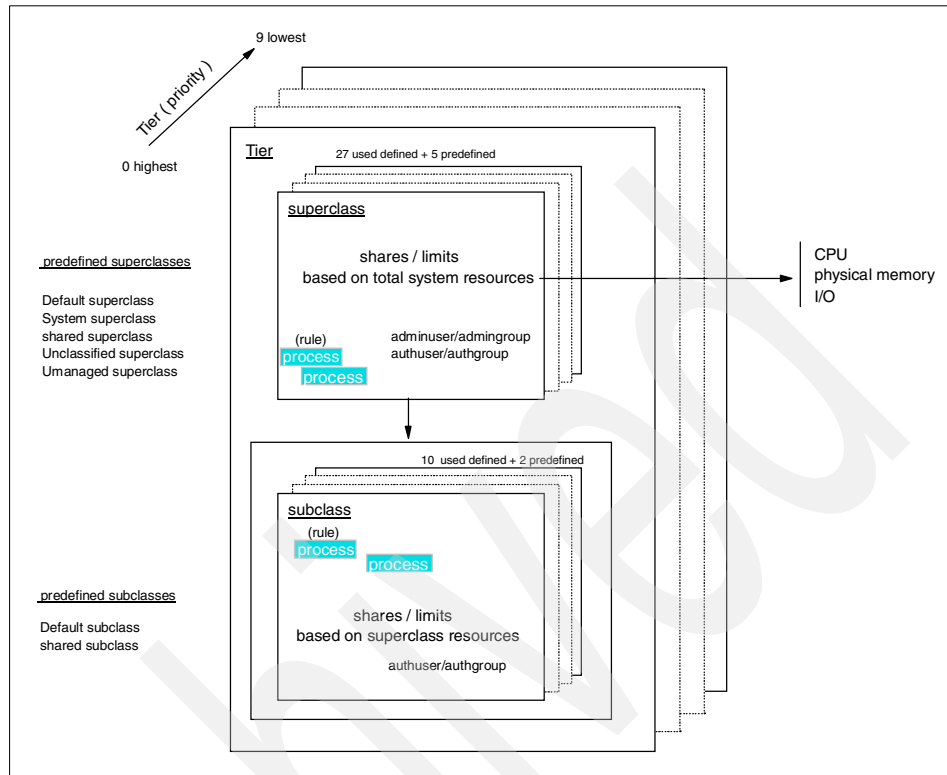


Figure 1-11 WLM concept

WLM does not produce multiple systems on one server hardware, but it divides the system into multiple classes, which are collections of processes or jobs. System resources such as CPU time, physical memory pages, and disk I/O bandwidth are distributed to the classes according to the defined shares and limits.

Because WLM is implemented by manipulating the resource allocation algorithm of the AIX kernel (such as scheduler, virtual memory manager, and device driver calls), its setup does not require rebooting or installing additional O/S instances and applications. Also, its reconfiguration is fully dynamic, and its granularity can be very fine because its resource allocation is not bound to any internal hardware boundaries, such as CPU cards, or memory banks.

Idle resources can be automatically redistributed from inactive classes to classes that need more resources according to their classes, tiers, shares, limits, and rules.

Logical partitioning or workload management

Logical partitioning and WLM are depicted in Figure 1-12.

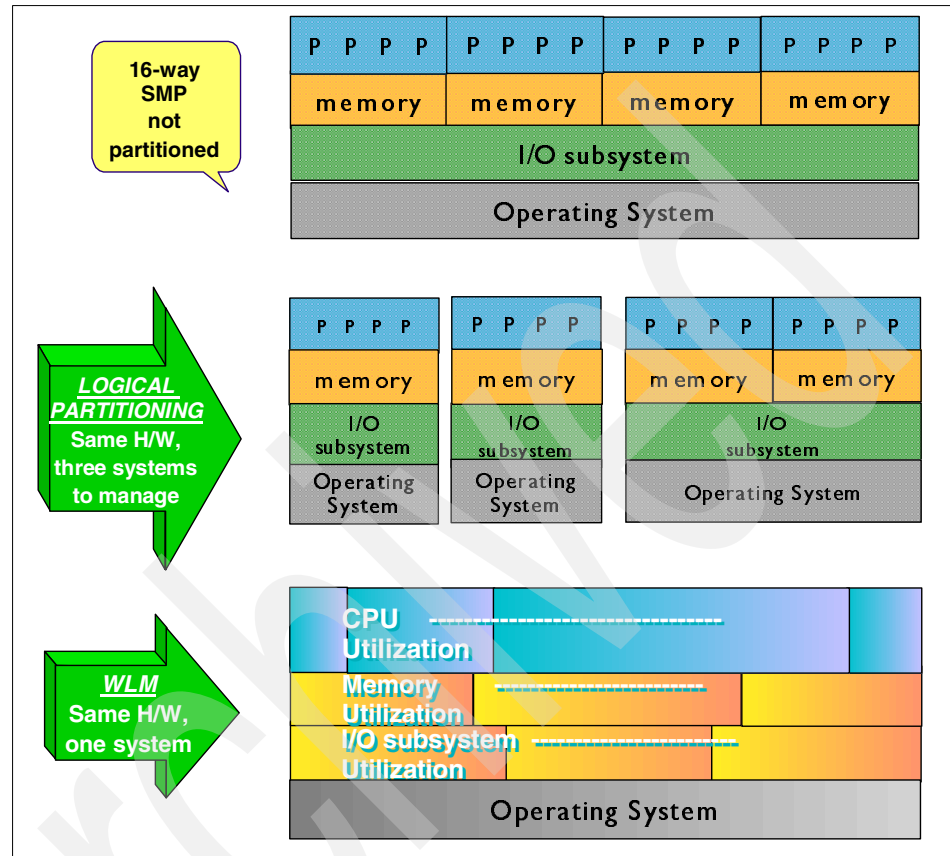


Figure 1-12 Logical partitioning or WLM

When WLM is a better solution

Logical partitioning might be a better solution than WLM:

- ▶ When there are different application dependencies:
 - If they require different versions (or fix levels) of the operating system.
 - If they require separate change management procedures of the OS.
- ▶ When there are different security requirements:
 - If they require different owners/administrators.
 - If they require strong separation of sensitive data.
 - If they require distributed applications with network firewalls between.

Partitioning offers excellent isolation between partitioned domains. Events in one domain cannot affect the others, and there is no resource contention of applications which belong to different domains. Also, even if the O/S crashes in one domain, the other domains remain intact. This allows running a test system and a production system on one server hardware without potential danger of service downtime. This can be very useful in times of application or O/S migration.

However, even with partitioning, failures in a shared hardware part such as the system backplane can lead to a total service going down. The fact that there are multiple instances of OSs means more overhead in overall system resource utilization and in administrative efforts.

Extra resources are needed due to the fact that each partition requires its own copy of the operating system, which has to be managed as an individual system.

In current partitioning implementations resources cannot easily be switched from one partition to another since they have constraints that restrict flexibility. Free resources on one partition will be wasted, if they are not used by any other application on that partition.

These limitations might lower the overall system resource utilization, compared with WLM.

When WLM is a better solution

WLM might be a better solution over logical partitioning when:

- ▶ Resource control and allocation are within a single OS instance.
- ▶ Allocations are on a per-application (process) basis.
- ▶ Typically very fine-grained control is required.

WLM uses only one instance of O/S, thus representing less administration burden. By its nature, WLM is much easier to configure and maintain than partitions.

No resource is wasted or kept idle, due to self-adjusting nature of target shares. If no application in a certain class is active at a moment, the resources assigned to that class is automatically and dynamically distributed to other classes to match the policies set by the system administrator. The target shares and limits themselves can be dynamically redefined, without stopping the applications. All these contribute to better resource utilization than in the case of partitioning.

But WLM does not provide hardware-level isolation as partitioning does, so the applications are not completely free from influences of other applications. Also, the limitation to one O/S image may prove to be a single point of failure; if the O/S crashes, the whole application service is unavailable.

Controlling the workload using WLM

To monitor the workload, DB2 UDB has his set of standard monitoring tools provided with DB2 UDB such as:

- ▶ DB2 snapshots that require to turn on the different monitor switches (bufferpools, locks, sort, statement, table, unit of work, timestamp) by:
db2 update monitor switches using <monitor_switch> on
before taking the instant snapshot by:
db2 get snapshot for <monitor_switch> on bigdb
- ▶ DB2 event monitor (to collect monitoring information on deadlocks, connections, statements, transactions during a certain period):
 - **db2 create event monitor db2evmon1 for connections write to file '/evmon/connections/' maxfiles 4 maxfilesize 1000** (to define the event monitor on connections for example)
 - **db2 set event monitor db2evmon1 state=1** to activate the event monitor
 - **db2evmon** or **db2eva** tools to analyze the event monitor output
- ▶ DB2 explain: **db2expln -database bigdb -schema % -package % -output explain.out**
- ▶ DB2 governor to favor or penalize queries: **db2gov**

DB2 UDB has also an extensive set of parameters (see Appendix D, “DB2 UDB configuration parameters” on page 355) that enables the database administrator to limit the memory (bufferpool, sort), CPU (query degree), and I/O (prefetchers, iocleaners, and so on) resources used by the database. But, DB2 UDB cannot prevent other applications from requesting resources that were initially supposed to be used by the database, nor avoid the operating system from denying those requests.

So, AIX Workload Manager can be used to control a specific data warehouse workload, and ensures that DB2 UDB gets the resources that the database administrator assumed would be available when the configuration was established, regardless of the needs of other software or user applications running on the same machine. Some skills on both DB2 UDB and Workload Manager are prerequisites.

When running the queries in the workload, some of them that mainly exhibit CPU and I/O boundaries (queries elapsed time and CPU consumption can be checked with **vmstat** and **db2batch**) can be configured using WLM to better suit your workload requirements.

For example, a long-running query that is an important query *must* be completed within certain time boundaries. How to favor the execution of this long-running

query in order to enable it to complete in its time boundaries while slowing down the execution of other queries, but giving them some portion of the resources?

To regulate the resources to allocate to queries, WLM can be used.

WLM allows a hierarchy of classes to be specified; processes to be automatically assigned to classes by the characteristics of a process; and manual assignment of processes to classes. Classes can be superclasses or subclasses. WLM controls processes.

With DB2 UDB, WLM will need to be configured to set up superclasses (for example, the instance), subclasses (set of processes such as heavy and light queries) with particular limits, CPU shares, and tiers; and of course, manually assigns **db2agents** processes to these subclasses. DB2 processes can be manually assigned to a class using the **wlm_assign** command.

To achieve a manual assignment, a possible sequence of operations may be:

1. Define the WLM configuration file with user IDs and subclasses.
2. Update the monitor switches:
db2 update monitor switches using statement on
3. Get the DB2 snapshot to generate lists of db2agents process IDs working on behalf of the different queries to control:
db2 get snapshot for applications on <database>
4. Assign the db2agents process IDs to the subclasses defined in the configuration file using **wlm_assign**

1.4 Case study and technical environment

This section describes the data used as a case study, and the hardware and software environment.

1.4.1 The case study based on TPC-H

To explore the new DB2 UDB V8.1 functionalities, we used the database schema (see Figure 1-13) and the data generation tool from the Transaction Processing Performance Council (TPC) called TPC-H.

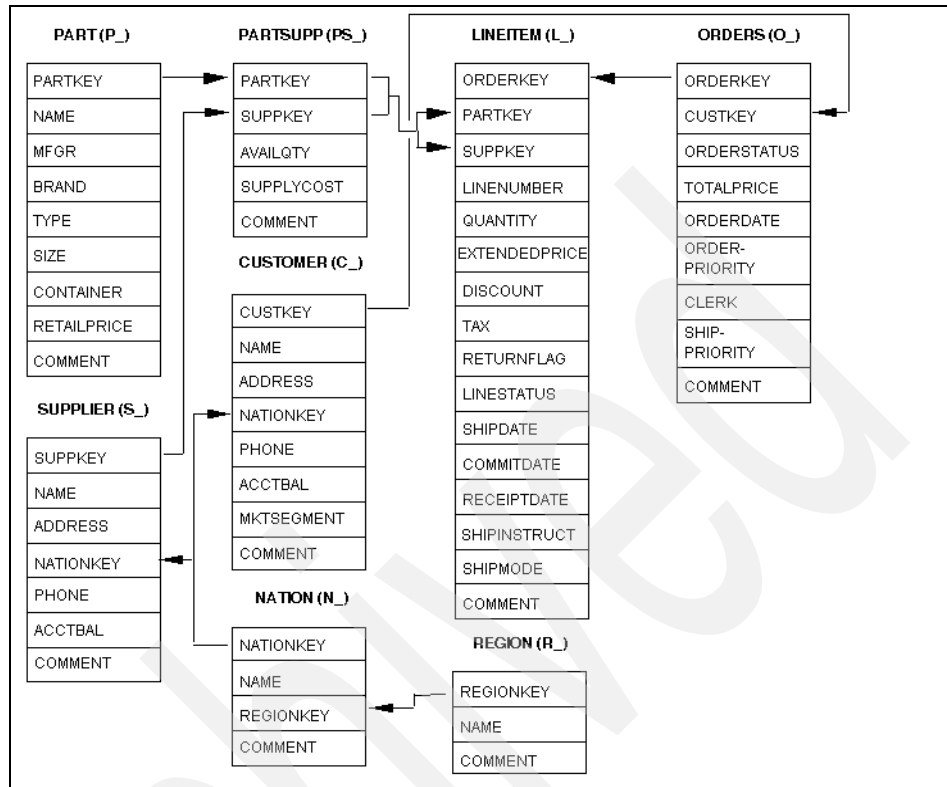


Figure 1-13 TPC-H tables design

Important: This activity was not a benchmark. TPC-H provided a readily available set of table definitions and a way to generate data to load into those tables. Using them, we were able to quickly setup a database with which to explore the functionalities of DB2 UDB V8.1.

Basically, it is an entity-relationship type model that represents a decision to support the database.

For more information, please refer to:

<http://www.tpc.org/tpch/>

1.4.2 Technical environment

To experiment DB2 UDB ESE V8.1, two configurations were used:

- Configuration 1 to test load: MDC tables were composed of one p690 32ways/128GB memory and 48 x 18GB SSA drives.
- Configuration 2 to test backup/recovery was composed of one p690 32ways/128 GB and ESS F20 of 128 x 36GB drives.

The database design chosen relies on the following assumptions:

- One database partition per CPU
- One additional database partition for the catalog, with data and indexes in the same table space. We chose to separate database partition for catalog partitioning for administrative purposes.

The level of software used is provided in Table 1-1.

Table 1-1 Software environment

Software	Level	Additional fix
AIX 5L	5.1 ML3	IY32468
DB2 UDB ESE with DPF	V8.1 GA code	

Configuration 1

Configuration 1 used to test load, and MDC tables are depicted in Example 1-14.

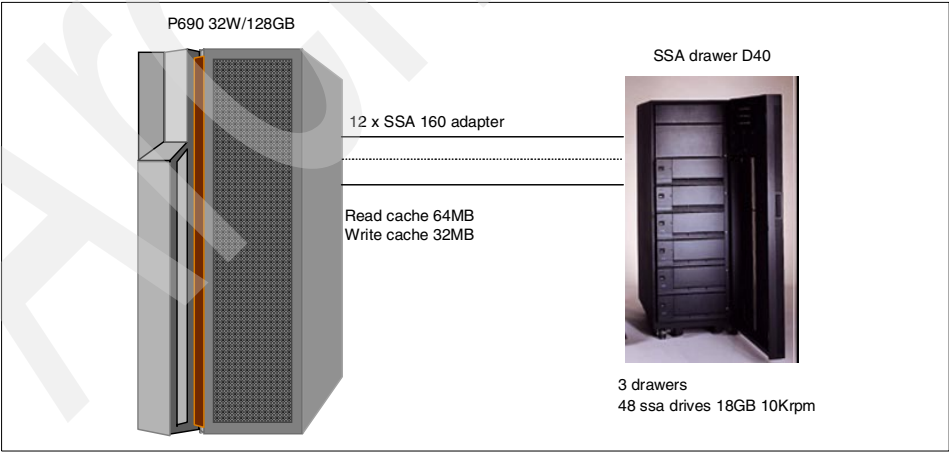


Figure 1-14 p690 and SSA drives configuration

Database configuration

The database partition configuration used is provided in Table 1-2.

Table 1-2 Database partition group configuration

db partition group	db partition
one	0
all	1 to 32
ibmcatgroup	0

The bufferpool configuration parameters used are provided in Table 1-3.

Table 1-3 Bufferpool configuration

Bufferpool	Page size	db partition group
IBMDEFAULTBP	4K	ibmdefaultgroup
BP16	16K	all ibmtempgroup
BP32K0000	32K	ibmtempgroup

The table spaces parameters used are provided in Table 1-4.

Table 1-4 Table space configuration

Table space	Page size	Extent size	Type	Containers
SYSCATSPACE	4	16	SMS	FS
TEMPSPACE1	4	16	SMS	FS
TBS_ONE	4	16	SMS	FS
TBS1	16K	page size x nb of containers	DMS	1 per disk
TEMPSPACE4	4K	16K	SMS	FS
TEMPSPACE16	16K	16K	SMS	FS
MDC	16K	page size x nb of containers	DMS	1 per disk
TBSP32K0000	32K	128K	SMS	FS
TBSP32KTMP0000	32K	128K	SMS	FS

I/O configuration and layout

The I/O configuration was based on:

- ▶ 48 SSA drives were attached to 12 SSA adapters (FC 6228); thus, 4 drives attached to each SSA adapter.
- ▶ 32 internal SCSI drives (36 GB / 10K rpm)

The number of external disks (48 as shown in Figure 1-15) does not allow a balanced configuration for 32 database partitions, so to balance the I/O load evenly, decision was made to create 4 volume groups of 12 disks each (each disk from a different adapter). Large table space hosting of the large tables spans across these 12 disks (1 raw table space container on each disk).

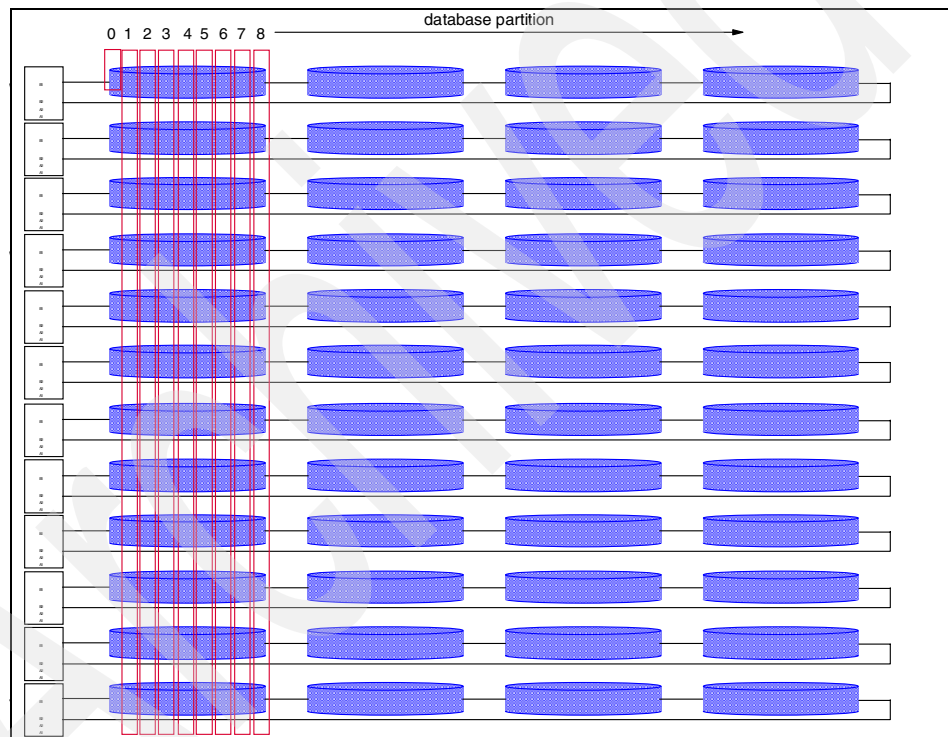


Figure 1-15 SSAs physical layout

Logs have been spread evenly on the faster SCSI internal disks, and designed to get one log file system per disk.

Tip: Keep in mind that on AIX, Logical Volume (LV) creation locks the volume group. So, as many volume groups there are as many simultaneous LV creations can be done.

Configuration 2

Configuration 2 used to test ESS flashcopy is depicted in Figure 1-16.

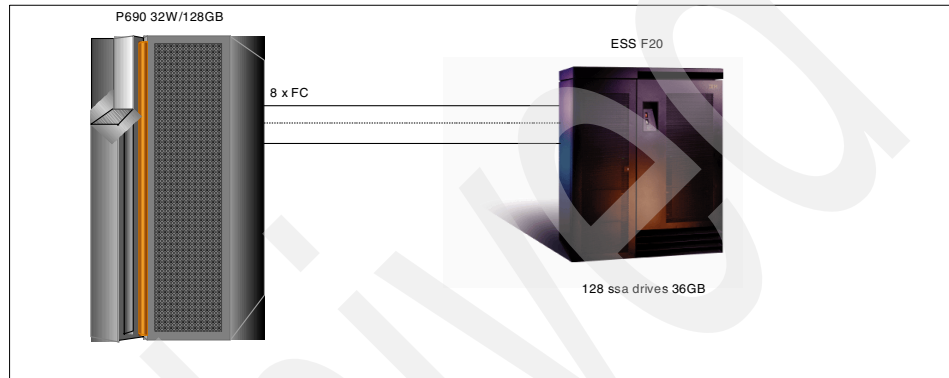


Figure 1-16 p690 and ESS configuration

The configuration is similar to configuration 1 except for the I/O placement on the ESS F20. ESS was formatted as RAID5 arrays and 8GB LUNs. Half of the disk space is allocated to the database, the other half is allocated to FlashCopy.

ESS is attached to p690 by 8 Fiber channels (FC 6228) with a throughput of 2 Gb/s.

Build a large data warehouse

The definition of where and how your data warehouse is placed is the one of the most important decisions for database performance.

It is not a hard task to do when you have the appropriate guidelines for your BI environment.

This chapter provides the main guidelines that will help you to build a large data warehouse.

This chapter contains the following sections:

- ▶ Steps for building a large data warehouse
- ▶ Disk considerations
- ▶ Database design considerations
- ▶ Installing and configuring DB2 UDB ESE V8.1

2.1 Steps for building a large data warehouse

In this topic we provide the necessary steps to build a large data warehouse with the main tasks, and the related sections as shown in Table 2-1.

Table 2-1 Steps for building a large data warehouse

Steps	Main tasks involved	Related sections
1. Understand your environment This first step is to understand your whole environment and have basic information to start planning your data warehouse building.	<ul style="list-style-type: none">▶ Information of machine configuration▶ Information of external storage type▶ Define type of workload▶ Information of raw data size	Section 2.2, "Disk considerations" on page 39
2. Determine disks layout and placement This step is all related to the disk definition layout based on the placement plan. You will actually allocate the data, log, DIAGPATH and temp space when you create your database and table spaces.	<ul style="list-style-type: none">▶ Have enough space for the database▶ Define RAID level▶ Plan data, log, DIAGPATH, temp space placement▶ Consider space for working and staging area	Section 2.2, "Disk considerations" on page 39 Section 2.3.10, "Configure DB2 UDB" on page 75
3. Define tables sizes The tables sizes information will be needed to the define the number of database partitions, the next step.	<ul style="list-style-type: none">▶ Define tables sizes▶ Consider MQT and MDC sizing	Section 2.3.7, "Size the tables" on page 74 Section 2.3.8, "Size for MDC utilization" on page 75 Section 2.3.9, "Size for MQT utilization" on page 75
4. Define database partitions	<ul style="list-style-type: none">▶ Define number of database partitions▶ Define database partitions group	Section 2.3.2, "Define the number of database partitions" on page 46 Section 2.3.3, "Define database partition groups" on page 49

Steps	Main tasks involved	Related sections
5. Set configuration parameters Set the parameters that are applicable to your environment.	<ul style="list-style-type: none"> ▶ Set AIX, Network and DB2 UDB parameters 	Section 2.3.11, "Recommended parameters for performance" on page 82
6. Design table spaces and bufferpools The focus of this step is to define all table spaces and mainly the containers layout. Bufferpools have to be defined since they are directly related to the table spaces.	<ul style="list-style-type: none"> ▶ Define containers layout ▶ Decide SMS or DMS ▶ Define table space definition parameters ▶ Define catalog table space placement ▶ Define temp space table space ▶ Define bufferpools 	Section 2.3.4, "Design the table spaces" on page 52
7. Choose the right partitioning key	<ul style="list-style-type: none"> ▶ Choose the partitioning key 	Section 2.3.6, "Choose the partitioning key" on page 68

2.2 Disk considerations

Deciding how to manage disk storage is one of the most important decisions a database administrator has to make.

How the data is physically laid out across the disks affects performance, as we explain in the following sections:

- ▶ Summary of the most popular RAID levels
- ▶ Data placement
- ▶ Data availability and performance

2.2.1 Summary of the most popular RAID levels

Each of the RAID levels supported by disk arrays uses a different method of writing data, and hence, provides different benefits:

RAID 0

Also known as striping, RAID 0 does not bring any additional data redundancy to a disk subsystem. The primary goal of RAID 0 is to improve I/O performance. When data is written, using a striped logical volume, the data is divided into small

pieces known as chunks or stripe units. These units are then written to the disks in parallel, in a round-robin fashion. During a read operation, the same units are read back in parallel, then written directly to the caller's memory area where they are re-assembled into the original data.

The main problem with RAID 0 is obviously that any disk failure in the array will render the data on the remaining disks unusable. For that reason, only consider RAID 0 if you require good performance, but are not concerned about availability.

Tip: You should not use AIX striping, since DB2 UDB has its own concept of striping, using multiple containers for each table space.

RAID 1

Also known as mirroring, RAID 1 is simply keeping a copy of the data in a different location to protect against data loss in the event of a disk failure. So, from the availability perspective, the copies should be kept on a separate physical volume, which ideally should be attached to a separate I/O adapter. To take this one stage further, another possibility would be to have the second disk in a separate disk cabinet to protect against events such as a power loss to one of the disk cabinets.

In the event of a disk failure, read and write operations will be directed to the mirrored copy of the data. In performance terms, mirroring has some advantages over striping. If a read request is received and the primary data copy is busy, then the AIX LVM (Logical Volume Manager) will read the data from the mirrored copy. Write performance is affected by the write scheduling policy configured using the AIX LVM options, which are PARALLEL or SEQUENTIAL.

Although mirroring can improve the read performance, the penalty is write performance.

Note: In AIX, when mirroring data, *only* the logical volumes are mirrored, not the physical disk. Mirroring is not a tool to provide a disk copy.,

Tip: For SSA disks, you should use mirroring for maximum performance (for data, logs, temporary space) and also use the inherent striping of DB2 UDB, since it has its own concept of striping when using multiple containers for each table space.

RAID 5

In a RAID 5 array, data and parity are spread across all disks. For example, in a 5+P RAID 5 array, six disks are used for both parity and data. In this example, 5/6s of the available space is used for data and 1/6th is used for parity.

Because of the parity data used by RAID 5, each write to the array will result in four I/O operations; this is known as the RAID 5 write penalty:

1. Read old data
2. Read old parity
3. Write new data
4. Write new parity

Fast Write Cache (FWC), which exists on some RAID adapters, can reduce the effects of this write penalty.

Note: Full-stripe writes to RAID 5 devices are optimized into a single write.

RAID 5 is commonly chosen because it provides an extremely good price/performance ratio combined with good availability.

Using the Enterprise Storage Server (ESS), which has a large cache, can significantly decrease the effects of the write penalty.

Tip: For ESS, use the built-in RAID5, which has a large cache, and can significantly decrease the effects of the write penalty. But, make sure you spread the logical disks evenly across all of ESS internal devices.

RAID 10

Sometimes called RAID 0+1, this RAID level provides better data availability at the cost of extra disks. Consider a striped logical volume, where failure of one disk renders the entire logical volume unusable. RAID 10 provides the ability to mirror the striped logical volumes to prevent this. When data is written to the disks, the first data stripe is data and the subsequent stripe copies (maximum of three copies, including first copy) are the mirrors, and are written to different physical volumes. RAID 10 is useful for improving DB2 log performance.

2.2.2 Data placement

To choose a physical disk layout, some factors should be considered:

- ▶ Workload considerations
- ▶ Available disk space and number of disks

Workload considerations

The type of workload of your database have to be considered when you define the physical database layout.

For example, if the database is part of a Web application, then typically its behavior will be similar to On-line Transaction Processing (OLTP) as it can be in an Operational Data Store. There will be a great deal of concurrent activity, such as large numbers of sessions active, simple SQL statements to process, and single row updates. Also, there will be a requirement to perform these tasks in seconds.

If the database is part of a Decision Support System (DSS), there will be small numbers of large and complex query statements, fewer transactions updating records as compared to transactions reading records, and many sequential I/O operations.

This information gives the database administrator some idea which can be used to aid in the physical design of a database which will be well-suited to that type of workload.

Available disk space and number of disks

The basic rule of thumb is that three to four times the amount of raw data is needed to build a database including indices and temp space without factoring in any data protection such as mirroring or RAID5. With mirroring, it will be closer to eight times the raw data. For details about RAID levels, see topic 2.2.1, “Summary of the most popular RAID levels” on page 39.

Consider space for staging and working area.

Take into account the number of disks required to achieve a balanced system: too many small disks can cause system bus contention. Such a system may also produce increased system administration overhead. But, too few large disks can cause I/O bottlenecks.

In an average system, a good balance would be a minimum of 6 to 10 disks per CPU. In summary, the basic goal is to design a system, so that no one disk or set of disks becomes a performance bottleneck.

2.2.3 Log placement

Logging performance such as log file, number of logs, and the type of storage for the logs will be discussed in “Database logs” on page 76.

Before any log files even exist, the database administrator needs to decide on two main factors:

- ▶ On which disks are the logs to be placed?
- ▶ Availability

On which disks are the logs to be placed?

Due to the importance of the log files, it is recommended that the log files should always reside on their own physical disk(s), separate from the rest of the database objects. The disks ideally should be dedicated to DB2 logging to avoid the possibility of any other processes accessing or writing to these disks.

Availability

Irrespective of the type of logging you choose, whether it be circular logging or archival logging, the availability of the physical disks is crucial to the database. For this reason, it is strongly recommended that you protect the log against single disk failures by using RAID 1 (mirroring the log devices), recommended for SSA disks; or by storing log files on a RAID 5 array, recommended for ESS disks, since it has a large cache, and the write penalty effects can be significantly decreased.

For details about RAID levels, see 2.2.1, “Summary of the most popular RAID levels” on page 39.

2.2.4 Data availability and performance

For an e-BI environment, with DB2 UDB databases commonly being used to support Web applications, 24x7 availability is an important consideration. For example, for any distributor company that sells the products using Web applications, and needs to operate 24x7 to manage its products inventories and ship the products.

It is worth remembering that in most cases increasing availability is a trade-off against a potential reduction in performance. The most common solutions implemented today are RAID 5 and data mirroring. For more details about RAID 5 and mirroring, see 2.2.1, “Summary of the most popular RAID levels” on page 39.

2.2.5 General storage performance recommendations

The following are general performance considerations to bear in mind when creating logical volumes on AIX:

- ▶ Try to spread heavily-accessed logical volumes across as many physical volumes as possible to enable parallel access.
- ▶ Use the inherent striping of DB2 UDB, placing containers for a table space on separate logical disks, which reside on separate RAID arrays. This will

eliminate the need for using underlying operating system or logical volume manager striping.

- ▶ Try to create your logical volumes with sufficient size for your data; this will avoid fragmentation if the logical volumes have to be increased later.
- ▶ Sequential write-intensive logical volumes are best placed on the outer edge of the physical volumes. Logical volume placement can be controlled during its creation.
- ▶ Sequential read-intensive logical volumes (for example MQTs) are best placed in the center of the physical volumes.
- ▶ Set inter-policy to maximum to spread each logical volume across as many physical volumes as possible.
- ▶ Set write verify to OFF.

Note: It is probably not necessary to use both logical volume manager's maximum interleave policy and DB2 UDB table space striping at the same time.

2.3 Database design considerations

Deciding the database design is one of the most important decisions, since it directly affects the database performance.

In this section, we provide considerations and recommendations about database design and the topics included are:

- ▶ Understand data partitioning
- ▶ Define the number of database partitions
- ▶ Define database partition groups
- ▶ Design the table spaces
- ▶ Size the tables
- ▶ Understand partitioning map
- ▶ Choose the partitioning key
- ▶ Size for MDC utilization
- ▶ Size for MQT utilization
- ▶ Configure DB2 UDB
- ▶ Configure AIX/Network parameters

2.3.1 Understand data partitioning

DB2 UDB ESE can be configured with multiple database partitions, and utilizes a *shared nothing* architecture. This means that no data or memory is shared between database partitions, which gives good scalability.

A *partitioned database* environment allows a database to be transparent to most users, despite being physically divided across more than one partition. Work can be divided among the database managers; each database manager in each partition works against its own part of the database.

In a partitioned database environment there are two terms: physical partition or physical node, and DB2 UDB V8.1 logical database partition (also called logical node in previous DB2 UDB versions).

The *physical partition* is the server where the partitioned database resides. In the case of the p690 server, you can have hardware partitioning (by processor set or LPAR); for example, if you partition the p690 in two LPARs, you will have two physical partitions. See Figure 2-1 on page 46.

Tip: When processing large volumes of data, multiple logical database partitions can be configured within a single multi-processor physical server if a single server is sufficient for workload needs.

Generally, hardware partitioning (by processor set or LPAR) is not the best way to support DB2 UDB ESE partitioning for the reason that one hardware partition allows processors to be used where needed, rather than the risk of having some processors idle while others are fully utilized.

A *logical database partition* differs from a *physical partition* in that it is not given control of an entire server. Although the server has shared resources, database partitions do not share the resources. Processors are shared, but disks and memory are not. Logical database partitions provide scalability.

Because data is divided across database partitions, you can use the power of multiple processors on multiple physical partitions to satisfy requests for information. Data retrieval and update requests are decomposed automatically into sub-requests, and executed in parallel among the applicable database partitions. The fact that databases are split across database partitions is transparent to users issuing SQL statements.

The data is physically stored across more than one database partition, and yet can be accessed as though it were located in the same place. Applications and users accessing data in a partitioned database do not need to be aware of the physical location of the data.

User interaction occurs through one database partition, known as the coordinator partition for that user/application. In this way, coordinator partitions (and the associated workload) can be spread across the partitioned database.

When processing large volumes of data, multiple database partitions can be configured within a single multi-processor physical server, or if a single server is not sufficient for workload needs, across multiple physical servers.

Note: A p690 may be physically partitioned into 4 8-way partitions if it is to be part of a cluster with a high speed switch.

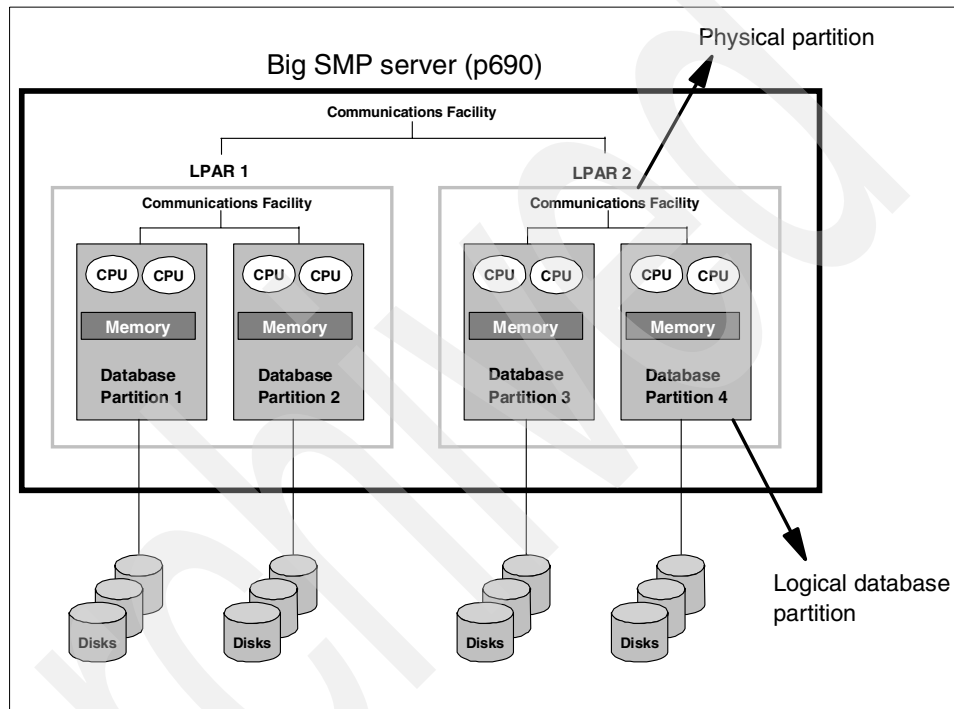


Figure 2-1 Physical partitions and logical database partitions

2.3.2 Define the number of database partitions

The number of logical database partitions is defined at instance level in the file `db2nodes.cfg`, which is located at `/$INSTHOME/sql/lib` directory in UNIX.

Important: Each *database partition* has its own:

- Logs
- Database configuration file (DB CFG)
- Bufferpools

A *partition group* contains one or more database partitions.

In Figure 2-2, we there are two physical machines with three logical database partitions on each machine. The logical database partitions are defined in the db2nodes.cfg file at instance level.

For more details about db2nodes.cfg configuration, see 2.4, “Installing and configuring DB2 UDB ESE V8.1” on page 95.

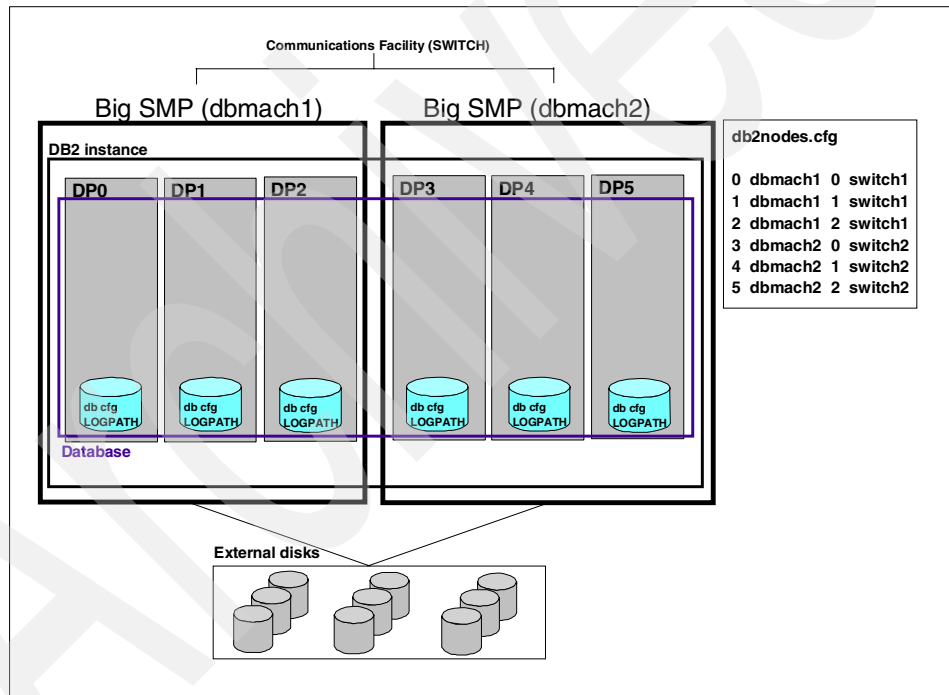


Figure 2-2 Number of logical partitions example

For more details about db2nodes.cfg configuration, see 2.4, “Installing and configuring DB2 UDB ESE V8.1” on page 95.

The topics covered are:

- Considerations

- Recommendations

Considerations

Follow some considerations for database partitions definition:

- When determining the number of database partitions, a number of considerations need to be taken into account:
 - The number of CPUs/memory available
 - The disk layout
 - The total amount of data residing under each partition
 - The largest table size under each partition
 - Future growth plans
 - The number of insert/update/delete operations
 - Amount of logging required per partition
 - Time required for batch/maintenance operations
- DB2 UDB ESE partitioned database can support up to 1000 partitions.
- Any database partition can be used as a coordinator partition to balance the workload.
- You are not restricted to having all tables divided across all database partitions in the database. DB2 UDB supports partial declustering, which means that you can divide tables and their table spaces across a subset of database partitions in the system. For example, you can create a database partition group containing a single database partition to store the dimension tables, which are always small.

Recommendations

The best partitioning strategy generally results in the following:

- Symmetrical number of processors per database partition

Tip: On a p690 machine, the general recommendation is:

- 1 CPU per each logical database partition

Tip: If you plan to run more than 20-30 concurrent queries, you should consider more CPUs per partition and also consider memory usage.

- Balanced disk layout between partitions. It is important to have data from each database partition distributed over sufficient physical disks, potentially with multiple containers. At least four disks per table space.

- ▶ The total amount of data residing under each database partition should not be too high to improve the parallelism of partition wide operations such as backups. Many factors come into play when deciding how much data there should be per database partition, such as the complexity of the queries, response time expectations, and other application characteristics. For example, a high percentage of non queried data allows more data per database partition, while a low percentage suggests less data per database partition.

Tip: Rules of thumb based on CPU vary, but may be in the region of 50-200 GB raw data per partition, with the real requirement to keep the individual table size about 100 GB/partition for utility performance and table scans.

- ▶ The largest table size under each partition should not be too high. There is an architectural limit per partition of 64 Gb for 4 k page size; 128 Gb for 8 k page size; 256 Gb for 16 k page size; and 512 Gb for 32 k page size. Beyond this, the maximum table size per partition affects the performance of processes such as **create index** or **runstats**.
- ▶ The future growth plans affect the number of partitions since it is desirable not to have to repartition within a period of 2 years for example. It is sometimes desirable, therefore, to create a partitioning strategy based on a 2 year sizing, rather than the current position. It is much more straightforward to add processors/memory or even migrate to multiple physical machines than to repartition a large database.

Important: Ensure that the catalog partition has been set correctly when the database is first created, since it cannot be changed at a later date. The catalog partition is determined by being the partition connected to when the database is first created. For example:

```
export DB2NODE=0
db2 terminate
db2 create database <dbname>.....
```

or:

```
db2_all "<<+0< db2 create database <dbname> ..."
```

2.3.3 Define database partition groups

A *database partition group* is a set of one or more database partitions that determine over which partitions each table space is distributed.

Figure 2-3 shows the database partition groups example.

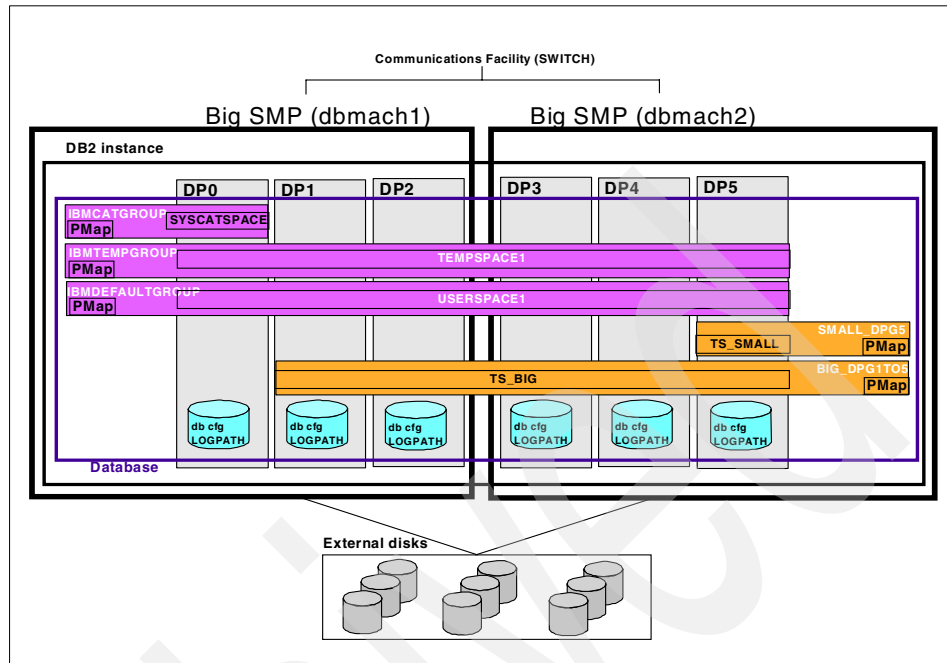


Figure 2-3 Database partition groups

Three default database partition groups are created when the database is created:

- ▶ IBMCATGROUP is the default database partition group for the table space containing the system catalogs. This database partition groups only contain the catalog partition.
- ▶ IBMTMPGROUP is the default database partition group for system temporary table spaces, which contain all partitions defined in db2nodes.cfg file.
- ▶ IBMDEFAULTGROUP is the default database partition group for the table spaces containing the user defined tables that you may choose to put there. This partition group contains all data partitions defined in db2nodes.cfg. The data partitions in IBMDEFAULTGROUP can be changed.

You can create new database partition groups (shown in Figure 2-3) using the **CREATE DATABASE PARTITION GROUP** command.

The commands described in Example 2-1 will create two database partition groups:

- ▶ The database partition group “BIG_DPG1TO5” on partitions 1, 2, 3, 4 and 5 for the table space TS_BIG, which contains the big tables.
- ▶ And another database partitions group “SMALL_DPG5” which contains only the partition 5 for the table space TS_SMALL which contains the small tables.

Example 2-1 Create database partition group command

```
CREATE DATABASE PARTITION GROUP big_dpg1to5 ON DBPARTITIONNUMS (1 TO 5);  
  
CREATE DATABASE PARTITION GROUP small_dpg5 ON DBPARTITIONNUM (5);
```

The topics covered are:

- ▶ Design points
- ▶ Design recommendations

Design points

You have to consider the following design points:

- ▶ In a multipartition database partition group, you can only create a unique index if it is a superset of the partitioning key.
- ▶ Depending on the number of database partitions in the database, you may have one or more single-partition database partition groups, and one or more multipartition database partition groups present.
- ▶ Each database partition must be assigned a unique partition number. The same database partition may be found in one or more database partition groups.

Design recommendations

You should consider the following design recommendations:

- ▶ It is suggested that a simple design of a single multipartition group across all data partitions be adopted for large tables (fact and the large dimension tables).
- ▶ For small tables, you should place them in a single-partition database partition groups, except when you want to take advantage of collocation with a larger table since the tables must be in same database partition group.

Tip: A single partition group can be created for the remainder of the dimension tables. It may be desirable to replicate the majority of the dimension tables across all partitions using DB2 UDB replicated summary tables, which would effectively *pre-broadcast* the smaller tables once a day for example, rather than having it take place for every query.

- ▶ You should avoid extending medium-sized tables across too many database partitions. For example, a 100 MB table may perform better on a 1 or 2-partition database partition group rather than on a 32-partition database partition group.
- ▶ You can use database partition groups to separate Operational Data Store OLTP-like tables from decision support (DSS) tables, to ensure that the performance of ODS transactions is not adversely affected.

2.3.4 Design the table spaces

How you set up your table spaces has a large impact on your database performance. In this section we provide some guidelines to define the table spaces placements getting the benefits from DB2 UDB and storage resources.

The topics included in this section are:

- ▶ Table space containers layout
- ▶ Table space SMS or DMS
- ▶ Table space definition parameters
- ▶ Tables per table space
- ▶ Catalog table space considerations
- ▶ Temporary table space considerations

Attention: Before creating the table spaces, check if there are any DB2 UDB parameters that apply to your environment (must be set up first). See 2.3.11, “Recommended parameters for performance” on page 82.

Table space container layout

Before you create your table spaces, you need to think about how the container configuration will affect the overall performance of the table spaces.

In this section we provide some guidelines about the table space containers layout definition.

The topics included are:

- ▶ Prerequisites
- ▶ Considerations
- ▶ Recommendations

Prerequisites

Ensure that you read 2.2, “Disk considerations” on page 39.

Considerations

The following general principles can be used to your advantage:

- ▶ For a partitioned database environment, establish a policy that allows partitions and containers within partitions to be spread evenly across storage resources.
- ▶ DB2 query parallelism allows workload to be balanced across CPUs, and if DB2 UDB ESE with the partitioned database option is installed, across data partitions.
- ▶ DB2 UDB I/O parallelism allows workload to be balanced across containers.
- ▶ You may intermix data, indexes, and temporary spaces on RAID arrays. Your I/O activity will be more evenly spread and avoids the skew which you would see if the components were isolated. This has the added advantage of making available more disk arms for each of these objects.

Recommendations

When determining the table space storage:

- ▶ If you have lots of disk, define one table space container per disk, otherwise, spread everything across all disks.
- ▶ At least 4 disks per table space (the more the better)

For example, for the big single table space “TS” (defined on the database partition group with 8 database partitions) we can define one container per disk for each database partition as shown in Figure 2-4, following the *spread and balance* approach.

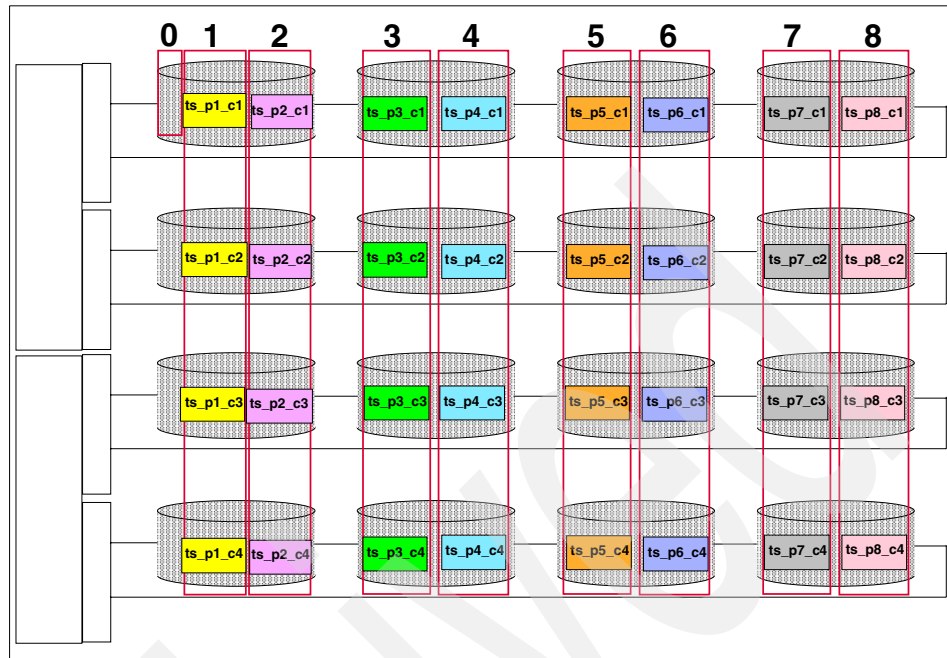


Figure 2-4 Table space containers layout example

Table space SMS or DMS

Once the table space and container design is complete, the next step is to decide between System Managed Storage (SMS) and Database Managed Storage (DMS).

Attention: Separate decisions need to be made for the base table data and temporary space.

The topics included are:

- ▶ SMS table spaces
- ▶ DMS raw table spaces
- ▶ DMS file table spaces
- ▶ Considerations and recommendations

SMS table spaces

SMS stands for system managed space, and you can only define a directory as a container type. DB2 UDB does nothing to manage the space assigned to the table space. The operating system owns the directory, creates, grows, shrinks, and removes the files within it, and controls all of the I/O.

On the negative side, when SMS table spaces grow dynamically, they compete for system resources at a time when those resources are in high demand. Also, you cannot add containers to an SMS database dynamically like you can with DMS table spaces. The only way to add containers is through a backup and redirected restore.

There is also a performance penalty to be paid for using SMS table spaces. First, system I/O buffers are used in addition to the DB2 UDB buffers already allocated. Not only is this an additional layer in the I/O path, but DB2 UDB must compete dynamically with other applications for system I/O buffer resources. The penalty may be heaviest with a write intensive workload.

Another performance penalty is paid by the system journaling of I/O activities and subsequent flushing of that journal to disk. Journaled file systems have come a long way in the past few years, and that penalty is not as severe as it used to be, but it is still present. More importantly, this journaling is more or less duplicated by DB2 logs.

All in all, the conventional wisdom is that you will pay some performance penalty for choosing SMS table spaces, but many people are willing to pay that penalty for the convenience.

DMS raw table spaces

DMS stands for database managed storage. Although the operating system knows that the storage devices exist, that storage is not made available to the system. It is given over to DB2 *raw* and unsullied by any operating system intervention.

In table spaces composed of raw containers, there is no file system overhead. DB2 UDB creates, grows, shrinks, and deletes tables; allocating and freeing space within the container. Table objects, such as Indexes and LOBs can reside in separate DMS table spaces, and containers can be added to them or extended.

Furthermore, no system overhead is expended in preparing and launching an I/O in addition to that already done by DB2 UDB. The data remains grouped in extents; and it is buffered in memory reserved exclusively for DB2 UDB. The system I/O layer is bypassed.

DMS raw table spaces are the best performing for user data, especially for OLTP workloads and large DSS reads, such as applications which insert into append tables.

DB2 raw table spaces are limited to the total size of the containers assigned to them; unused space in those containers cannot be shared.

DMS file table spaces

DMS file table spaces are a compromise between SMS and DMS raw table spaces. In this case, filesystem files are created and then given to DB2 UDB to manage. DB2 UDB handles the files as if they were raw devices.

The drawbacks are that there is a little system overhead in maintaining the container files; the table spaces are limited to the total size of the files assigned; and unused space cannot be shared. The benefits are improved performance over SMS table spaces, and the ability to perform conventional filesystem backups.

Considerations and recommendations

Some basic considerations between SMS and DMS table spaces are shown in Table 2-2.

Table 2-2 Basic considerations between SMS and DMS table spaces

Considerations	SMS	DMS
Managing containers (drop / add, reduce size)	NO	YES
Separating indexes and long from data	NO	YES
No tuning of operating system parameters	NO	YES
Space allocated as needed	YES	NO
Easy administration	YES	NO

We describe some performance considerations for SMS and DMS table spaces as follow:

- ▶ SMS
 - You might want to pay a little more attention to container layout for SMS table spaces, particularly if those table spaces are not spread across multiple containers on separate arrays.
 - Because file managers dynamically allocate disk storage, storage can become fragmented, and therefore it can become more difficult to maintain sequential access on disk.
 - Using SMS, be sure to allocate a sufficient number of containers on a single RAID array, particularly if you have few arrays, and if database load performance is especially important. We recommend 4 to 8 SMS containers for every RAID array, depending on how many arrays will be used. (For example, if a single array is to be used, then we recommend 8 containers, however, if you are using more than 3 arrays, 4 four containers per array may be sufficient.)

► DMS

DMS raw table spaces generally provide the best performance for several reasons:

- The layers of file management are removed from the I/O path.
- Data generally maps fairly simply to underlying storage. Data that appears to DB2 UDB as sequential access also appears to ESS as sequential access, making best use of caching algorithms.
- When using DMS raw devices, DB2 UDB will ensure that pages are placed contiguously on the disk drives; with SMS containers this is not the case, as the filesystem decides where to locate the pages (this can also apply to DMS file containers). Contiguous placement of pages is important, as pages make up DB2 extents. Contiguous pages will speed up operations like table scans. Note that DMS file table spaces require tuning of opsys parameters.

Tip: Some tests indicated much better performance using DMS raw table spaces on DB2 UDB ESE V8.1 for AIX 5 environment on a p690 machine.

Table space definition parameters

In this topic we describe some considerations about page size, extent size, and prefetch size settings that primarily affect the movement of data to and from the disk subsystem and how they work together. We also provide information about overhead and transfer rate parameters that are taken into account when making optimization decisions, and help to determine the relative cost of random versus sequential accesses:

- Container size
- Page size
- Extent size
- Prefetch size
- Overhead
- Transfer rate

Attention: Pay attention to defining the page size and extent size, since you cannot change them after the table space creation, unless you recreate the table space.

Container size

One of the factors which will affect the parallel I/O performance is the size of the containers used. If you use multiple containers of different sizes in a table space,

then the effectiveness of any parallel prefetches performed against that table space will be reduced.

For example:

- ▶ You created a new table space containing one small container and one large container. Eventually, the small container will become full and any additional data will be written to the large container. The two containers are now unbalanced. For example, in BI environment, when you have queries that have large amount of rows, having unbalanced containers reduces the performance of parallel prefetching to nothing, as a **SELECT** statement may result in some data being required from the small container, but a large amount may be required from the large container.
- ▶ You have a table space which contains a number of large containers; these start to fill up, so you add a small container. The table space will then be rebalanced and the small container will contain far less data than will now reside in the larger containers, and once again the database manager will not be able to optimize parallel prefetching.

Tip: Keep the containers with the same size for best performance.

Note: Introduced in DB2 UDB V8.1, is the ability to drop a container from a table space, reduce the size of existing containers, and add new containers to a table space such that a rebalance does not occur using **DROP**, **RESIZE** and **BEGIN NEW STRIPE SET** clauses of the **ALTER TABLESPACE** command. For more detailed information, see *SQL Reference Volume 2 Version 8* SC09-4845.

Page size

The overall performance of your table space will be affected by the page size you choose. Page sizes are defined for each table space during the creation. There are four supported page sizes: 4 K, 8 K, 16 K, and 32 K. Some factors that affect the choice of page size include:

- ▶ The maximum number of records per page is 255. To avoid wasting space on a page, do not make page size greater than 255 times the row size plus the page overhead.
- ▶ For a dimensional model, it is recommended that small page size should be used for the dimension tables, because dimension tables normally are small, and using a small page size, it will not waste disk space and space in the bufferpools.

- Because of the fact that tables which normally are large tables and accesses rows are sequentially, then larger page sizes will provide better performance. Remember the limit of 255 rows per page to avoid wasting disk and bufferpool space. For example, if your row length is 39 bytes, so in one page you can have at the maximum (39 bytes*255 rows)=10 KB. So, if you allocate the page size of 32 KB, you will waste 22 KB per page.
- The maximum size of a table space is proportional to the page size of its table space as shown in Table 2-3. These limits apply at the table space level per database partition. So, if you have a multi-partitioned table space, the maximum size limit would be the correspondent size limit described in Table 2-3 times the number of database partitions where the table space was defined.

Table 2-3 Page size relative to table space size

Page size	Max data/index size per DP	Max row length	Max columns
4KB	64 Gb	4,005 bytes	500
8KB	128 Gb	8,101 bytes	1012
16KB	256 Gb	16,293 bytes	1012
32KB	512 Gb	32,677 bytes	1012

Tip: It is suggested that all table spaces be standardized to 1 page size because there are significant advantages such as:

- Reduce wasting memory by requiring multiple bufferpools for each size.
- Reduce wasting temporary space by either having multiple temporary table spaces or forcing smaller page size table spaces to use the temporary space of the largest page size table space.
- Reduce the factor of determining if multiple page sizes are relevant when doing problem determination.

Important: DB2 catalog table space (SYSCATSPACE) will always use 4 Kb page size, and the 4 Kb bufferpool can be of a small size to satisfy this.

Extent size

The extent size you choose for a table space determines the number of table data pages written to one container before DB2 UDB writes to the next container; the type of table space used is a major factor.

When using DMS table spaces, we allocate space one extent at a time; when one extent is full, a whole new extent is allocated. Data is striped across the container(s) by extent, as in Figure 2-5. Note that extents are mapped in a round-robin fashion, and extents for different objects (table 1 and table 2) need not be contiguous.

This differs from SMS table spaces which allocate space one page at a time.

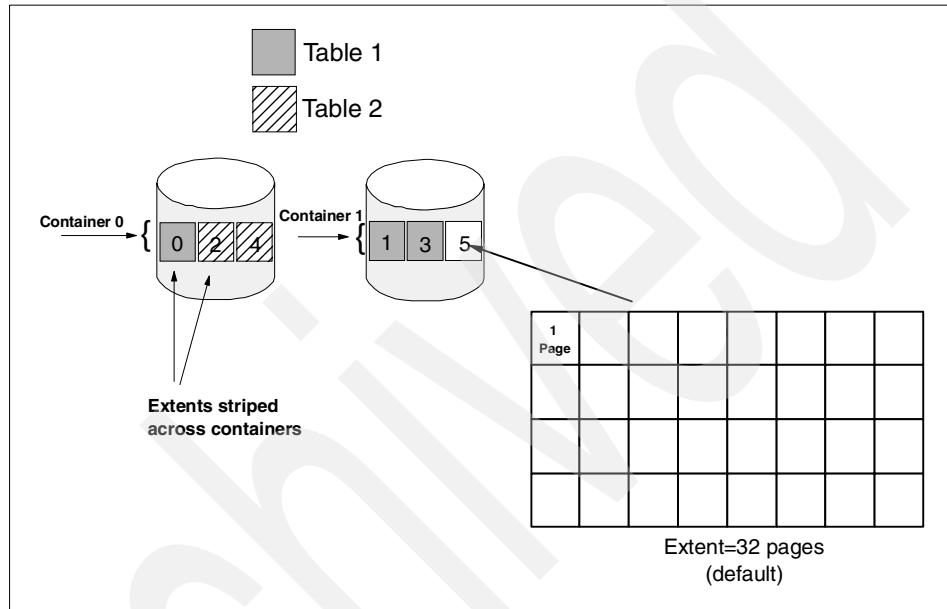


Figure 2-5 DMS table space structure

The optimum extent size value will be related to the actual disk layout used.

Tip: It is good to match extent sizes to operating system limits (for example, AIX supports 16 buffers per read call.)

Use larger values for fact tables that are mostly scanned or have a very high growth rate. If your DMS table space does contain large tables, and you do not increase the value for EXTENTSIZE parameter, but instead use a small extent size, then you will incur an overhead when the database has to continually allocate these small extents (either when a table is expanding or new tables are being created).

When using striped containers (like RAID), extent size should be set to a multiple of the stripe size (such as RAID stripe size).

Tip: In ESS for example, a good starting point would be to set extent size equal to 1 complete stripe of disks in the LUN (for example, for a LUN striped across a 6+P array, set it to 6*32 K = 196 K.

Prefetch size

Setting the prefetch size correctly will improve the database manager's ability to read pages in advance and reduce execution times. You can change the prefetch size value using **ALTER TABLESPACE** command.

If the database's main workload requires good sequential I/O performance, such as a DSS workload, then the *PREFETCHSIZE* becomes even more important.

Follow some considerations and recommendations:

- ▶ For most environments, the recommendation would be to set the *PREFETCHSIZE* equal to (*EXTENTSIZE* * number of containers). For optimal parallel prefetching, the containers in your table space should reside on separate physical devices. This is important when using a DMS raw container, as there will be no prefetching or caching performed by the operating system.
- ▶ For *PREFETCHSIZE* with RAID devices, we recommend that the *PREFETCHSIZE* is the RAID stripe size multiplied by the number of parallel disk drives (do not include parity disk) and also a multiple of the *EXTENTSIZE*. For example, if you have the following table space as shown in Example 2-2:

Example 2-2 Table space example for PREFETCHSIZE with RAID

```
CREATE TABLESPACE TBS1
MANAGED BY DATABASE USING
(DEVICE '/dev/rtbs11v' 2000) PAGESIZE 8K
EXTENTSIZE 8
PREFETCHSIZE 24
```

A relatively small RAID array configuration is used to illustrate these points:

Using an array = 3 + P (3 parallel plus one parity) and a strip size = 64 k, we can say that the above **CREATE TABLESPACE** command will give us the following results:

- Using an *EXTENTSIZE* of 8 pages and a *PAGESIZE* of 8 K, each extent will span 1 drive. The value used for *EXTENTSIZE* should always be equal to or a multiple of the RAID stripe size.

- Using a PREFETCHSIZE of 24 pages, each prefetch done will utilize the 3 parallel disks. The calculation is $\text{strip size} * N / \text{PAGESIZE}$ which equates, in this example, to $64 \text{ KB} * 3 / 8 = 24$. We recommend that the PREFETCHSIZE is the RAID stripe size multiplied by the number of parallel disk drives (do not include parity disk) and also a multiple of the EXTENTSIZE.

Tip: The optimum prefetch size would retrieve data from all physical disks in one I/O operation.

Tip: The prefetch size should be set such that as many arrays as desired can be working on behalf of the prefetch request. In non-ESS storage, the general recommendation is to calculate prefetch size to be equal to a multiple of the extent size times the number of containers in your table space. For ESS, you may work with a multiple of the extent size times the number of arrays underlying your table space.

Note: To help you to tune PREFETCHSIZE for your table spaces, use the database system monitor and other monitor tools. You might gather information about whether:

- ▶ There are I/O waits for your query, using monitoring tools available for your operating system.
- ▶ Prefetching is occurring, by looking at the pool_async_data_reads (bufferpool asynchronous data reads) data element in database monitor output (snapshot output for example).

If there are I/O waits and the query is prefetching data, you might increase the value of PREFETCHSIZE. If the prefetcher is not the cause of the I/O wait, increasing the PREFETCHSIZE value will not improve the performance of your query.

If you wish to increase the PREFETCHSIZE, then you also need to consider increasing the NUM_IOSERVERS database configuration parameter to a least $\text{PREFETCHSIZE} / \text{EXTENTSIZE}$.

The NUM_IOSERVERS database configuration parameter specifies the number of the I/O servers that perform prefetch I/O for the database. You should normally set the same number for the NUM_IOSERVERS parameter as the number of physical disks. If PREFETCHSIZE is increased above (the suggestions outlined

so far for *additional* prefetching), then the NUM_IOSERVERS database configuration parameter should be set to at least PREFETCHSIZE /EXTENTSIZE.

Note: To help you to tune NUM_IOSERVERS, use the snapshot or event monitoring information.

Check the data element prefetch_wait_time. It shows the time an application spent waiting for an I/O server (prefetcher) to finish loading pages into the bufferpool. If the prefetch wait time is considerable, you might increase the NUM_IOSERVERS.

Although larger prefetch values might enhance throughput of individual queries, mixed applications would generally operate best with moderate-sized prefetch and extent parameters. You will want to engage as many arrays as possible in your prefetch, to maximize throughput.

It is worthwhile to note that prefetch size is tunable. By this we mean that prefetch size can be altered after the table space has been defined and data loaded. This is not true for extent and page size which are set at table space creation time, and can not be altered without re-defining the table space and re-loading the data.

Overhead and transfer rates

DB2 UDB defines two parameters related to I/O operational performance: overhead and transfer rate. They are defined during the table space creation, but they can be altered after using **ALTER TABLESPACE** command.

These parameters are taken into account when making optimization decisions, and help determine the relative cost of random versus sequential accesses.

► Overhead

This provides an estimate (in milliseconds) of the time required by the container before any data is read into memory. This overhead activity includes the container's I/O controller overhead as well as the disk latency time, which includes the disk seek time.

The following formula can be used to estimate overhead cost (I/O controller, latency, seek time). A detailed explanation of this formula can be obtained in the *DB2 UDB Administration Guide: Performance, SC09-4821*.

OVERHEAD = average seek time in milliseconds + (0.5 * rotational latency)

In this formula, 0.5 represents an average overhead of one half rotation and rotational latency (milliseconds / full rotation) is given as:

$$(1 / \text{RPM}) * 60 * 1000$$

So, for a disk with an RPM = 10000, we can obtain:

$$(1 / 10000) * 60 * 1000 = 6.0 \text{ milliseconds}$$

So, assume an average seek time of 5.4 milliseconds for this disk type; this gives the following estimated OVERHEAD:

$$5.4 + (0.5 * 6.0) = 8.4 \text{ milliseconds}$$

► Transfer rate

Which provides an estimate (in milliseconds) of the time required to read one page of data into memory.

When calculating the TRANSFERRATE you need to take into account the following considerations if your table space containers are made up of more than one single physical disk (for example, an array such as RAID):

- If the spec_rate value is small, then multiply by the number of physical disks on which the container resides, assuming bottleneck at disk level.
- If the number of disks is large, then I/O bottleneck is not likely to be due to disks, but more likely due to disk controller or I/O buses (system). If this is the case, then you will need to test the actual I/O rate. Refer to the *DB2 UDB Administration Guide: Performance, SC09-4821* for further suggestions on how to do this.

We recommend that if your container spans multiple physical disks, then these disks should ideally have the same OVERHEAD and TRANSFERRATE characteristics.

If they are not the same, or you do not have the hardware documents that contain the information you require for the formulas given, then set the OVERHEAD value to the average of all the disks in the table space. As for TRANSFERRATE, for a DSS environment, set TRANSFERRATE to the sum of all the disks.

The following values described in Table 2-4, are recommended for use with ESS.

Table 2-4 Suggested overhead and transfer rates for ESS

Overhead	12.5
Transfer rate (4KB)	0.1
Transfer rate (8KB)	0.3
Transfer rate (16KB)	0.7
Transfer rate (32KB)	1.4

Tables per table space

Here are some considerations for placing tables, which might help you decide which tables should go:

- ▶ For a dimensional model, it is generally best to place dimension tables in the same table space to conserve space and for the fact tables in their own table space. Snapshot monitoring occurs at the table space level, for bufferpool analysis, so separating tables as much as possible is advantageous. The disadvantage of having too many small table spaces is that considerable disk space may be wasted from over allocation and table space overhead.
- ▶ Consider that the rules of thumb vary but may be in the region of 50-200Gb raw data per partition.
- ▶ Keeping important tables in their own table space will simplify administration of this table space and will speed recovery should the table space need to be restored (because you will only have one table to restore), for example for the fact tables.
- ▶ Consider table space capacity limits: 64 GB (4K pages), 128 GB (8K pages), 512 GB (32K pages), 2 TB for long table spaces. In a partitioned database, they are the limits per partition.
- ▶ Place tables that need to be monitored more in their own table spaces, for example MQTs, since the **LIST TABLESPACE SHOW DETAIL** command gives the number of used pages. If the table space is a DMS table space, the command also reports the number of free pages and the high water mark information. For example, you can monitor the growth of a fact table of a star schema by putting it in its own table space.

Catalog table space considerations

In this topic we describe some considerations about the catalog table space:

- ▶ Catalog table spaces are restricted to using the 4 KB page size.
- ▶ An SMS table space is recommended for database catalogs, for the following reasons:
 - The database catalog consists of many tables of varying sizes. When using a DMS table space, a minimum of two extents are allocated for each table object. Depending on the extent size chosen, a significant amount of allocated and unused space may result. When using a DMS table space, a small extent size (two to four pages) should be chosen; otherwise, an SMS table space should be used.
 - There are large object (LOB) columns in the catalog tables. LOB data is not kept in the bufferpool with other data, but is read from disk each time it is needed. Reading LOBs from disk reduces performance. Since a file system usually has its own cache, using an SMS table space, or a DMS

table space built on file containers, makes avoidance of I/O possible if the LOB has previously been referenced.

Important: Ensure that the catalog partition has been set correctly when the database is first created, since it cannot be changed at a later date. The catalog partition is determined by being the partition connected to when the database is first created. For example:

```
export DB2NODE=0
db2 terminate
db2 create database <dbname>.....
```

or:

```
db2_all "<+0< db2 create database <dbname> ..."
```

Temporary table space considerations

Temporary table spaces are used during SQL operations for internal temporary tables, for sorts, to store intermediate results, table reorganizations, and other transient data. So, in BI environment, which normally has heavy SQL operations and sorts, it is very important that the temporary table space be correctly defined.

Follow some recommendations and considerations for the temporary table space:

- ▶ SMS is almost always a better choice than DMS for temporary table spaces because:
 - There is more overhead in the creation of a temporary table when using DMS versus SMS.
 - Disk space is allocated on demand in SMS, whereas it must be pre-allocated in DMS. Pre-allocation can be difficult: Temporary table spaces hold transient data that can have a very large peak storage requirement, and a much smaller average storage requirement. With DMS, the peak storage requirement must be pre-allocated, whereas with SMS, the extra disk space can be used for other purposes during off-peak hours.
 - The database manager attempts to keep temporary table pages in memory, rather than writing them out to disk. As a result, the performance advantages of DMS are less significant.
- ▶ If you have table spaces utilizing different page sizes, then our suggestion would be to create one temporary table space with a page size equal to that of the majority of your data tables. When selecting temporary table spaces, DB2 UDB ignores page sizes which are too small to accommodate an

operation which requires temporary space. For this reason you will need to ensure that the page size of your temporary table space(s) can accommodate the row lengths and column used in your database; see Table 2-3 on page 59 for a list of page size row and column limits.

- The reason we suggest only having one temporary table space for a given page size is because when you have multiple temporary table spaces using same page sizes, the DB2 optimizer will generally choose a temporary table space size by selecting the temporary table space with the largest bufferpool.

For example, you had two 16 KB temporary table spaces defined called TEMP1 and TEMP2. Now assume TEMP1 has a large bufferpool and TEMP2 has only a small bufferpool; UDB will select TEMP1 as it has largest bufferpool. Once selection is made, UDB will then alternate between ALL temporary table spaces, which have the chosen page size (16 KB).

This means that when performing an operation such as a sort, we alternate between TEMP1 and TEMP2 because they both use a 16 KB page size. The disadvantage here is that we only use TEMP1's large bufferpool half the time, because we are alternating between the two table spaces. A better solution in this example would be to have a single 16 KB temporary table space with one bufferpool.

Note: If you reorganize table spaces in the temporary table space, then you will need a temporary table space that has the same page size as the table space being reorganized.

Note: Catalog table spaces are restricted to using the 4 KB page size. As such, the database manager always enforces the existence of a 4 KB system temporary table space to enable catalog table reorganizations.

2.3.5 Understand partitioning map

In a partitioned database environment, the database manager must have a way of knowing which table rows are stored on which database partition. The database manager must know where to find the data it needs, and uses a pmap called a partitioning map, to find the data.

A partitioning map is an internally generated array containing either 4 096 entries for multiple-partition database partition groups. The partition numbers of the database partition group are specified in a round-robin fashion.

For example, in Figure 2-6 we provide a partitioning map example for a database partition group that contains the partitions 1, 2, 3, and 4.

The 4096 hash buckets with the partition numbers represent the partitioning map (pmap) and the partition numbers are specified in a round-robin manner.

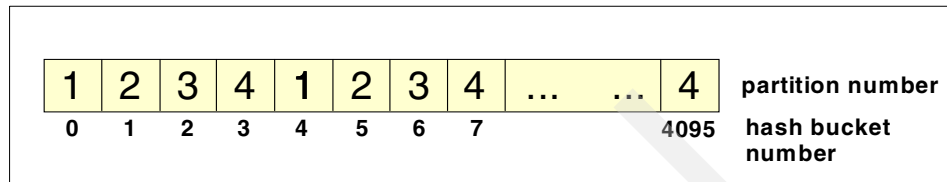


Figure 2-6 Partitioning map

2.3.6 Choose the partitioning key

A partitioning key is a column (or group of columns) that the database manager uses to determine how data is distributed between the partitions; that is, the location (the database partition) where the data is stored. A partitioning key is defined on a table using the **CREATE TABLE** statement.

Hash partitioning is the method by which the placement of each row in the partitioned table is determined. The method works as follows:

1. The hashing algorithm is applied to the value of the partitioning key, and generates a partition number between zero and 4095.
2. The partitioning map is created when a database partition group is created. Each of the partition numbers is sequentially repeated in a round-robin fashion to fill the partitioning map.
3. The partition number is used as an index into the partitioning map. The number at that location in the partitioning map is the number of the database partition where the row is stored.

In this section the topics covered are:

- ▶ Why choose a good partitioning key?
- ▶ Why an inappropriate partitioning key should not be chosen?
- ▶ How to choose a good partitioning key?

Why choose a good partitioning key?

Choosing a good partitioning key is important for the following main reasons:

- ▶ An even distribution of data across the partitions
- ▶ Maximum use of colocated joins as this avoids the database manager having to ship data between partitions.

Note: Collocation between two joining tables means having to have the matching rows of the two tables always in the same database partition.

Why an inappropriate partitioning key should not be chosen?

An inappropriate partitioning key can cause uneven data distribution. Columns with unevenly distributed data, and columns with a small number of distinct values should not be chosen as a partitioning key. The number of distinct values must be great enough to ensure an even distribution of rows across all database partitions in the database partition group. Example 2-3 shows the ORDERS table creation command.

Example 2-3 ORDERS table creation command

```
create table ORDERS
( O_ORDERKEY integer not null, O_CUSTKEY integer not null,
  O_ORDERSTATUS char(1) not null, O_TOTALPRICE float not null,
  O_ORDERDATE date not null, O_ORDERPRIORITY char(15) not null,
  O_CLERK char(15) not null, O_SHIPPRIORITY integer not null,
  O_COMMENT varchar(79) not null)
```

If you choose the O_ORDERSTATUS as a partitioning key, probably you will have uneven data distribution, since this column certainly has a small number of distinct values. You should not choose this column as a partitioning key.

A good candidate for the partitioning key would be O_CUSTKEY or O_ORDERKEY, because of a high number of distinct values. Also, the queries should be considered for the table collocation to define good partitioning key candidate as describe in “How to choose a good partitioning key?” on page 70.

Another consideration about inappropriate partitioning key is the cost of applying the partitioning hash algorithm that is proportional to the size of the partitioning key. The partitioning key cannot be more than 16 columns, but fewer columns result in better performance. Unnecessary columns should not be included in the partitioning key.

Tip: To check if you have uneven data distribution, you can run the following statement. This statement will show you the number of rows per partition you have in a table:

```
db2 "select dbpartitionnum(partkey), count(*) from <schema.table> *  
group by dbpartitionnum(partkey) order by dbpartitionnum(partkey)"
```

Substitute the appropriate values for schema, table and partitioning key column.

Follow an example result with number of rows for each partition where the table resides:

1	2
-----	-----
	1 281645
	2 281417
	3 282250
	4 281854
	5 281444
	6 280939
	7 281515
	8 281141
	9 281684
	10 280886
	11 280267
	12 279949

12 record(s) selected.

How to choose a good partitioning key?

In this section we provide the main considerations about choosing a good partitioning key.

The topics covered are:

- ▶ Rules for good partitioning key candidate
- ▶ Rules govern the use of partitioning keys
- ▶ Partition compatibility
- ▶ Table collocation
- ▶ Replicated Materialized Query Table

Rules for good partitioning key candidate

To choose a good partitioning keys candidate, you should consider the following rules:

- ▶ Frequently joined columns

- ▶ Columns that have a high proportion of different values to ensure an even distribution of rows across all database partitions in the database partition group.
- ▶ Integer columns are more efficient than character columns which are more efficient than decimal.
- ▶ Equijoin columns
- ▶ Use the smallest number of columns possible.

Rules for the use of partitioning keys

The following rules are for the use of partitioning keys:

- ▶ Creation of a multiple partition table that contains only long data types (LONG VARCHAR, LONG VARCHARIC, BLOB, CLOB, or DBCLOB) is not supported.
- ▶ The partitioning key definition cannot be altered.
- ▶ The partitioning key should include the most frequently joined columns.
- ▶ The partitioning key should be made up of columns that often participate in a GROUP BY clause.
- ▶ Any unique key or primary key must contain all of the partitioning key columns.
- ▶ Tables containing only long fields cannot have partitioning keys and can only be placed into single-partition database partition groups.
- ▶ If a partitioning key is not defined, one is created by default from the first column of the primary key. If no primary key is specified, the default partitioning key is the first non-long field column defined on that table. (Long includes all long data types and all large object (LOB) data types.)
- ▶ If you are creating a table in a table space associated with a single-partition database partition group, and you want to have a partitioning key, you must define the partitioning key explicitly. One is not created by default. If no columns satisfy the requirement for a default partitioning key, the table is created without one. Tables without a partitioning key are only allowed in single-partition database partition groups.

Restriction: To ADD a partitioning key at a later time, the table must be defined in a table space on a single-partition database partition group and must not already have a partitioning key. If the partitioning key already exists for the table, the existing key must be dropped before adding the new partitioning key.

A partitioning key cannot be added to a table that is a subtable.

Partition compatibility

The base data types of corresponding columns of partitioning keys are compared and can be declared partition compatible. Partition compatible data types have the property that two variables, one of each type, with the same value, are mapped to the same partition number by the same partitioning algorithm.

Partition compatibility has the following characteristics:

- ▶ A base data type is compatible with another of the same base data type.
- ▶ Internal formats are used for DATE, TIME, and TIMESTAMP data types. They are not compatible with each other, and none are compatible with CHAR.
- ▶ Partition compatibility is not affected by columns with NOT NULL or FOR BIT DATA definitions.
- ▶ NULL values of compatible data types are treated identically; those of non-compatible data types may not be.
- ▶ Base data types of a user-defined type are used to analyze partition compatibility.
- ▶ Decimals of the same value in the partitioning key are treated identically, even if their scale and precision differ.
- ▶ Trailing blanks in character strings (CHAR, VARCHAR, GRAPHIC, or VARGRAPHIC) are ignored by the hashing algorithm.
- ▶ BIGINT, SMALLINT, and INTEGER are compatible data types.
- ▶ REAL and FLOAT are compatible data types.
- ▶ CHAR and VARCHAR of different lengths are compatible data types.
- ▶ GRAPHIC and VARGRAPHIC are compatible data types.
- ▶ Partition compatibility does not apply to LONG VARCHAR, LONG
- ▶ VARGRAPHIC, CLOB, DBCLOB, and BLOB data types, because they are not supported as partitioning keys.

Table collocation

In a partitioned database, the performance of join queries can be greatly enhanced through collocation of rows of the different tables involved in the join.

Collocation between two joining tables means having to have the matching rows of the two tables always in the same database partition.

Note: If tables are in the same database partition group, they have the same pmap or partitioning map. For collocation, there must be equijoin predicates between all corresponding partitioning key columns.

Figure 2-7 describes an collocated join example, where the CUST and ORDERS tables have been partitioned on *cust_id* column. An SQL query that requires a join on the *cust_id* column will see significant performance benefits from this partitioning scheme, because of the greater parallelism achieved through collocated joins.

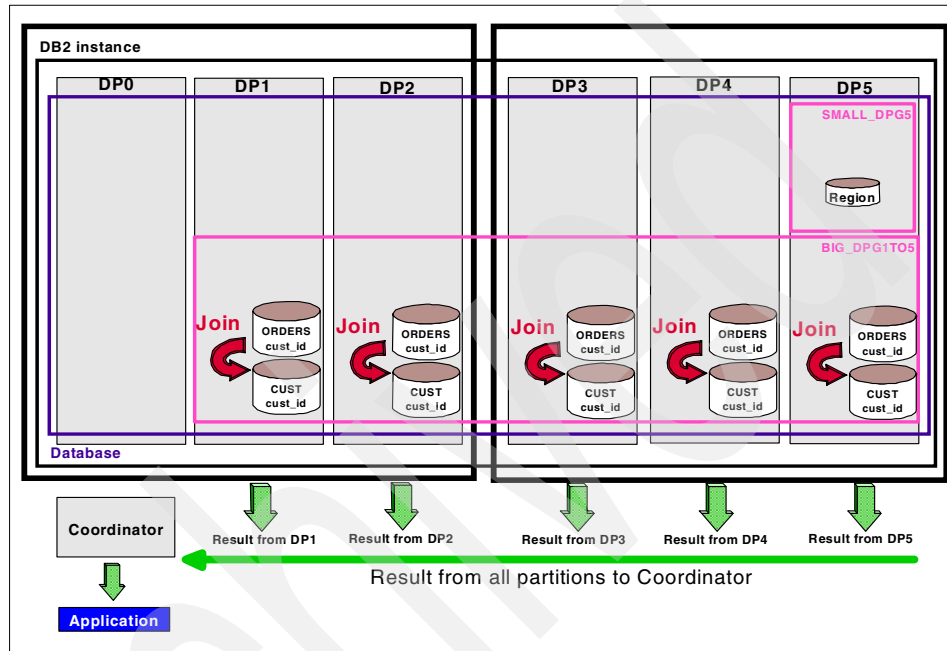


Figure 2-7 Table collocation

A collocated join will occur if two collocated tables are joined using all of the columns in the partitioning key.

Tables are collocated when they satisfy the following requirements:

- ▶ The tables must be in same database partition group.
- ▶ The database partition group of both the tables must have the same partitioning map.
- ▶ The partitioning keys from both tables must have the same number of columns.
- ▶ Corresponding partitioning key columns must be partition compatible (the same or similar).

We always have to use the maximum of collocated joins as this avoids the database manager having to ship data between partitions.

You can now use MQTs to replicate tables to other database partitions to enable collocated joins to occur even though the all tables are not joined on the partitioned key. An example is provided in 5.4, “Intra database replicated tables and partitioning” on page 173.

2.3.7 Size the tables

The environment example we used to calculate the table sizes is:

- ▶ 2 p690 (32-way / 128GB memory)
- ▶ 32 database partitions on each machine (Total = 64 database partitions)

Figure 2-8 details the estimated table sizes used in our test.

Table Name	Row Count	Row length	Total Size (GB)	Size / SMP (GB)	Size / DP (GB)
lineitem	5999989709	146.3	817.5	408.7	6.38
orders	1500000000	128	178.8	89.4	1.39
customer	150000000	195	27.2	13.6	0.21
part	200000000	163.8	30.5	15.25	0.24
supplier	10000000	170.7	1.6	0.8	0.01
partsupp	800000000	163.8	122	61	0.95
nation	25	n/a	0		
region	5	n/a	0		
TOTAL	8659989739		1177.6	588.8	9.2

Figure 2-8 Table sizes

Follow the meaning of each columns described in Figure 2-8:

- ▶ **Row count**
The number of rows in the table.
- ▶ **Row length**
The byte counts sum of all columns in the table.
- ▶ **Total size**
 $\text{Total size} = \text{Row count} * \text{Row length}$
- ▶ **Size / SMP**
 $\text{Size / SMP} = \text{Total size} / \text{number of physical machines}$
In this example, we used 2 p690 machines.
- ▶ **Size / DP**
 $\text{Size / DP} = (\text{Size / SMP}) / \text{number of database partitions per machine}$

In this example, we used 32 partitions per machine.

2.3.8 Size for MDC utilization

You will get significantly improved performance and maintenance advantage when using MDC, if you choose the right set of dimensions for clustering a table and the properly extent sizes. On the other hand, if incorrect choices are made, utilization of space might be poor and performance could degrade. To design good MDC tables you must first identify potential dimension candidates, and then evaluate the storage space needed for an MDC table using some variation of those dimensions.

For details how to design MDC, see Chapter 4., “Speed up performance with MultiDimensional Clustering” on page 135.

2.3.9 Size for MQT utilization

A materialized query table is a table which contains pre-computed results whose definition is based on the result of a query to attend many applications that have a pattern of analysis using more or less similar queries with minor variations in a query's predicates, and they are often issued repetitively against large volumes of data.

Based on this, the size for a MQT is estimated using the same calculation as for a base table. For more details about table sizing, see 2.3.7, “Size the tables” on page 74.

But, if your MQT is also defined as MDC, you also have to consider the MDC sizes. For more details about MDC sizes, see Chapter 4., “Speed up performance with MultiDimensional Clustering” on page 135.

2.3.10 Configure DB2 UDB

In this section we provide some important DB2 UDB configurations and recommendations for your data warehouse.

The topics included are:

- ▶ Database logs
- ▶ Bufferpool considerations
- ▶ DIAGPATH directory path placement

Database logs

In this section we will describe management and performance considerations relating to database logging. If set up inappropriately for your workload, database logging could become a significant overhead.

The topics covered are:

- ▶ On which disks are the logs to be placed?
- ▶ Circular or archival log
- ▶ Logging performance
- ▶ Filesystem or raw logical volume
- ▶ Mirroring
- ▶ Number of log files
- ▶ Size of logs

On which disks are the logs to be placed?

Due to the importance of the log files, it is recommended that the log files should always reside on their own physical disk(s), separate from the rest of the database objects. The disks ideally should be dedicated to DB2 logging to avoid the possibility of any other processes accessing or writing to these disks.

Tip: It is highly recommended to place the database logs on separate disks from the data.

Availability

Irrespective of the type of logging you choose, whether it be circular logging or archival logging, the availability of the physical disks is crucial to the database. For this reason, it is strongly recommended that you protect the log against single disk failures:

- ▶ By using RAID 1 (mirroring the log devices) recommended for SSA disks
- ▶ By storing log files on a RAID 5 array recommended for ESS disks. Since a RAID 5 array has a large cache, the write penalty effects can be significantly decreased.

For details about RAID levels, see topic 2.2.1, “Summary of the most popular RAID levels” on page 39.

Circular or archival log

DB2 UDB can operate in one of two modes relating to transaction logging:

- ▶ Circular logging

This means that writes to the database are logged to enable rollback of outstanding units of work. However, after a commit, log records relating to that

unit of work are not retained, and the log files will be reused in a circular manner. This means that it is not possible to replay transactions to recover data when operating in this mode. Online backups, table space backups, and roll forward are not allowed when using circular logging.

► Archival logging

In this mode writes are logged in the same way, but log records are retained after a commit. This means that following a restore of the database, it is possible to replay transactions or roll forward through log records, up to a point in time or the end of the log records.

There are some other advantages, for example, a database in archival logging mode can be backed up using an online backup (reads and writes to the database are allowed at the same time), and also can be backed up and restored at the table space level.

The choice of logging mode is generally determined by the type of data being written and the availability requirements. If the data is *first time data*, cannot be reproduced elsewhere, and has a significant cost associated with any loss, then generally archival logging is required. Also, if the database is required for read/write access 24x7 then online backup may also be a requirement.

If this is not the case, then for ease of management and simplicity, circular logging is generally preferred.

If you have sufficient time available to take offline backups, perhaps you may plan a cycle of full backups followed by one or more incremental backups.

Note: In your BI environment, if your data load is scheduled (for example every night) it is recommended that you use circular logging if you have sufficient time available to take offline backups.

But, if your BI environment has several updates during the day, and if the database is required for read/write access 24x7, then archival logging is recommended, since online backups have to be taken. In this case, it is important to have good log administration (for example, using TSM, because logs are required for the database restore.)

Logging performance

The log files exist to provide the ability to be able to recover your environment to a consistent state, and preserve the integrity of your data, so logging performance is very important.

Placement of these files needs to be optimized, not only for write performance, but also for read performance, because the database manager will need to read the log files during database recovery.

Note: In DB2 UDB V7.1, the total log files may be up to 32 GB. In DB2 UDB V8.1, total log files can now be up to 256 GB in size, and you can perform extremely large amounts of work within a single transaction.

Filesystem or raw logical volume

We now have two options when deciding how to store the DB2's transaction log files: either in the operating system's filesystem or in a raw logical volume. The most familiar method here is to use the filesystem, as this is all we have supported in the past for logs. Using raw logical volumes for logging will bring performance improvements, specifically for Solaris environments for improving log writes performance, similar to those improvements gained from switching from table space file containers to DMS raw devices. However, it is worth pointing out that in most situations there are advantages and disadvantages to using raw logical volumes for your log devices:

Advantages

- ▶ The I/O path is shorter, since you are now bypassing the operating system's filesystem.
- ▶ Raw device striping may provide faster I/O throughput.
- ▶ No restrictions are imposed by a filesystem.
- ▶ A raw device can span multiple disks.

Disadvantages

- ▶ The device (logical volume) created for the logs must be dedicated to DB2 UDB.
- ▶ The device (logical volume) created for the logs cannot be operated upon by any operating system utility or third-party tools, which would back up or copy from the device.
- ▶ Currently, if you are using IBM data replication products, the read log API will not call the user exit if you use the raw log device. The recommendation is that IBM data replication products should not be used when using raw log devices. Similarly, do not use raw log devices if you use the **sqlurlog** API.

Mirroring

As the log files are crucial to your database, you can either mirror your log devices to ensure that the disks they reside on are mirrored, or use the database configuration parameter **mirrorlogpath**. It is recommended that the log disks be

entirely dedicated to DB2 UDB, and that no other processes writes to these disks. If possible, place the disks on separate disk controllers to maximize availability.

Number of log files

Your active log space will be determined the number of log files you define. As we have seen already, the active log space is defined by the parameters (LOGPRIMARY + LOGSECOND)*LOGFILSZ. You need to take into account your database workload, and therefore the potential logging activity.

Most DSS workloads with little update, insert, or delete activity may benefit from a smaller number of log files. However, in this environment, where logging activity is low, it may be worthwhile using a larger log file size to reduce administration overhead. For example, if you are using Archival logging, the log files must be well administrated, since they are required for database restoration, so having larger log files and a smaller number of log files is easier to administrate instead of having smaller log files and larger number of log files.

Size of logs

Remember that the total active log space can now be up to 256 GB. For partitioned database, this limit is per partition.

So the calculation would be:

$$(\text{LOGPRIMARY} + \text{LOGSECONDARY}) * \text{LOGFILSZ} * 4k < 256 \text{ GB}$$

In this formula, LOGFILSZ represents the Log File Size database configuration parameter.

If you are using raw devices for logging, then remember that log records are still grouped into log extents; the size of each is LOGFILSZ (4 KB pages). DB2 UDB places the log extents consecutively in the log device, and every log extent carries with it a two page overhead for extent header information. This affects the number of log extents that can be written to your raw log device. The total number can be calculated by the formula:

$$\text{raw-device-size} / (\text{logfilsz} + 2)$$

The device must be large enough to support the active log space that you are going to allocate. By this we mean that the number of log extents that can be contained on your raw device must be greater than (or equal to) the value of LOGPRIMARY.

Bufferpools considerations

In a DSS environment, the main memory allocation is for bufferpool and sorting. For information about sort parameters, see “DB2 UDB parameters” on page 82.

The general rule of thumb for a DSS environment is to start with about 50% of your system's main memory devoted to bufferpool(s), but this rule has to be considered *very* carefully before being followed. The biggest factor for consideration is if this is a dedicated database server.

Tip: For the first run, we provide some recommendations to estimate bufferpool size for a DSS environment:

$MDB \text{ (Memory for DB)} = \text{Total Memory} - \text{AIX Memory} - \text{Network Memory}$

$MDP \text{ (Memory per DP)} = (MDB / \text{Number DPs}) - (\text{all other DB parameters})$

$BP = MDP * (\text{between } 0.5 \text{ and } 0.75) \Rightarrow \text{it means } 50\% \text{ to } 75\% \text{ of memory}$

$MDB = \text{Memory for DB}$

$MDP = \text{Memory for database partition}$

$DP = \text{database partitions}$

$BP = \text{bufferpools}$

AIX Memory estimation = 16MB

Network Memory estimation = 16MB

All other DB parameters estimation = 16MB

Depending on plans, you can give more memory to sorting, since the DSS environment involves large joins and complex aggregations, which can require heavy sorting.

At least one bufferpool per table space page size is required. In a DSS environment we would suggest starting with as few bufferpools as possible (1 per page size) with one exception, which is to use a small bufferpool for the fact table (since you are never going to get much of the table in memory, the hit rate will not be good), but big enough of course to get a few prefetches running concurrently. Then you can allocate a bigger bufferpool to the dimension tables, so they become memory resident without having pages flushed out every time the fact table is scanned. In general, bufferpools do not need to be as large as in a OLTP environment, and memory may be better utilized for sorting (SORTHEAP and SHEAPTHRES).

To help you to tune bufferpool size, use the database system monitor (snapshot or event monitor) to gather information to get the *bufferpool hit ratio* information.

The bufferpool hit ratio indicates the percentage of time that the database manager did not need to load a page from disk in order to service a page request. That is, the page was already in the bufferpool. The greater the bufferpool hit ratio, the lower the frequency of disk I/O. The bufferpool hit ratio can be calculated as follows:

$(1 - ((\text{pool_data_p_reads} + \text{pool_index_p_reads}) /$

$(\text{pool_data_l_reads} + \text{pool_index_l_reads})) * 100\%$

This calculation takes into account all of the pages (index and data) that are cached by the bufferpool. You can find the values for calculation using the database system monitor (snapshot or event monitor).

Note: The bufferpool size specified in the **CREATE BUFFERPOOL** command is per partition. So, DB2 UDB will allocate this amount for each database partition, which belongs to the database partition group you specified in the **CREATE BUFFERPOOL** command.

For example, if you have the database partition group “DPG_ALL” which contains five database partitions, and then you execute the following command:

```
create bufferpool BP_16KB database partition group DPG_ALL size 32000  
pagesize 16k
```

The calculation is:

BPTotal Size = (size in pages*page size) * number of partitions in DP group

So, in the example there will be $(32000 * 16 \text{ kb}) * 5 = 2.4 \text{ Gb}$ for BP Total Size.

Note 1: For 64-bit database, remember that you can define larger bufferpools, SORTHEAP, package caches, and the other resources that can consume large amounts of memory. For more details about 64-bit support, see 9.2, “64-bit considerations” on page 270.

Note 2: Each bufferpool page allocated uses space in the memory allocated for the database heap (DBHEAP), for internal control structures. To reduce the need for increasing the DBHEAP when bufferpools increase in size, nearly all bufferpool memory comes out of the database shared-memory set and is sized *online, automatically*.

DIAGPATH directory placement

DIAGPATH is a database manager parameter that allows you to specify the fully qualified path for DB2 UDB diagnostic information.

In addition to defining where the db2diag.log, traps, and dump files will be written, with DB2 UDB ESE V8.1 the DIAGPATH parameter controls where the

new Administration Notification log is written. The name of this log is the combination of the instance name and *.nfy*; for example, *db2inst1.nfy*. This new type of diagnostic log is used to write notification messages intended for use by database and system administrators. For example, any alerts generated by the new health indicators are written to this log. Detailed diagnostic information is still written to the *db2diag.log*.

It is recommended to separate the diagnostic information and data to different disks to avoid disk contention, preferentially in local disks. This separation can also be used to ensure that information written into the diagnostics directory will not be able to fill up your data or log file systems, and can cause DB2 UDB to crash.

The *DIAGPATH* is a parameter that affects all databases in the instance. In a partitioned database environment that involves more than one physical machine, the path you specify must either reside on a shared file system accessible by all machines, or the same path must be defined on each machine. If the path is on a shared file system, diagnostic information from all partitions will be written to the same diagnostic logs. If the same path is defined on each machine on local file systems, the diagnostic logs on each machine will contain only the information from the partitions that are located on it.

2.3.11 Recommended parameters for performance

The information below gives hints, tips, and suggestions for your partitioned data warehouse and have been gathered from various IBM experiences with customers using DB2 UDB ESE V8.1.

The topics included are:

- ▶ DB2 UDB parameters
- ▶ AIX parameters
- ▶ Network parameters

DB2 UDB parameters

In this topic, we describe the main recommended DB2 UDB parameters which includes:

- ▶ DB2 variables
- ▶ DB2 database manager parameters
- ▶ DB2 database parameters

DB2 variables

Follow the main recommended DB2 variables:

Attention: A `db2stop` and `db2start` must be done in order for the DB2 variables to take effect.

DB2_STRIPED_CONTAINERS

When creating a DMS table space container, a one-page tag is stored at the beginning of the container. The remaining pages are available for storage by DB2 UDB, and are grouped into extent-size blocks of data.

When using RAID devices for table space containers, it is suggested that the table space be created with an extent size that is equal to, or a multiple of the RAID stripe size. However, because of the one-page container tag, the extents will not line up with the RAID stripes, and it may be necessary during the I/O request to access more physical disks than would be optimal. This is particularly important when the RAID devices are not cached, and do not have special sequential detection and prefetch mechanisms (such as native SSA disks).

The reason this variable does not affect DMS file containers is that the pages allocated for the DMS file container are still allocated by the filesystem, so we cannot guarantee that the pages in the container will be contiguous across the disk(s). With raw devices, DB2 UDB controls this and allocates the pages contiguously.

When DB2_STRIPED_CONTAINERS is set to ON when the table space is created, the initial container tag will take up a full extent. This will avoid the problem described above.

Note: We recommend that DB2_STRIPED_CONTAINERS is set to on. With ESS, sequential detection and prefetch may alleviate this problem.

DB2_PARALLEL_IO

We recommend to set DB2_PARALLEL_IO variable if using multiple physical disks for each container.

For example, if the container is created on a RAID device that is composed of more than one physical disk, you may want to issue parallel read and write calls.

When you specify the table space's ID to the DB2_PARALLEL_IO registry variable, the database is able to issue parallel I/O requests to that table space, even though it consists of a single container. DB2 UDB achieves this by enabling multiple prefetchers simultaneously for the table space, which results in improved utilization of the underlying physical drives. If DB2_PARALLEL_IO is on then the number of prefetchers that DB2 UDB dispatches is equal to `prefetchsize/extentsize`.

You can enable parallel I/O for every table space using the DB2_PARALLEL_IO variable, as shown in the following example:

```
db2set DB2_PARALLEL_IO=*
```

You can also enable parallel I/O for a list of table spaces IDs (comma-separated) as in the following example:

```
db2set DB2_PARALLEL_IO=1,2,4
```

A list of table space IDs can be obtained by running the **LIST TABLESPACES** command when connected to your database, or you could run this command:

```
SELECT TBSPACE,TBSPACEID FROM SYSCAT.TABLESPACES
```

After setting the DB2_PARALLEL_IO registry variable, you must run **db2stop** and then **db2start** for the change to take effect.

DB2_PARALLEL_IO should also be enabled for striped containers when PREFETCHSIZE > EXTENTSIZE.

DB2_FORCE_FCM_BP

Enable the DB2_FORCE_FCM_BP variable since it is recommended for partitioned database environments, which has multiple logical database partitions in a physical machine. It uses shared memory for inter-partition communication instead of the network, and provides a faster communication between the database partitions. When the DB2_FORCE_FCM_BP variable is set to Yes, the FCM buffers are always created in a separate memory segment, so that communication between FCM daemons of different logical partitions on the same physical partition occurs through shared memory. Otherwise, FCM daemons on the same partition communicate through UNIX sockets and communicating through shared memory is faster.

The following command enables DB2_FORCE_FCM_BP variable:

```
db2set DB2_FORCE_FCM_BP=YES
```

DB2 database manager parameters

Follow the main recommended DB2 database manager configuration parameters:

► **SHEAPTHRES and SHEAPTHRES_SHR**

In a DSS environment, the main memory allocation is for bufferpool and sorting (SHEAPTHRES, SHEAPTHRES_SHR and SORTHEAP). For information about bufferpool estimate size, see “Bufferpools considerations” on page 79.

SHEAPTHRES is used to control the total amount of memory allocated for sorting on the DB2 UDB server, and applies on a per partition basis. This parameter should be set to the maximum amount of memory for all

SORTHEAP that should be allowed to be allocated at any given time, specifically when using connection concentrator.

The rule of thumb to calculate SHEAPTHRES is:

$$\text{SHEAPTHRES} = (\text{SORTHEAP} * \# \text{ concurrent sorts})$$

The SHEAPTHRES parameter is used differently for private and shared sorts:

- For private sorts, this parameter is an instance-wide soft limit on the total amount of memory that can be consumed by private sorts at any given time. When the total private-sort memory consumption for an instance reaches this limit, the memory allocated for additional incoming private-sort requests will be considerably reduced.
- For shared sorts, this parameter is a database-wide hard limit on the total amount of memory consumed by shared sorts at any given time. When this limit is reached, no further shared-sort memory requests will be allowed (until the total shared-sort memory consumption falls below the limit specified by sheapthres).

This parameter is completely related to SORTHEAP parameter. For sort tune tips, see “DB2 database configuration recommendations” on page 87, SORTHEAP parameter.

SHEAPTHRES_SHR represents a hard limit on the total amount of database shared memory that can be used for sorting at any one time. When the total amount of shared memory for active shared sorts reaches this limit, subsequent sorts will fail (SQL0955C). If the value of SHEAPTHRES_SHR is 0, the threshold for shared sort memory will be equal to the value of the SHEAPTHRES database manager configuration parameter, which is also used to represent the sort memory threshold for private sorts. If the value of SHEAPTHRES_SHR is non-zero, then this non-zero value will be used for the shared sort memory threshold. SHEAPTHRES_SHR is only meaningful in two cases:

- If the intra_parallel database manager configuration parameter is set to yes, because when INTRA_PARALLEL is set to no, there will be no shared sorts.
- If the Concentrator is on (that is, when max_connections is greater than max_coordagents), because sorts that use a cursor declared with the WITH HOLD option will be allocated from shared memory.

INTRA_PARALLEL

This should only be set on if you have more than 2 CPUs per partition and you have low query concurrency. This is particularly important for large SMPs, where we could end up with 100s of agents dispatched for a single query, and 1000s of DB2 UDB processes running concurrently, in which case, task switching becomes a significant overhead. The large numbers of processes running is another reason to consider physical partitioning.

When running INTRA_PARALLEL or DFT_DEGREE set to YES with multiple data partitions per SMP, set MAX_QUERYDEGREE to number of processors per data partition for example on a 32-way p690 with 8 data partitions, set MAX_QUERYDEGREE to 4.

CPUSPEED

Set this database manager configuration parameter to -1. This will be the most recommended value for DB2 UDB. By doing this, DB2 UDB will calculate the value.

FCM_NUM_BUFFERS

If you have multiple logical database partitions on the same machine, you may find it necessary to increase the value of this parameter.

It is recommended to start with a default value (4 096). If you run out of message buffers because of the number of users on the system, or the number of database partition servers on the system, or the complexity of the applications, you may find it necessary to increase it.

Tip: To tune FCM parameters, use the database monitor to monitor the low watermark for the free buffers, free message anchors, free connection entries, and the free request blocks. If the low water mark is less than 10 percent of the number of the corresponding free data item, increase the value of the corresponding parameter.

What follows is a snapshot output example from database partition 0, and no FCM low water mark is less than 10 percent of the corresponding free data item:

```
db2 get snapshot for database manager | grep -i fcm
```

Node FCM information corresponds to	= 0
Free FCM buffers	= 3068
Free FCM buffers low water mark	= 2816
Free FCM message anchors	= 1530
Free FCM message anchors low water mark	= 1484
Free FCM connection entries	= 1536
Free FCM connection entries low water mark	= 1504
Free FCM request blocks	= 2001
Free FCM request blocks low water mark	= 1973
Number of FCM nodes	= 33
Memory Pool Type	= FCMBP Heap

FCM message anchors, FCM connection entries, and FCM request blocks are now self-tuning.

DB2 database configuration recommendations

Follow the main DB2 database configuration recommendations:

► **Use DMS with raw logical volumes**

Using DMS table spaces with raw logical volumes on the p690 is proving to provide an even greater improvement to I/O performance than it has been on other AIX systems.

► **SORTHEAP**

SORTHEAP is allocated to each agent, and agents are started per partition for a query accessing the table on a partition. The agent does not necessarily allocate a full SORTHEAP. If the optimizer calculates less amount of SORTHEAP and this amount is enough for the sort, then smaller amount of sort memory will be requested by the agent at runtime.

Tip: For the first run, we provide some recommendations to estimate SHEAPTHRES and SORTHEAP for a DSS environment:

$MDB \text{ (Memory for DB)} = \text{Total Memory} - \text{AIX Memory} - \text{Network Memory}$

$MDP \text{ (Memory per DP)} = (MDB / \text{Number DPs}) - (\text{all other DB parameters})$

$SHT = MDP * (\text{between } 0.5 \text{ and } 0.25) \Rightarrow \text{it means } 50\% \text{ to } 25\% \text{ of memory}$

$SH = SHT / \# \text{ of concurrent sorts}$

MDB =Memory for DB

MDP =Memory for database partition

DP = database partitions

SHT =SORTHEAP threshold

SH =SORTHEAP

AIX Memory estimation = 16MB

Network Memory estimation = 16MB

All other DB parameters estimation = 16MB

Depending on your plans, you can give more memory to sorting, since the DSS environment involves large joins and complex aggregations, which can require heavy sorting.

Important: It is important to allocate as much memory as possible to the SORTHEAP (and set the threshold accordingly) without over allocating memory and causing memory paging to occur. It is possible for a sort to be done entirely in sort memory. We will recommend setting SHEAPTHRES to 50% of the DB2 UDB memory, and then divide by the number of concurrent sorts expected to get SORTHEAP. Typically, this gives SORTHEAP in the region of 5000 to 10000 pages.

However, if this causes the operating system to perform page swapping, then you can lose the advantage of a large SORTHEAP. So, whenever you adjust the SORTHEAP and SHEAPTHRES configuration parameters, use an operating system monitor to track any changes in the system paging.

In the sort phase, the sort can be categorized as overflowed or non-overflowed. When considering the return of the results of the sort phase, the sort can be categorized as piped or non-piped:

- Overflowed and non-overflowed

If the information being sorted cannot fit entirely into the SORTHEAP (a block of memory that is allocated each time a sort is performed), it overflows into temporary database tables. Sorts that do not overflow (non-overflowed) always perform better than those that do.

- Piped and non-piped

If sorted information can return directly without requiring a temporary table to store a final, sorted list of data, it is referred to as a *pipedReader sort*. If the sorted information requires a temporary table to be returned, it is referred to as a *non-pipededReader sort*. A pipedReader sort always performs better than a non-pipededReader sort.

The DB2 optimizer will determine if a non-overflowed sort can be performed, and if a pipedReader sort can be performed by comparing the expected result set with the value of the SORTHEAP database configuration parameter.

Tip: For good sort performance, always try to have non-overflowed and pipedReader sorts.

To help you to tune SORTHEAP and SHEAPTHRES, you can use the system performance monitor, snapshot for example:

- To check if you have overflowed sorts:

You have to consider the total sorts and sort overflows elements as shown in Example 2-4.

You can calculate the percentage of overflow sorts by sort overflows/total sorts, and use the value to determine if the optimizer is attempting to use a SORTHEAP and fails. If the percentage is high, consider increasing the SORTHEAP and/or SHEAPTHRES values.

Example 2-4 Snapshot for database output

db2 get snapshot for database on pfp grep -i sort	
Total Private Sort heap allocated	= 2059
Total Shared Sort heap allocated	= 0
Shared Sort heap high water mark	= 0
Total sorts	= 5
Total sort time (ms)	= 55749
Sort overflows	= 0
Active sorts	= 0

- To check pipedReader and non-pipededReader sorts:

You have to consider the *PipededReader sorts requested* and *PipededReader sorts accepted* elements as shown in Example 2-5.

You can calculate the percentage of piped sorts accepted by piped sorts accepted/piped sorts requested, and determine if piped sorts are being chosen by the optimizer, but not accepted. If the percentage is low, consider increasing the SORTHEAP and/or SHEAPTHRES.

Example 2-5 Snapshot output for database manager

```
db2 get snapshot for database manager |grep -i sort
Private Sort heap allocated          = 0
Private Sort heap high water mark    = 2059
Post threshold sorts                 = 0
Piped sorts requested                 = 1
Piped sorts accepted                  = 1
Sorting Information                   (SORT) = ON 12-06-2002 16:54:21.712838
```

Important: On a partitioned database, you can take snapshot of the current partition, a specified partition, or all partitions using global snapshot.

NUM_IOSERVERS

The NUM_IOSERVERS database configuration parameter specifies the number of the I/O servers which perform prefetch I/O for the database. You should normally set the same number for the NUM_IOSERVERS parameter as the number of physical disks. If PREFETCHSIZE is increased above the suggestions outlined so far (for *additional* prefetching), then the NUM_IOSERVERS database configuration parameter should be set to at least PREFETCHSIZE /EXTENTSIZE. For a large system, beware not to make this too big. Remember that it is specified per partition. To calculate it, if you stripe every table space across each rank you can easily end up with a total number of prefetchers (NUM_IOSERVERS*#partitions) that would be as many times as the total number of disks. A realistic number may be a maximum of 20-30 IO servers per partition.

Note: To help you to tune NUM_IOSERVERS, use the snapshot or event monitor information.

Check the data element prefetch_wait_time. It shows the time an application spent waiting for an I/O server (prefetcher) to finish loading pages into the bufferpool. If the prefetch wait time is considerable, you might increase the NUM_IOSERVERS.

NUM_IOCLEANERS

This parameter allows you to specify the number of asynchronous page cleaners for a database. These page cleaners write changed pages from the bufferpool to disk before the space in the bufferpool is required by a database agent. This means that the agents will not wait for changed pages to be written out. As a result, your application's transaction can run faster.

Important: DB2 UDB V8.1 uses AIO for page cleaners on some platforms (for example AIX) to improve performance of activities such as page cleaning and prefetching. AIO is most effective if data in containers is distributed across multiple disks.

So, you need to tune AIO on the operating system. To configure the number of AIO servers on AIX, use the AIO **minservers** and **maxservers** parameters. The manuals suggest not to increase the number of AIO servers to more than 80.

You can start setting it up to the number of CPUs, and you can always decrease from there.

You may use the database system monitor to help you tune **num_iocleaners** configuration parameter using information from the event monitor about write activity from a bufferpool:

- The parameter can be reduced if both of the following conditions are true:
 - **pool_data_writes** is approximately equal to **pool_async_data_writes**
 - **pool_index_writes** is approximately equal to **pool_async_index_writes**.
- The parameter should be increased if either of the following conditions are true:
 - **pool_data_writes** is much greater than **pool_async_data_writes**
 - **pool_index_writes** is much greater than **pool_async_index_writes**.

Tip: Since page cleaner IO is now asynchronous, a smaller value can be set in **num_iocleaners** than in DB2 UDB V7. You can start with 2.

Multi-page file allocation

SMS table spaces are expanded on demand. This expansion is done a single page at a time by default. However, in certain work loads (for example, when doing a bulk insert) you can increase performance by using the **db2empfa** tool to tell DB2 UDB to expand the table space in groups of pages or extents. The **db2empfa** tool is located in the bin subdirectory of the sqllib directory. Running

it causes the **multipage_alloc** database configuration parameter to be set to YES. In a multipartition database, **db2empfa** must be run on every partition.

AIX parameters

In this section, we provide the main AIX parameters that should be taken into account when you use DB2 UDB. With some parameters we can provide a general recommendation for DB2 UDB V8.1, and for other ones, the recommended values will depend on your hardware configuration:

► Asynchronous IO (AIO)

DB2 UDB has made improvements to page cleaner I/O. To take full advantage of DB2 UDB ESE V8.1 ability to use AIO to improve I/O performance, the AIX AIO must be enabled. To determine the current status in your environment, from the AIX command window run **lsdev -Cc aio**

Your system has AIO already setup if the output received is as below:

```
aio0 Available Asynchronous I/O
```

It is not configured properly if it returns the message displaying Defined instead of Available

```
aio0 Defined Asynchronous I/O
```

To activate it, from the AIX command window run **mkdev -l aio0**

► maxuproc

Maxuproc is the maximum number of process per user, and the initial recommended value is 4096.

Note: The command to change the **maxuproc** parameter is:

```
chdev -l sys0 -a maxuproc='4096'
```

► maxperm

Specifies the point above which the page stealing algorithm steals only file pages. This value is expressed as a percentage of the total real-memory page frames in the system. The specified value must be greater than or equal to 5. Check using **vm tune** AIX tool.

► minperm

Specifies the point below which file pages are protected from the repage algorithm. This value is a percentage of the total real-memory page frames in the system. The specified value must be greater than or equal to 5. Check using **vm tune**.

► **maxservers / minservers**

These parameters are used to configure the number of asynchronous I/O (AIO) servers. You might tune AIO on the AIX since DB2 UDB uses AIO, such as page cleaning and prefetching.

Here are some suggested rules of thumb for determining what value to set maximum number of servers to:

- a. The first rule of thumb is to limit the maximum number of servers to a number equal to ten times the number of disks that are to be used concurrently, but no more than 80. The minimum number of servers should be set to half of this maximum number.
- b. Another rule of thumb is to set the maximum number of servers to 80 and leave the minimum number of servers set to the default of 1 and reboot. Monitor the number of additional servers started throughout the course of normal workload. After a 24-hour period of normal activity, set the maximum number of servers to the number of currently running AIO + 10, and set the minimum number of servers to the number of currently running AIO - 10. In some environments, you may see more than 80 AIO KPROCs running. If so, consider the third rule of thumb.
- c. A third suggestion is to take statistics using **vmstat -s** before any high I/O activity begins, and again at the end. Check the field **iodone**. From this you can determine how many physical I/Os are being handled in a given wall clock period. Then increase the maximum number of servers, and see if you can get more iodone in the same time period.

Tip: Manuals suggest not to increase the number of AIO servers to more than 80

► **extshm**

The extend shared memory option from AIX for extending the shared memory segment or region up to 256 MB, is not recommended when using with DB2. Do *not* use it with DB2 UDB.

► **vm tune**

Tip: When using vmtune, use the following parameters:

```
/usr/samples/kernel/vmtune -p 5 -P 10 -r 128 -R 512 -f 3600 -F  
18960 -u 16 -t 10 -b 4096 -B 9216
```

Network

In this topic, we provide some initial recommendations for network parameters values for DB2 UDB V8.1 for a partitioned database:

```

thewall      =65535
sb_max       =1310720
rfc1323      =1
tcp_sendspace =221184
tcp_recvspace =221184
udp_sendspace =65536
udp_recvspace =65536
ipqmaxlen    =250
somaxconn    =1024

```

Note: To change these parameters (tcp_sendspace, tcp_recvspace, rfc1323) in AIX, execute the following command:

```
no -o <parameter> = <value>
```

Gigabyte Ethernet

If you are using Gigabyte Ethernet, consider the following:

- ▶ Enable jumbo frame
- ▶ Use interface specific no options
- ▶ Enable rfc1323

SP switch

If you are using SP switch and a Corsaire adapter (the Corsaire adapter does not use interface specific options), consider these recommended values:

```

spoolsize = 33554432 (32MB)
rpoolsize = 33554432 (32MB)

```

Note: To set these parameters, enter the following commands:

```
/usr/lpp/ssp/css/chgcss -l css0 -a spoolsize=33554432
```

```
/usr/lpp/ssp/css/chgcss -l css0 -a rpoolsize=33554432
```

If you are encountering packet drops over the switch, setting the following parameters to the suggested settings will enable faster recovery of any lost packets, because of the lower fasttimo value:

- ▶ **no -o fasttimo = 50** (Down from 200)
- ▶ tcp_nagle_limit = 0 (The default is 65535, setting it to 0 will cause the packets to be sent right away. This is equivalent to tcp_nodelay)

2.4 Installing and configuring DB2 UDB ESE V8.1

In this section we provide a generic tasks checklist about DB2 UDB ESE V8.1 installation and information about the DB2 objects creation based on case study

Details about DB2 UDB ESE V8.1 installation, refer to manual *Quick Beginnings for DB2 Servers Version 8, GC09-4836-00*.

The sections included in this chapter are:

- ▶ Generic tasks checklist to install and configure DB2 UDB ESE V8.1
- ▶ Creating DB2 objects based on case study

2.4.1 Generic checklist to install and configure DB2 UDB ESE V8.1

In this topic we provide the checklist to install and configure the DB2 UDB ESE V8.1.

To install and configure DB2 UDB ESE V8.1, we performed the following tasks:

1. Check prerequisites and update AIX environment settings
2. Create DB2 home directory and export NFS
3. Create groups and users required
4. Install DB2 UDB ESE V8.1
5. Create DB2 instance
6. Check and add profile entries
7. Add services entries
8. Edit db2nodes.cfg
9. Create .rhosts
10. Start the DB2 instance

Check prerequisites and update AIX environment settings

The prerequisites should be checked before installation and you can see the manual *Quick Beginnings for DB2 Servers Version 8, GC09-4836-00*.

The main AIX environment settings recommendation values for the DB2 UDB ESE V8.1 installation are:

- ▶ AIX parameter
`maxuproc=4096`
- ▶ Network parameters

```

thewall      =65535
sb_max       =1310720
rfc1323      =1
tcp_sendspace =221184
tcp_recvspace =221184
udp_sendspace =65536
udp_recvspace =65536
ipqmaxlen    =250
somaxconn    =1024

```

If you are using switch, consider these recommended values:

```

spoolsize = 33554432 (32MB)
rpoolsize = 33554432 (32MB)

```

Create a DB2 home file system and export with NFS

When you create the DB2 home file system you have to make it available to all the computers that will participate in your partitioned database system.

You have to do 3 tasks:

1. Create a DB2 home file system
 - To create a DB2 home file system, you can use SMIT JFS.
 - In our environment, we created the **/db2home** file system.
 - Minimum space required is 90 MB.
2. Export the DB2 home file system to all participating partitioned servers
 - You can use SMIT to export the **/db2home** file system.

```

smitty nfs
-->Network File System (NFS)
---->Add a Directory to Exports List

```

Note: It is important to list the host names of the other servers that participate in your partitioned system in **HOSTS allowed root access** filed during the definition of **Add a Directory to Exports List**.

3. Mount the DB2 home file system from all participating partitioned servers
 - The exported **/db2home** directory has to be mounted from all participating partitioned servers. The SMIT option is:

```

smitty nfs
-->Network File System (NFS)
---->Add a File System for Mounting

```

Create groups and users required

Three users and groups are required to operate DB2 and they have to be created on all computers that participate in your partitioned database environment.

Important: Users and groups must be identical on all partitions in the cluster (same UIDs and GIDs).

Table 2-5 describes the required users and groups.

Table 2-5 Required users and groups

Required user	user name	group name
Instance owner	db2inst1	db2iadm1
Fenced user	db2fenc1	db2fadm1
Administration server user	db2as	db2asgrp

You can specify your own user and group names as long as they adhere to your system naming rules and DB2 UDB naming rules.

Note: If you will use DB2 Setup wizard to install the DB2 UDB ESE V8.1, it allows you to create the groups and users or use the existing ones during the instances creation steps. See topic , “Create DB2 instance” on page 101.

Install DB2 UDB ESE V8.1

It is high recommended that the DB2 UDB ESE V8.1 code be installed in all computers that participate in your partitioned system, because it is better for performance. You can use a response file to install on other servers.

But, if you use NFS instead, you will have low administration but suboptimal performance.

You can install the DB2 UDB ESE V8.1 product manually or using DB2 Setup wizard.

- ▶ If you do a manually installation, you can use **smit installp** and in **SOFTWARE to install** field, you select the DB2 filesets you want to install.
- ▶ But if you want to use DB2 Setup wizard, you have to execute **db2setup** in the installation path.

Note: The DB2 Setup wizard, besides installing the product, it also create and configure DAS and DB2 instances for partitioned environment. So, the steps “Create DB2 instance” on page 101, “Check and add profile entries” on page 105 and “Add services entries” on page 105 are done during the DB2 Setup wizard installation.

In our environment we executed the **db2setup** command and in Figure 2-9 we show the DB2 Setup wizard welcome window.

```
{loire:root}/db2esev81aix5L64/ese.sbcs ->. ./db2setup
```

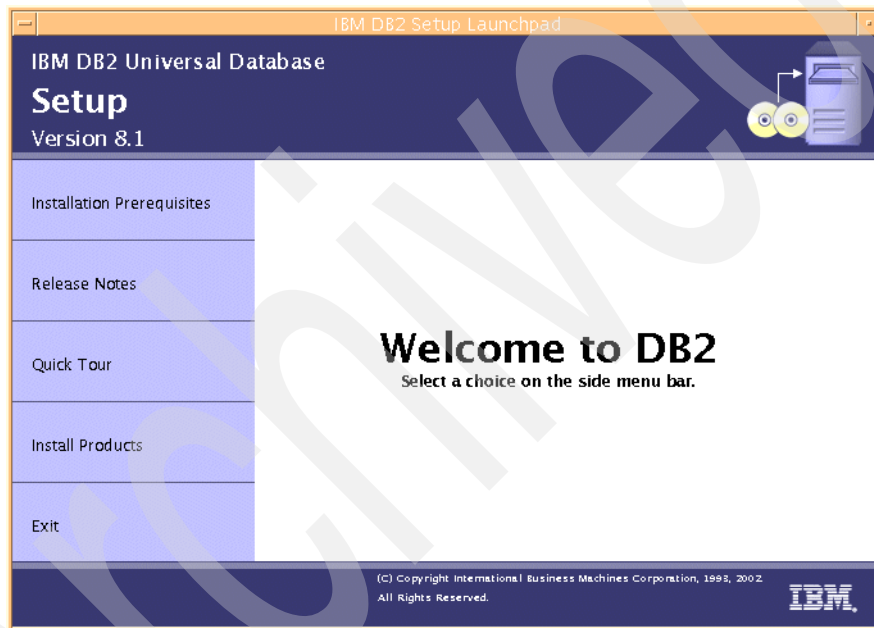


Figure 2-9 DB2 Setup welcome

In Figure 2-10, we show the DB2 Setup wizard window to select the features you want to install. And notice that the left side of the window you have all the installation and configuration steps.

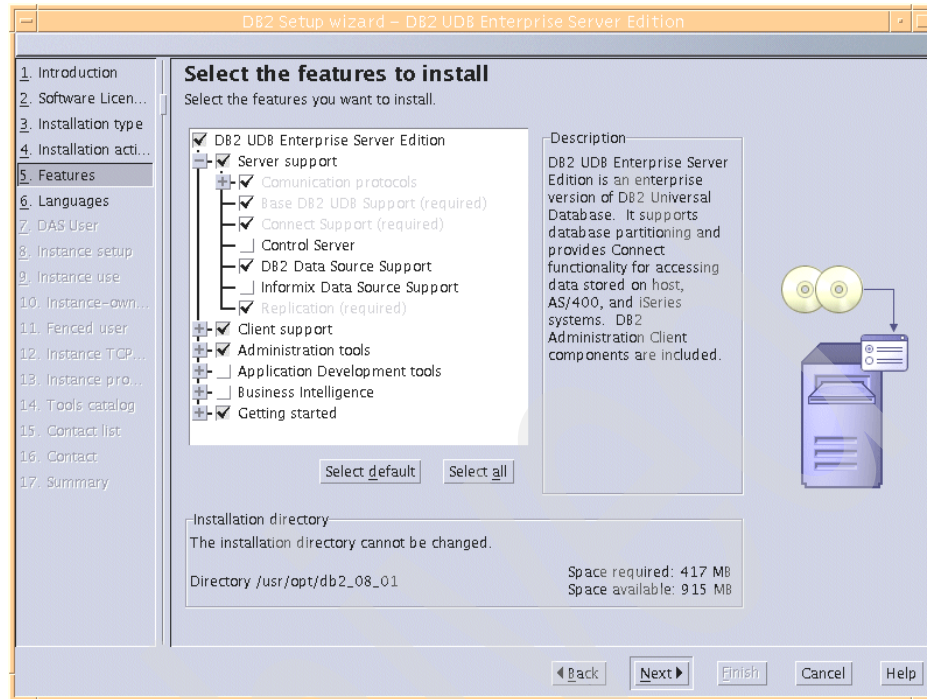


Figure 2-10 DB2 Setup wizard features

Note: The DB2 UDB V8.1 for AIX installation path now is `/usr/opt/db2_08_01` directory as shown in Figure 2-10.

If you want to install other DB2 features after the installation, besides using `smiinstallp` and select the additional filesets you want to install, you can also run the DB2 Setup wizard again as shown in Figure 2-11 and then you can select the additional feature you want to install as shown in Figure 2-12.

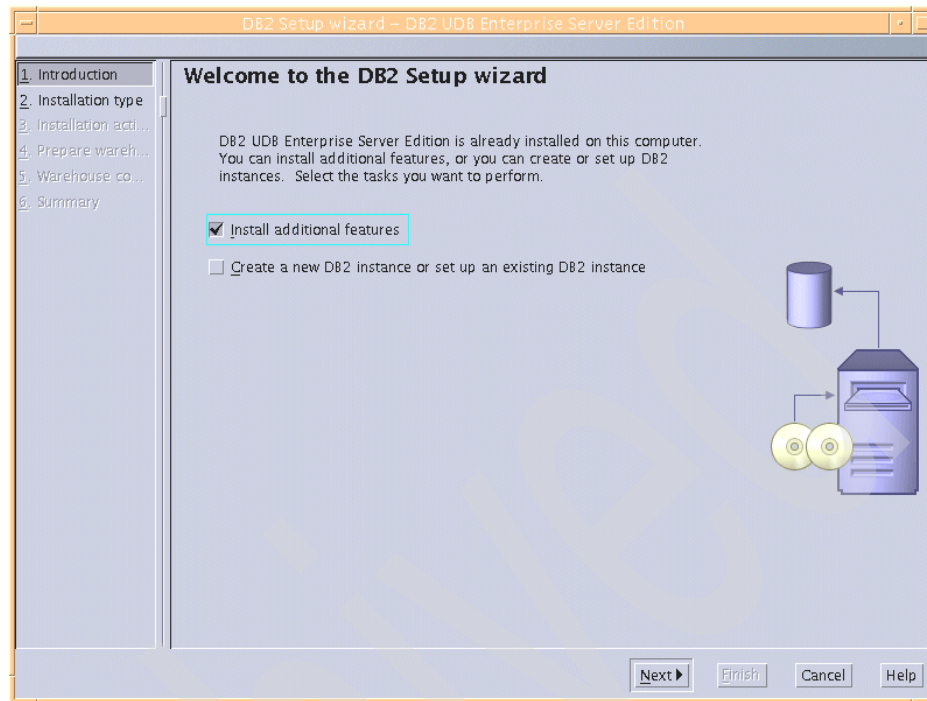


Figure 2-11 DB2 Setup install features

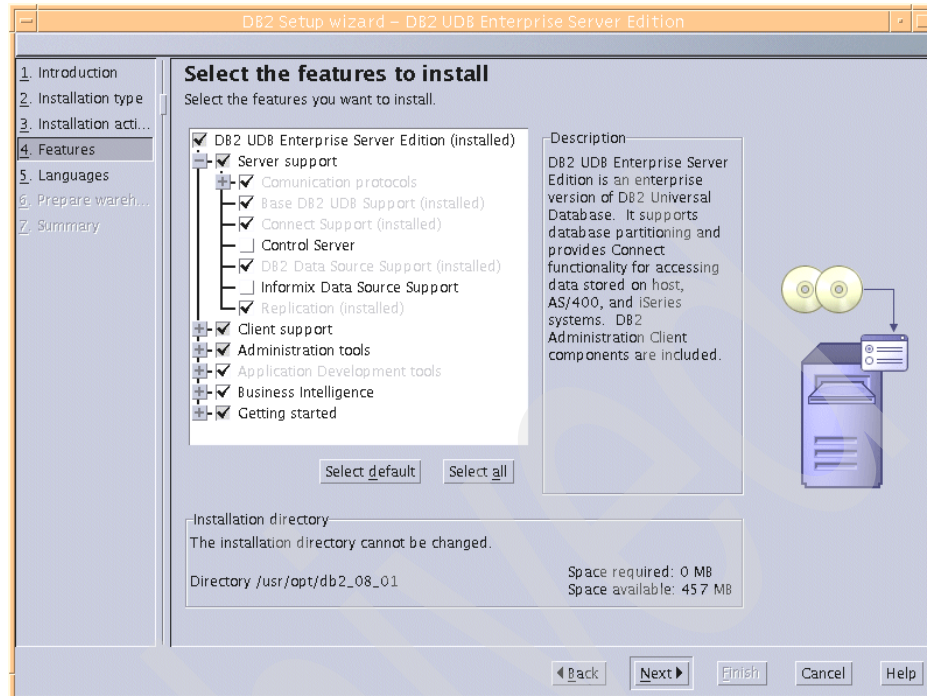


Figure 2-12 DB2 Setup select features

Create DB2 instance

You can create the DB2 instance manually or through DB2 Setup wizard.

- To create the instance manually, you should be logged in as root using the `db2icrt` command.


```
{loire:root}/usr/opt/db2_08_01/instance -> ./db2icrt -u db2fenc1 db2inst1
```
- If you chose to install DB2 using DB2 Setup wizard, there are three steps to define the DB2 instance.

The first one, is the **Instance setup** step. You have to specify if your DB2 instance will be 32-bit or 64-bit, as shown in Figure 2-13.

The second one, is the **Instance use** step where you have to select if this DB2 instance will be a Single-partition instance or it will be a Partitioned instance. Choose the Partitioned instance option as shown in Figure 2-14.

And the last one, is the **Instance-owning user** step. This is the step where you have to define the user and group that will own the DB2 instance. You can specify a new or existing user and group as shown in Figure 2-15.

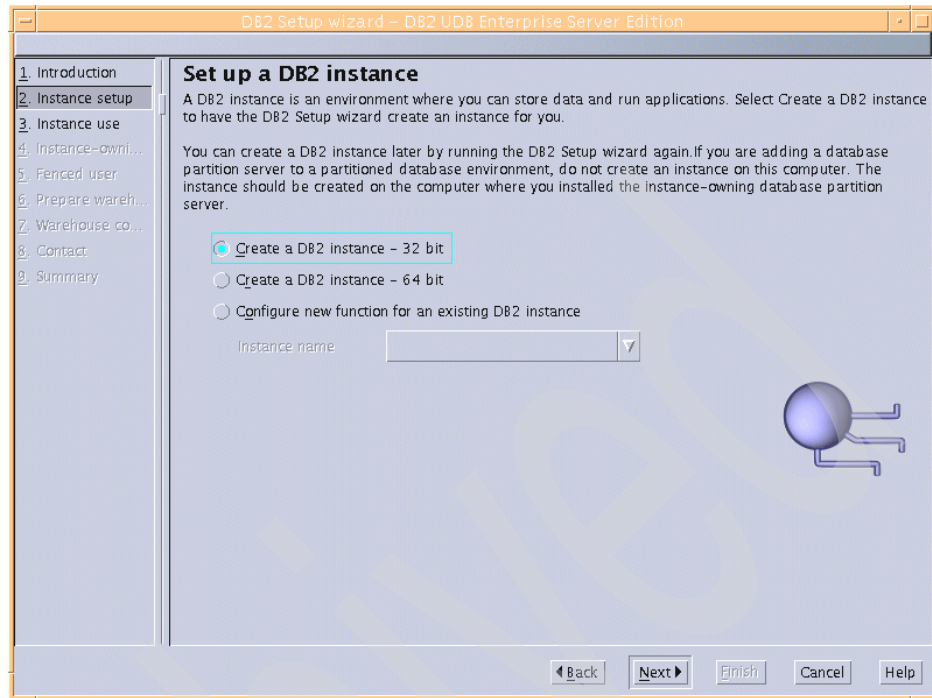


Figure 2-13 DB2 Setup instance setup

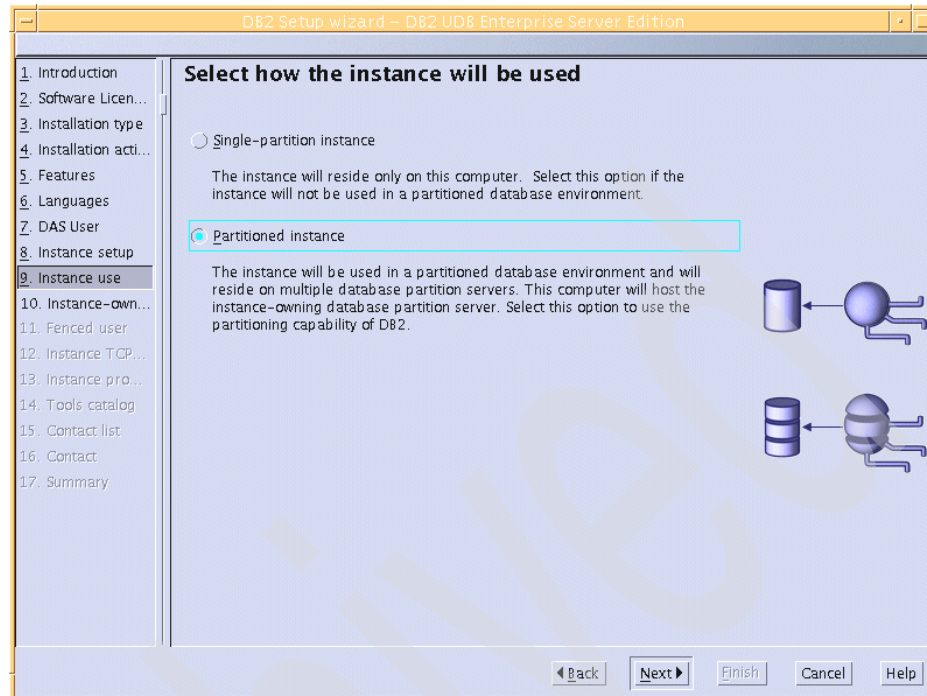


Figure 2-14 DB2 Setup instance use

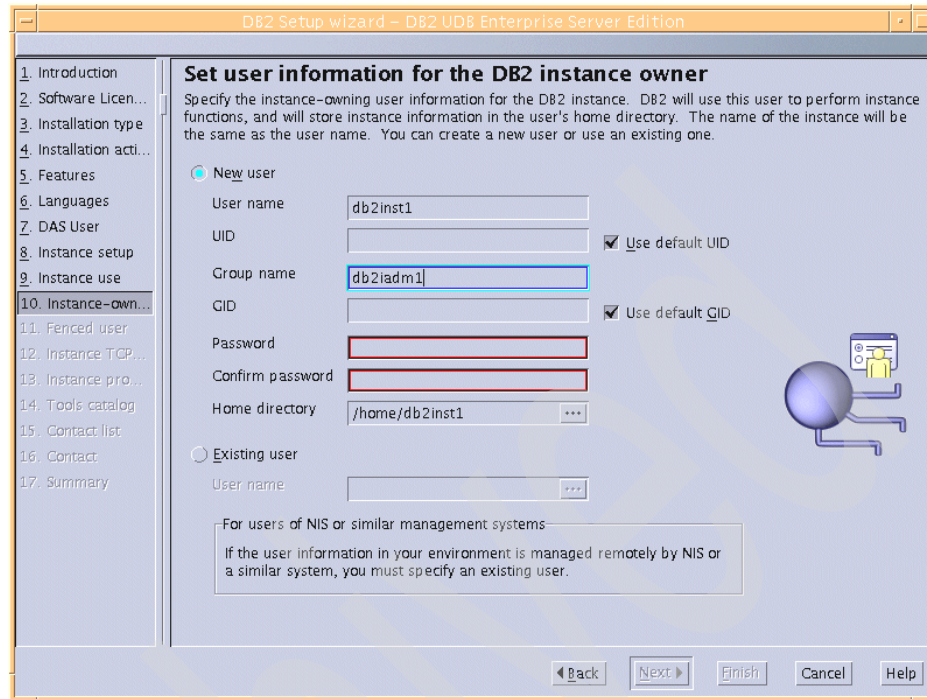


Figure 2-15 DB2 Setup instance owner

If you want to create other DB2 instance after the installation, besides executing **db2icrt** command, you can run the DB2 Setup wizard again and it will give you an option to create new instances as shown in Figure 2-16.

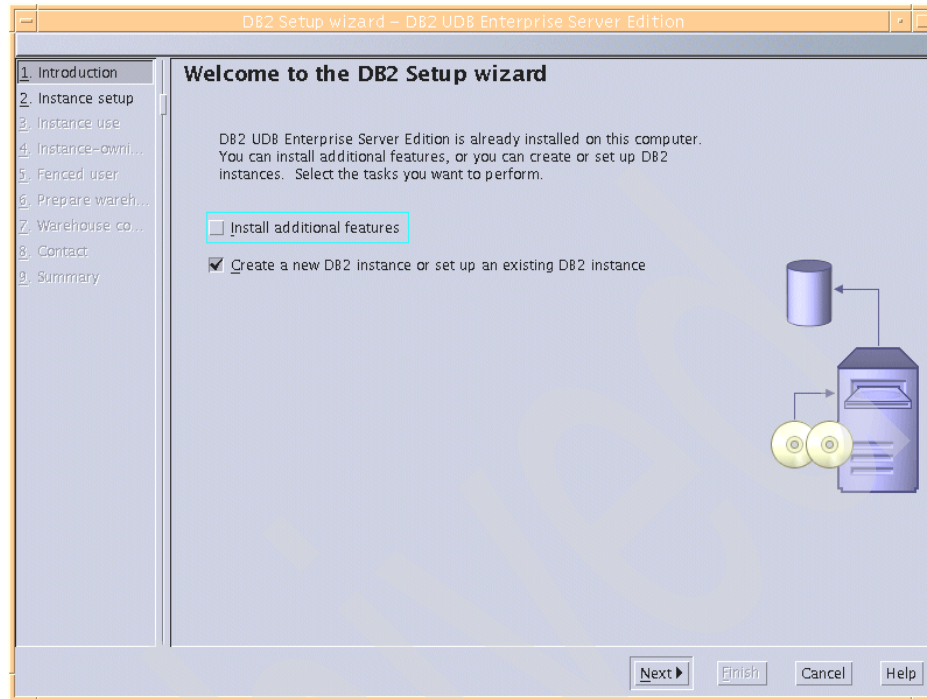


Figure 2-16 DB2 Setup create new instance

Check and add profile entries

The .profile from instance owner user must contain an entry to execute the \$INSTHOME/sql/lib/db2profile file.

If you created the DB2 instance manually, you should add the db2profile execution entry in .profile.

But, if you created the DB2 instance through DB2 Setup wizard, the following lines are added automatically in .profile:

```
# The following three lines have been added by UDB DB2.
if [ -f /db2home/db2inst1/sql/lib/db2profile ]; then
    . /db2home/db2inst1/sql/lib/db2profile
fi
```

Add services entries

To enable communication between database partition servers, ports for each database partition must be reserved in the /etc/services file on all servers.

When you create the DB2 instance manually, the following 4 entries are created automatically in `/etc/services` file as shown in Example 2-6, it means that if you have 4 ports reserved, you can have at the maximum 4 database partitions in your DB2 instance. But then you can change them for your partitioned environment.

Example 2-6 /etc/services entries for DB2 instance

```
DB2_db2inst1 60000/tcp
DB2_db2inst1_1 60001/tcp
DB2_db2inst1_2 60002/tcp
DB2_db2inst1_END 60003/tcp
```

Note: The only mandatory entries are the beginning (DB2_InstanceName) and ending (DB2_InstanceName_END) ports. The other entries are reserved in the services file so that other applications do not use these ports.

If you create the DB2 instance using DB2 Setup wizard, you can specify the number of database partitions in the **Maximum logical nodes** field as shown in Figure 2-17 and then during the DB2 instance creation, the entries are added in `/etc/services` automatically as in Example 2-6.

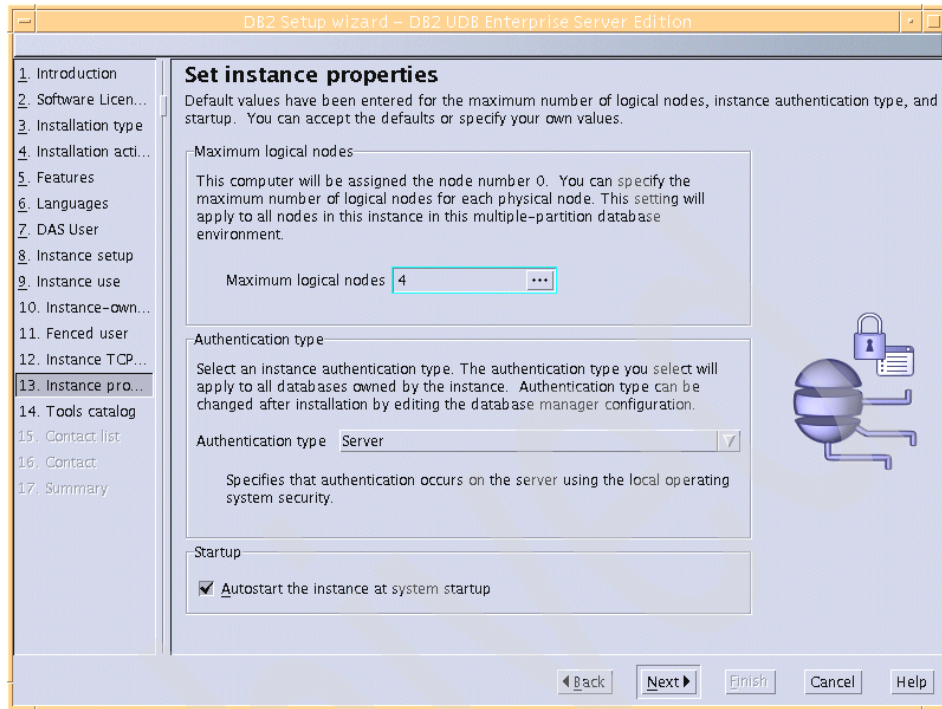


Figure 2-17 DB2 Setup number of database partitions

In our environment, we have 33 database partitions and the `/etc/services` file is configured as shown in Example 2-7:

Example 2-7 `/etc/services` based on case study

```
DB2_db2inst1      60000/tcp
DB2_db2inst1_END 60032/tcp
```

Important: The same port range must be reserved in the `/etc/services` file on each computer in your partitioned database system.

Edit `db2nodes.cfg`

The partition configuration file `db2nodes.cfg` is located in the `$INSTHOME/sqllib` directory and contains information that tells DB2 which database partition servers participate in an instance. There is a `db2nodes.cfg` file for each instance in a partitioned database environment.

The `db2nodes.cfg` file must contain one entry for each database partition server that will participate in the instance.

Because the home directory for the instance is on the shared DB2 home file system, there is only one db2nodes.cfg.

The db2nodes.cfg contains 4 columns as shown in a example in Table 2-6.

Table 2-6 db2nodes.cfg file

DB Partition Number	Hostname	Logical Port	Netname
0	host1	0	switch1
1	host1	1	switch1
2	host2	0	switch2
3	host2	1	switch2

Follows the description of each column:

► DB Partition Number

This is the number of the database partition and it must be sequential and unique. In this example, there are 4 database partitions.

► Hostname

You have to specify the hostname of the server which will contain the corresponded DB Partition Number. In this example, there are two servers, host1 and host2. The host1 contains the database partitions 0 and 1 and the host2 contains the database partitions 2 and 3.

► Logical Port

In this column you have to specify the logical port for the database partition on the each server. This value must be sequential and unique per server.

► Netname

This column is optional and you should use it if you are using switch for communication between servers.

In our environment, we have 33 database partitions in one machine then the db2nodes.cfg is configured as shown in Example 2-8.

Example 2-8 db2nodes.cfg based on case study

0	loire	0
1	loire	1
2	loire	2
3	loire	3
4	loire	4
5	loire	5

6	loire	6
7	loire	7
8	loire	8
9	loire	9
10	loire	10
11	loire	11
12	loire	12
13	loire	13
14	loire	14
15	loire	15
16	loire	16
17	loire	17
18	loire	18
19	loire	19
20	loire	20
21	loire	21
22	loire	22
23	loire	23
24	loire	24
25	loire	25
26	loire	26
27	loire	27
28	loire	28
29	loire	29
30	loire	30
31	loire	31
32	loire	32

Create .rhosts

The `.rhosts` file is created to enable the execution of remote commands.

You have to create this file manually in the DB2 instance home directory as the following command:

```
vi /db2home/db2inst1/.rhosts
```

The `.rhosts` file has the following format:

```
hostname instance_owner_user_name
```

For example, based on the Table 2-6 on page 108, the `.rhosts` file would be configured as follows:

```
host1 db2inst1
host2 db2inst1
```

In our environment, we have only one machine and the DB2 instance is db2inst1, then we have the following entry in `.rhosts` file:

Start the DB2 instance

To start the DB2 instance, log on as instance owner (db2inst1) and then execute the **db2start** command. The **db2start** is issued in parallel and each partition returns starting information as it starts, not specifically in order.

In Example 2-9 we show the **db2start** command in our environment.

Example 2-9 db2start command

```
{loire:db2inst1}/db2home/db2inst1 ->db2start
12-18-2002 15:49:21    1    0    SQL1063N    DB2START processing was successful.
12-18-2002 15:49:21    5    0    SQL1063N    DB2START processing was successful.
12-18-2002 15:49:21    6    0    SQL1063N    DB2START processing was successful.
12-18-2002 15:49:21   13    0    SQL1063N    DB2START processing was successful.
12-18-2002 15:49:21    2    0    SQL1063N    DB2START processing was successful.
12-18-2002 15:49:21   26    0    SQL1063N    DB2START processing was successful.
12-18-2002 15:49:21   23    0    SQL1063N    DB2START processing was successful.
12-18-2002 15:49:21    0    0    SQL1063N    DB2START processing was successful.
12-18-2002 15:49:21   12    0    SQL1063N    DB2START processing was successful.
12-18-2002 15:49:22   10    0    SQL1063N    DB2START processing was successful.
12-18-2002 15:49:22   29    0    SQL1063N    DB2START processing was successful.
12-18-2002 15:49:22   24    0    SQL1063N    DB2START processing was successful.
12-18-2002 15:49:22   31    0    SQL1063N    DB2START processing was successful.
12-18-2002 15:49:22   27    0    SQL1063N    DB2START processing was successful.
12-18-2002 15:49:22    4    0    SQL1063N    DB2START processing was successful.
12-18-2002 15:49:22   19    0    SQL1063N    DB2START processing was successful.
12-18-2002 15:49:22   16    0    SQL1063N    DB2START processing was successful.
12-18-2002 15:49:22   20    0    SQL1063N    DB2START processing was successful.
12-18-2002 15:49:22   32    0    SQL1063N    DB2START processing was successful.
12-18-2002 15:49:22   17    0    SQL1063N    DB2START processing was successful.
12-18-2002 15:49:22   25    0    SQL1063N    DB2START processing was successful.
12-18-2002 15:49:22   21    0    SQL1063N    DB2START processing was successful.
12-18-2002 15:49:22    7    0    SQL1063N    DB2START processing was successful.
12-18-2002 15:49:22   11    0    SQL1063N    DB2START processing was successful.
12-18-2002 15:49:22   30    0    SQL1063N    DB2START processing was successful.
12-18-2002 15:49:22    9    0    SQL1063N    DB2START processing was successful.
12-18-2002 15:49:22   28    0    SQL1063N    DB2START processing was successful.
12-18-2002 15:49:22   22    0    SQL1063N    DB2START processing was successful.
12-18-2002 15:49:22   14    0    SQL1063N    DB2START processing was successful.
12-18-2002 15:49:22    3    0    SQL1063N    DB2START processing was successful.
12-18-2002 15:49:22   15    0    SQL1063N    DB2START processing was successful.
12-18-2002 15:49:22   18    0    SQL1063N    DB2START processing was successful.
12-18-2002 15:49:23    8    0    SQL1063N    DB2START processing was successful.
SQL1063N    DB2START processing was successful.
```

LOAD and populate the data warehouse in parallel

With apologies to the data warehouse, datamart, and OLTP purists, many database installations today would have to be characterized as hybrids. Some installations are collapsing their datamarts back into a corporate data warehouse. Workloads are being consolidated onto large symmetric multiprocessors (SMP). Globalization, competition, and e-business factors may factor into the construction of your database. Many installations use service level agreements, which among other things, state when data must be available and what response time is acceptable.

This chapter considers two stages for populating the data warehouse:

- ▶ Initially, the data needs to be loaded into the database.
- ▶ Then the data content must be maintained to keep it current and relevant. New data must be entered either by full refresh or more incrementally. If you select the incremental approach, you must consider how to remove old and obsolete data from the database.

3.1 Initial mass load

So now you have this really large, really empty database, and you need to get you data into it. For our study, the initial population of the database was done using the DB2 LOAD utility. LOAD supports both bulk loading and incremental loading.

This section describes the following:

- ▶ The highlights of the DB2 UDB V8.1 enhancements to the LOAD utility
- ▶ The schematic of our database environment
- ▶ An example of a bulk table load
- ▶ Using LOAD with a multipartition database
- ▶ Overview the DB2 UDB V8.1 LOAD process model

3.1.1 Online LOAD

New functionality added to the LOAD utility simplifies loading data in both the single partition and multiple partition database environments. The impact that LOAD has on availability of data is greatly reduced. More activities can be done concurrently with less effort. The highlights of the DB2 UDB V8.1 LOAD improvements include:

- ▶ Locking now occurs at the table level.

A major drawback of the pre-Version 8.1 loader is that it acquired an exclusive table space lock even to load a single table. For table spaces containing multiple tables, it was bad enough that users could not access the table being loaded (incrementally, of course). Even worse, none of the tables in the table space could be read, much less written to while the loader was running. The entire table space was offline.

Often, this feature was circumvented by creating staging tables in a separate table space. Rows were loaded into the staging table. Later, the staged rows were moved to the target table using **INSERT INTOSELECT FROM**. Note that because **LOAD** locked the staging table space, only one table could be loaded at a time unless multiple staging table spaces were used.

The new V8.1 LOAD utility does not lock the table space. Users can continue to query, update, add, and delete rows in other tables even while LOAD is running. By default, users are denied access to the table being loaded. But, by including the new *ALLOW READ ACCESS* clause, users can query preexisting data in the table even while new data is being loaded.

For those of you who have them, you may not want to discard your staging tables yet. For while the new data is being loaded to a table, users are still unable to write to the table.

Because LOAD cannot proceed if any user has a write lock on the table, another new option *LOCK WITH FORCE* is available. This option forces applications to release locks on the table so that **LOAD** can continue.

- ▶ The LOAD utility directly supports multipartition databases.

The multipartition support, which prior to V8.1 was provided by the autoloader, **db2at1d**, and its configuration file has been embedded into the LOAD utility. This functionality is described more fully in 3.1.4, “Multipartition LOAD support” on page 119, and in 3.1.5, “Partitioning LOAD subagents in DB2 UDB V8.1” on page 126.

Note: **db2at1d** scripts and syntax are still supported for compatibility with previous versions.

- ▶ Generated columns are populated during load.

Prior to DB2 UDB V8.1, if a table contained generated columns (other than identity columns), the table would be placed in check pending state while the load completed. To actually generate the column values, and to take the table out of check pending state, the Example 3-1 command must be run.

Example 3-1 SET INTEGRITY prior to DB2 UDB V8.1

```
SET INTEGRITY FOR your.tablename IMMEDIATE CHECKED FORCE GENERATED
```

Now with Version 8.1 the column values are generated during LOAD; the table is not placed in check pending state (at least not because of generated columns), and **SET INTEGRITY** is not required. Your table is available sooner and with less effort.

- ▶ The DB2 Control Center now includes a LOAD wizard to assist the set up of a LOAD command.

3.1.2 Loading the case study data

To explore the new functionality of the LOAD utility, we used the database schema and the data generation tool from the TPC-H as mentioned in 1.4.1, “The case study based on TPC-H” on page 31.

The TPC-H data generation tool, **dbgen**, was used to create the data to be loaded into the tables. The output of **dbgen** is a delimited ASCII source file; the vertical bar (|) delimits the data columns.

Eight tables must be loaded. The six large multipartition tables (customer, part, supplier, partsupp, orders, and lineitem) will be loaded using **DB2 LOAD**. The two

small tables (nation and region) will be stored in a single database partition table space, and loaded using **DB2 INSERT**. The loading sequence will be as follows:

- ▶ The two small tables will be created from input files, which will be generated by dbgen and stored on disk. Those input files will in turn be used by DB2 **IMPORT** to populate the tables.
- ▶ Because there was insufficient disk space to house the generated input files, which would normally be used to load the six large tables, we decided to create named pipes to which dbgen would write the generated load data, and from which **DB2 LOAD** would populate the tables. To facilitate this activity, script `gen_and_load_one.ksh` was created.
- ▶ Those tables on which foreign key and primary key constraints were created are in *CHECK PENDING* state after DB2 LOAD. So, we will need to use DB2 **SET INTEGRITY** to get the tables into normal state.

Scripts and the step by step procedures are provided in Appendix A, “Using DB2 LOAD for the case study” on page 299.

The same procedure was used to load each of the six multipartition tables: customer, part, supplier, partsupp, orders, and lineitem. The input was generated by dbgen, using a scaling factor of 60, and using four input processes for all tables except order and lineitem, which used 12. The procedure shown here documents the load of the lineitem table, the largest table of the set.

For the lineitem table load, twelve dbgen processes create input files. While this load was running, the 32-processor p690 machine was running at 79% utilization. All multipartition options were allowed to default. That is, no *PARTITIONED DB CONFIG* list was specified. Consequently, LOAD was allowed to select the partitioning partitions (it picked partitions 1 through 9), the number of output partitions (all partitions where the lineitem table is defined), and so forth.

An extract of the DB2 load command on the line item table and its results is provided in Example 3-2.

Example 3-2 Output from gen_and_load_one.ksh on lineitem table

```
{loire:db2inst1}/db2home/db2inst1/scripts -> gen_and_load_one.ksh L 60 12
```

TPC-H (c) dbgen command is='dbgen -s60 -C12 -TL '.

```
DB2 LOAD command is 'db2 load from /db2work/dbgen_data/lineitem.tbl.1,
/db2work/dbgen_data/lineitem.tbl.2, /db2work/dbgen_data/lineitem.tbl.3,
/db2work/dbgen_data/lineitem.tbl.4, /db2work/dbgen_data/lineitem.tbl.5,
/db2work/dbgen_data/lineitem.tbl.6, /db2work/dbgen_data/lineitem.tbl.7,
/db2work/dbgen_data/lineitem.tbl.8, /db2work/dbgen_data/lineitem.tbl.9,
/db2work/dbgen_data/lineitem.tbl.10, /db2work/dbgen_data/lineitem.tbl.11,
```

```
/db2work/dbgen_data/lineitem.tbl.12 of del modified by coldel| fastparse  
anyorder messages /db2home/db2inst1/messages/lineitem replace into  
tpcd.lineitem nonrecoverable'.
```

Waiting for all processes to complete....

DBGEN output follows:
TPC-H Population Generator (Version 1.3.0)
Copyright Transaction Processing Performance Council 1994 - 2000
DBGEN_RC=0

DB2LOAD output follows:

Agent Type	Node	SQL Code	Result
LOAD	001	+00000000	Success.
LOAD	002	+00000000	Success.
LOAD	003	+00000000	Success.
LOAD	004	+00000000	Success.
LOAD	005	+00000000	Success.
LOAD	006	+00000000	Success.
LOAD	007	+00000000	Success.
LOAD	008	+00000000	Success.
LOAD	009	+00000000	Success.
LOAD	010	+00000000	Success.
LOAD	011	+00000000	Success.
LOAD	012	+00000000	Success.
LOAD	013	+00000000	Success.
LOAD	014	+00000000	Success.
LOAD	015	+00000000	Success.
LOAD	016	+00000000	Success.

LOAD	017	+00000000	Success.
LOAD	018	+00000000	Success.
LOAD	019	+00000000	Success.
LOAD	020	+00000000	Success.
LOAD	021	+00000000	Success.
LOAD	022	+00000000	Success.
LOAD	023	+00000000	Success.
LOAD	024	+00000000	Success.
LOAD	025	+00000000	Success.
LOAD	026	+00000000	Success.
LOAD	027	+00000000	Success.
LOAD	028	+00000000	Success.
LOAD	029	+00000000	Success.
LOAD	030	+00000000	Success.
LOAD	031	+00000000	Success.
LOAD	032	+00000000	Success.
PARTITION	001	+00000000	Success.
PARTITION	002	+00000000	Success.
PARTITION	003	+00000000	Success.
PARTITION	004	+00000000	Success.
PARTITION	005	+00000000	Success.
PARTITION	006	+00000000	Success.
PARTITION	007	+00000000	Success.
PARTITION	008	+00000000	Success.
PARTITION	009	+00000000	Success.

```
PRE_PARTITION 000      +00000000    Success.

RESULTS:      32 of 32 LOADs completed successfully.

Summary of Partitioning Agents:
Rows Read      = 360011594
Rows Rejected   = 0
Rows Partitioned = 360011594

Summary of LOAD Agents:
Number of rows read      = 360011594
Number of rows skipped   = 0
Number of rows loaded    = 360011594
Number of rows rejected  = 0
Number of rows deleted   = 0
Number of rows committed = 360011594

DB2LOAD_RC=0
DB20000I The TERMINATE command completed successfully.
```

During the load of line item, the **DB2 LOAD QUERY** was run several times to monitor the load progress on each partition:

```
db2_a11 "\"|| echo DB2PART=##; db2 load query table tpcd.lineitem
```

Please look at the results in Appendix A, “Using DB2 LOAD for the case study” on page 299.

3.1.3 Some considerations when using LOAD

The LOAD statement in Example 3-2 on page 114 omits a couple of options which you might wish to use:

- ▶ *DUMPFIL*E modifier: The dumpfile contains rows that cannot be loaded, because they are invalid or have syntax errors.
- ▶ *FOR EXCEPTION* *schema.table*: This clause identifies an exception table into which rows are copied, which violate a unique or primary key constraint. In other words, if a table column is supposed to be unique, and a duplicate value is encountered during LOAD, one of those rows (which one cannot be predicted) will be deleted. This action occurs during the DELETE phase of LOAD. The deleted row will be stored in an exception table, if one exists. The exception table definition looks just like the definition for the table being loaded with two additional columns. One method for maintaining exception

tables is to create a separate schema, say one named EXCP, in which the exception tables are defined as in Example 3-3.

Example 3-3 Create exception schema and table

```
-- Create a schema 'excp' for exception tables
CREATE SCHEMA excp ;

-- Define an exception table 'excp.table1' for 'table1'
CREATE TABLE excp.table1 AS
    (SELECT table1.*,
        CURRENT_TIMESTAMP AS TIMESTAMP,
        CAST ('' AS CLOB(32K)) AS MSG
    FROM table1
    ) DEFINITION ONLY ;
```

The LOAD statement in Example 3-2 on page 114 includes several options that you may or may not wish to use:

- ▶ **ANYORDER** modifier: This modifier works in conjunction with the **CPU_PARALLELISM** parameter for intra-partition loading. And in a multipartition environment, **ANYORDER** must be specified to use multiple **SPLIT_NODES**. Do not use **ANYORDER** if the sorted order of the input file must be maintained.
- ▶ **FASTPARSE** modifier: This option can enhance performance by reducing the syntax checking done by LOAD. Use this option when you are certain that your input data is valid.

The final LOAD parameters to consider are **COPY=NO** and **NONRECOVERABLE** for databases with retention logging activated.

- ▶ **COPY=NO**: Understand the implication of coding **COPY=NO** on the LOAD statement. When retention logging is turned on for a database, if you specify **COPY=NO** on the LOAD statement, the table space containing the table being loaded will be placed in backup pending state. Deletes and/or updates cannot be done until either a table space backup, or a full database backup is done. **COPY=YES** only has a small (<10%) overhead.
- ▶ **NONRECOVERABLE**: If this option is coded, table spaces are not put into backup pending state when the LOAD operation completes. A copy of the loaded data does not have to be made. By specifying that the load transaction is to be marked as non-recoverable, you will not be able to recover it by a later roll forward action. The **ROLLFORWARD** utility will skip the LOAD transaction and mark the table as invalid. The **ROLLFORWARD** utility will also ignore subsequent logged transactions to that table. After **ROLLFORWARD** completes, the invalid table can only be dropped or restored from a backup (full or table

space) taken after a commit point following the completion of the non-recoverable load operation.

For details on the **LOAD** and **ROLLFORWARD** utilities, see manuals:

- ▶ *DB2 Command Reference, SC09-4828*
- ▶ *DB2 Data Movement Utilities Guide and Reference, SC09-4830*

3.1.4 Multipartition **LOAD** support

The function previously available only with the autoloader has been built into the DB2 UDB ESE V8.1 **LOAD** utility. The autoloader, **db2at1d**, and the autoloader control file are no longer needed. The **db2at1d** system command is still supported, but it has been changed to a single transaction model.

Version 8 **LOAD** allows *CLIENT* option and input type of *CURSOR*.

Partitioning **LOAD** supports the following data file formats:

- ▶ Delimited ASCII (DEL)
- ▶ Non-delimited ASCII (ASC)
 - Fixed length ASCII
 - Variable length ASCII
- ▶ *CURSOR*, a new data input type offered in V8.1, loads the result set of an SQL **SELECT** statement.

The partitioning **LOAD** does not support the PC/IXF data format file, because it cannot be partitioned. But a work around exists using *LOAD_ONLY_VERIFY_PART* (see the Notebox in section “MODE modename” on page 122).

DB2_PARTITIONEDLOAD_DEFAULT registry variable

The default behavior of the new DB2 UDB V8.1 load utility is that the load operation will occur on all partitions on which the table is defined. This happens even without coding the *PARTITIONED DB CONFIG* clause of **LOAD**. An installation can choose to maintain the pre-Version 8.1 behavior of **LOAD** in a partitioned environment. Perhaps they wish to avoid modifying existing scripts that **LOAD** to single database partitions. But for whatever reason, the pre-Version 8.1 load operation can be obtained by setting the *DB2_PARTITIONEDLOAD_DEFAULT* registry variable as in Example 3-4.

Example 3-4 Setting DB2_PARTITIONEDLOAD_DEFAULT

```
db2set DB2_PARTITIONEDLOAD_DEFAULT=NO
```

PARTITIONED DB CONFIG clause of LOAD

The information previously provided by the autoloader control file is passed to LOAD via the PARTITIONED DB CONFIG clause. An abbreviated LOAD syntax that shows support for partitioned environments appears in Example 3-5.

Example 3-5 PARTITIONED DB CONFIG clause of LOAD utility

```
LOAD FROM .... PARTITIONED DB CONFIG partitioned_db_option1
              partitioned_db_option2 ....
```

Each of the options for the partitioned database environment will now be described. The options will look very familiar to anyone who has used the autoloader.

HOSTNAME hostname

This option is used in conjunction with the *FILE_TRANSFER_CMD* option. If the *FILE_TRANSFER_CMD* option is not coded, the *HOSTNAME* option is ignored even if it is coded. Replace *hostname* with the name of the remote machine where the data file resides. This may be a z/OS host or another workstation. If this option is not specified, and the *FILE_TRANSFER_CMD* option is specified, the hostname *nohost* will be used.

FILE_TRANSFER_CMD command

Specify command to identify a program, a batch file, or a script that will be called before data is loaded onto any partitions. The value specified must be a fully qualified path. The full path, including the execution file name, must not exceed 254 characters. The command will be invoked with the syntax as in Example 3-6.

Example 3-6 Syntax for LOAD invocation of FILE_TRANSFER_CMD

```
<command> <logpath> <hostname> <basepipename> <nummedia> <media_list>
```

where:

<command> is the value identified by the *FILE_TRANSFER_CMD* parameter
<logpath> the directory name for the file from which the data is being loaded. Diagnostic or temporary data may be written to this path
<hostname> is the value identified by the *HOSTNAME* option
<basepipename> is the stem used for named pipes that the load operation will create and from which expect to receive data. A pipe is created for each input source identified to LOAD FROM. The pipe name will be <basepipename> followed by a period and then an index number that corresponds to each input file to LOAD. As an example, if two files were to be loaded, two named pipes would be created. Assuming that the <basepipename> is 'pipeabc' the names of the two pipes would be 'pipeabc.000' and 'pipeabc.001'.
<nummedia> specifies the number of media arguments that follow. It will be the count of the files to be loaded.

<media_list> is the list of filenames to be loaded.

***PART_FILE_LOCATION* pathname**

In *PARTITION_ONLY* mode, *pathname* is the fully qualified location to which the partitioned files will be written. In *LOAD_ONLY* mode, *pathname* is the fully qualified location of the partitioned files to be loaded. This location must exist on each partition specified by the *OUTPUT_DBPARTNUMS* option. If the load mode is *PARTITION_ONLY*, this option must be coded, or an error will be returned. In *LOAD_ONLY* and *LOAD_ONLY_VERIFY_PART* modes, this parameter must be specified whenever the data file name specified in the **LOAD** command is not fully qualified.

This parameter is not valid when the *CLIENT* parameter is specified and the load operation is taking place in the *LOAD_ONLY* or the *LOAD_ONLY_VERIFY_PART* modes.

For all file types other than *CURSOR*, the location refers to the path where the partitioned files exist or are to be created. For the *CURSOR* file type, the location does not refer to a path, but to a fully qualified file name. This will be the fully qualified base file name of the partitioned files that are created on each output partition. Multiple files may be created with the specified base name if there are LOB columns in the target table.

***OUTPUT_DBPARTNUMS* partition_list**

The partition numbers listed in *partition_list* identify the database partitions on which the load operation is to be performed. The partition numbers must be a subset of the database partitions on which the table is defined. The default is a list of all database partitions on which the table is defined. The list must be enclosed in parentheses, and the items in the list must be separated by commas. Ranges are permitted. For example, if (0, 2 to 10, 15) was specified as the *partition_list* the table would be loaded on partition 0, on partitions 2 through 10, and on partition 15.

***PARTITIONING_DBPARTNUMS* partition_list**

partition_list is the set of partition numbers that will be used in the partitioning process. The list must be enclosed in parentheses, and the items in the list must be separated by commas. Ranges are permitted as in (0, 2 to 10, 15). The database partitions specified as partitioning (these were formerly called “*splitters*”) can be different from the database partitions being loaded. If not specified, the **LOAD** command will determine how many database partitions are needed, and which database partitions to use in order to achieve optimal performance.

Note: For performance reasons, try to separate the partitioning database partitions from the database partitions being loaded if that is possible.

If the *ANYORDER* modifier is not specified in the **LOAD** command, only one partition agent will be used in the load session. Furthermore, if there is only one partition specified for the *OUTPUT_DBPARTNUMS* option, or the coordinator partition of the load operation is not an element of *OUTPUT_DBPARTNUMS*, the coordinator partition of the load operation is used as the partitioning partition. Otherwise, the first partition (not the coordinator partition) in *OUTPUT_DBPARTNUMS* is used as the partitioning partition.

If the *ANYORDER* modifier is specified, the number of partitioning partitions is determined as follows:

number of partitions in *OUTPUT_DBPARTNUMS*)/4 + 1.

Then, the number of partitioning partitions is chosen from the *OUTPUT_DBPARTNUMS*, excluding the partition being used to load the data.

MODE modename

MODE specifies the mode in which the load operation will take place when loading a partitioned database. *PARTITION_AND_LOAD* is the default. Valid values are:

- ▶ *PARTITION_AND_LOAD*: Data is partitioned (perhaps in parallel) and loaded simultaneously on the corresponding database partitions.
- ▶ *PARTITION_ONLY*: Data is partitioned (perhaps in parallel) and the output is written to files in a specified location on each loading partition.
- ▶ *LOAD_ONLY*: Data is assumed to be already partitioned. The partitioning process is skipped, and the data is loaded simultaneously on the corresponding database partitions.
- ▶ *LOAD_ONLY_VERIFY_PART*: Data is assumed to be already partitioned, but the data file does not contain a partition header. The partitioning process is skipped, and the data is loaded simultaneously on the corresponding database partitions. During the load operation, each row is checked to verify that it is on the correct partition. Rows containing partition violations are placed in a dumpfile if the dumpfile file type modifier is specified. Otherwise, the rows are discarded. If partition violations exist on a particular loading partition, a single warning will be written to the load message file for that partition.

Note: This new mode, *LOAD_ONLY_VERIFY_PART*, can be used as a work around to load IXF data in the partitioned database environment. The IXF file to be loaded must be available to each partition, either by NFS mounting, or by physically copying the file to every partition to be loaded. By specifying the loading mode *LOAD_ONLY_VERIFY_PART*, each partition will pick out only those records that belong to that partition.

Ignore warnings about partitioning errors. Use caution when allocating a dump file. If specified, each row rejected for partitioning error, will be written there.

ANALYZE: This mode generates an optimal partitioning map with even distribution across all database partitions. If *ANALYZE* mode is specified, the *MAP_FILE_OUTPUT* parameter is required. See the *MAP_FILE_OUTPUT* filename below.

MAX_NUM_PART_AGENTS count

count specifies the maximum numbers of partitioning agents to be used in a load session. The default is 25.

ISOLATE_PART_ERRS action

Indicates how the load operation will react to errors that occur on individual partitions. The default is *LOAD_ERRS_ONLY*.

Attention: The default action *LOAD_ERRS_ONLY* is incompatible with the *ALLOW READ ACCESS* and/or *COPY YES* clauses of **LOAD**. Consider using *NO_ISOLATION* instead.

Valid action values are:

► ***SETUP_ERRS_ONLY***

Errors that occur on a partition during setup, such as problems accessing a partition, or problems accessing a table space or table on a partition, will cause the load operation to stop on the failing partitions, but to continue on the remaining partitions. Errors that occur on a partition while data is being loaded will cause the entire load operation to fail and roll back to the last point of consistency on each partition.

► ***LOAD_ERRS_ONLY***

Errors that occur on a partition during setup will cause the entire load operation to fail. When an error occurs while data is being loaded, the partitions with errors will be rolled back to their last point of consistency. The load operation will continue on the remaining partitions until a failure occurs, or until all the data is loaded. On the partitions where all of the data was

loaded, the data will not be visible following the load operation. Because of the errors in the other partitions, the transaction will be aborted. Data on all of the partitions will remain invisible until a load restart operation is performed. This will make the newly loaded data visible on the partitions where the load operation is completed, and resume the load operation on partitions that experienced an error.

Data is only committed when all partitions are successfully loaded.

Attention: The default action for multipartition LOAD is *LOAD_ERRS_ONLY*. *LOAD_ERRS_ONLY* cannot be used if the *ALLOW READ ACCESS* or *COPY YES* option of the **LOAD** command is used.

SETUP_AND_LOAD_ERRS: Partition-level errors, which occur during setup or loading data cause processing to stop only on the affected partitions. As with the *LOAD_ERRS_ONLY* mode, when partition errors do occur while data is being loaded, the data on all partitions will remain invisible until a load restart operation is performed. For those familiar with autoloader, *SETUP_AND_LOAD_ERRS* action is equivalent to the **db2atld** *FORCE YES* configuration file option.

Attention: *SETUP_AND_LOAD_ERRS* cannot be used when *ALLOW READ ACCESS* or *COPY YES* option of **LOAD** is used.

NO_ISOLATION: Any error during the load operation causes the transaction to fail.

STATUS_INTERVAL number

number represents how often you will be notified of the volume of data that has been read. The unit of measurement is megabytes (MB). The default is 100. Valid values are whole numbers from 1 to 4000.

PORT_RANGE (lower_port, higher_port)

Specifies the range of TCP ports used to create sockets for internal communications. The default range is from 6000 to 6063. If defined at the time of invocation, the value of the *DB2ATLD_PORTS* DB2 registry variable will override the value of the *PORT_RANGE* load configuration option.

To set the *DB2ATLD_PORTS* registry variable, the range should be provided in the following format specified in Example 3-7.

Example 3-7 DB2ATLD_PORTS registry variable

```
db2set DB2ATLD_PORTS=51000:51300
```

Attention: Make sure that your *PORT_RANGE* (or *DB2ATLD_PORTS* registry variable) is sufficient to support your LOAD requirements. DB2 UDB V8.1 removal of the table space lock makes it more likely that multiple LOADs can run in parallel. Couple that with support for partitioning, and it is quite easy to use many AIX ports for sockets.

CHECK_TRUNCATION

There is no parameter for this option. Coding *CHECK_TRUNCATION* directs the load utility to check for truncation of data records at input/output. The default behavior is that data will not be checked for truncation.

MAP_FILE_INPUT filename

The *filename* parameter specifies the input file name for the partitioning map. This parameter must be specified if the partitioning map is customized, as it points to the file containing the customized partitioning map. A customized partitioning map can be created by using the *db2gpmmap* program to extract the map from the database system catalog table, or by using the *ANALYZE* mode of the MPP Loader to generate an optimal map. The map generated by using the *ANALYZE* mode must be moved to each database partition in your database before the load operation can proceed.

MAP_FILE_OUTPUT filename

The *filename* parameter represents the output filename for the partitioning map. This parameter is required when the *ANALYZE* mode is specified. An optimal partitioning map with even distribution across all database partitions is generated. If this modifier is not specified, and the *ANALYZE* mode is specified, the program exits with an error.

TRACE count

Specifies the number of records to trace when you require a review of a dump of the data conversion process and the output of the hashing values. The default is 0.

NEWLINE

Code this option when the input data file is an ASCII file with each record delimited by a new line character and the *RecLen* parameter of the LOAD command is specified. When this option is specified, each record will be checked for a new line character. The record length, as specified in the *RecLen* parameter, will also be verified.

DISTFILE filename

Specifies the name of the partition distribution file. The default is *DISTFILE*.

OMIT_HEADER

Code this option to direct that a partitioning map header should not be included in the partition file. If not specified, a header will be generated.

RUN_STAT_DBPARTNUM database_partition_number

If the *STATISTICS YES* parameter has been specified in the **LOAD** command, statistics will be collected only on one database partition. This parameter specifies on which database partition to collect statistics. If the value is -1 or not specified at all, statistics will be collected on the first database partition in the output partition list *OUTPUT_DBPARTNUMS*. If *OUTPUT_DBPARTNUMS* is not specified, statistics will be collected from the first partition on which the table's containing table space is defined.

3.1.5 Partitioning LOAD subagents in DB2 UDB V8.1

This section might classify as more than you ever wanted to know about the partitioned LOAD utility. For in it, we list the different categories of partitions with reference to LOAD, and identify the subagents related to partitioning and loading. Lastly, we provide a formula for determining how many ports are required for various types of loading. This is useful for coding the *PORT_RANGE* option of *PARTITIONED DB CONFIG* parameter of individual LOADs. It may also benefit those who wish to figure out how to set the global *DB2ATLD_PORTS* registry variable.

Partition category

To better understand the individual subagents involved with various **LOAD** utility operations, we begin by describing how partitions are used by **LOAD** to coordinate, to partition, and to load.

Coordinating partition

This is the partition to which the user connects to perform the load operation. There will always be a coordinating partition.

Partitioning partitions

These are the partitions where the partitioning agents run. There will be one or more for load modes *PARTITION_AND_LOAD*, *PARTITION_ONLY*, and *ANALYZE*. If load is not *MODIFIED BY ANYORDER* only a single partitioning partition will be used.

Loading partitions

Loading partitions are those where the data is actually loaded. They derive from load utility modes of *PARTITION_AND_LOAD*, *LOAD_ONLY*, or *LOAD_ONLY_VERIFY_PART*.

LOAD subagents

The individual subagents involved with various **LOAD** utility operations are listed below.

Load coordinator, db2agent

Coordinates the entire load job and requests all additional subagents depending upon mode of **LOAD**.

Catalog subagent, db2lcata

This subagent executes only on the catalog partition, and performs various setup functions. There is one catalog subagent per load job.

Initialize subagent, db2linit

This subagent is responsible for initializing during the setup phase.

Userexit subagent, db2lurex

This subagent is responsible for running the program or script specified by the *FILE_TRANSFER_CMD* option of the **LOAD** command. There is no more than one userexit subagent per load job.

Prepartitioning subagent, db2lpprt

This subagent pre-partitions user data from one input stream into multiple output streams. Basically, it hands off rows in round-robin fashion to the partitioning subagents.

Partitioning subagent, db2lpart

This subagent applies a hashing algorithm to a row received from the partitioning subagent, and directs the row to the proper output partition. There is one partitioning subagent for each partitioning partition.

Load subagent, db2lload

This subagent performs the partition load. There is one subagent per loading (or output) partition.

db2lmibm

This subagent generates the partitioned data file(s).

The architecture overview of *PARTITION_AND_LOAD* mode is provided in Figure 3-1.

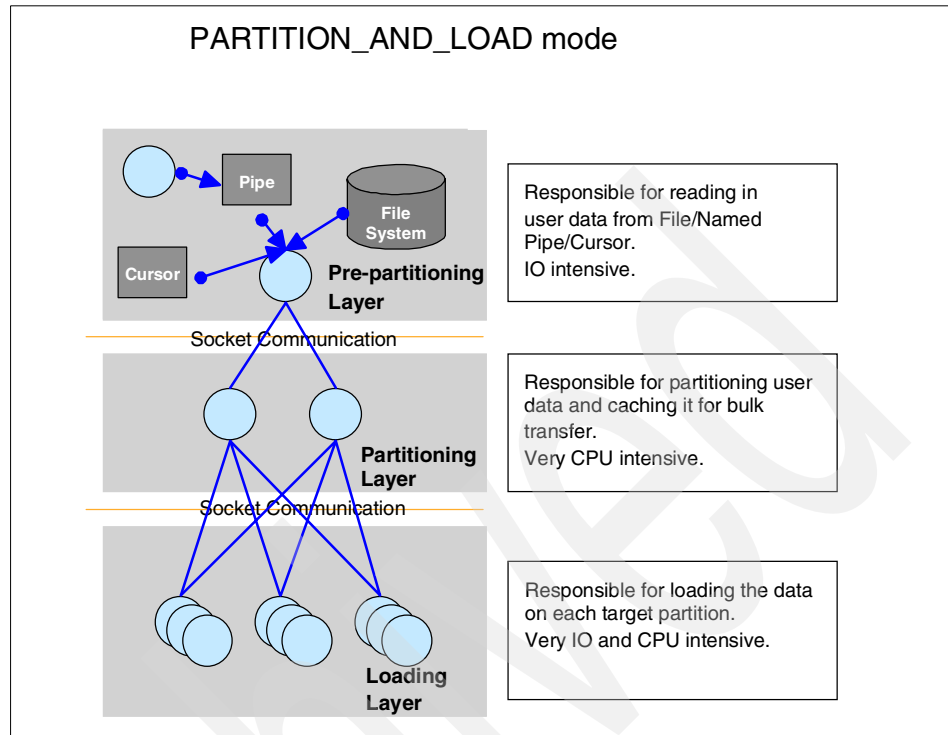


Figure 3-1 PARTITION_AND_LOAD mode architecture overview

Ports and sockets used by multipartition loading are in discussed in Figure 3-2.

Ports and Sockets Used by Partitioning Load

- By default, Partitioning Load will use ports in the range 6000-6063.
- The Partitioning LOAD's port range can be redefined through the *PORT_RANGE* option of *PARTITIONED DB CONFIG* list.
`LOAD FROM ... PARTITIONED DB CONFIG PORT_RANGE (51200,51270)`
- Total ports used per load job out of the port range =
(# partitioning sub-agents + # loading sub-agents)
- If ANYORDER AND # input sources > 1
Total sockets used = (# input sources * # partitioning sub-agents) +
(# partitioning sub-agents * # loading sub-agents)
- else
Total sockets used = (# partitioning sub-agents) +
(# partitioning sub-agents * # loading sub-agents)

Figure 3-2 Ports and sockets used by multipartition LOAD

3.1.6 LOAD QUERY command

The **LOAD QUERY** already discussed in 3.1.2, “Loading the case study data” on page 113 is done by partition. DB2 Control Center can be used to get a consolidated view of all partitions.

3.2 Incremental updates and LOADs

Unless you plan to use periodic full refresh to keep your database up to date, you must plan how to keep the data relevant. New data must be loaded; obsolete rows must be deleted or updated. Many techniques exist to get new data into a database. Among them:

- ▶ **LOAD INSERT:** This method uses the *INSERT* mode of the **LOAD** utility. (There are four modes in which **LOAD** can be run: *REPLACE*, *INSERT*, *RESTART*, or *TERMINATE*). By default, the table which is being incrementally loaded will be locked. Query, update, and delete are prevented. By coding the *ALLOW READ ACCESS* on the **LOAD** statement, queries can run against pre-existing data in the table being loaded. If full access to the table being loaded is required, consider using a staging table.
- ▶ **IMPORT:** The import utility inserts data from an input file into a table or updateable view. If the table or view receiving the imported data already contains data, you can either replace or append to the existing data.

Note: In general, **IMPORT** should be used for less than 5,000 rows and consider binding it with the buffered insert option.

On large tables, **IMPORT** is significantly slower than **LOAD**. For frequent mass inserts, the performance of **IMPORT** might be improved by rebinding **IMPORT** to allow *buffered insert*. Also, the *COMMITCOUNT* parameter can be coded on the import statement to cause periodic commits. This helps with logging and with concurrency.

By default, the import utility is bound to the database with isolation level RR (repeatable read). If a large number of rows is being imported into a table, the existing lock may escalate to an exclusive lock. If another application working on the same table is holding some row locks, a deadlock will occur if the lock escalates to an exclusive lock. To avoid this, the import utility requests an exclusive lock on the table at the beginning of its operation.

Holding a lock on the table has two implications. First, if there are other applications holding a table lock, or row locks on the import target table, the import utility will wait until all of those applications commit or roll back their changes. Second, while import is running, any other application requesting locks will wait until the import operation has completed.

- ▶ Extract-Transform-Load (ETL) application: An ETL application as DB2 Warehouse Manager integrated in DB2 or Ascential Data Stage can extract, transform, and load new data into your data warehouse.
- ▶ Application program: You might wish to develop, either in-house or under contract, applications to tailor how new data is put into your database.
- ▶ IBM Data Replication products can be used to replicate updates and refresh the data warehouse.
- ▶ Stored procedures, scripts (KornShell, Perl, and so on) which contain SQL statements, triggers, and a myriad of other ways can be used to update your database.

3.2.1 Deleting data

Another database maintenance task is to delete obsolete rows. Just as for load, there are many ways to delete. This section presents a couple of methods for your evaluation. Many shops have a requirement to maintain some number of months of data. When a new month is added, the oldest month is deleted.

- ▶ Use a multi-dimensionally clustering (MDC) table with the month as one of the dimensions (please refer to Chapter 4, “Speed up performance with MultiDimensional Clustering” on page 135). Because MDC tables use block

indexes, deletes can be faster than non MDC tables, because of reduced logging and index maintenance.

- ▶ Structure the data in multiple tables, with one table for each month. For queries, create a view which is the union of the rows in all the tables. When a new month is to be added, create a new table, drop the view, recreate the view with the new table and without the oldest table, and finally, drop the old table. (A **drop table** is really fast!) One of the drawbacks is that you will need to re-grant any permissions that this view requires.

3.3 LOAD and real-time notion

Another challenge that enterprises may face today in regards to competition and improving business performances, is to be able to make business decisions as soon as possible, and for business analysts to be alerted to any new event that might change the business state as it shows up in the operational system. The *real-time notion* is a way to compress time in business decisions between the cause and the effect, but business cycles require to define how much or little latency is acceptable. Different industries have different requirements. So, are different the business requirements too, within a single organization as you go from operational to strategic viewpoints.

Lately, there has been much talk about real-time data, especially in the Business Intelligence arena. Certainly business units would say they favor real-time data. But, just as likely they want really inexpensive data processing. Vendors of course, have been quick to promote the former while not so much the latter. Contrary to claims, providing near real-time access to data is a complex and time consuming undertaking. Organizations which wish to tackle that challenge today will find that a number of software products must be cobbled together. Probably even some specialized programs will need to be developed in-house or under contract.

Each organization has to answer the questions about:

How much latency can I accept for my data in regard to my business processes and my business cycle? When do I need instantaneous information?

From a data point-of-view, a real-time or near real-time solution usually requires the following capabilities to refresh the data:

- ▶ Continuous parallel load
- ▶ Message queueing for continuous feed from legacy and other operational systems
- ▶ Concurrent load with read/write queries and other operations activities

For example Web personalization may require interactive communication and instantaneous refreshing of data for a customized and appropriate answer to the customer. On the other hand, a real-time supply chain management will provide integrated, up-to-the-second views of supply chain profitability forecasting. Fraud detection and customer relationship management may require real-time scoring to score customers, and send alerts in the campaign management system. Some examples of real-time scoring are provided in the redbook *Enhance Your Business Applications: Simple integration of Advanced Data Mining Functions*, SG24-6879.

Beyond the scalability, performances, availability, flexibility required for the RDBMS, real time is a whole solution to design whose architecture involves many components, as shown in Figure 3-3.

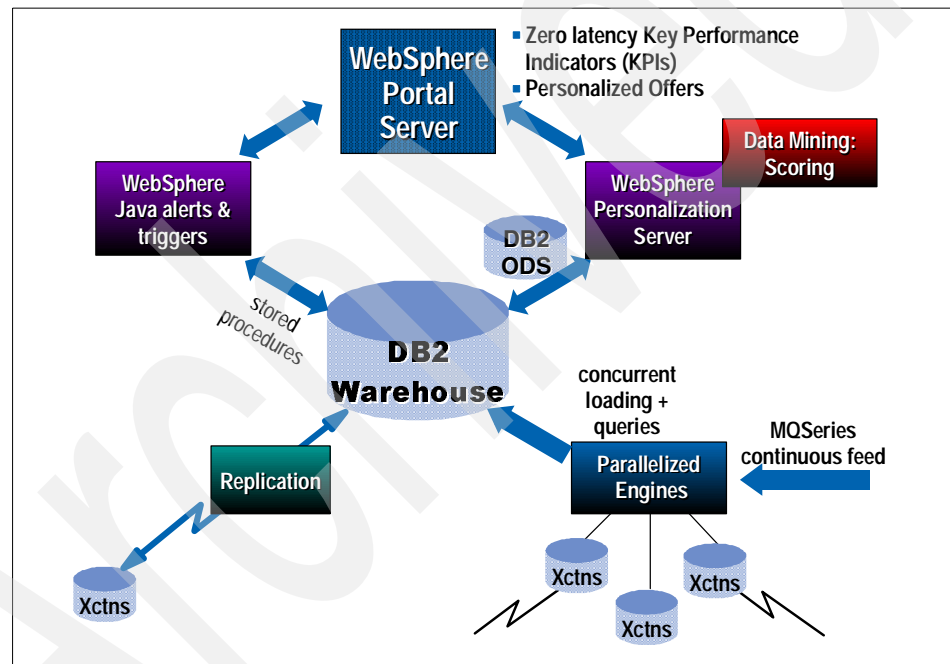


Figure 3-3 Real time components

Here is an example on designing and using DB2 UDB capabilities to better answer your real time requirements.

OLTP on one database partition group, with MQTs based on the OLTP tables, are maintained in a separate database partition group. This will allow separating the decision support and adhoc query environment from the online processing environment. In fact, they can be on different LPAR or logical partitioning on the

same p690, and even in a cluster database environment; they could even be on different hardware boxes.

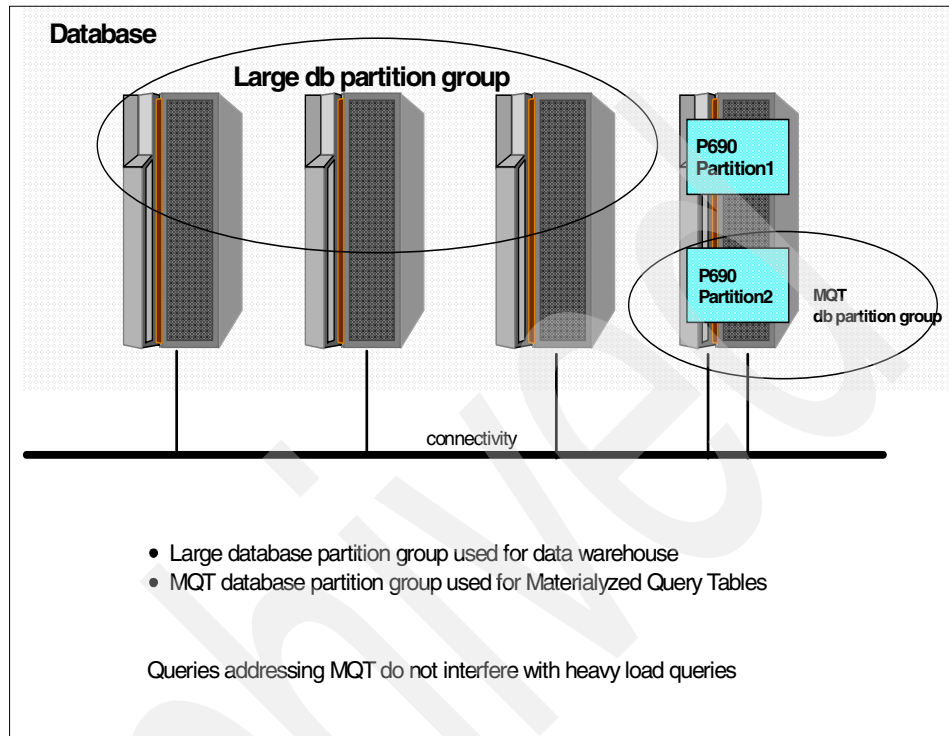


Figure 3-4 Separating load on a large cluster of p690

Speed up performance with MultiDimensional Clustering

DB2 UDB V8.1 introduces a new patent pending data storage technology called *MultiDimensional Clustering* (MDC). MultiDimensional Clustering provides a method for guaranteed, automatic, and continuous clustering of data along multiple dimensions. The main objectives of this chapter follow:

- ▶ To introduce the concept of MDC
- ▶ To show how to create a MDC table
- ▶ To cite the benefits of MDC
- ▶ To offer guidelines for using MDC

4.1 What is MultiDimensional Clustering?

This section introduces the terminology of MDC.

4.1.1 Overview of the clustering index

Prior to V8.1, clustering was implemented via the clustering index. Clustering indexes, like all indexes, were record-based. Clustering was supported in only a single dimension, perhaps by store, perhaps by period, but not by both store and period. To establish clustering required either sorting prior to loading, or loading, and then reorganizing with the clustering index. Unless the database was read-only, the clustering could degrade over time. Updates which lengthened variable length columns could cause relocation of the record. Inserts to pages where free space was insufficient would have to be moved to another page.

The clustering index degrades over time. The region index is initially clustered, but the year index cannot be clustered. After some time, row inserts and updates cause the rows to become unclustered by region. As rows become scattered within the table, a query such as:

```
SELECT * FROM TABLE1 WHERE REGION=5
```

might generate a significant amount of random and sequential I/O, and would likely return pages that contain a mix of qualifying and nonqualifying rows.

Once the clustering factor drops below a certain threshold, the optimizer no longer considers the table to be clustered.

4.1.2 Introducing MultiDimensional Clustering (MDC)

Figure 4-1 presents a conceptual diagram of MultiDimensional Clustering in three dimensional region, year, and color.

Dimension:

an axis along which data is physically organized in an MDC table

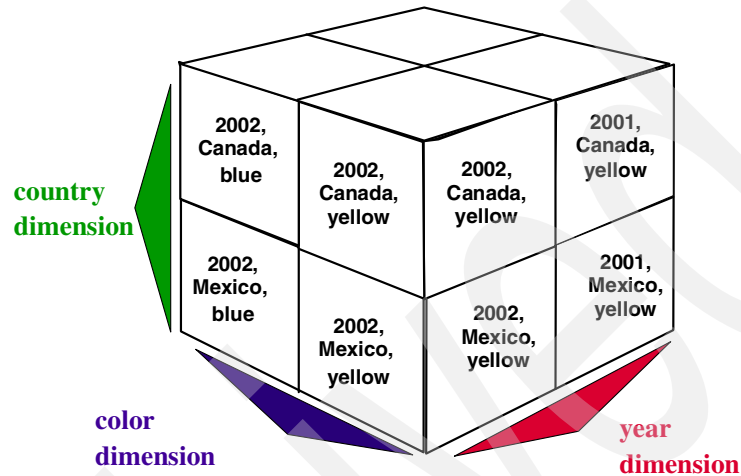


Figure 4-1 MDC: dimensions

DB2 UDB manages MDC tables by block according to dimensions defined when the table was created, instead of row ID as done by clustering index (see Figure 4-2).

Row Indexes, Block Indexes

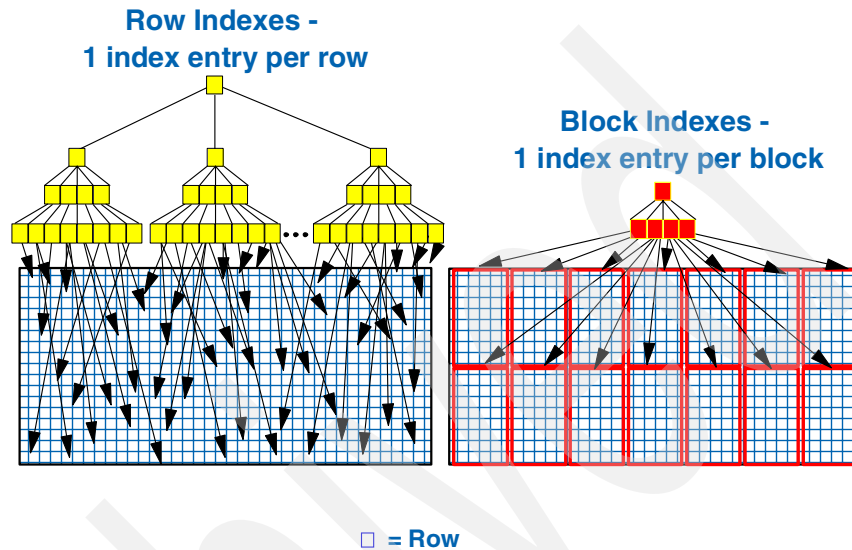


Figure 4-2 MDC is using block indexes

Clustering is guaranteed. If an existing block satisfies all dimensions, insert will put the row into that block if there is sufficient space. If there is not enough space in the existing block, or if no block exists with the dimensions, a new block is created to store the row.

Some terminology

The previous paragraph referred to dimension and block. While these terms probably make a little sense, let us define them more precisely. A few additional terms will be presented too.

Block

A block is the smallest allocation unit of the MultiDimensional Clustering table. In DB2 UDB V8.1, a block is equivalent to an extent, which in turn has that non-alterable number of pages specified or defaulted when the table space was created.

Dimension

A dimension is an axis along which data is organized in a multidimensional clustered table. A dimension is an ordered set of one or more columns, which may be thought of as one of the clustering keys of the table.

Slice

A slice is the portion of the table, which contains all rows having a specific value for one of the dimensions, as in Figure 4-3, Figure 4-4, and Figure 4-5.

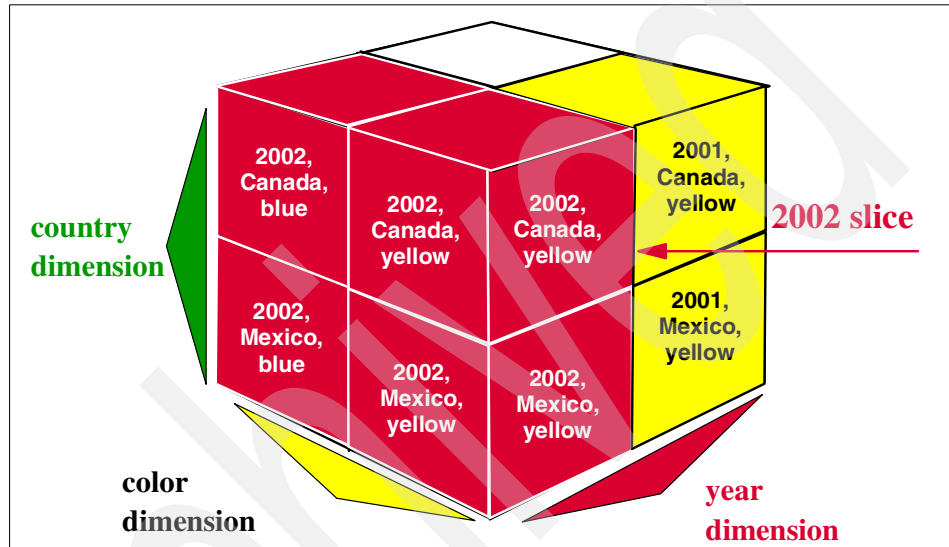


Figure 4-3 The slice for year = 2002

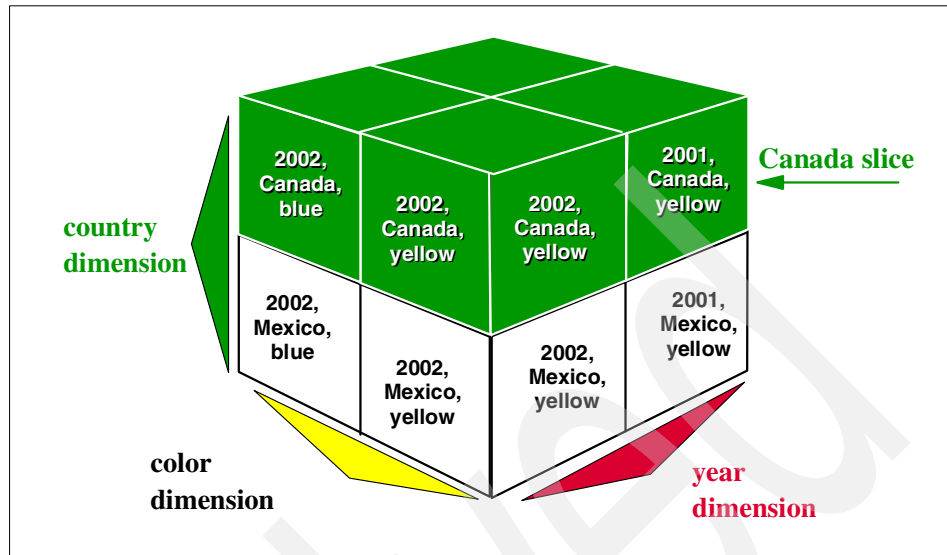


Figure 4-4 The slice for country = Canada

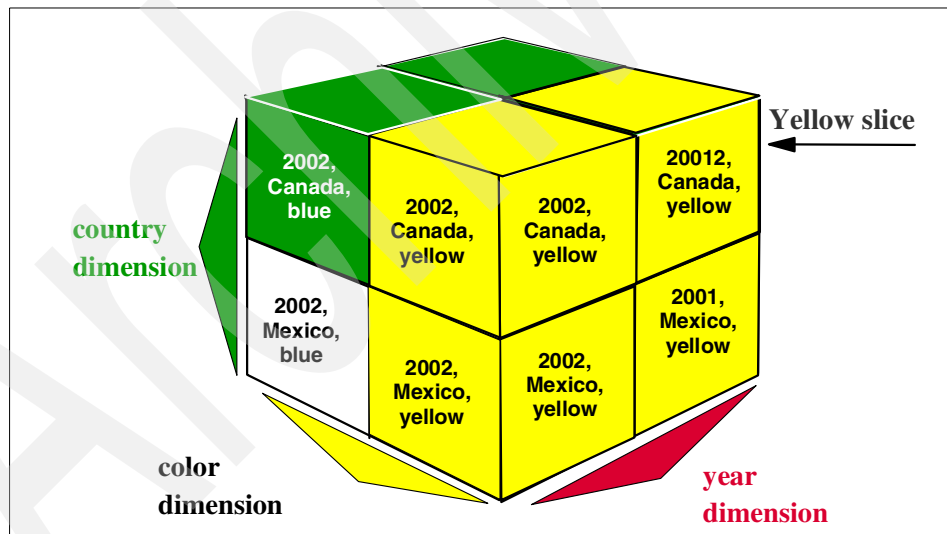


Figure 4-5 The slice for color = yellow

Cell

A cell is the portion of the table that contains rows having the same unique set of dimension values. The cell is the intersection of the slices from each of the dimensions.

The size of each cell is at least one block (=extent) and possibly more.

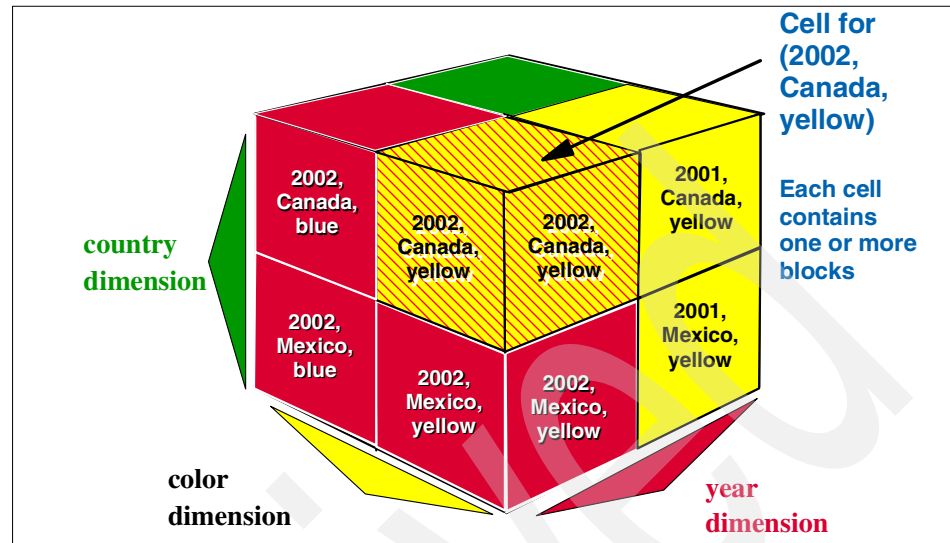


Figure 4-6 The cell for dimension values (2002, Canada, yellow)

Block index

The structure of block index is almost identical to a regular index. The major difference is that the leaf pages of a regular index is made up of pointers to rows, while the leaf pages of a block index contain pointers to extents. Because each entry of a block index points to an extent, whereas the entry in a RID index points to a row, a block index will be much smaller than a RID index, but it still points to the same number of rows.

In determining access paths for queries, the optimizer can use block indexes just like it uses RID indexes. The block indexes can be ANDed and ORed with other block indexes. They can also be ANDed and ORed with RID indexes. Block indexes can be used to perform reverse scans.

Because the block index contains pointers to extents, not rows, a block index cannot enforce uniqueness of rows. A RID index on the column would be necessary.

Dimension block index

The dimension block index is automatically generated by the system when the multidimensional table is created. A one dimension block index will be created for each column (or set of columns in the case of a multicolumn dimension) specified in the *ORGANIZE BY DIMENSIONS* clause of the *CREATE TABLE* statement.

Dimension block indexes are required. Consequently, they cannot be dropped. A dimension block index can be renamed. So, if you are not pleased with the weird system generated name, you can provide a more meaningful name.

4.2 Design guidelines for MDC tables

Properly chosen dimensions and extent sizes can offer significantly improved performance, depending on the data and workload. Unfortunately, the converse is also true. If incorrect choices are made, utilization of space might be poor, and performance could degrade. To design good MDC tables, you must first identify potential dimension candidates (the critical task), and then evaluate the storage space needed for a MDC table using some variation of those dimensions.

4.2.1 Step 1: Identify dimension candidates

The first step toward MDC is to examine the existing or planned workload seeking queries which will benefit from clustering. For an existing database, the workload can be captured from the SQL snapshot on statement, and SQL statement event monitor, from queries of the dynamic statement cache, or from application logs or records.

Note: The DB2 Query Patroller, when it is available for DB2 UDB V8.1, can also assist with workload evaluation.

If your database is still in the planning stage, you will need a good idea of the SQL that will be run.

The following are candidates for clustering:

- ▶ Columns used for range, equality, and IN predicates. Examples are:
`shipdate>'2002-05-14', shipdate='2002-05-14', year(shipdate) in (1999, 2001, 2002)`
- ▶ Roll-in or roll-out of data. For example:
`delete from table where year(shipdate) = '1999'`
- ▶ Columns referenced in GROUP BY clause
- ▶ Columns referenced in ORDER BY clause
- ▶ Foreign key columns in fact table of star schema database
- ▶ Combinations of the above

From the above list, discard columns where the values are frequently changed.

4.2.2 Step 2: Evaluate storage requirements

The second consideration that applies to dimension selection is to determine how much space will be needed to store the multidimensional clustering table. If you neglect to account for space requirements, or make inaccurate calculations, you can be in for quite a shock. It is worth repeating:

Important: Be sure to consider the storage requirements of the multidimensional clustering table!

So, how much space is required? Recall that the unit of storage allocation for an MDC table is an extent. For each unique combination of values for the chosen dimensions, at least one extent will be required. If many rows have a particular combination of values, they may fill an extent, and expand to one or more additional extents. But if only one row occurs with a particular combination of values, that single row will occupy the entire extent. The ideal MDC table is the one where every cell has just enough rows to exactly fill one extent. It is also a fantasy. The objective of this section is to outline a set of steps to get as close to the ideal MDC table as possible.

Perhaps you discover that most of the cells in your table will each occupy two or more extents. As the number of extents per cell increases, more I/O operations are needed to retrieve the rows for the cell. Performance might be improved if the number of blocks could be reduced by consolidation. Unless the number of extents per cell is excessive, this situation is not considered a problem.

On the other hand, if your choice of dimensions leads to a lot of extents, each having only a few rows, you can eat up a lot of space in a hurry. These mostly empty blocks are sometimes called *sparse blocks* or *sparse extents*. Having a table with a large number of mostly empty blocks is generally considered to be a problem. Especially if the table has millions or billions of rows. Of course, there are always exceptions. Perhaps you know that over time new rows will be added, which will use the excess space. In this case, we call the extra space free space instead of wasted space.

How well or poorly space is utilized depends upon the number of cells, the number of rows per cell, and the size of a block. For both the non-MDC and the MDC table, the unit of storage allocation is the extent.

The next couple of paragraphs illustrate how to estimate storage requirements for the non-MDC table and for the MDC table. The intent is not to insult your intelligence, but to focus on the difficulty of estimating storage and to point out potential sources of inaccuracy. Calculating the storage requirement for a non-MDC table is straightforward. Basically, we just have to determine how many rows (with no more than 255) will fit in a page after accounting for overhead and

freespace. Divide the rows per page into the total number of rows to determine the number of pages needed. Toss in any pages for free space, and then enough pages to fill the final extent and you have a pretty good idea of your storage requirement. The accuracy of your answer really depends upon two things: how well you estimate the total number of rows and the use of an average row length. If using the average length of a row causes you some distress, use the maximum row length instead.

How to calculate the space required to store an MDC table implemented with a potential set of candidate dimensions is easy to describe. To do it accurately depends on the assumptions you make to simplify the estimation.

Estimate the number of cells

Once we select a candidate dimension set, we must identify how many potential cells are likely to occur. The number of cells will be equal to the number of unique combinations of the dimension attributes. For an existing non-MDC table, the query in Example 4-1 will provide the exact number of cells.

Example 4-1 Counting the number of cells for MDC

```
WITH cell_table as (  
    SELECT DISTINCT dimension_col1, dimension_col2, ... , dimension_colN  
    FROM table )  
SELECT COUNT(*) as cell_count  
FROM cell_table
```

If a non-MDC table does not exist in the database, an upper bound for the number of cells can be determined by computing the cartesian product of the expected number of unique values in each dimension column. This simple estimate will be inaccurate if there are correlations between the dimension columns. For example, consider an orders table. Assuming normal delivery of items, shipdate and receiptdate columns are correlated. For any given shipdate, the receiptdate will be a couple of days later; receiptdate will certainly not be earlier than shipdate. On average, all orders with the same shipdate will have the same receiptdate. The number of unique combinations of these two columns will be somewhere between the number of unique values (column cardinality) of shipdate, and some small multiple of that number of unique values (column cardinality multiplied by two).

Estimate the space occupancy per cell

A good way to determine whether the space utilization of an MDC table will be acceptable, is to compare the projected MDC storage requirement to the storage required for the non-MDC table. In most cases, non-MDC tables in existing databases will be evaluated for organizing by dimensions. Where a non-MDC

table does not exist, the evaluation will be much easier if you can create a representative sample table.

For an existing non-MDC table, we recommend that the table be reorganized or freshly loaded. Then execute **RUNSTATS** to update the table statistics. Now the *FPAGES* utilization statistic can be used to accurately determine the current number of pages allocated and to provide a baseline for evaluating dimension choices.

Having determined the number of cells, the next task is to estimate how much space is needed to store each cell. First, determine the total number of rows in the table. For an existing table, a count of existing rows will establish this value precisely. Otherwise, make your best guess. Also, calculate the size of a storage unit for a cell, *BlkSz*, in bytes. *BlkSz* equals page size in K multiplied by 4096 bytes/K multiplied by the number of pages in an extent)

The SQL in Example 4-2 is useful for estimating rows per cell, *RpC*. The minimum and maximum rows per cell, *MinRpC* and *MaxRpC*, report the range of values for *RpC*, and provide an indication of the accuracy of the *RpC* value.

Example 4-2 SQL to examine rows per cell

```
WITH cell_table (dimension_col1, dimension_col2, ... , dimension_colN, RpC)
  as (SELECT DISTINCT dimension_col1, dimension_col2, ... , dimension_colN,
        COUNT(*) FROM table
        GROUP BY dimension_col1, dimension_col2, ... , dimension_colN)
SELECT avg(RpC) as RpC, min(RpC) as MinRpC, max(RpC) as maxRpC
FROM cell_table
```

Estimate the space occupied per cell, *SpC*, by multiplying *RpC* by the average row size.

Finally, dividing the space required per cell (in bytes) by the size of a block (the size of and extent in bytes), we can get an idea of how well (or how poorly) storage space will be utilized by the current set of dimensions an a specific block size.

A utilization value of 1.0 indicates that one cell will fully occupy one extent. At this point, create the MDC table using the candidate dimension set. Compare the space required to by the MDC table against the baseline from the non-MDC table.

A really large utilization value, say 10 or higher, indicates that little storage is unused, but you might be able to boost performance by increasing the size of a block, and thereby reduce I/O operations. As the utilization gets higher, you may be able to improve performance by adding additional dimension keys.

An extremely small utilization (0.1 or less) indicates that a great deal of storage space allocated to the table will be unused. A utilization of 0.1 indicates that if an MDC table were to be created now, it could require 10 times as much space as needed to store the non-MDC table.

If the storage of your MDC table is or likely will be unsatisfactory, there are some tuning knobs you can apply.

Tuning space requirements for MDC

If the utilization of space by an MDC table is unacceptable, you must either change the number of cells, change the size of a cell's allocation unit (block = extent), or both. Specifically, you can:

- ▶ Change the extent size
- ▶ Change the page size
- ▶ Change the granularity of one or more dimensions
- ▶ Change the number candidate dimensions
- ▶ Try a different combination of dimensions

Adjusting page size and extent size

After analyzing your space requirements, you may determine that the resulting cells contain too few rows to adequately fill an extent. Unless the number of rows is really small, you can examine what might happen if you reduced the size of an extent. The page size and extent size are set when table space is created. Neither value can be altered. The only choices are either to create an additional table space, or to delete and recreate the existing table space. The choice of page size is limited to specific values: 4096, 8192, 16384, or 32768. Regardless of the size, a page can contain no more than 255 rows. The table space page size must match the page size of the bufferpool assigned to the table space.

The size of a block and the allocation unit of a cell can be varied by changing some combination of the size of a page and/or the number of pages in an extent, by reducing the size of an extent (page size multiplied by number of pages in an extent).

Let us start with a simple example. Assume that we are developing a star schema database where the fact table has dimensions of region, store, and day. The table is planned to store 5 years of sales records for 200 stores in four regions. The potential number of cells needed can be roughly calculated as the cartesian product of the number of unique values for each of the dimensions. For the example, there are potentially $4 \times 200 \times 365 \times 5 = 730,000$ unique combinations of values or cells. One way to reduce space requirements for the table is to rollup one or more dimensions to a more coarse granularity and a lower cardinality.

You have a couple of solutions to evaluate: first, consider reducing the extent size. Another possibility is to provide some higher level of grouping along one or more dimensions.

The potential number of extents needed to store a table can be roughly calculated as the cartesian product of the number of unique values for each of the dimensions. For example, consider a MDC table with dimensions of region, store, and day. Assuming four regions, 200 stores, and 5 years of sales records, you have the potential for $4 \times 200 \times 365 \times 5 = 730,000$ unique combinations. To reduce space requirements for the table, consider rolling up one or more of the dimensions to a more coarse granularity and a lower cardinality. Referring back to the previous example, perhaps you can replace the day dimension with a year_and_month dimension. With the same values for region and store dimensions, but replacing the day dimension with year_and_month, the possible number of cells is reduced to $4 \times 200 \times 12 \times 5 = 48,000$. With this change, each cell will potentially contain more rows, and will be less likely to contain only a few.

When considering whether to change the granularity of a dimension, take account of the predicates commonly used on that column. If most predicates are on specific days, and you have clustered the table on year_and_month, DB2 will be able to use the block index on year_and_month to quickly narrow down which months contain the desired days and access only those associated blocks. But when scanning the blocks, DB2 UDB must apply day predicate to each row to determine which days qualify.

Having selected the dimensions, you only have the question of how to order those dimensions. When a MDC table is created, a composite block index is automatically created in addition to the dimension block indexes. This composite index is used when records are inserted into the table. The index may also be used, however, as part of an access path to satisfy a query -- just like an index on compound columns might be used today.

4.3 Creating a MDC table

After the dimensions are chosen, to defining the table is easy!

DB2 UDB V8.1 adds the *ORGANIZE BY DIMENSIONS* clause to the **CREATE TABLE** statement to indicate that a table is multidimensionally clustered. The keyword *DIMENSION* is optional.

Several examples will be used to illustrate the MDC table creation process.

Example 4-3, creates a simple MDC table with three dimensions.

Example 4-3 Create a MDC table with single-column dimensions

```
CREATE TABLE mdc1 (date DATE, store CHAR(8), region CHAR(2),  
    year_and_month generated as INTEGER(date)/100)  
    ORGANIZE BY DIMENSIONS (region, store, year_and_month)
```

In addition to table mdc1, three dimension block indexes will be automatically created: one for region, one for store, and one for the generated column year_and_month. Another index, the composite block index, will be created for the tables. Defining a dimension for a table is as easy as specifying a key definition for an index. Indeed, it is even more simple -- a separate **CREATE INDEX** statement is not required!

Note: With MDC, block indexes are always built. Set SORTHEAP and SHEAPTHRES appropriately.

The *INTEGER* function used for generating one of the dimension columns was extended in DB2 UDB V8.1 to support the date type (and a time value). For example, *INTEGER*('10/31/2002') yields 20021031. Conveniently, dividing by 100 gives the year and month, or 20021031/100 = 200210.

Note: Range scans on generated columns can only be done when the expression used to generate the column is monotonic. Monotonic means:

```
if (A > B) then expr(A) >= expr(B) and  
if (A < B) then expr(A) <= expr(B)
```

Or in English, as A increases in value, the expression based upon A also increases or remains constant.

Examples of monotonic operations include: A + B, A * B, integer(A).

Examples of non-monotonic operations are: A - B, month(A), day(A). The expression month(A) is non-monotonic because as A increases, the value of the expression fluctuates. month(20010531) equals 05; month(20021031) equals 10; but month(20020115) equals 01. So, as the date value increases, the value of the month fluctuates.

If the compiler cannot determine whether or not an expression is monotonic, the compiler assumes that the expression is *not* monotonic.

If the compiler identifies an expression as non-monotonic, then only equality predicates will be used on the generated column. Range scans will not be done.

Each dimension can be made up of one or more columns. Example 4-4 on page 149, illustrates the syntax for creating a MDC table where one of the dimensions is a composite key.

Example 4-4 Create MDC table with one multicolumn dimension

```
CREATE TABLE mdc2 (col1 INTEGER, col2 CHAR(8), col3 CHAR(2),  
                    col4 FLOAT)  
ORGANIZE BY DIMENSIONS (col1, (col3, col4), col2)
```

Table mdc2 will be created. At the same time the system will automatically create the three dimension block indexes, and the composite block index. The block index for dimension col3,col4 will be a composite key.

4.4 Using a MDC table

As a standard table, data will be loaded, inserted, updated, deleted in the MDC table.

Tip: When using the **LOAD** command on a MDC table, you can try to presort the records to speed up the LOAD.

How does the MDC table behave during insert and delete processing?

4.4.1 Insert processing

When a row is to be inserted in an MDC table, DB2 UDB probes the table's composite block index to determine whether or not an entry for the cell already exists.

If an entry for the cell exists, DB2 UDB proceeds as outlined in step 1 below; otherwise, the flow of step 2 is followed:

1. The cell exists. DB2 UDB scans the list of BIDs (block identifiers). For each BID, DB2 UDB uses the Free Space Control Record (FSCR) in the block to locate room to store the record.

Note: A FSCR is a special record stored on the first page of each block. It has a 20 byte fixed length section plus bit for each page in the block.

If, after all block FSCRs have been checked, no space is found, a new extent is required and the logic of step 2 applies. Space has been found on a page in the block. DB2 UDB logs the row, inserts it, and updates the FSCR. Note that

no entry is required to any of the dimension indexes nor to the composite index. If one or more row ID (RID) indexes exist, log and insert the key and RID in each.

2. Either a new cell is required or all blocks of an existing cell are full. First, DB2 UDB scans the block map trying to find a free block. DB2 UDB first tries to use an empty block, one from which all rows have been deleted, but not yet released by **REORG**. If none is found, a new extent is obtained. When a block has been obtained, update the log and change the block status to in use. Log and add the new key and block ID to each dimension block index, and to the composite block index. DB2 UDB logs and inserts the row into the new block, and updates the FSCR. Finally, if any RID indexes exist, DB2 UDB logs and inserts the RID in each.

During insert processing, hints bits in the composite block index are used to avoid an exhaustive search of the FSCRs.

During load of unsorted data, set *UTILHEAP* high, and allow the load buffer size to default to 25% of *UTILHEAP*.

4.4.2 Delete processing

When a row is deleted from an MDC table, the following occurs:

1. The row is deleted, but the block still exists.

The record to delete will be logged and deleted from the table as row ID indexes if they exist.

2. Both the record and the block are deleted

The record to delete will be logged and deleted from the table. The block entry status the record belongs to will be logged and changed to free before logging and removing the block-ids from any block indexes (if they exist). Any row ID indexes will also be removed if they exist.

4.5 MDC and multipartition database

If you plan to use MDC in a DB2 UDB ESE V8.1 multipartition environment, you need to consider the potential impact of partitioning upon space utilization of MDC tables. If your partitioning key matches one or more of the MDC dimension keys, you have no problem. But if your partitioning key is not one of the table dimensions, you need to evaluate the impact of partitioning on your space calculations. Partitioning potentially reduces the granularity of your dimension keys.

Example 4-5 shows a query on a potential MDC table spread across 32 database partitions. The table to be evaluated is the lineitem table, which has 360 million rows. The average row size is 127 characters. First, we look at the lineitem table without regard for the effect of partitioning.

Example 4-5 Rows per cell in lineitem table

```
select distinct l_shipdate, l_receiptdate, count(*) as rowcount
  from tpcd.lineitem
 group by l_shipdate, l_receiptdate
 order by 3 desc, l_shipdate, l_receiptdate
 fetch first 20 rows only
```

L_SHIPDATE	L_RECEIPTDATE	ROWCOUNT
01/24/1998	02/17/1998	5287
07/25/1992	08/07/1992	5279
09/19/1995	09/29/1995	5276
03/30/1993	04/03/1993	5275
10/29/1992	11/26/1992	5266
09/08/1997	09/21/1997	5266
09/30/1992	10/28/1992	5265
07/20/1998	07/27/1998	5259
12/23/1995	01/04/1996	5253
03/17/1995	03/19/1995	5251
07/28/1996	07/29/1996	5248
06/23/1992	07/14/1992	5247
07/03/1992	08/02/1992	5247
02/26/1995	03/19/1995	5245
06/21/1995	07/08/1995	5245
11/24/1996	12/09/1996	5245
06/04/1993	06/25/1993	5244
01/22/1998	02/19/1998	5244
01/23/1996	02/17/1996	5243
07/01/1998	07/03/1998	5242

20 record(s) selected.

In Example 4-5, each of the 20 output lines is one cell in MDC terminology. Assuming that these first 20 cells is representative, lets estimate that each cell contains around 5,200 rows. (It turns out that this might be a bad assumption if we examine the twenty cells with the fewest number of rows. See Example 4-6.)

Example 4-6 Twenty lineitem table cells with lowest row counts

```
select distinct l_shipdate, l_receiptdate, count(*) as rowcount
  from tpcd.lineitem
 group by l_shipdate, l_receiptdate
```

```
order by 3 desc, l_shipdate, l_receiptdate
fetch first 20 rows only
```

L_SHIPDATE	L_RECEIPTDATE	ROWCOUNT
01/02/1992	01/14/1992	24
01/02/1992	01/23/1992	27
01/02/1992	01/04/1992	30
12/01/1998	12/16/1998	31
01/02/1992	01/06/1992	32
01/02/1992	01/08/1992	33
01/02/1992	01/03/1992	34
01/02/1992	01/19/1992	34
12/01/1998	12/08/1998	34
12/01/1998	12/10/1998	34
01/02/1992	01/11/1992	35
01/02/1992	01/15/1992	35
01/02/1992	01/16/1992	35
01/02/1992	01/27/1992	35
01/02/1992	02/01/1992	36
01/02/1992	01/09/1992	37
01/02/1992	01/12/1992	37
12/01/1998	12/12/1998	37
12/01/1998	12/28/1998	37
01/02/1992	01/17/1992	38

20 record(s) selected.

The defaults for page size and extent size in our sample database are 16 K and 32 pages, respectively. So, with *head in the sand* and 5200 row estimate, each cell would occupy around 27 percent of an extent. With the other 73 percent unused, there is clearly a problem. If lineitem were to be recreated as an MDC table, it would require approximately four times the space that is currently allocated.

Example 4-7 summarizes the calculation.

Example 4-7 Calculating the number of extents

With 16K page, 16293 bytes are usable.

The average lineitem row size is 127 bytes, so one database page can contain up to $\text{integer}(16293/127) = 603$ rows.

With 32 16K pages per extent, a cell of as many as 19276 rows will occupy a single extent. Recall that for MDC an extent equates to a block.

The estimate of 5200 rows per cell indicates that each cell will use around 27 percent of an extent, the minimum unit of space allocation for a cell as shown below:

$$(5200/19276)*100 = 27 \text{ percent (approximately)}$$

At this time, we could play with the 4 tuning knobs (page size, extent size, dimension granularity, number of dimensions) to derive a reasonable allocation. But before going to this trouble, there is still a problem. The lineitem table is partitioned on column L_ORDERKEY. The columns planned for dimensions are L_SHIPDATE and L_RECEIPTDATE. The following query, Example 4-8, repeats Example 4-5 on page 151, except that we now account for the impact of partitioning on lineitem.

Example 4-8 Accounting for the impact of partitioning

```
select distinct l_shipdate, l_receiptdate, count(*) as rowcount
  from tpcd.lineitem
 where dbpartitionnum(l_shipdate) = 13
 group by l_shipdate, l_receiptdate
 order by 3 desc, l_shipdate, l_receiptdate
 fetch first 20 rows only
```

L_SHIPDATE	L_RECEIPTDATE	ROWCOUNT
11/26/1992	12/18/1992	211
09/01/1995	09/08/1995	210
09/25/1994	10/05/1994	206
01/13/1998	02/03/1998	206
02/13/1998	03/12/1998	206
06/20/1996	07/14/1996	205
12/21/1993	01/07/1994	204
06/21/1994	07/08/1994	204
06/14/1992	07/02/1992	203
01/12/1993	01/14/1993	203
12/21/1996	12/28/1996	203
02/11/1998	02/20/1998	203
01/08/1993	01/27/1993	202
04/11/1993	05/03/1993	202
06/03/1993	06/14/1993	202
10/03/1996	10/04/1996	202
02/18/1994	03/07/1994	201
07/26/1994	08/06/1994	201
02/25/1995	03/14/1995	201
06/04/1995	06/12/1995	201

20 record(s) selected.

When partitioning is taken into account, the number of rows per cell drops from around 5200 to around 200. The drop in rowcount is roughly equal to the original rowcount (without accounting for partitioning) divided by the number of partitions over which the table is spread. Instead of a particular set of dimension key values mapping to one cell for the entire table, the identical set of values can occur on every partition. So, each partition will likely have a cell instead of one cell for the entire table. Example 4-9 compares the first 10 sorted cells in database partition 9 to the first 10 sorted cells from partition 13. In this small list, six of ten dimension key values match.

Example 4-9 Compare dimension values in two partitions

```
-- DB partition 9
select distinct l_shipdate, l_receiptdate, 9 as partition
  from tpcd.lineitem
  where dbpartitionnum(l_shipdate) = 9
  group by l_shipdate, l_receiptdate
  order by l_shipdate, l_receiptdate
  fetch first 10 rows only
```

L_SHIPDATE	L_RECEIPTDATE	PARTITION
01/02/1992	01/03/1992	9
01/02/1992	01/04/1992	9
01/02/1992	01/06/1992	9
01/02/1992	01/07/1992	9
01/02/1992	01/08/1992	9
01/02/1992	01/09/1992	9
01/02/1992	01/11/1992	9
01/02/1992	01/12/1992	9
01/02/1992	01/16/1992	9
01/02/1992	01/17/1992	9

10 record(s) selected.

```
-- DB partition 13
select distinct l_shipdate, l_receiptdate, 13 as partition
  from tpcd.lineitem
  where dbpartitionnum(l_shipdate) = 13
  group by l_shipdate, l_receiptdate
  order by l_shipdate, l_receiptdate
  fetch first 10 rows only
```

L_SHIPDATE	L_RECEIPTDATE	PARTITION
------------	---------------	-----------

01/02/1992	01/03/1992	13
01/02/1992	01/04/1992	13
01/02/1992	01/05/1992	13
01/02/1992	01/06/1992	13
01/02/1992	01/08/1992	13
01/02/1992	01/11/1992	13
01/02/1992	01/13/1992	13
01/02/1992	01/17/1992	13
01/02/1992	01/18/1992	13
01/02/1992	01/19/1992	13

10 record(s) selected.

Important: In a partitioned database environment, be aware of the possible impact of partitioning when calculating space requirements for a potential MDC table.

If the partitioning key of the table matches one (or more) of the planned dimension keys, the space calculation will be easier. But if the partitioning key is not the same as one of the dimension keys, you will need to consider the impact of partitioning on your space calculations.

This does not mean that you should make the partitioning key the same as a dimension key. The considerations for choosing a dimension key are different from those of choosing a partitioning key.

4.6 Performance

As an illustration, this section reports and analyzes the results of a *proof of performance and scalability* called POPS for short.

For part of this test the DAILY_SALES fact table was set up as a regular (non-MDC) table. For the other part the DAILY_SALES table was a MDC table. The same set of queries was run against both the non-MDC and the MDC versions of the table. The results are reported below.

As illustrated in Figure 4-7, the database was a star schema design. The DAILY_SALES fact table was organized by period (PERIOD_KEY) and by store (STORE_KEY). Five dimension tables were used: period, store, customer, promotion, and product.

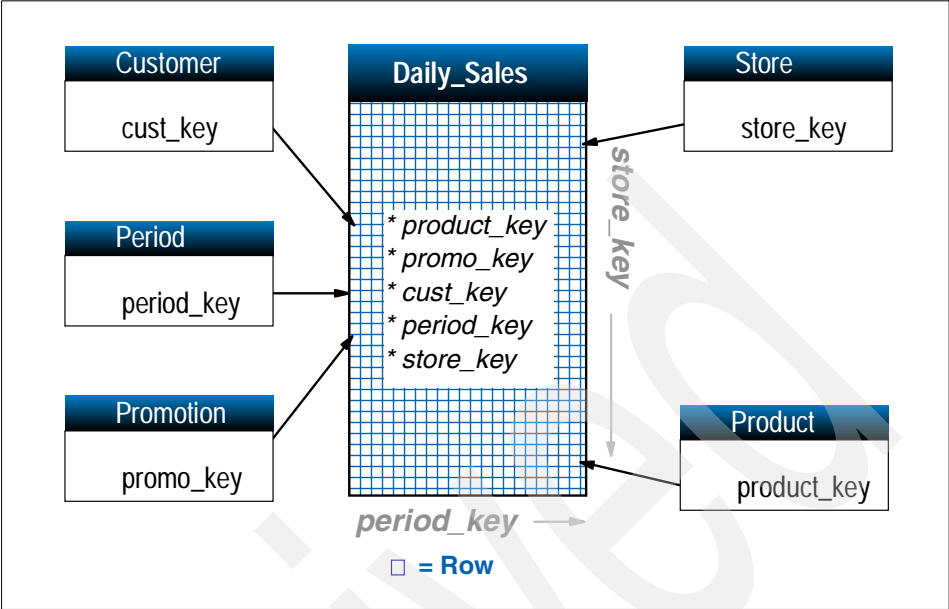


Figure 4-7 Star schema database for POPS

During the non-MDC runs, the DAILY_SALES table had RID indexes created on all star dimensions. For the MDC tests, DAILY_SALES was organized by dimensions PERIOD_KEY and STORE_KEY with standard RID indexes on the other three star dimensions. An additional multicolumn RID index was created on STORE_KEY, PERIOD_KEY.

The data table space used page size 32 K with an extent size 16 K. The index table space used the page size 4 K, and extent size 16 K. All table spaces were Database Managed Storage (DMS) raw.

Table 4-1 provides a comparison of the size requirements of the non-MDC and MDC configurations of the database. Note that the STORE_KEY and PERIOD_KEY block indexes are much less than 1% of the corresponding row indexes. When created as a multidimensional clustering table, DAILY_SALES required approximately 1% more disk space because of extent allocations.

Table 4-1 Size comparison

	store_key	period_key	product_key	daily_sales table
MDC index pages	71	72	222,086	--

	store_key	period_key	product_key	daily_sales table
non_MDC index pages	222,054	222,054	222,086	--
MDC index levels	2	2	4	--
non-MDC index levels	4	4	4	--
MDC table size (pages)	--	--	--	689,264
non-MDC table size (pages)	--	--	--	681,903

Query 1: point query on block versus RID index

As shown in Figure 4-8 on page 157, selecting a multidimensional slice (STORE_KEY=1) is much faster.

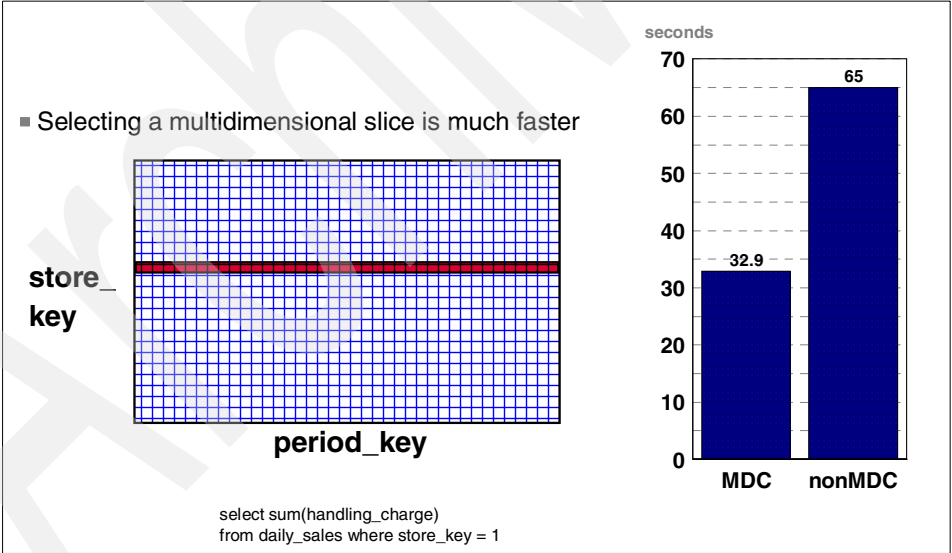


Figure 4-8 Query 1: point query on block versus row ID index

Query 2: range query on block versus RID index

This query selected all rows in a range of dates. Figure 4-9 compares the results of the MDC and non-MDC test.

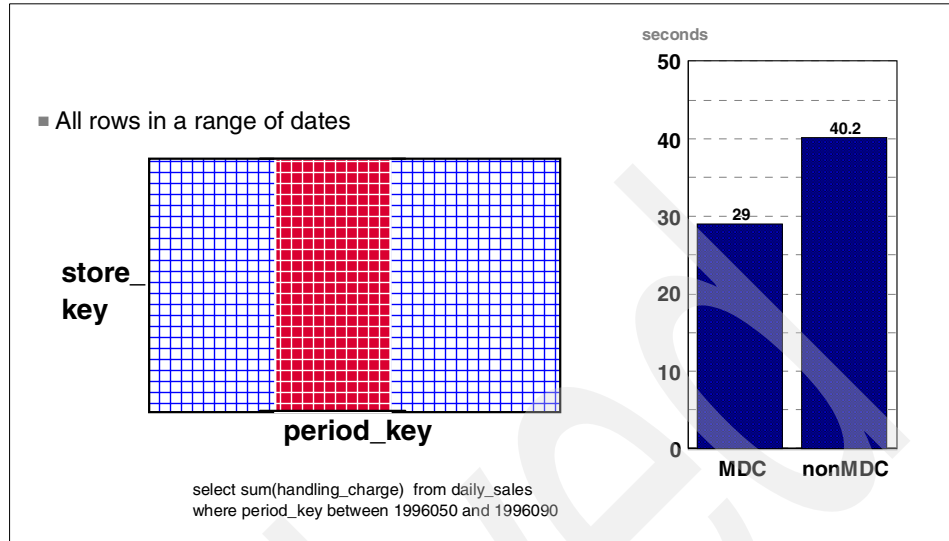


Figure 4-9 Query 2: range query on block versus row ID index

Query 3: range query on two dimensions

The results of a query in which two dimensions are qualified appears in Figure 4-10.

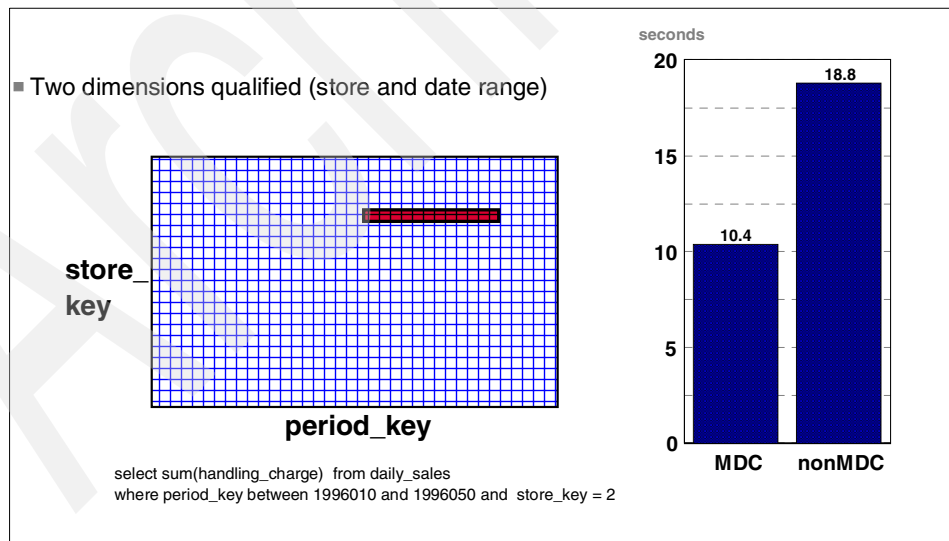


Figure 4-10 Query 3: range query on two dimensions

Query 4: query on a cell

Recall that a cell contains all rows which map to a specific set of dimension values. The allocated storage is at least one whole extent and possibly more depending upon how many rows have the same values for each of the dimensions. Query 4, graphed in Figure 4-11, scanned a cell worth of data. The benefit, though small, was due to a smaller index tree for MDC (block vs. RID index).

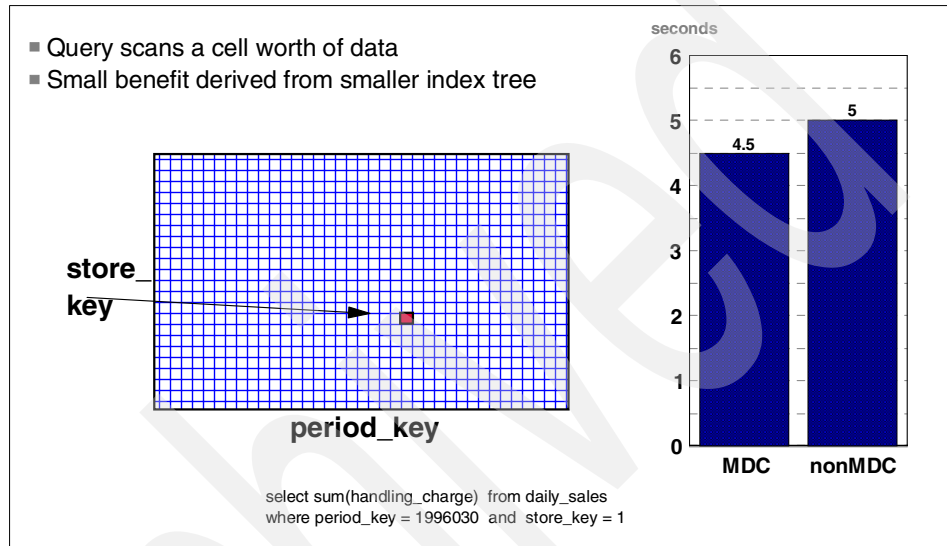


Figure 4-11 Query 4: query on a cell

Query 5: full table scan

The query scanned the entire table, and as Figure 4-12 shows, MDC offered little or no advantage.

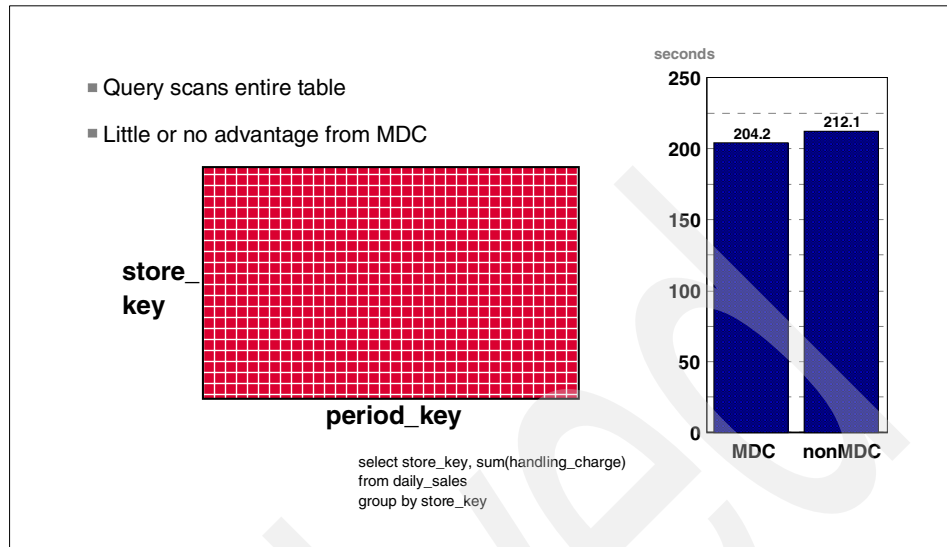


Figure 4-12 Query 5: full table scan

Query 6: table scan with block predicate

For the MDC table, the query scanned the entire table. Three slices were skipped. Performance of the MDC query was better, because the WHERE clause of the query only had to be applied once per block, whereas it had to be applied once for every row in the non-MDC table. See Figure 4-13.

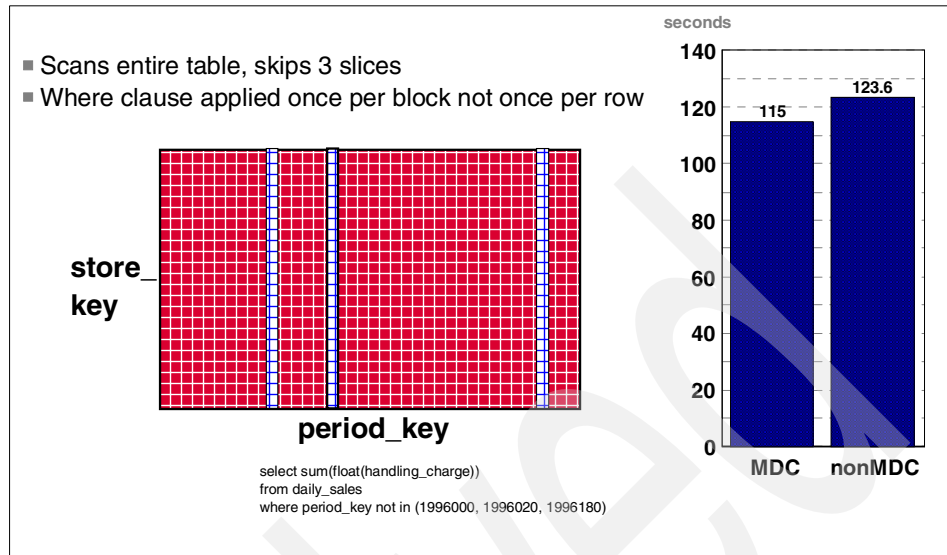


Figure 4-13 Query 6: table scan with block predicate

Query 7: index ANDing of block and RID indexes

With the MDC query, the result was only rows belonging to qualifying blocks (PERIOD_KEY). Index ANDing combined the block identifiers (BIDs) and row identifiers (RIDs) to determine the rows to return. The non-MDC query, as seen in Figure 4-14, was slower because two RID indexes had to be ANDed and then rows retrieved one by one.

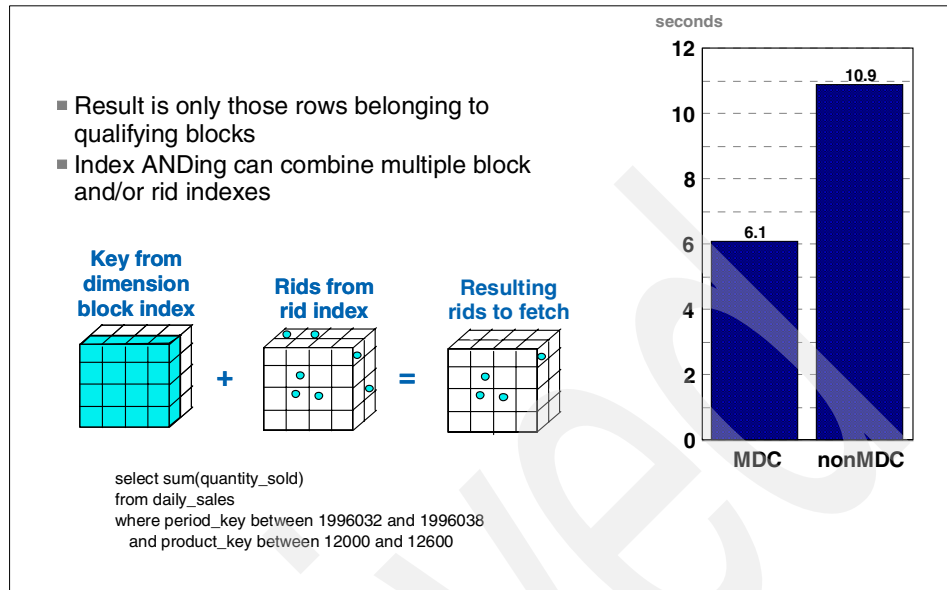


Figure 4-14 Query 7: Index ANDing of block and row ID indexes

Query 8: index ORing of block and RID indexes

Index ORing can combine multiple block and/or RID indexes. The better performance of the MDC query is that rows which matched the block key were read in large batches. Only the records qualified by PERIOD_KEY, which were outside of the blocks, had to be read page by page. With the non_MDC query, all retrieval was row by row. Figure 4-15 on page 163 shows the large advantage of the MDC query.

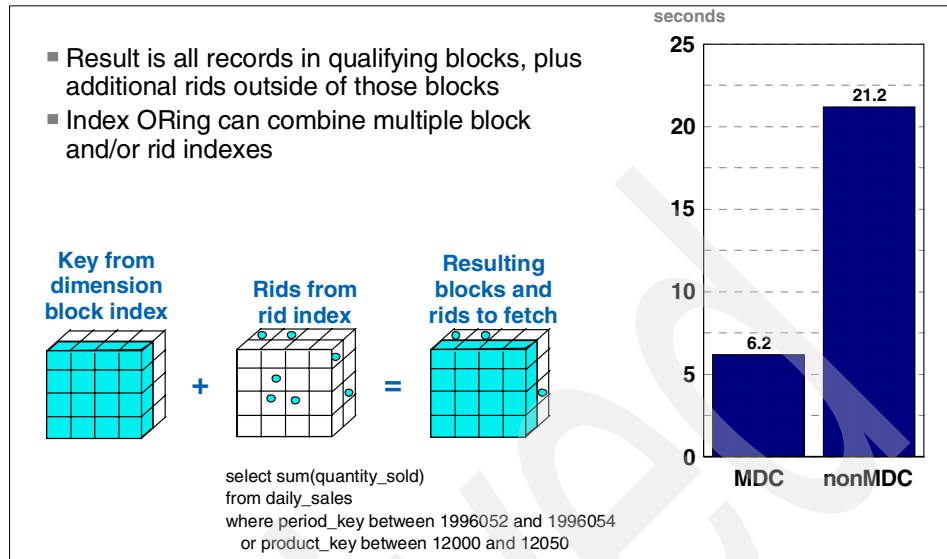


Figure 4-15 Query 8: Index ORing of block and row ID indexes

Query 9: nested loop join

Wow! Look at the performance comparison as shown in Figure 4-16.

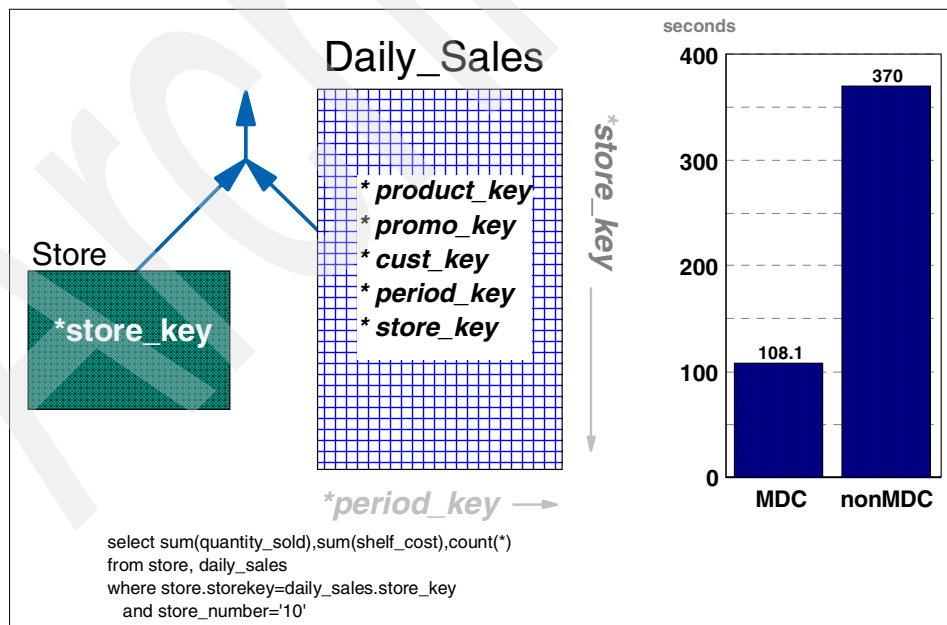


Figure 4-16 Query 9: nested loop join

Query 10: joining multiple tables

For MDC, two of the three join columns were dimensions. As shown in Figure 4-17, the performance of the MDC query was much better than the non-MDC query.

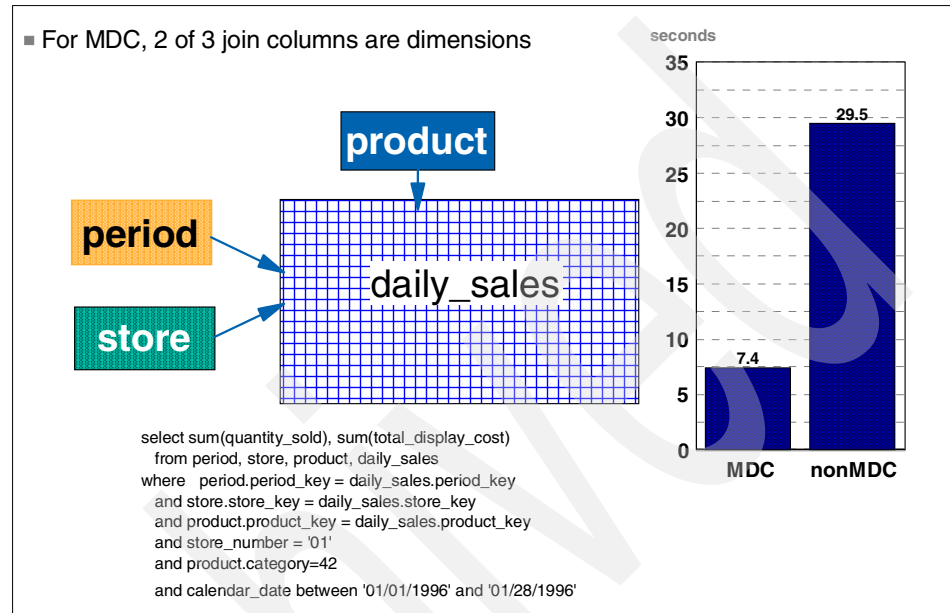


Figure 4-17 Query 10: joining multiple tables

4.7 Benefits of MDC

The benefits of MDC are simple:

- ▶ Easy to create table and indexes are automatic.
- ▶ Guaranteed clustering
- ▶ Performance, performance, performance!

4.8 Considerations and recommendations

When creating MDC tables the following should be considered:

- ▶ Know your SQL.
- ▶ MDC is not only for star schema database.

- ▶ If creating multidimensional clustering table(s) in an SMS table space, run command **db2empfa**. To check, **db2 get db cfg for <database>**.
In a multipartition database, **db2empfa** must be run on every partition.
- ▶ Verify that the cardinality of each dimension will not leave an excess number of sparse extents.
- ▶ In a partitioned environment, consider the potential impact of partitioning key, if the key is other than one of the dimension columns:
 - Chose partitioning key to maximize collocation of table joins
 - Choose dimensions based upon range queries.
- ▶ Verify that the extent size is not too big.
- ▶ Check the order in which columns are specified in the *ORGANIZE BY DIMENSIONS* clause.
- ▶ Clustering index is still available and can still be created.
- ▶ Although the name multidimensional clustering table implies that the number of dimensions must be greater than one, an MDC table can be build with just a single dimension.
- ▶ Load of MDC table always includes a build phase to create the required dimension indexes. So, set *SORTHEAP* and *SORTHEAPTHRES* to high values.
- ▶ At the current time, **REORG INPLACE** (online) cannot be used on MDC tables.
- ▶ **REORG** is not necessary unless you wish to consolidate sparse blocks.
- ▶ Be sure to install Fixpak 1 to correct the ordering problem with multicolumn dimension indexes.
- ▶ MDC tables are like ordinary tables in that referential integrity, views, and Materialized Query Tables can be defined upon them.
- ▶ The *SAVECOUNT* parameter of the **LOAD** command is not supported for MDC tables. Consequently, a **LOAD RESTART** will only take place at the beginning of the load, or the build, or the delete phase.

4.9 MDC design prototype

A starting attempt at a tool to assist the evaluation of how dimension choices will use storage is provided and described in Appendix B, “MDC dimension analyzer” on page 315.

Speed up performance with Materialized Query Tables

Like MDC tables, Materialized Query Tables (MQTs) offer a way to speed up response times on well known and repeatable queries against large volumes of data, where results are often expressed as summaries or aggregates. Automatic Summary Tables (ASTs) and replicated summary tables were first introduced to DB2 UDB in V5.2. Subsequent versions have extended the function and loosened restrictions. DB2 UDB V8.1 continues the trend. In fact, the functionality has been generalized to the point that a more fitting name was needed. That new name is *Materialized Query Table*, MQT to friends. AST functionality is now a subset of the MQT. Chapter 2, “DB2 UDB’s materialized views,” in the redbook *DB2 UDB’s High Function Business Intelligence in e-business*, SG24-6546 provides extensive coverage of MQTs (also called materialized views).

This chapter describes the MQT with emphasis on the enhancements added by DB2 UDB V8.1 and usage of partitioning. A MQT can:

- ▶ Use multidimensional clustering tables (MDCs) as base tables, as it can use regular tables as well.
- ▶ Be multidimensionally clustered themselves
- ▶ Be refreshed incrementally

5.1 MQTs overview

So what is a materialized query table and why does it exist? To the true believer in the relational model and the fully normalized database, any derived data is a sin. To derive an entire table is nothing less than heresy. And that in a nutshell is what an MQT is. It is a real (materialized) table built from the result set of a query. Like any other table, an MQT can have indexes and **RUNSTATS** should be used to create and store statistics about the table. MQTs primarily exist for three reasons: performance, performance, and performance.

For those of you in a federated database environment, there is actually a fourth reason that you might create a MQT: availability. You can locally cache data from one or more remote databases, so that when a remote database is not available, queries can continue using the cached data table.

Caching techniques are frequently applied to increase performance. And the increase is usually one or more orders of magnitude. As examples, consider that L2 cache is used as a kind of lookaside to minimize fetches from memory. The DB2 bufferpool caches data to minimize I/O to and from disk. You know that old saying: "There's no I/O like no I/O!" Similarly, MQTs enhance performance. Consider them as a kind of computational cache. Properly selected, MQTs perform a set of calculations that can then be used over and over by subsequent queries, but without going through the same operations again and again. The qualifying rows do not need to be refetched saving input time, the calculations need not be redone saving CPU time.

As shown in Figure 5-1, support for MQT also called materialized views requires the following:

- ▶ Having a DBA *pre-compute* an aggregate query, and *materialize* the results into a table. This summary table would contain a superset of the information that could answer a number of queries that had minor variations.
- ▶ Having an enhanced DB2 optimizer to *automatically rewrite* a query against the base tables to target the MQT instead (if appropriate) in order to satisfy the original query.

Since the MQT often contains precomputed summaries, and/or a filtered subset of the data, it would tend to be much smaller in size than the base tables from which it was derived. When a user query accessing the base table is automatically rewritten by the DB2 optimizer to access the MQT instead, then significant performance gains can be achieved.

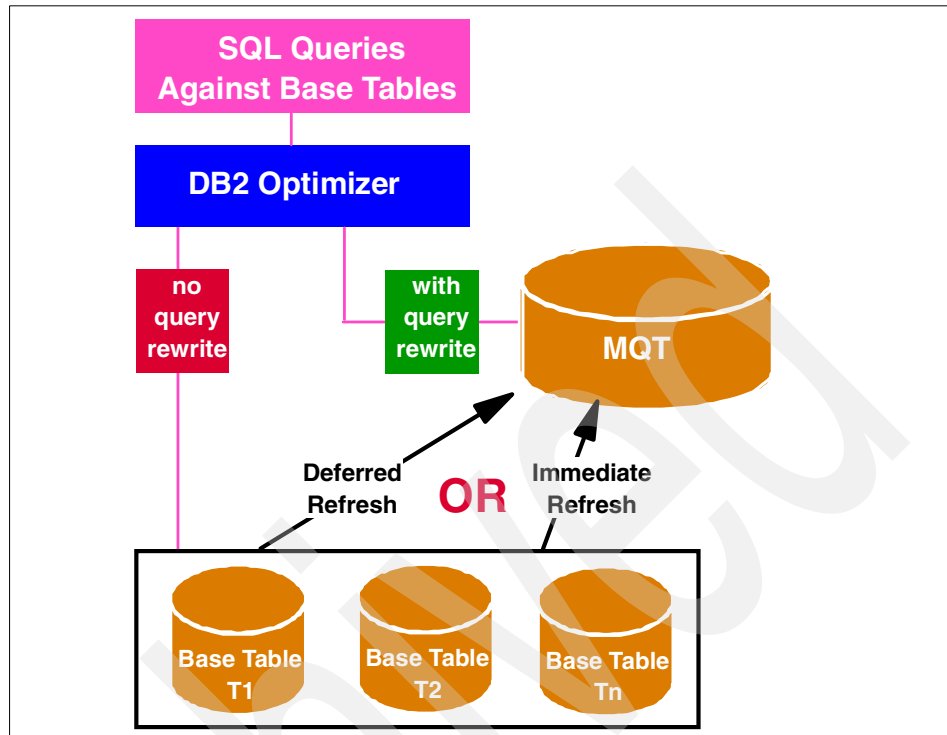


Figure 5-1 MQTs overview

Potential uses for the MQT follow:

- ▶ To store precalculated summary and aggregate data (the automatic summary table)
- ▶ To replicate tables or subset of tables
- ▶ To partition where the table does not exist
- ▶ To repartition a table to facilitate collocated joins
- ▶ To locally cache data from one or more remote databases (new in V8.1)
- ▶ To materialize a frequently performed inner join (new in V8.1)

Note: Outer joins are not currently supported with *REFRESH IMMEDIATE* tables or *REFRESH DEFERRED* tables associated with staging tables. They can still be used to create an AST.

- ▶ To separate hot rows and/or hot columns of a table into a table that might be better tuned for performance (by separating hot data, you could define better indexes, buffering possibilities and so on).

To create and populate the MQT, see the summarized steps in Figure 5-2:

1. Create the MQT.

When the MQT creation DDL is executed, MQT that has not yet been populated is placed in CHECK PENDING NO ACCESS. Dependencies regarding the base tables and the MQT are recorded in SYSCAT.TABLES, SYSCAT.TABDEP, SYSCAT.VIEWS just as any other table or view definition creation. All packages that update the base tables on which the MQT is built are invalidated and rebound to use the MQT.

2. Populate the MQT.

DB2 UDB supports MQTs that are either:

- *MAINTAINED BY SYSTEM* (default): MQTs are updated by DB2 UDB either when the underlying tables are updated or when a *REFRESH* statement is issued. Such MQTs may be defined as either *REFRESH IMMEDIATE*, or *REFRESH DEFERRED*. If the *REFRESH DEFERRED* option is chosen, then either the *INCREMENTAL* or *NON INCREMENTAL* refresh option can be chosen during refresh.
- *MAINTAINED BY USER*: In this case, it is up to the user to maintain the MQT whenever changes occur to the base tables. Such MQTs *must be* defined with the *REFRESH DEFERRED* option. Even though the *REFRESH DEFERRED* option is required, unlike *MAINTAINED BY SYSTEM*, the *INCREMENTAL* or *NON INCREMENTAL* option does *not* apply to such MQTs.

3. Tune the MQT.

This involves the creation of appropriate indexes, and executing the **RUNSTATS** utility on the MQT to ensure optimal access path selection.

In particular, for *REFRESH IMMEDIATE* tables or for those that we will have *INCREMENTAL* refresh done on it, we must have an index on the appropriate columns. This is absolutely necessary to enhance the performance. If an index is not defined, the *INCREMENTAL* refresh might take even longer than a full refresh. Defining an index might improve the performance by orders of magnitude.

Note: If the *GROUP BY* columns of an AST can be defined NOT NULL, performance of *INCREMENTAL* refresh might be further improved as compared to NULL columns.

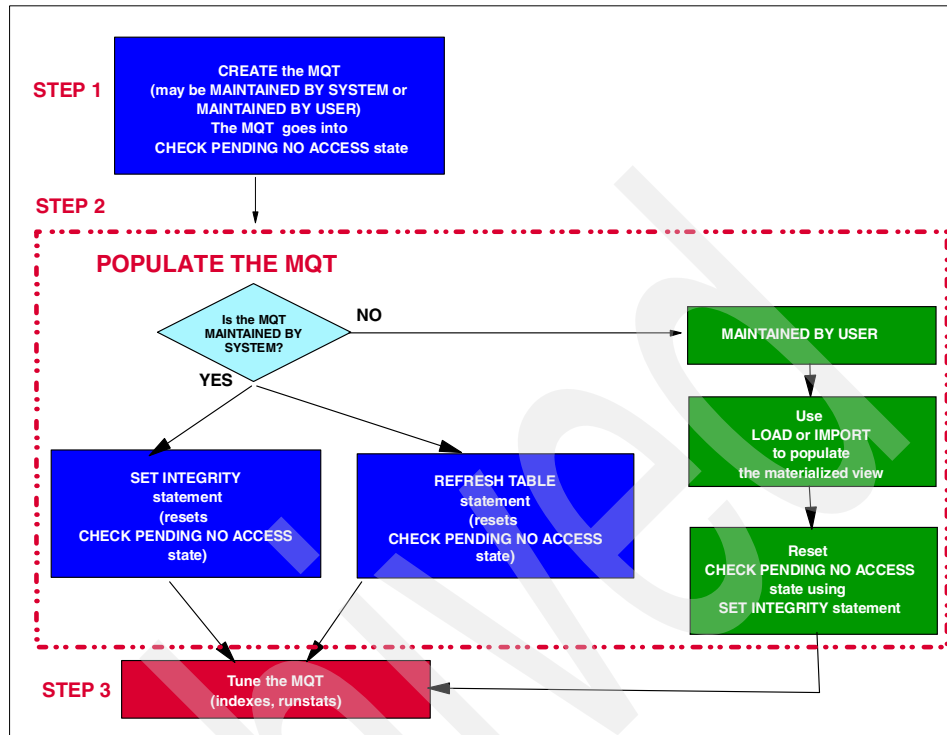


Figure 5-2 Create and populate the MQT

To maintain the MQ, two approaches may be adopted:

- ▶ In the deferred refresh approach, DB2 UDB does not automatically keep the MQT in synchronization with the base tables, when the base tables are updated. In such cases, there may be a latency between the contents of the materialized view, and the contents in the base tables. MQTs no longer require that the MQT be completely regenerated. An incremental refresh is now available through the use of a staging table.
- ▶ In the immediate refresh approach, the contents of the MQT are always kept in sync with the base tables. An update to an underlying base table is immediately reflected in the MQT as part of the update processing. Other users can see these changes after the unit of work has completed on a commit. There is no latency between the contents of the materialized view and the contents in the base tables.

For more details on how to implement MQTs, please refer to *DB2 UDB's High Function Business Intelligence in e-business*, SG24-6546, and to *DB2 Administration Guide: Implementation Version 8*, SC09-4820 for incremental maintenance of MQTs, and how to create a staging table.

5.2 When to consider a MQT?

The design of good materialized query tables requires adequate up-front planning and analysis. The designer needs to be familiar with the query workload to identify patterns for accessing tables, and frequently performed aggregation and summarization.

When deciding whether or not to create a materialized query table, consider the following:

- ▶ Will the MQT significantly increase performance?
- ▶ Will many queries benefit? Will the most frequent or most critical or most expensive/long running queries benefit?
- ▶ Will the MQT offer resource savings: communication, I/O, and CPU?
- ▶ Is the loss of disk space that will contain the MQT and its indexes a worthwhile trade for the performance that will be gained?
- ▶ What is the cost of updating or refreshing the MQT?
- ▶ What are the patterns for accessing groups of tables, for aggregation, and for grouping requirements?
- ▶ How current does the data in the MQT need to be? Does it need to be up to the minute?
- ▶ For MQTs that are maintained in real time, will automatic updates be too slow?
- ▶ In a partitioned environment, will collocation of data provide any benefit?
- ▶ What will be the logging requirement when large MQTs are refreshed?
- ▶ Should the MQT be system or user maintained?

5.3 When will the MQT be used?

Having gone to the trouble of creating a MQT, it would be nice if it were really used. This section discusses the two portions of the SQL compiler that determine whether or not an MQT will be used.

First, the query rewrite portion must be able to determine that a (maybe more) MQT exists and that it can be used to satisfy the current query.

Second, the optimizer examines the costs of using the MQT, and makes the final determination to use or not use the MQT.

For the optimizer to consider using an MQT in the access plan, certain conditions must preexist:

- ▶ The materialized query table must be created with *ENABLE QUERY OPTIMIZATION* clause of **CREATE TABLE**. This is the default option when an MQT is created.
- ▶ For REFRESH DEFERRED MQTs, the *CURRENT REFRESH AGE* special register must be set to *ANY*. This setting informs the optimizer that any version of the MQT can be used to determine the answer to the query.

REFRESH IMMEDIATE MQTs are always current and are candidates for optimization regardless of the setting of the *CURRENT REFRESH AGE* register.
- ▶ The optimization level must be equal to 2, or greater than or equal to 5.

The default value for the query optimization level is 5. It can be changed by updating the database configuration parameter *DFT_QUERYOPT*.

The value can also be set as follows:

Example 5-1 Set CURRENT QUERY OPTIMIZATION level

```
-- Valid values are 0, 1, 2, 3, 5, 7, and 9
SET CURRENT QUERY OPTIMIZATION 7
```

- ▶ The MQT must not be in REFRESH PENDING mode or CHECK PENDING NO ACCESS state.
- ▶ The special register *CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION* must be set for the appropriate type of MQT to be considered. Acceptable settings are *ALL*, *NONE*, *SYSTEM*, or *USER*. If set to *SYSTEM*, user maintained MQTs will not be used for optimizing.

Example 5-2 shows how to set this register to the value *SYSTEM*.

Example 5-2 Set CURRENT MAINTAINED TABLE TYPES register

```
SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION = SYSTEM
```

- ▶ The isolation level currently in effect must be equal to or lower than the optimization level that was in effect when the MQT was created.

5.4 Intra database replicated tables and partitioning

Attention: This form of replication is in no way related to the replication provided by IBM replication products and tools as DB2 DataPropagator that is interdatabase replication (replication of updates between databases).

In the partitioned database environment, the performance of joins increases dramatically when the rows of the different tables in the join are collocated, and we can avoid shipping data between partitions.

We always have to use maximum of collocated joins as this avoids the database manager having to ship data between partitions.

Figure 5-3 describes a environment with collocated join between CUST and ORDERS tables, which have been partitioned on cust_id column, as showed in , “Table collocation” on page 72.

However, when the REGION table is also involved in the join, then a collocated join is not possible, since the REGION table does not have a cust_id column, and therefore, cannot be partitioned by cust_id. DB2 UDB can choose to perform a directed join in this particular case, but the performance of such joins is less efficient than a collocated join, since the movement of rows is inline with query execution.

You can now use MQTs to replicate tables to other database partitions to enable collocated joins to occur even though the all tables are not joined on the partitioned key.

In Figure 5-3, REGION is replicated to the other partitions using the MQT infrastructure in order to enable collocated joins for superior performance.

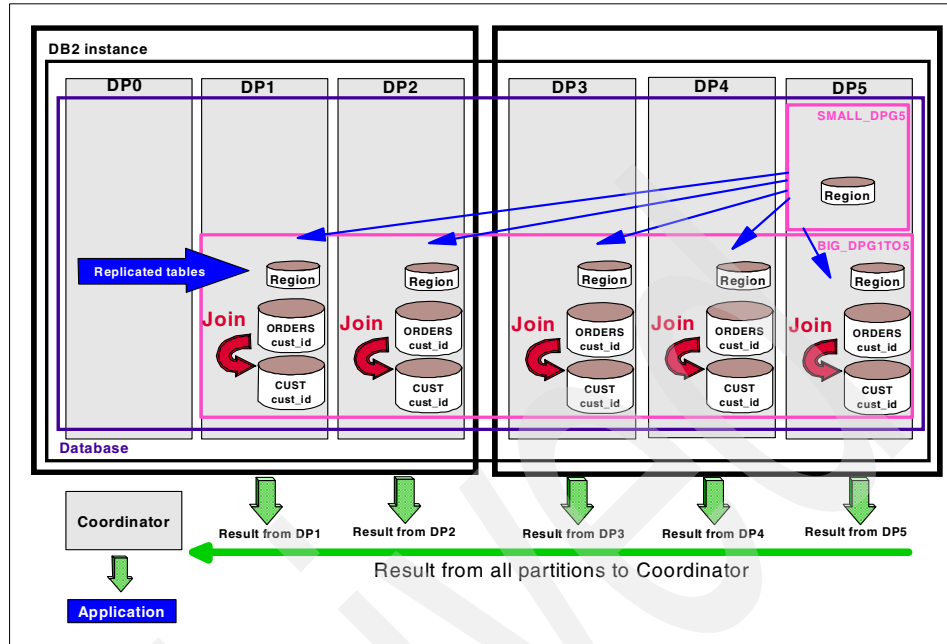


Figure 5-3 Replicated Materialized Query Table

Tip: Replicated MQTs may be ideal on a star schema configuration where dimension tables are not huge, and disk space is not an issue.

Example 5-3 shows how to create a replicated table. Table `tpcd.nation` is stored in a single-partition table space. In order to facilitate a colocated join with a table in multipartition table space `tbs1`, we create the replicated MQT `tpcd.nation_r`. With the *REPLICATED* clause we requested that it be duplicated on all the partitions in the partition group over which table space `tbs1` is defined.

Example 5-3 Create a replicated Materialized Query Table

```
create table tpcd.region_r as
  (select * from tpcd.region)
  data initially deferred refresh immediate
  replicated in tbs1
DB20000I The SQL command completed successfully.

refresh table tpcd.region_r
DB20000I The SQL command completed successfully.

refresh table tpcd.region_r incremental
DB20000I The SQL command completed successfully.
```

```
refresh table tpcd.region_r not incremental
DB20000I The SQL command completed successfully.
```

Note: If data in a database partition group contains replicated MQTs, the MQTs must be dropped before data redistribution.

5.5 MQT and MDC?

DB2 UDB V8.1 introduced the multidimensional clustering (MDC) table (see Chapter 4, “Speed up performance with MultiDimensional Clustering” on page 135).

There is a mutual relationship between MQT and MDC. An MQT can be defined on a MDC base table that means have base tables that are multidimensionally clustered. Furthermore, MQTs can themselves be multidimensionally clustered as MDCs as shown in Example 5-4.

Example 5-4 Create a MQT table multidimensionally clustered

```
create table MQT2 As
    (select c1, sum(c2) as sc2, count(c2) as cc2, count(*) as cs
     from t1 group by c1)
data initially deferred refresh immediate organize for dimensions (c1);
```

Enhance query performance in a partitioned environment

Data warehouse systems often involve answering complex ad-hoc queries containing aggregation, joins of multiple tables and a large amount of data. Efficient query processing in these systems is a critical requirement. In DB2 UDB ESE with DPF, one of the critical facilities for efficient query processing is the horizontal partitioning of the tables. This data distribution across several partitions allows for parallel processing of the query that helps reduce the overall query execution time.

There are many factors that affect query performance. In this chapter, we look at the role that partitioning plays in query performance. We also look at how DB2 accounts for partitioning when it plans the execution of the query. This information might be useful in establishing a good way to partition the tables, as well as to help analyze query performance, which is affected by the way tables are partitioned. The following aspects are considered:

- ▶ Query performance and the DB2 compiler
- ▶ Communication between partitions
- ▶ Join partitioning strategies
- ▶ Join planning using the join partitioning strategies
- ▶ Sort and aggregation parallelization
- ▶ Statistics in a partitioned environment
- ▶ Analyzing query plans in a partitioned environment

6.1 Query performance and the DB2 compiler

Complex queries, particularly those involving many tables, and those involving large tables, are significantly influenced by the DB2 optimizer. There are two components within the compiler: namely the query rewrite and the optimizer, which try to come up with an optimum access plan. Information about the data and the environment is used to arrive at the plan. Information used includes bufferpool, configuration, schema, partitioning, indexes, and table statistics among other factors.

The query rewrite component internally rewrites the query to better but semantically equivalent representations of the query. By and large the query rewrite component does not make too many decisions that are different between a partitioned and a non-partitioned environment. One rewrite which is very relevant to the partitioned environment, is one where queries containing correlated subqueries are decorrelated. What this means is that instead of executing the subquery by passing the values from the outer query block a row-at-a-time, we pass all the pertinent values in one block, and evaluate the subquery once before joining back to the main block. This significantly optimizes the communication overhead if these rows have to be sent across partitions.

The optimizer enumerates through the many base table access plans, and the various join methods and join orders to determine the optimum plan by comparing costs of various alternatives. In a partitioned environment, the optimizer also considers various join strategies, taking into account the partitioning of the various tables and intermediate query results. In addition, it considers optimizing other operations like those that sort and aggregate data. These will be discussed in subsequent sections. Before we do this, we will briefly discuss the basic communication mechanism between partitions.

6.2 Communication between partitions

Processing within the database is carried out using database agents, which are either processes or threads depending on the operating system platform. The coordinator subagent is the one that interfaces with the client process or thread. In a partitioned database environment, the coordinator agent is the main controlling agent that distributes database processing requests to other subagents. All these subagents, including the coordinator subagent, perform the various pieces of the access plan that the DB2 optimizer lays out. The mechanism used to transfer data from one agent or subagent to another is known as a *table queue*.

6.2.1 Table queues

From an external viewpoint, the table queue mechanism is the same whether you have shared nothing physical partitions (separate machines) or shared nothing logical partitions (on the same machine). The underlying mechanism is a little different as one might expect.

A similar table queue mechanism is used to transfer data between agents when we have the intra-partition facility turned on. Intra-partition parallelism works best with symmetric multi-processor (SMP) machines. Depending on your machine and application environment, you could have both inter-partition and intra-partition parallelism active. The portion of a query that is executing on a given partition can be further parallelized when more than one agent works on a given task.

Table queues pass data from one subagent to another. These are appropriately put into the access plan by the optimizer. There are different types of table queues that are introduced in the query access plan to handle specialized processing.

6.2.2 Table queue concepts

There can be *asynchronous* or *synchronous table queues*. Specifying **FOR FETCH ONLY** on the **SELECT** statement is recommended if the application is not interested in a row-at-a-time output from the database. If you do not specify this option, or if you are doing an **UPDATE**, synchronous table queues are used whereby, at each database partition, the cursor is positioned on the next row to be read from that database partition.

In terms the planning for how subagents are to be distributed and coordinated to execute the statement, table queues can assume different forms to transfer the data. There are *directed table queues* that send data to a specific partition. This could be based on hashing the value of the relevant partitioning keys, or could be a more specialized form based on predicates containing specialized functions for example, `NODENUMBER() = 5`. Another special form of a directed table queue is one that is specifically targeted to the coordinator subagent. Sometimes, when serialization is required in the plan, the optimizer might decide to direct data to a random partition so as not to penalize one partition that it is aware of.

Broadcast table queues are used to broadcast rows to several partitions defined by the target or destination partitioning.

There is also the concept of *merging table queues*, which try and preserve the order of the stream of rows at the receiving side by merging the locally ordered rows from the sending side. Merging table queues also has the option of eliminating duplicates if there is uniqueness required for a specific key. Merging

table queues *can be either merging directed table queues or merging broadcast table queues.*

DB2 UDB also has another concept of a *listener table queues* that are used to implement correlated subqueries. Here the correlation values are passed down to the subagent executing the subquery, and the result is passed back to the subagent driving the subquery.

When using the intra-partition parallelism feature, there are also *local table queues* that transfer data between the local agents that handle this form of parallelism. *Merging local table queues* are those table queues that merge sorted data from multiple agents in the intra-partition parallelism context.

6.2.3 Local bypass

There are some situations where DB2 UDB can detect that there is no need to use a table queue to transfer the result table data from one subagent to another within the same partition through a mechanism called local bypass. If processing is known to be limited to a single partition, for example, local equality predicates on partitioning columns, then, the execution is done completely by the coordinator without involving other subagents and a two-phase commit.

This is something that is relevant to ODS OLTP_like applications. In order to consciously exploit this optimization, applications need to be aware of the partitioning using functions like *get_table_partition ()* and *get_row_partition ()*. Using such functions, applications can connect directly to the partition where the data is processed. This may not be something to be overly concerned about in a data warehouse environment.

6.3 Join partitioning strategies

In Section 2.3.6, “Choose the partitioning key” on page 68, we discussed how to choose a good partitioning key in order to partition a table. We also discussed the concept of colocated joins that provide significant performance benefit through greater parallelism. While the aim must be to strive for collocation, this is not always achievable. Let us look at the partitioning issues in the context of the TPC-H schema described in Figure 1-13 on page 32.

The ORDERKEY in the LINEITEM table is a foreign key tied to the ORDERKEY column in the ORDERS table. Most queries joining these two tables would have the equality predicate `LINEITEM.ORDERKEY = ORDERS.ORDERKEY`. To achieve a colocated join between these two tables, it would be necessary to partitioning these tables on the ORDERKEY. Now the LINEITEM table is also tied up to the PARTSUPP table through the PARTKEY and the SUPPKEY.

Several queries also stipulate the join between these two tables. We have already decided to partition the LINEITEM table on the ORDERKEY. This implies that it would not be possible to get a collocated join between the LINEITEM and the PARTSUPP table. This is an issue with a typical star schema. At most we can partition one dimension table for a collocated join between it and the fact table. It is usual to pick the dimension table that is large and commonly joined in most queries. How, then, do we join tables that are not collocated?

Let us begin by looking at the various join strategies that are considered in the DB2 optimizer. Without considering partitioning, there are three main join methods that are considered the nested loop join, the merge scan join and the hash join. It is beyond the scope of this book to describe these join methods. For details of these joins methods, one can refer to the *DB2 Administration Guide: Implementation Version 8, SC09-4820*. Another access plan consideration is the join order. The join order deals with which tables are joined first and which side of the join the table or group of tables are placed. The DB2 optimizer enumerates the various join methods and orders and tries to find the optimum access plan by costing the various alternatives. A partitioned environment adds to the complexity of this process. The tables are partitioned in different ways and there are additional strategies that need to be considered to decide how best to deal with the fact that the data is distributed across the partitions and it may be necessary to move data from one partition to another. The aim is to minimize the cost of executing the query by accounting for data movement between partitions.

There are four join strategies that the optimizer considers to arrive at the execution plan in a partitioned environment:

- ▶ Collocated joins
- ▶ Directed joins
- ▶ Broadcast joins
- ▶ Repartitioned joins

6.3.1 Collocated joins

This join has already been discussed in Section 2.3.6, “Choose the partitioning key” on page 68 in the context of table collocation. Collocation is achieved when there are equality join predicates on the partitioning columns of the two tables that use the same partitioning map. These tables are distributed on the same set of partitions and matching rows reside on the same partitions. The output partitioning of the result of the join is the same as the partitioning of the inputs of the join. Note that the inputs to the join could be the result of previous joins.

To summarize, a collocated join is possible if:

- ▶ The inputs to the join have the same partitioning map and
- ▶ The inputs to the join have the same number of partitioning key columns and

- ▶ The corresponding partitioning key columns are partition compatible and
- ▶ The tables are joined by equijoin predicates on the corresponding partitioning key columns

In this join strategy, there is no data movement between the partitions to carry out the join.

6.3.2 Directed joins

Another join strategy that is considered is the directed join. This strategy is applicable when one of the tables has its partitioning key columns involved in the equijoin predicates that are used to join the tables. The join strategy involves directing the rows of the other table to the appropriate partition of the table that is partitioned on the same column used in the join predicate.

Consider, for example, the equijoin predicate `LINEITEM.ORDERKEY = ORDERS.ORDERKEY` and the `ORDERS` table is partitioned on the `ORDERKEY` column while the `LINEITEM` is partitioned on the `PARTKEY` and `SUPPKEY` columns. In this case, each row of the `LINEITEM` table is sent to one database partition of the `ORDERS` table decided by hashing the `LINEITEM.ORDERKEY` and using the partition map of the `ORDERS` table. The join of this `LINEITEM` row occurs on this database partition.

To summarize, a directed join between tables `T1` and `T2` is possible if:

- ▶ There is no collocated join possible and
- ▶ There is an equijoin predicate on all partitioning key columns of one table (say `T1`)

In this join strategy, the optimizer chooses a join where each row of the table `T2` is directed to one partition of `T1` by hashing `T2`'s join columns according to the partitioning of the columns of `T1` and directing the rows to the corresponding partition of `T1`.

6.3.3 Broadcast joins

This join strategy can be used if we do not have equijoin predicates. One of the inputs of the join is broadcast to all the partitions of the input on the other side of the join.

It can also be used with equijoin predicates on columns other than partitioning keys if it is found to be a cheaper join compared to the repartitioned join that is discussed in the next section. Typically, this join strategy is chosen if we have a small table joined to a very large table. In this case, the rows of the small table are broadcast to all the partitions of the big table. The join is carried out on each partition of the big table.

To summarize, a broadcast join is considered if

- ▶ No collocated or directed join possible and
- ▶ One of the tables is small or there are no equijoin predicates between the joined tables

In this case, the optimizer considers plans where each row of one table is broadcast to all partitions of the other table

6.3.4 Repartitioned joins

This join strategy competes with the broadcast joins when we have equijoin predicates but the columns differ from the partitioning keys of the inputs to the join. Here rows of both the inputs of the joins are directed to a set of partitions corresponding to some interesting partition map and based on the hashing of the join columns.

A repartitioned join is considered if

- ▶ No collocated or directed join possible and
- ▶ There is an equijoin predicate

In this join strategy, the optimizer considers a plan where each row of the inner table and the outer table is sent to a set of partitions based on a rehashing of the joining columns

6.4 Join planning using the join partitioning strategies

The optimizer looks for ways to cut down on the number of permutations and combinations of alternate plans that it considers. A query could involve tables with different partition characteristics. The optimizer keeps track of the interesting partitioning classes and when planning the query that it does bottom-up, it considers these interesting partitioning classes as it plans each join.

If the optimizer is to do an exhaustive study of the alternatives, the more the number of tables with different partitioning characteristics the greater the number of alternate plans. In order to lower the amount of time and space it uses to optimize queries in a partitioned environment, it applies certain heuristic rules.

The optimizer considers the following hierarchy during join planning:

- ▶ Collocated joins
- ▶ Directed joins
- ▶ Broadcast joins or repartitioned joins

As we have said many times before collocated joins are by far the best option to parallelize the joins with no movement of data from one partition to another. If this is possible between two inputs to the join, the optimizer simply ignores the rest of the join partitioning strategies. The next best choice is the directed join strategy. If this is possible, ignore the rest of the join strategies. If neither the collocated join nor the directed join is possible, then choose between a broadcast join and a repartitioned join based on the cost.

As an example consider the join between the CUSTOMER table that is partitioned on CUSTKEY with the NATION table that is on a single partition. Consider the join predicate CUSTOMER.NATIONKEY = NATION.NATIONKEY.

The optimizer establishes two interesting partition classes. The first is that of the NATION table and the second is that of the CUSTOMER table. During planning, the steps involved are to:

- ▶ Consider the partition of NATION
 - No collocation is possible (different set of partitions)
 - Direct CUSTOMER to NATION (single partition table NATION could be considered as partitioned on NATIONKEY)
 - Ignore other strategies
- ▶ Consider the partitioning of CUSTOMER
 - No collocation possible (different set of partitions)
 - No directed join possible (CUSTOMER is not partitioned on the join column NATIONKEY)
 - Broadcast NATION to CUSTOMER
 - Consider repartitioned joins (for example repartition CUSTOMER hashed on NATIONKEY and repartition NATION on the partitions of the CUSTOMER table also hashed on NATIONKEY)

The plan with the lowest estimated cost among the possible alternatives listed above is the one that is chosen as the optimum access plan.

6.4.1 Replicated table joins

Another important consideration is replication within the database. This is not to be confused with replication between databases. This facility, described in more detail in section 5.4, allows a copy of data from a table to be replicated in the partition of another table. If the optimizer considers the replicated table to answer the query, the join to this table could be considered to look like a broadcast join without the need to move the data from the original table that was replicated.

The optimizer may choose this plan if this is the cheaper alternative. It will cost plans using both the original table as well as the replicated table. This is

necessary since there could be circumstances where the indexes defined on the tables are so different that they affect the cost of the execution. You could imagine a situation for example, where you have no indexes on the replicated table and a convenient index on the original table. This index might be exploited to filter data based on a predicate on the query. The cost of accessing the replicated table may be so much more than that of broadcasting the filtered subset of the original table referenced in the query. It is therefore very important to define a good set of indexes on the replicated tables particularly if this is by and large a read-only table.

If the tables are large, it may be possible to replicate only a subset of frequently accessed columns and rows to cut down on the amount of disk space used.

Consider defining a replicated table if some tables are commonly joined and:

- ▶ Cannot be partitioned on same keys as other tables joined and
- ▶ Are modest in size (or the hot data can be filtered for replication) and
- ▶ Are typically read-only

6.5 Sort and aggregation parallelization

Other than joins, sorts and aggregations may also exploit the partitioned environment to achieve parallelism.

6.5.1 Sort parallelization

Consider a simple SQL statement where you are selecting the data from a huge table but require the output to be ordered on a column. Let us assume that you have no index on this column and the only option is to have the data sorted. There are two options. You may read all the data and send it to the coordinator agent through a normal table queue and then sort the data at the coordinator. As you might expect this sort may be huge and may require data to be split to disk in order to complete the sort. In addition all the processing is done by one agent and is therefore serialized.

Another option that is often chosen by the optimizer is to sort the data on each partition. This can be done in parallel within each partition and being on a smaller subset of the data in the entire table, DB2 UDB may not have to spill to disk and incur any I/O to carry out the sort. Once the data is ordered on each partition, it can then be merged at the coordinator using the merging table queue.

6.5.2 Aggregation parallelization

Aggregation is another operation that exploits parallelism in a partitioned environment. Similar to the sort optimization, there is a possibility to aggregate data at the partition level and further consolidate at the coordinator level. In terms of the level of aggregation, there are three forms of aggregation that DB2 UDB deals with:

- ▶ **Partial aggregation**

This is when aggregation is done while the data is being sorted. As soon as the *SORTHEAP* is full, the aggregation is discontinued.

- ▶ **Intermediate aggregation**

This is when the grouping columns do not cover the partitioning keys, then there might be rows across partitions belonging to the same group. In this case, we can do some amount of aggregation at the partition level and then have a final aggregation step at the coordinator. This form of partial aggregation is referred to as intermediate aggregation.

- ▶ **Final aggregation**

This is the final aggregation done with the full *GROUP BY* column set. It also completes a partial or intermediate aggregation phases. Note that we could have all three phases of aggregation active to complete one aggregation operation.

The DB2 optimizer costs the different ways of aggregating the data to arrive at the best plan.

6.6 Statistics in a partitioned environment

The DB2 optimizer uses statistics to estimate the size of the intermediate results and the costs in order to choose an optimum access plan. In a partitioned environment, the collection of statistics is done on a single partition. It is difficult to extrapolate some statistics to the table level. If the table is very small compared to the number of partitions, the statistics may not be reflective of the whole table. It could happen that there are no rows on the partition where the statistics are collected. Small tables, particularly those with significant skew in the partition key values, may give the optimizer better information to use during planning if the tables are placed in a single partition.

6.7 Analyzing query plans in a partitioned environment

This section looks at characteristics of a plan that are relevant to a partitioned environment. In order to explain some of the concepts explained in the sections above, an example based on the TPC-H schema is shown in Example 6-1. This query has a join of six tables with an aggregation.

The `db2exfmt` tool output that formats the information from the `EXPLAIN` tables is shown in Example 6-1 too. The numbers in parenthesis below the operator is the operator number. The numbers above the operators are the cardinalities relevant to the operation.

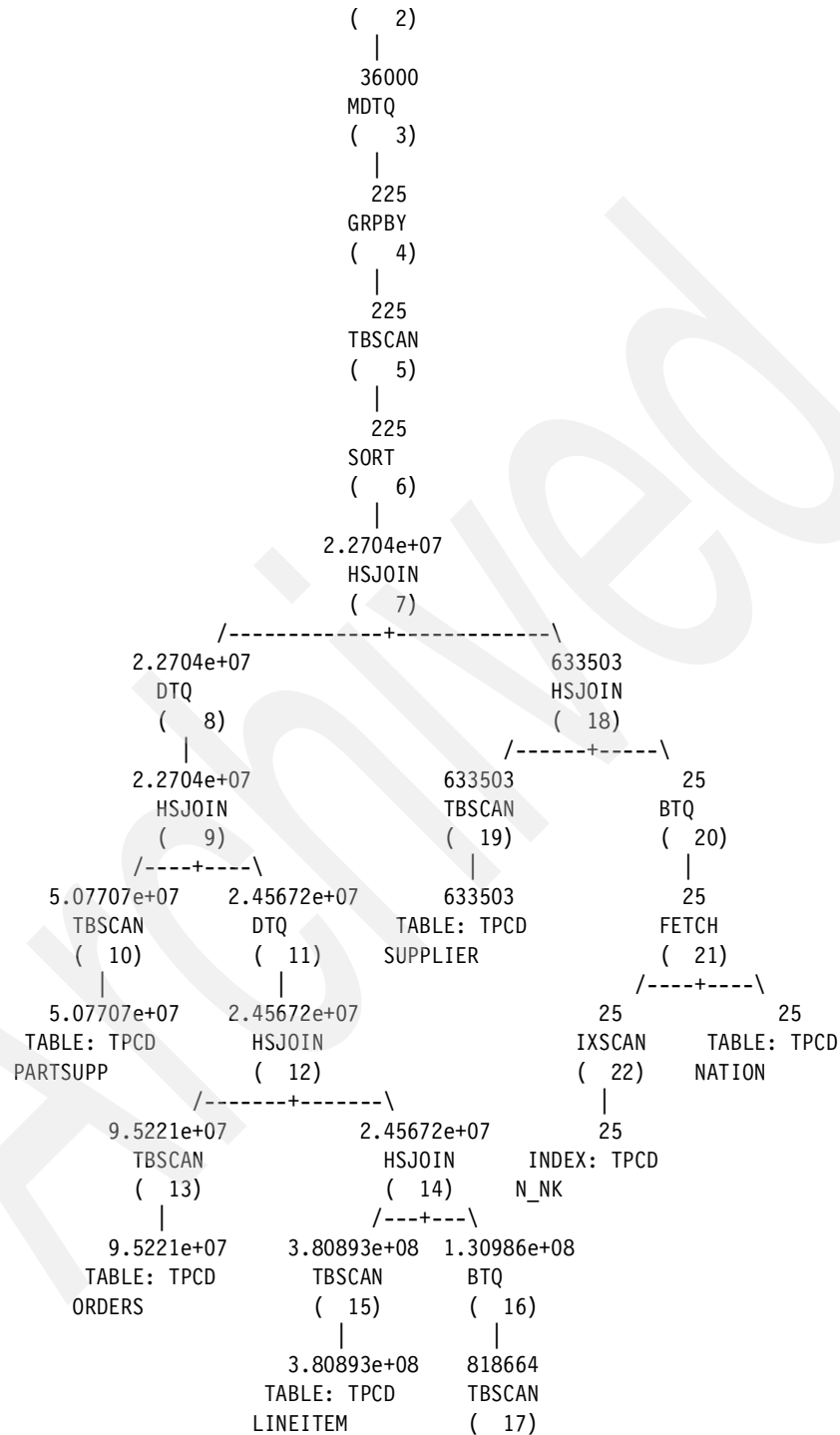
Example 6-1 Analyzing query access plan

```
select nation,
       o_year,
       sum(amount) as sum_profit
from (select n_name as nation,
            year(o_orderdate) as o_year,
            l_extendedprice * (1 - l_discount) - ps_supplycost * l_quantity as
amount
      from tpcd.part,
           tpcd.supplier,
           tpcd.lineitem,
           tpcd.partsupp,
           tpcd.orders,
           tpcd.nation
     where s_suppkey = l_suppkey and
           ps_suppkey = l_suppkey and
           ps_partkey = l_partkey and
           p_partkey = l_partkey and
           o_orderkey = l_orderkey and
           s_nationkey = n_nationkey and
           p_name like '%coral%' ) as profit
group by nation, o_year
order by nation, o_year desc
```

Access Plan:

Total Cost: 2.99008e+07
Query Degree:1

Rows
RETURN
(1)
|
225
GRPBY



|
1.26927e+07
TABLE: TPCD
PART

The two biggest tables LINEITEM and ORDERS are partitioned on ORDERKEY. The query joins these two tables on the ORDERKEY. The join (HSJOIN(12)) is collocated and this is probably a good thing considering the size of data being joined.

The PART table is partitioned on PARTKEY but the LINEITEM is partitioned on ORDERKEY. Even though the LINEITEM could have been directed to the PART table partitions, the optimizer would rather not choose to move the rows of the LINEITEM table. Instead, after filtering some of the PART table rows, the optimizer chooses to broadcast (BTQ(16)) the PART table to be joined to the LINEITEM table. This join is done before the join to the ORDERS tables since it helps reduce the size of the LINEITEM table. Note that the partitioning of the result of this join is the same as that of the LINEITEM table. Note also that the cardinality has been increased from 818664 to 1.30986a+08 after broadcasting the rows to the 160 partitions of the LINEITEM table.

The join between PARTSUPP and LINITEM is through the columns PARTKEY and SUPPKEY. These columns are also the partitioning keys by which the PARTSUPP table is partitioned. The optimizer chooses a directed table queue (DTQ(11)) to push each of the result rows containing the LINEITEM data to the corresponding partitions of the PARTSUPP table.

The NATION table is joined to the SUPPLIER table on the NATIONKEY. Since the SUPPLIER is partitioned on the SUPPKEY, this join (HSJOIN(18)) uses a broadcast table queue (BTQ(20)). The resulting partitioning of this join is still the same as the SUPPLIER was partitioned with SUPPKEY as the partitioning column.

The final join (HSJOIN(7)) is chosen with a directed table queue (DTQ(8)) to send each rows of the result of the join between LINITEM, PART, ORDERS and PARTSUPP to the corresponding partitions of the result of SUPPLIER and NATION. This is because the join predicate is on the SUPPKEY.

Finally at the top of the plan, after all the joins, we have a partial sort on the N_NATION, O_YEAR columns on each partition. This also helps the intermediate aggregation (GRPBY(4)) that collapses the size of the result on each partition. The final aggregation is done through the GRPBY(2) operator. The order required by the query is maintained by merging the rows from each partition through the merging directed table queue (MDTQ(3)) sent to the coordinator partition before the result is returned to the user.

While it may seem a daunting task to analyze these plans with more complex queries, it is hoped that this example helps gain an insight into how DB2 UDB lays out the access plan in a partitioned environment.

Improve 24x7 operations with new online utilities

This chapter focuses on the enhancements in DB2 UDB ESE V8.1, which contribute to smaller and fewer planned outages, thus allowing the database to come closer to having a true planned 24x7 availability. Many of these enhancements allow the database administrators to perform database maintenance tasks online, which previously required an outage, and which also will contribute to fewer unplanned outages.

Companies utilizing e-Business Intelligence are enabling access to the data to more and more types of users, and are showcasing these features in national and even international advertising campaigns. These new database users are often the companies customers, such as a person tracking a shipment, placing an on-line order, or submitting an insurance claim. This type of business requirement requires reducing or eliminating many of the planned change windows or outages that historically have been done on weekends. Even though these types of users may not be the main users of the data, they can contribute to the companies image and success in the market place. High-level management is placing more and more emphasis on reducing planned outages to an absolute minimum, and they would prefer to completely eliminate them.

7.1 Availability

This section covers the major enhancements in DB2 UDB ESE V8.1, which will provide the tools needed to increase the availability of environments with large, multi-partitioned databases. Highlights of DB2 UDB features that enable it to provide high availability are shown in Figure 7-1. The topics focused on are:

- ▶ Load improvements - Loading into multi-partition databases is easier, and now allows some concurrent access to the data in both the table and table space during the load.
- ▶ Table reorganization improvements - The ability to reorganize a table online with read and write access throughout the process is sure to improve the availability of your database and reduce the number of offline reorganizations that need to be run during planned outages.
- ▶ Indexes - A new type of index - type-2, is now supported. This will improve concurrency since next-key locking is reduced.
- ▶ Online updates are now allowed for the bufferpool, and for additional database manager and database configuration parameters. This online change capability will increase the availability of your system by requiring fewer planned outages to implement configuration parameter changes. It will also improve the DBAs ability to implement performance changes into production in a timely fashion, and improves the ability to prevent unplanned outages.
- ▶ A sound backup and restore strategy can improve both the availability and the performance of the system. Enhancements such as faster table space recovery and support of XBSA industry standard interfaces are now available, and are reviewed with a focus on how to incorporate them into your backup and recovery plans. Proper planning and preparation can reduce any unplanned outages that may be needed to recover the database.
- ▶ Improvements to DMS container operations and an online **INSPECT** command provide additional ways to perform database maintenance online that previously required an outage.

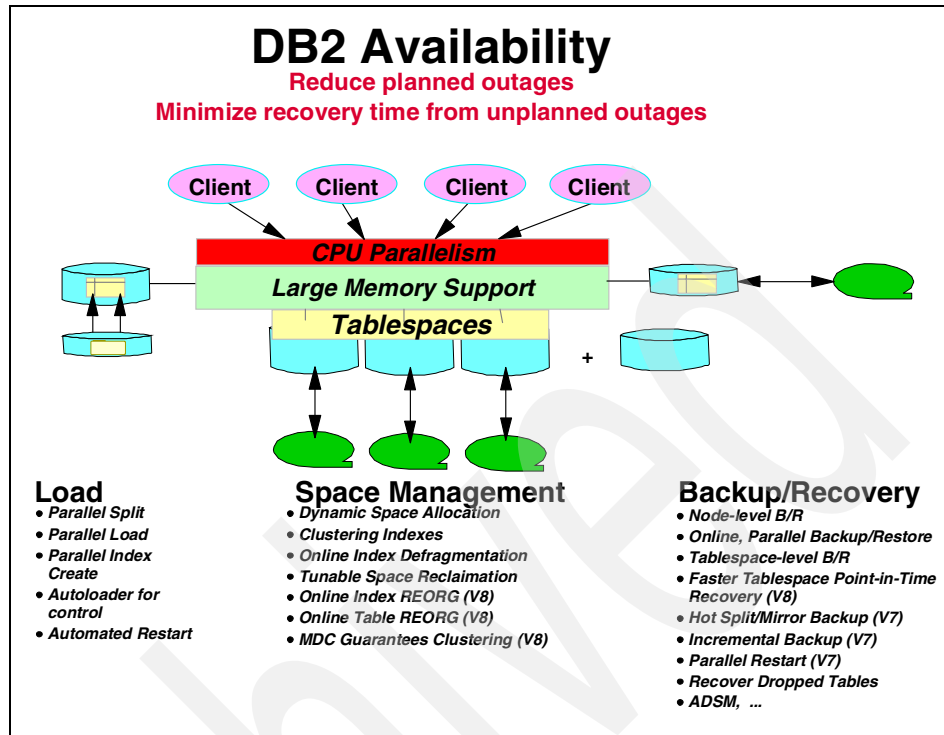


Figure 7-1 DB2 UDB high availability features

7.2 Online table load

The good news for warehouse environments with large DB2 UDB databases is that with DB2 UDB ESE V8.1 it is simpler than ever before to load data into multipartition databases and have access to it.

7.2.1 Improved methods of loading

Loading data into a multi-partitioned database is now done the same way as loading data into a single partitioned database.

The autoloader utility is no longer needed. All LOAD functions used for loading into single partition databases can now be used for loading into multi-partition databases. These include the commands (**LOAD**, **db2load**) and APIs (**db2load**, **db2LoadQuery**). This compatibility of commands also makes a transition from a single partitioned database to a multi-partitioned database more streamlined

since many of the processes used to load to the single partition database may now continue to be used for the multi-partition database, with little alteration.

If your DB2 UDB environment requires loading data into both single and multi-partition databases, this compatibility between the load processes will reduce the maintenance required to support both types of environments, and enables the processes to be more easily standardized across the environment by not having to maintain two separate sets of processes and documentation.

An additional improvement to the ability to load data is you can now use SQL to select data from one table and load it into another table without having to export it first. This is done by using a new file type name *CURSOR*. By being able to bypass the export step you do not need to have as much disk space available to hold the intermediate results, and the entire operation may be completed in one step. This operation is demonstrated in 7.2.4 “Load wizard utility” on page 195.

7.2.2 Improved table space and table access while loading

Improved access to the data objects equates to higher availability for your applications.

Not only is it easier to get the data into the warehouse, but since the lock on the table space has been removed, applications can now have full access to tables co-existing in the table space with the table that is being loaded. Additional new features such as the ability to load the data online will enable your warehouse to reach a higher level of availability than ever before. In fact, your applications can even have read access to the pre-existing data in the table that you are actually loading to.

This is true not only for normal tables, but also for materialized query tables (MQT) and automatic summary tables (AST), which are a specific type of MQT. Previously, the data in both MQTs and ASTs was unavailable during a load on the parent table, and were completely rebuilt after the load was done, which can seriously impact the overall availability of the database. Now, the data in the MQTs and ASTs can remain available for access during a load operation on its parent table. After the load is complete on the parent table, both MQTs and ASTs no longer require complete rebuilds, but can be updated incrementally. The data is not available during this maintenance phase, but the time to perform it is significantly reduced from the previous requirement of having to rebuild them in their entirety. Overall database availability is greatly enhanced by having access to the data up to the starting point of the maintenance activity, and the greatly shortened maintenance activity.

7.2.3 Why, how, and when these features can be used

For how to use and take advantage of these new features, please refer to Chapter 3, “LOAD and populate the data warehouse in parallel” on page 111; 3.1.1, “Online LOAD” on page 112; 3.1.4, “Multipartition LOAD support” on page 119; and 3.1.5, “Partitioning LOAD subagents in DB2 UDB V8.1” on page 126.

7.2.4 Load wizard utility

A load wizard utility is now provided with DB2 UDB ESE V8.1. This wizard will assist and guide you in performing loads to both single and multipartitioned databases, even if you have never performed a load before.

This section will document the capabilities of the wizard and the steps to use it are demonstrated using a sample business decision that may be similar to one you have experienced in your environment.

For example, to boost sales, the marketing sales director has decided it that all promotional material that is sent to the customers (every row in the customer table) should also be sent to the suppliers to encourage them to also become customers. Since there is additional marketing data kept in the customer table, marketing has requested that the columns supplier key, name, address, nation key, and phone number from the supplier table be loaded into the customer. The program that inserts the information for new suppliers will be updated to also insert the data into the customer table at the same time as the supplier is added, so this should be a one time load operation.

The load wizard is launched from the Control Center. There are several ways to do this; one is illustrated in Figure 7-2. All screens used to perform this task are shown to enable you to see all of the options that are available through the wizard.

1. Launch the **Control Center** ->**Connect to database**.
2. Select the table you wish to load and right-click.
3. Select load. This will launch the **Load Wizard**.

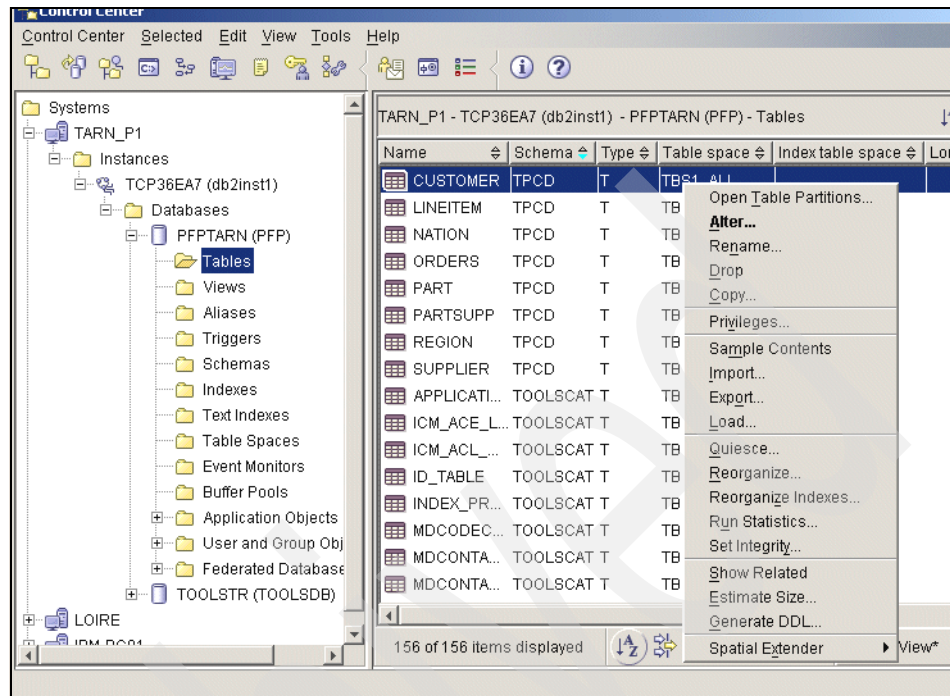


Figure 7-2 Selection of tables to be loaded

The first window contains options for the load operations that can be performed. For this example, the **Split** and **Load** data is selected as shown in Figure 7-3.

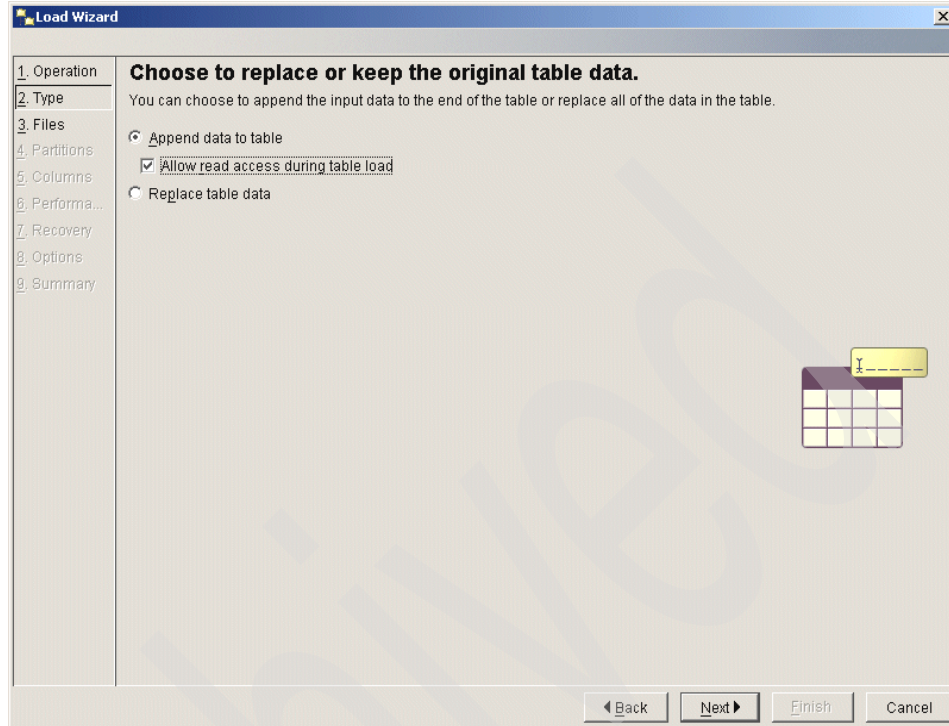


Figure 7-4 Type of load to be performed

The third window of the Load wizard is where the type of input file to be used is selected and the location of any output or messages files is specified. The input file choices are Positional Text (ASC), Delimited Text (DEL), or Load from Cursor. If either the Positional Text or Delimited Text option is chosen, Input fields for the Input file location and several fields to enter the full path and file name for various output files that will be generated by these options. For this example, we want perform a load from CURSOR, so that is the option selected, as displayed in Figure 7-5. The Load from Cursor option has the option to launch the SQL Assist wizard, which we will use to generate the SQL statement to select the data from the supplier table. One of the screens from the SQL Assist wizard is show in Figure 7-6. The full path and file name for any messages generated by the load has also been entered.

The image shows a 'Load Wizard' dialog box with a sidebar on the left containing steps 1 through 10. Step 3, 'Files', is selected and highlighted. The main area is titled 'Specify input and output files.' and contains the following elements:

- A paragraph: 'Most load operations will have at least one input or output file. Other minor file specifications can be found on the 'Options' page.'
- An 'Input file format' section with a dropdown menu set to 'Load from Cursor' and a 'No Options' button.
- A paragraph: 'Manually compose a SELECT statement, or use SQL Assist. The result set of the SELECT statement will be used as input data for this load operation.'
- A text box containing the SQL query:

```
SELECT SUPPLIER.S_SUPPKEY, SUPPLIER.S_NAME, SUPPLIER.S_ADDRESS, SUPPLIER.S_NATIONKEY, SUPPLIER.S_PHONE  
FROM TPCD.SUPPLIER AS SUPPLIER
```
- An 'SQL Assist...' button.
- A label: 'Full path and filename to store progress messages:'
- A text box containing the path: `/db2work/customer_load/load_cust.msg`
- Navigation buttons at the bottom: 'Back', 'Next', 'Finish', and 'Cancel'.

Figure 7-5 Input and output file specifications

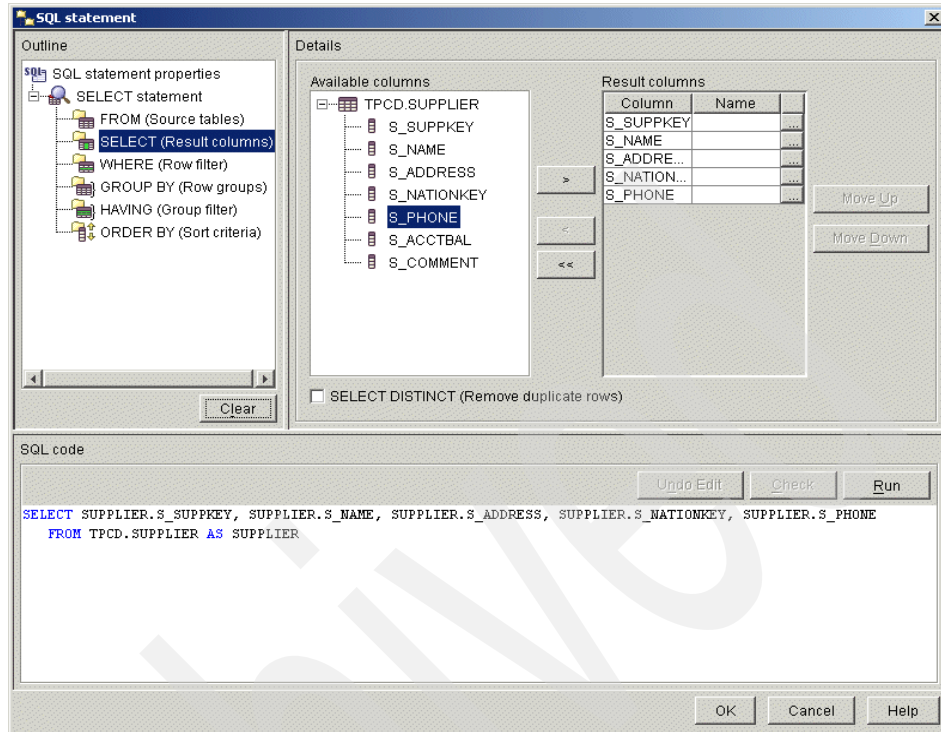


Figure 7-6 An example of the SQL Assist wizard

The next window shows where which partitions that should participate in the load is determined. By default, DB2 UDB will use all partitions and determine where the data will be loaded. For our example, the defaults were chosen. See Figure 7-7 for the screen image.

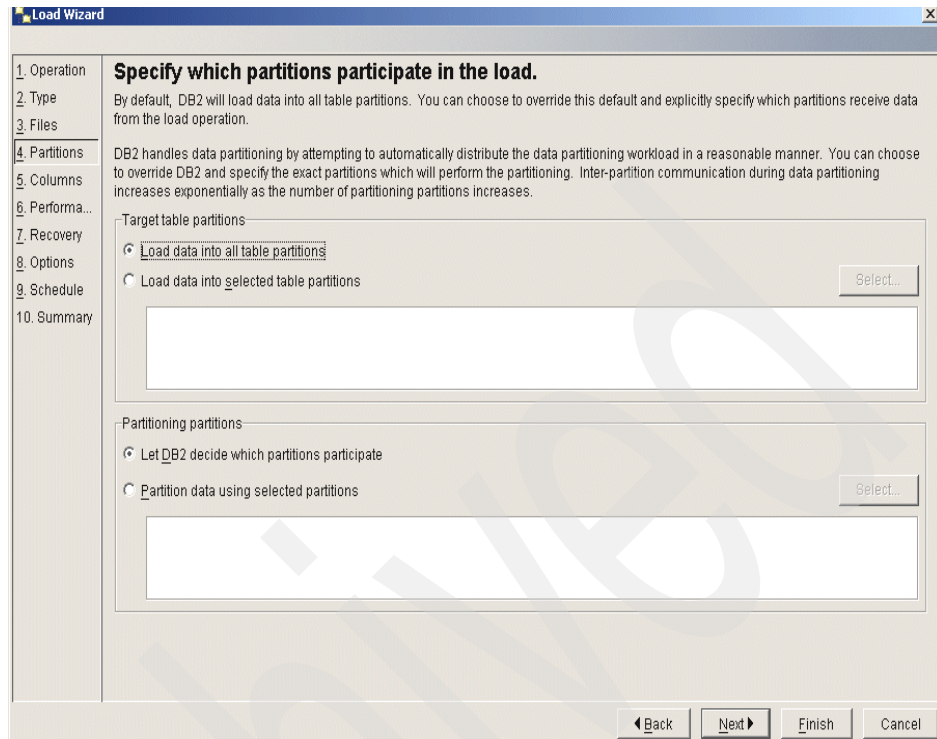


Figure 7-7 Partitions to participate in the load

The next window in the load wizard assists you in mapping the input columns from the source table to their mapping to the output columns in the target table. In our case, it is a simple one-to-one match as displayed in Figure 7-8.

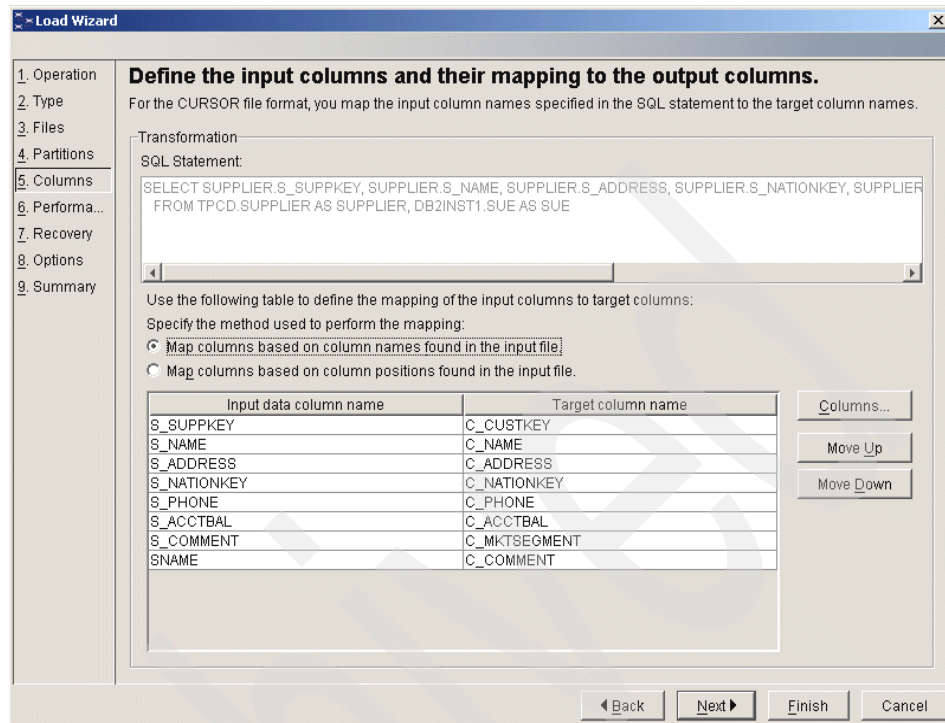


Figure 7-8 Definition of the mapping of input columns to output columns

Selection of the performance and statistics collections options is done on the sixth screen of the load wizard. Since in our case we loaded around 600 thousand rows into a table with 9 million rows already in it, the option to let the load utility update existing indexes is selected. DB2 UDB can update the indexes incrementally, or rebuild all of the indexes. For this load, the option to let DB2 UDB make the decision between the two options is selected. Figure 7-9 shows this window.

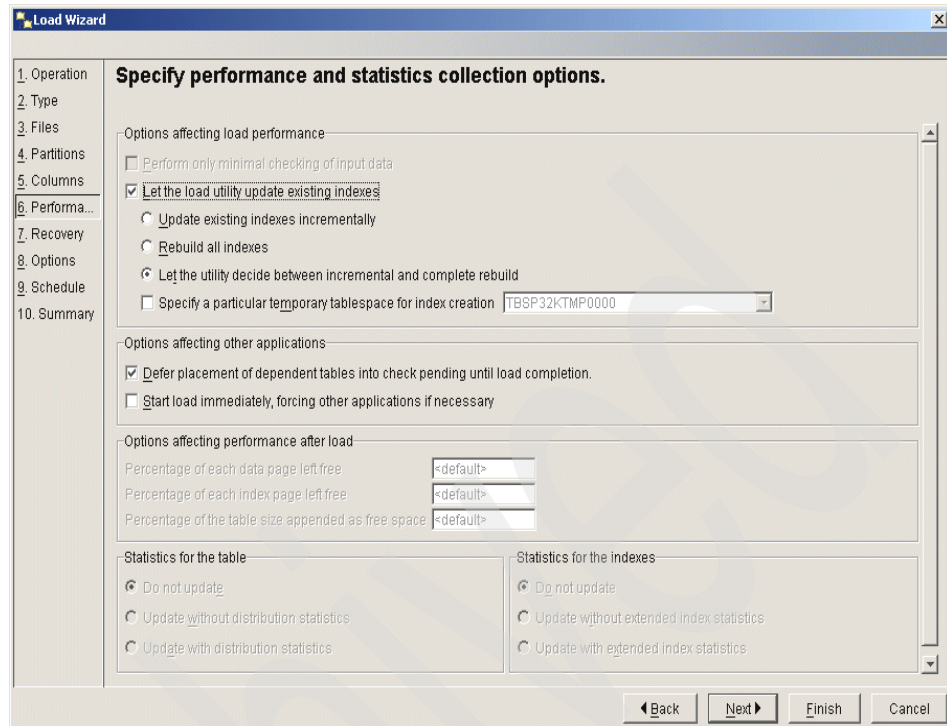


Figure 7-9 Selection of performance and statistics options

Failure options and recovery strategy options are selected on the next window of the load wizard. For this example, we would like to leave the table in a usable state at the end of the load, instead of in backup pending. Since we are only loading about 500,000 rows, the space to keep a copy of the input data is not significant. These selections are shown in Figure 7-10.

Load Wizard

1. Operation
2. Type
3. Files
4. Partitions
5. Columns
6. Performa...
7. Recovery
8. Options
9. Schedule
10. Summary

Select failure options and recovery strategy.

Use this page to specify failure and recovery options for a load failure. You can also specify the target table is configured for forward recovery.

Crash recovery

☐ Establish consistency points and generate row count messages during data load

Number of rows between consistency points:

☒ Load job fails if any partition encounters errors in the setup phase

☐ Load will rollback immediately on all partitions if any fail after setup is complete

Forward recovery

The following options will be valid only if the database is configured for forward recovery. If you perform an unrecoverable load, a subsequent roll forward through this load will succeed.

☒ Perform a recoverable load

☐ Do not make a copy of the input data. This will put the tablespace into backup pending state.

☒ Save a copy of the input data. The table will be usable after load completion.

Copy target:

Directory names:

Figure 7-10 Failure options and recovery strategy

The advanced option selection page is next. Since our example is a simple, non-complex one, no options on this screen have been selected for our case. A picture of the screen is in Figure 7-11 to illustrate what type of advanced options are available.

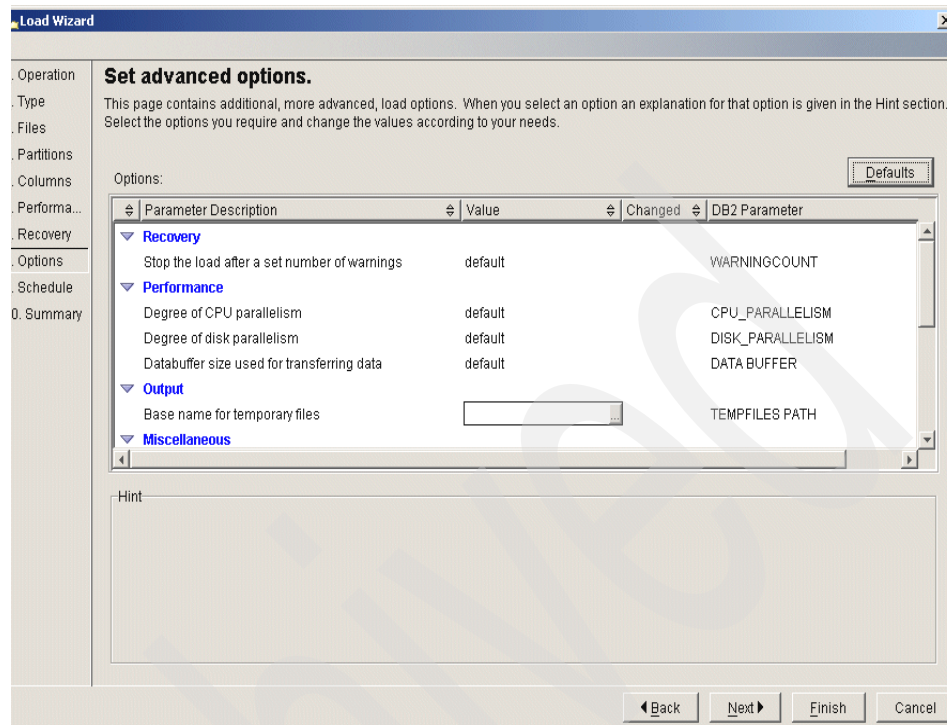


Figure 7-11 Advanced options window

We are now ready to either run the load task immediately or to schedule it in the Task Center. Even though this should be a one-time load, the option to create this load as a task in the Task Center, and to save it only at this time, are chosen. This gives us the flexibility to decide at a later time when to schedule the load and by saving the task, it provides us with the potential for re-use as a pattern if a similar type of task needs to be done.

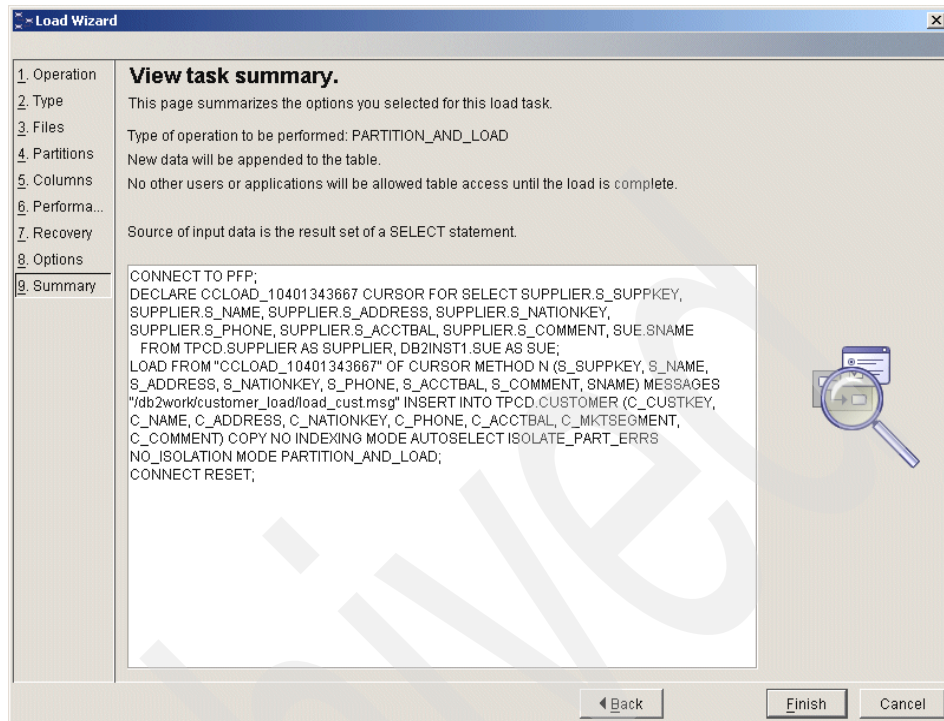


Figure 7-12 Scheduling the execution of the load

The final panel of the load wizard provides a summary of the task creation as displayed in Figure 7-13.

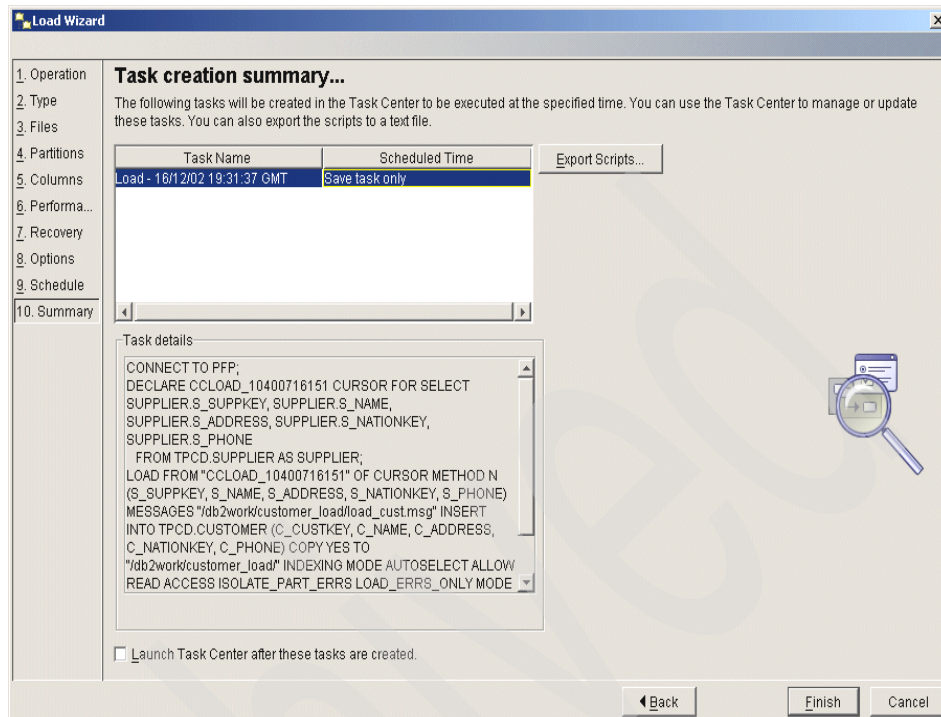


Figure 7-13 Task creation summary

At this point, if review of new processes are required in your environment, the review could be done. If you wish to make any changes to the load options:

Start at the **Task Center** and select **Load Tasks** and then select **Edit with dialog**.

This will cause the load wizard to be launched and populated with the data from the task. You may then select any of the load wizard screens where you wish to make changes.

When you are ready to run the load, simply go to the Task Center. It may be used to either run the load immediately, select the **Run now** choice, or to schedule the load to be done at some time in the future select the **Schedule** option.

The screen where all three of these actions, **Edit with dialog**, **Run now**, and **Schedule**, may be selected is shown in Figure 7-14. After the load task has run, an additional selection called **Show Results** will be displayed, which you can select to view the results of the task.

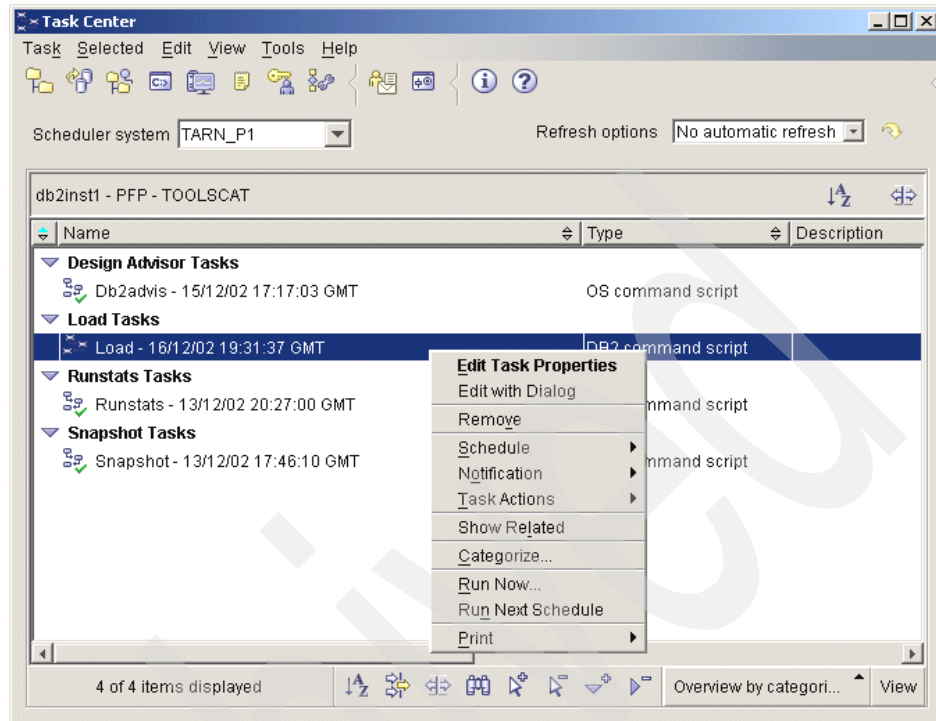


Figure 7-14 Task Center showing load task just created

7.3 Online and classic reorganization of tables

The **REORG** command available in previous versions of DB2 UDB has been so greatly expanded in DB2 UDB ESE V8.1, that it is now referred to as the *classic REORG*. Environments with large data warehouses and databases that currently run classic offline reorganizations will improve the availability of their databases by reducing the need for planned outages by implementing the use of the new **INPLACE**, **DBPARTITIONNUM**, and **INDEXES ALL FOR TABLE** options of the **REORG** command. Organizations that have reduced the frequency of **REORGs**, or avoided offline **REORGs** because of the lack of maintenance or outage windows, also will benefit from these same options. These options are described in detail in the following sections.

7.3.1 Why reorganize the data in a table?

The two main reasons that a **REORG** is done is to reclaim disk space and to physically reorganize the data on the disk to improve performance.

If the primary reason for considering reorganizations in your environment is to be able to use the space freed up by the **REORG**, then a clustering index should not be used on the table. This is because a clustering index, by definition, will be used to determine the physical placement of the data, and will not compact the data pages to the same degree as done without a clustering index.

7.3.2 How does an online reorganization benefit my environment?

Online table reorganization has been implemented through the **REORG TABLE INPLACE** option. It will increase the availability of your environment, because applications that require data in the table will be able to run during the **REORG**, since the table is available for both read and write activity. SQL statements going against the table do not even have to wait until the **REORG** is complete to benefit from the performance improvements gained from the **REORG** activity, since results can be seen incrementally as the reorganization is performed, and pages are consolidated, as illustrated in Figure 7-15.

The **REORG TABLE INPLACE** may also reduce the amount of replace by disks required to perform the **REORG**. The classic **REORG** requires enough work space for a second complete copy of the table, either in the table's existing table space, or in a temporary table space. With the **INPLACE** option, rows are reconstructed within the existing table object, thus minimizing the extra amount of space required. However, because of the inplace nature of the reorganization, the amount of space allocated to the table object's table space should be re-evaluated before performing an online reorganization to determine if additional space is required.

If business needs require write access to the table throughout the reorganization process, specify the **NOTRUNCATE TABLE** option when reorganizing the table, as shown in Example 7-1. This is because by default in order to reclaim space, the in-place reorganization truncates the table as the last phase of the reorganization. During this phase, the table is S-locked, and not able to be written to.

Example 7-1 REORG with nottruncate option

```
db2 reorg table customer inplace nottruncate table
```

Please note that the in-place reorganization is performed only on the table data objects, and not on the index, long fields, or LOB objects.

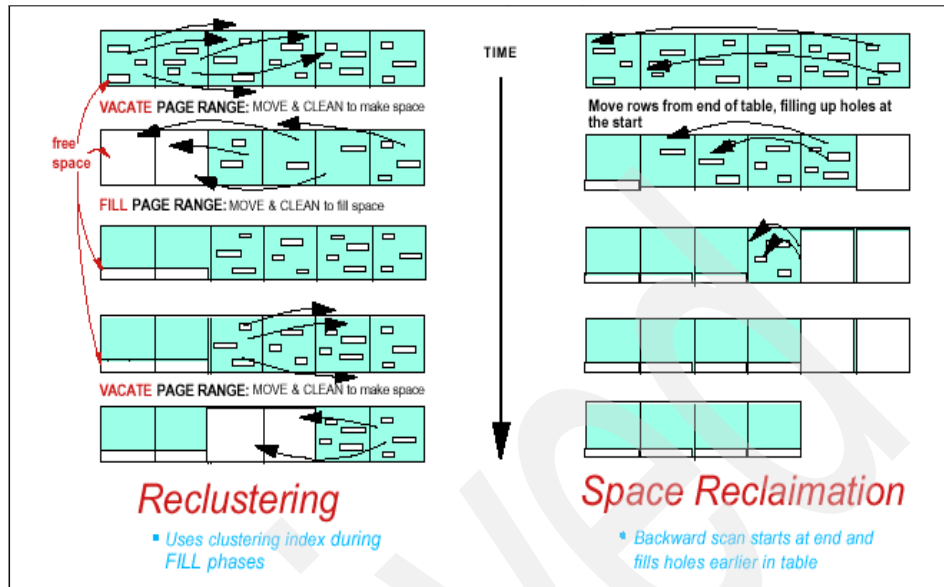


Figure 7-15 Online reorganization flow

Note: Online table reorganizations may only be done on tables with type-2 indexes and without extended indexes. For details on type-2 indexes, please refer to 7.4, "Index enhancements" on page 216.

7.3.3 What benefits will REORG by database partition provide?

Both the **REORG INPLACE** and the classic **REORG** can use the **DBPARTITIONNUM** option, which allows more control on how and when the database resources required to perform the **REORG** will be used.

Separating the **REORG** by database partition can be used to control the CPU usage and I/O activity by spreading the resources used out across both time and I/O devices, allowing more control on how and when those resources are used. This can be used to improve the scheduling of the **REORG**, or to remove any performance impact on the production environment that may have been caused by running the **REORG** on all database partitions concurrently.

For example, if the database is partitioned by national or international geographical area, running the **REORG** by database partition would provide the ability to schedule by time zone, even further decreasing any impact on production performance. The ability to pause, resume, or stop a **REORG TABLE** by database partition at any point has also been added. In addition, if a **REORG** fails,

only the failed partition will be rolled back. Please note that the classic offline **REORG** can also take advantage of all database partition options.

7.3.4 When should the indexes on a table be reorganized?

The **REORG INDEXES ALL FOR TABLE** should be planned after a **REORG TABLE INPLACE** has been run. It is not needed after a classic **REORG**, because the indexes are rebuilt as the last phase of the **REORG**.

A complete discussion of this option is presented in 7.4.4, “Online index reorganization” on page 219.

7.3.5 Is there a way to monitor the REORG?

Yes, the progress of the **REORG** may be monitored by table snapshots. The monitoring data for a table reorganization is always collected, regardless of the Database Monitor Switch value. The **GET SNAPSHOT FOR TABLES** provides status on the progress of the **REORG** and the information from the snapshot may be used to answer questions that are often asked such as:

“How long until the **REORG** will be finished?”

or

“What percentage of the table has already been reorganized?”

A **REORG** of the lineitem table was started with the command in Example 7-2.

Example 7-2 Sample of starting an online reorganization

```
DB2 REORG TABLE TPCD.LINEITEM INDEX SYSIBM.SQL021204184342620 INPLACE ALLOW  
WRITE ACCESS START
```

The output shown in Figure 7-16 has **REORG** information the table line item for partition 1. The global snapshot provides this information for each partition. It was obtained by executing **db2 get snapshot for tables on pfp global**

Using this data, we know that the **REORG** started at 15:15:25, is a little over 10% done on partition 1, that it is being performed **INPLACE**, and **ALLOW WRITE ACCESS** is in effect. Its status is started and the end time does not have a value.

```

Table Schema      = TPCD
Table Name       = LINEITEM
Table Type       = User
Rows Read        = 1800057970
Rows Written     = 0
Overflows        = 0
Page Reorgs      = 3508448
Table Reorg Information:
Node number      = 1
  Reorg Type     =
    Reclustering
    Inplace Table Reorg
    Allow Write Access
  Reorg Index    = 1
  Reorg Tablespace = 3
  Start Time     = 12-17-2002 15:15:25.437097
  Reorg Phase    =
  Max Phase      =
  Phase Start Time =
  Status         = Started
  Current Counter = 10281
  Max Counter    = 95120
  Completion     = 0
  End Time       =

```

Figure 7-16 REORG status from a table snapshot

7.3.6 Is there a way to control the REORG?

Businesses are often anxious about allowing any additional applications or processes on their system due to concerns about the potential impact the additional workload may have on their production performance statics and service level agreements. This type of issue is often faced by DBAs when they propose that required maintenance tasks such as reorganization be done at a different time, or when the length of time that a reorganization takes starts to exceed a maintenance widow. The ability to pause, resume, and even completely stop the reorganization at any time will be of great assistance when addressing these types of business concerns. These options will also be useful in simplifying and quantifying problem determination issues regarding performance degradation by being able to immediately remove the reorganization process from the mix of running applications.

To **PAUSE** the **REORG** that was started by the statement in Example 7-2 on page 211, executes the command shown in Example 7-3.

Example 7-3 Sample of pausing an online reorganization

```
DB2 REORG TABLE TPCD.LINEITEM INDEX SYSIBM.SQLO21204184342620 INPLACE PAUSE
```

The command **db2 get snapshot for tables on pfp global** is executed again. The output produce pertaining to partition 1 is shown in Figure 7-17. The status is now showing as **PAUSED**, and 23085 rows were processed before the **REORG** was **PAUSED**. Notice that the end time now has a value of 15:15:55.

```

Table Schema      = TPCD
Table Name       = LINEITEM
Table Type       = User
Rows Read        = 1800057970
Rows Written     = 0
Overflows       = 0
Page Reorgs     = 8073601
Table Reorg Information:
Node number      = 1
  Reorg Type     =
    Reclustering
    Inplace Table Reorg
    Allow Write Access
  Reorg Index    = 1
  Reorg Tablespace = 3
  Start Time     = 12-17-2002 15:15:25.437097
  Reorg Phase    =
  Max Phase      =
  Phase Start Time =
  Status         = Paused
  Current Counter = 23085
  Max Counter    = 95331
  Completion     = 0
  End Time       = 12-17-2002 15:55:55.079464

```

Figure 7-17 Table snapshot showing that the REORG status is Paused

To **RESUME** the **REORG** that was **PAUSED**, the statement in Example 7-3 executes the command shown in Example 7-4.

Example 7-4 Sample of resuming an online reorganization

```

DB2 REORG TABLE TPCD.LINEITEM INDEX SYSIBM.SQLO21204184342620 INPLACE ALLOW
WRITE ACCESS RESUME

```

The command **db2 get snapshot for tables on pfp global** is executed again. The output produce pertaining to partition 1 is shown in Figure 7-18. The status is once again showing as Started, and the value for current counter has increased. However, please note that the Start Time is now 15:57:14, which is not the original start time of the **REORG**, but the time when the action **RESUME** was done, and once again End Time has no value.

Fortunately DB2 UDB keeps track of which type of **REORG** command was executed and when. The history file has an entry for every **REORG** command that is performed with a timestamp of when it was executed. These entries may be seen by executing the **DB2 LIST HISTORY REORG** command for the database and partitions in which the table resides.

The output from **db2 list history reorg since 20021217 for pfp** command is shown in Figure 7-18. The entries for the **REORG START**, **REORG PAUSE**, **REORG RESUME** and **REORG Done** for the table line item are all clearly labeled.

```

Table Schema      = TPCD
Table Name       = LINEITEM
Table Type       = User
Rows Read        = 1800057970
Rows Written     = 0
Overflows        = 0
Page Reorgs      = 8412657
Table Reorg Information:
Node number      = 1
  Reorg Type     =
    Reclustering
    Inplace Table Reorg
    Allow Write Access
  Reorg Index    = 1
  Reorg Tablespace = 3
  Start Time     = 12-17-2002 15:57:14.913111
  Reorg Phase    =
  Max Phase      =
  Phase Start Time =
  Status         = Started
  Current Counter = 23880
  Max Counter    = 95331
  Completion     = 0
  End Time       =

```

Figure 7-18 Table snapshot showing that the REORG status is again started

Op	Obj	Timestamp+Sequence	Type	Dev	Earliest Log	Current Log	Backup ID
G	T	20021217151525	N			S0000025.LOG	
Table: "TPCD" ".LINEITEM"							
Comment: REORG START Start Time: 20021217151525 End Time: 20021217151525							
00004							
Op	Obj	Timestamp+Sequence	Type	Dev	Earliest Log	Current Log	Backup ID
G	T	20021217155555	N			S0000539.LOG	
Table: "TPCD" ".LINEITEM"							
Comment: REORG PAUSE Start Time: 20021217155555 End Time: 20021217155555							
00005							
Op	Obj	Timestamp+Sequence	Type	Dev	Earliest Log	Current Log	Backup ID
G	T	20021217155714	N			S0000539.LOG	
Table: "TPCD" ".LINEITEM"							
Comment: REORG RESUME Start Time: 20021217155714 End Time: 20021217155714							
00006							
fake fake							
Op	Obj	Timestamp+Sequence	Type	Dev	Earliest Log	Current Log	Backup ID
G	T	20021217173916	N		S0016740.LOG	S0016747.LOG	
Table: "TPCD" ".LINEITEM"							
Comment: Comment: REORG Done Start Time: 20021217173916 End Time: 20021217173916							

Figure 7-19 REORG entries from the history file for partition 1

7.3.7 Are there availability improvements for the classic REORG?

Yes. The classic **REORG** now allows read access to the table up to the copy phase. Also, performing the **REORG** by database partition provides assistance to finely tune the timing, duration, and impact of the REORG on the databases system resources, as covered in 7.3.3 “What benefits will REORG by database partition provide?” on page 210. In addition, the ability to exclude both long field and LOB

data from the **REORG** can decrease the amount of time required, as well as the CPU use and I/O required, both of which will improve the availability of your database.

7.4 Index enhancements

In order to implement the most significant new availability features of DB2 UDB ESE V8.1, such as multidimensional clustering, online loads, online reorganization of tables, and online reorganization of indexes, type-2 indexes are required. DB2 UDB now supports type-2 indexes. Additional changes, such as the ability to rename an index and create indexes, can also contribute to a higher level of availability.

Indexes may now be reorganized separately from the table data, and used in combination with the **REORG TABLE INPLACE**, which may be used to reduce or eliminate outages formally taken to perform offline reorganizations of the table and index.

Many warehouse environments with very large tables already use DMS table spaces to store the table data and separate DMS table spaces to store the index data. It also seems as more and more applications require access to the data, more and more indexes are needed to provide high performance for these new applications. Before DB2 UDB ESE V8.1, only regular DMS table spaces could be used for indexes. With V8.1, indexes may now use DMS large table spaces. However, if the use of large DMS table spaces for existing tables is something that your environment may need, careful planning to perform the change is needed, since this type of change may well require the following: the data to be exported or unloaded from the table; the table to be recreated with the proper DMS table space for both data and indexes; and the data to be imported or loaded back into the table, and the indexes created. These actions are beyond the scope of this book and will not be covered here.

If your table already has data in it when you create the indexes, use the **COLLECT STATISTICS** option with the **create index**. This eliminates the need to execute a separate **RUNSTATS** command after the index is created.

7.4.1 How do the changes to the create index and rename help?

The changes to the create index statement that now allow concurrent read and write access to the table throughout the index creation, and the ability to rename an index both will improve availability by reducing the number of planned outages previously needed to do most work with indexes on production tables.

A business example that demonstrates how availability can be improved by taking advantage of both the improve concurrency during an index creation, and the ability to rename indexes is described below.

Suppose that when the customer table was first built, an unique index named `ci_key_name_phone` was created. Several months later, while monitoring the applications that access the customer table, you noticed that the `c_address` field was requested by over 98% of the queries, providing a good index hit ratio, but requiring DB2 UDB to go to the table row to get the address information. If that read from the table data could be eliminated, then the performance of those applications (doing 98% of the queries) could be dramatically improved.

The naming standards enforced in your environment require that the name of the index be derived from the columns that make up the actual index, and should not reflect any include columns. In this situation, this would require that both of the indexes, the current one without the include column, and the new one with the include column have the exact same name. This is not allowed.

Previously, the only way to accomplish this was to drop the existing index and create the index again with the changes to the definition, and then execute a **RUNSTATS** command against the table. With the very large tables used by most warehouse environments (millions to billions of rows) these three steps often take many hours to complete. To ensure the data integrity, which is meant to be enforce by this *unique index*, an outage would need to be planned in order to perform this work, with a duration long enough complete all three steps. This type of trade off, long outages versus performance improvements to be gained by taking the outage, are becoming more and more difficult to explain to management as the outage window required continues to grow along with the size of the database.

The previously required outage is no longer required because of the improvements that allow read and write access to the table during an index creation, combined with the ability to rename an index, effectively provide a means to alter an index online and almost instantaneously.

A rename may also be used as a method to *fall back*. As long as both indexes exist on the table, each will be kept up-to-date, so if a fall back to the old index is required, the indexes can just be renamed back to their original names. However, having both indexes on the table will may negatively impact performance. The trade-off between the ability to fall back, and the performance impact required to maintain that ability, will determine how long (after step 3 in Figure 7-5) to wait before executing step 4.

1. Creation of original index performed three months ago:

```
db2 create unique index db2inst1.ci_key_name_phone on  
tpcd.customer(c_custkey,c_name,c_phone)
```

2. Creation of the new index with the include column using the collect statistics option to remove the requirement to perform runstats against the table following the index creation, which were performed while the system is being used for production:

```
db2 create unique index db2inst1.cin_key_name_phone on  
tpcd.customer(c_custkey,c_name,c_phone) include (c_address) collect statistics
```

3. Rename of the indexes:

```
db2 rename index db2inst1.ci_key_name_phone to cio_key_name_phone  
db2 rename index db2inst1.cin_key_name_phone to ci_key_name_phone
```

4. Drop the old index: This step should performed only after the possibility of re-instituting the original index, because the new index does not exists.

```
db2 drop index db2inst1.cio_key_name_phone
```

The authorizations for the original index are transferred to the renamed index, and the system catalog tables are updated accordingly. The rename index statement does invalidate any package dependent upon it, but the package is automatically re-created the next time an application needs it.

7.4.2 What are type-1 and type-2 indexes, and what do they mean?

In this section we discuss on what are type-1 and type-2 indexes, and what they mean for online index reorganization.

Type-1 indexes

DB2 UDB versions prior to V8.1 use type-1 indexes. Type-1 indexes caused additional locking in certain occasions, which will reduce the performance of your applications by having SQL statements that may receive SQL0911 time-outs caused by the false lock contention.

Type-2 indexes

DB2 UDB ESE V8.1 with type-2 indexes support will improve concurrency, since next-key locking is reduced, which will result in fewer SQL statements receiving unnecessary SQL0911. This improvement is show graphically in Figure 7-20 on page 220.

Type-2 indexes also allow indexes to be created on columns with lengths greater than 255 bytes.

Note: All indexes created in V8.1 will be created as type-2 indexes, except if there is already an existing type-1 index on the table, the new index will be created as a type-1. Type-1 and type-2 indexes are not compatible and can not exist on the same table.

7.4.3 Conversion of type-1 indexes to type-2 indexes

If you are migrating from a pre DB2 UDB ESE V8.1 database to a V8.1 database, it is a very simply process to convert any type-1 indexes to type-2 indexes. A **REORG INDEXES ALL FOR TABLE** is all that is required to convert type-1 indexes to type-2 indexes. Type-1 indexes will not cause harm to your database, but the migrating to type-2 indexes will enable you to take full advantage of all of the features in V8.1 dependent upon them.

The DBAs or system administrators responsible for developing the migration plan to move to Version 8.1 should add this index migration as a task. Then everyone involved will be aware of when each table will have type-2 indexes on them, and will be able to plan the implementation of the new V8.1 processes that require them, such as online loads and online table reorganizations.

To determine if you have type-1 or type-2 indexes on your tables, you may use the **INSPECT** command. However, this command may take a while to run, and is not really necessary. If you just execute the **REORG INDEXES ALL FOR TABLE WITH THE CONVERT** option and if the indexes are already type-2, no action will be taken; and if they are type-1, they will be converted to type-2. If the indexes need to be reorganized anyway, this is a more efficient way to do it than to run the **INSPECT** followed by the **CONVERT**.

7.4.4 Online index reorganization

Prior to DB2 UDB ESE V8.1, there was no option to separate the reorganization of the table and its indexes. The indexes were completely rebuilt during the last phase of the off-line, classic **REORG**. DB2 UDB V8.1 has added the flexibility and capability to be able to perform the table and index reorganizations online and separately from each other. By default, a **REORG INDEX** will rebuild all indexes on the tables into unfragmented, physically contiguous pages.

One of the features of the **REORG INDEX** is that it creates a separate shadow storage object in the same table space as the existing index object, so there needs to be enough storage allocated to the table space to hold two copies of the indexes. This is shown graphically in Figure 7-20.



Figure 7-20 REORG INDEXES ALL FOR TABLE

Attention: It is important to note that the default access for a table reorganization is *write access*, but the default for an index reorganization is *allow no access*.

7.5 Online configuration parameters updates

DB2 UDB ESE V8.1 now provides the ability to create and alter the size of bufferpools online, and has increased the number of configurations parameters that may be updated online. (See Appendix D, “DB2 UDB configuration parameters” on page 355.) Online changes of configuration parameters are taking effect immediately, increasing the availability to get the database online, and are available while applying changes to increase performance.

Many of the parameters used to configure DB2 engine for performance are located in the database manager configuration file and the database configuration file. Figure 7-21 shows the relationships between these files and the database objects that they affect.

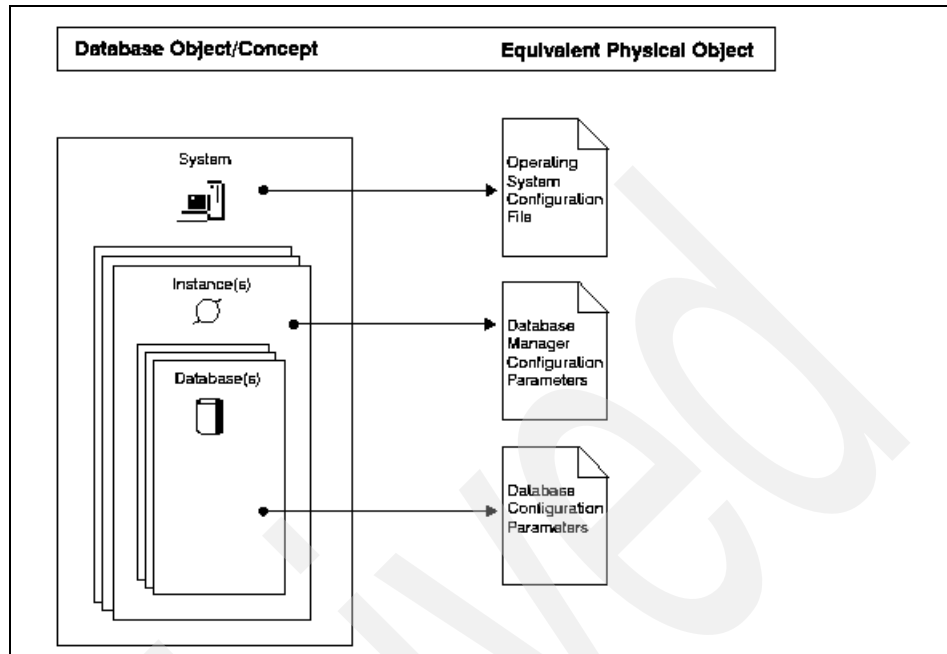


Figure 7-21 Configuration parameter files

These parameters, along with items that affect how memory, are used like bufferpools, and historically are able to be set with the database up and running, but do not take effect until the DB2 instance was recycled. In most large production environments, these changes would require change tickets to be opened, and a spot found for the change during a planned outage. As companies strive ever further towards 100% availability, these outage windows are shrinking in both number and duration. This causes intense scheduling issues between changes needed for items such as performance tuning on a database, applications upgrades, new hardware installs; the list goes on and on.

The default values are set using the assumption that the system is used only as a database server without a lot of memory available to it. If your environment, such as large data warehouse application, does not match the default environment, the default settings should be carefully reviewed to determine if they should be adjusted to match your environment. DB2 UDB ESE V8.1 now offers some guidance in setting the parameters based on what your particular environments need. This can be done through Control Center and launching the new Configuration Advisor wizard, covered in 8.3.3, "Configuration Advisor" on page 254, or by using the new **AUTOCONFIGURE** discussed in 8.4, "AUTOCONFIGURE command" on page 261.

The path to performance improvement recommended by most experts is to implement changes to only a few of these parameters at a time, and to change the values incrementally in small steps. This path allows impact of the changes to be evaluated after implementation, and can help determine what the next steps are. Changing only one or two parameters at a time also helps reduce the possibility of the parameter changes, in effect cancelling each other out when several may have had a positive improvement on performance, while others had a negative impact.

The need to recycle DB2 UDB during a planned outage window, and the recommended way to implement performance tuning are becoming more and more diametrically opposed as the windows shrink and the databases grow.

DB2 UDB ESE V8.1 has made some significant improvements in both the type and number of these configuration parameters, which may now be changed without requiring DB2 UDB to be recycled. These changes will go a long way in empowering DBAs and system administrators to be able to plan, implement, and monitor performance improvements incrementally, without an outage window. The end result should be better tuned, higher performing databases in a smaller amount of time. The decision of when to have the parameter changes and bufferpool resizing take effect is under the DBAs control. This is done by specifying either **IMMEDIATE** or **DEFERRED** when the command is executed. By default (as long as there is enough free memory available) the change will go into effect immediately, as long as there is an attachment to the instance when the command is run.

Tuning your DB2 UDB ESE V8.1 environment to provide the best possible performance is a very complex task. Determining the values and settings for the DBM and DB configuration parameters and the number of bufferpools to be created, and the amount of memory to be allocated to each, are just some of the items to be considered when designing and tuning your system.

This section's purpose is to illustrate how, once you have already determined the settings and values, these values may be changed online.

In many companies the push towards combining databases used for warehouses and databases used by OLTP applications into a single multi-partitioned database to be used by both, continues to evolve and become a reality. More and more of these environments are finding that the applications using the database make use of DB2 UDB resources such as memory and I/O, in vastly different ways. This increases the already complex task of determining the proper values to use. Previously, even if the type of workload in the environment varied distinctly in a predictable way, the values used were a compromise between the needs of all of the applications, because they could not be changed online.

7.5.1 Online bufferpool activities

Bufferpools are used by DB2 UDB to place data and index rows in memory.

In general, for multi-partitioned databases, bufferpools are created or altered on all of the database partitions in the database, or on specifically named database partition groups (formally known as node groups).

Bufferpools are often sized by the type application that will use them. Very generally, large bufferpools are used in environments that have a high volume of queries that infrequently require sorts. A data warehouse environment does not usually fit that description very well; its queries usually require large sorts, and so it is more likely to create smaller sized bufferpools, and allocate more memory to the configuration parameters that affect sorting.

7.5.2 Online changes to configuration parameters

Now (with many of these parameters able to be changed and take effect immediately) is the time to take a fresh look at the timing of the workload in your environment to determine if there are areas where your system can make use of this ability. One quick example would be to change them before large reports are run at the end of the month or at the end of the year.

DB2 UDBs now has the ability to automatically adjust the values of some of the parameters in both the database manager configuration file and the database configuration file based on current resource requirements. The initial release supports this option for the Size of Instance Shared Memory, (*instance_memory*) in the database manager configuration file, and both the Database Shared Memory size (*database_memory*) and the Maximum Number of Active Applications (*maxappls*) in the database configuration file.

Note: For parameters that are set to *AUTOMATIC*, the current value at the time the **get db** or **get dbm** command is run is shown in parenthesis. The *delayed values* column displays the value that will be used as a starting point when DB2 UDB is recycled.

The summary of configuration parameters from the Administration Guide: Performance may be found in Appendix D, “DB2 UDB configuration parameters” on page 355. It has a list of all of the parameters in the DBM and DB configuration file, the level of impact they may have on your system, and if it is configurable online or not, and if its value can be set to *AUTOMATIC*.

The commands **db2 get dbm cfg** and **db2 get db cfg** show the values for each of the parameters. However, until now, if the **db2 update** command had already been run, the new value would show even though it had not taken effect because

DB2 UDB had not yet been recycled. For example, the AIX operating system your p690 Regatta is scheduled for an upgrade next weekend and so DB2 UDB will be recycled and it has been several months since this has been done.

7.5.3 How does online configuration improve availability?

Some of these changes may seem relatively minor or trivial upon first review. You have been asked to verify that no changes will be made to DB2 UDB when it is started as part of the recycle. When you look at the database manager file for partition 0, a subset of the information output is shown in Figure 7-22. This is format of the information that you would receive from running **db2 get db cfg for pfp** whether the command was run on Version 8.1 or an earlier version.

```
db2 get db cfg for pfp

      Database Configuration for Database pfp

Database configuration release level      = 0x0a00
Database release level                  = 0x0a00

Database territory                      = US
Database code page                      = 819
Database code set                       = ISO8859-1
Database country/region code           = 1

.
.
.
Interval for checking deadlock (ms)      (DLCHKTIME) = 10000
Percent. of lock lists per application  (MAXLOCKS)  = 10
Lock timeout (sec)                      (LOCKTIMEOUT) = 60
.
.
Log file size (4KB)                     (LOGFILSIZ) = 1000
Number of primary log files              (LOGPRIMARY) = 3
Number of secondary log files            (LOGSECOND) = 75
.
.
```

Figure 7-22 Partial output received for partition 0 from **db2 get db cfg for pfp**

From this, there is no indication if the values shown in the output are really being used by DB2 UDB at the current time or not; and in older versions of DB2 UDB, there was not an easy way to determine this, so you may or may not have been able to provide an accurate answer to the question if DB2 UDB will have any changes made when it's recycled, which could lead to scenario #1.

Scenario #1

You are unaware of any changes which were not documented through the change control process, and you report that no changes have been made that will change the way DB2 UDB behaves after it is recycled.

All is well when DB2 UDB is first started early Sunday morning.

Monday morning

You start noticing SQL0911s (timeout) in the db2diag.log, the Health Center is issuing alerts, and users are calling saying that applications that worked on Friday, but took several minutes to process, are now not working.

You are asked once again if anything changed in DB2 UDB over the weekend, and you answer “No.”

Severity 1 conference calls are convened with you and your support staff from operations, AIX systems, applications, etc. People spend the next six hours performing problem determination.

The problem is determined to be the Locktimeout value. Unfortunately, it is not one of the parameters that can be changed online, so an unplanned outage is required to recycle of DB2 UDB to fix it. You most likely will spend most of the next several days performing a root cause analysis, writing reports, and attending meetings to determine how this happened, and how to ensure that it does not happen again.

Friday night

The filesystem used by DB2 UDB, which is dedicated for your DB2 transaction logs is 80% full at 9:00 PM. A new sales catalog is being mailed to all of your customers next week. The program that imports the items that will be in that catalog into the items table is started at 10:00 PM. This program had a new version installed into production last weekend. The programmer decided there was no reason to keep using the commit count parameter as part of the import and removed it. The program used does a commit every 5,000 rows, so having a total of five transaction logs per partition was OK, and the filesystem for your logs had room. Log retain is on and the userexit to archive the logs to TSM is in place. Now however, 75 secondary log files can be allocated instead of two. The filesystem fills up because most of the logs are still active, and TSM cannot archive them. Health Monitor alerts are received, and all applications that need a additional log space to continue will fail and rollbacks are performed (block log on disk full is set to NO), and in the worst case scenario happens, DB2 UDB crashes.

Possible resolutions:

The application performing the import must complete successfully by noon on Saturday. Several options that could be taken are described below:

1. Put the commit count back into the program. This would require the new version to be moved into production immediately without any testing.

Evaluation: High risk as there is no confidence that additional errors will not be introduced.

2. Put the old version of the program back into production.

Evaluation: Low to medium risk as the risk is depending on how it could be done in your environment.

3. If there is space available to use, increase the size of the filesystem to be able to hold at least 78 logs per partition.

File systems on AIX can not be made smaller. To reclaim the space later, the filesystem must be exported, dropped, rebuilt, and re-imported. An additional complication of this solution is the exact number of active logs that would be needed for the import program, and all other programs executing at the same time can not be determined with 100% accuracy, which raises the question “Are 75 secondary logs even enough to allow the import program to run successfully?” and if not, what value should be used and how large do the file systems really have to be to enable it to work?

Evaluation: High risk, mostly because no guarantee that this step alone will allow the import application to run successfully.

4. Take advantage of the new update abilities of the database manager parameters, which include a new way to allocate secondary log files and online updates to parameters.

Set logsecond to -1 and blk_log_disk_ful on. Both are updatable online.

Setting logsecond -1 allows active logs to be archived by the user exit. Block log on disk full, means that when the DB2 UDB attempts to create a new log file, and the filesystem is full, it will not cause the application to fail and initiate a rollback. DB2 UDB will keep trying to create the new log every 5 minutes. This will give the userexit time to free up space by archiving some logs to TSM.

Evaluation: Low risk. DB2 UDB does not need to be recycled, if desired the two parameters can be changed back to their original value when the import application finishes. There is a potential performance impact for applications that would wait for logs to be archived to continue.

Now, both the **get dbm cfg** and **get db cfg** commands have a new parameter **show detail**. This parameter will show both the current value being used and any changes that have been made to the parameters, but are not yet in effect and delayed until DB2 UDB is recycled. Figure 7-23 shows some of the rows returned when the show detail parameter was added. You can now see that the current value for lock timeout (locktimeout) is actually -1 (never timeout), and that the number of secondary log files (logsecond) that DB2 UDB is able to allocate is really only 2. When DB2 UDB is recycled, it will start using the new values of 60

seconds for locktimeout and 75 for logsecond. Scenario #2 shows the difference this knowledge can provide and how it contribute to increased availability.

```
db2 get db cfg for pfp show detail
```

Database Configuration for Database pfp			
Description	Parameter	Current value	Delayed value
Database configuration release level		= 0x0a00	
Database release level		= 0x0a00	
Database territory		= US	
Database code page		= 819	
Database code set		= ISO8859-1	
Database country/region code		= 1	
.			
.			
Interval for checking deadlock (ms)	(DLCHKTIME)	= 10000	10000
Percent. of lock lists per application	(MAXLOCKS)	= 10	10
Lock timeout (sec)	(LOCKTIMEOUT)	= -1	60
.			
.			
Log file size (4KB)	(LOGFILSIZ)	= 1000	1000
Number of primary log files	(LOGPRIMARY)	= 3	3
Number of secondary log files	(LOGSECOND)	= 2	75
.			
.			

Figure 7-23 Partial output for partition 0 from db2 get db cfg for pfp show detail

Scenario #2

You are aware of the two changes that have been done that were not documented through the change control process, and report what changes will occur in the way DB2 UDB behaves after it is recycled.

A timeout value of 60 seconds is not a reasonable value for a large data warehouse. Further monitoring and investigation needs to be done before determining if a value other than -1 is feasible for your environment. Locktimeout is not a parameter that can be updated and made effective online. The command **db2_a11 db2 update db cfg use locktimeout -1** is run before DB2 UDB is recycled, ensuring that when DB2 UDB starts again, locktimeout will be -1, thus avoiding the need for an unplanned outage mid day Monday.

Result: No problems reported Monday morning; no service level agreements missed; no problem determination to be done, and so on.

It can not be determined who updated the value of logsecond from 2 to 75 (using the deferred option) or why it was done. The command **db2_a11 db2 update db cfg for pfp logsecond 2** is executed before the change window. The following Friday night, the filesystem for the logs will not fill up, the DB2 UDB would not crash and the unplanned outage is avoided. The import application will still fail and rollback due to DB2s ability to allocate more logs; the rollback will be done much faster due to the smaller number of logs. The potential for some of the other applications to be effected still may also be force to rollback. Either option #2 or #4 could be implemented at this time to enable the import application to work. In addition, it has also most likely been established who made the changes and why (the import application programmer) and changes are in progress to restrict that type of authority to everyone that does not require it for production support.

Result: Problem does occur, but has a much smaller impact on production. Increased authorization security is planned for implementation to prevent this type of issue impacting availability in the future.

Tip: The basic form of the commands **get dbm cfg** and **get db cfg**, do not require an attachment to the instance or a connection to the database.

However, the expanded form of **get dbm cfg show detail** requires an attachment to the instance, and the expanded form of **get db cfg for dbname show detail** requires a connection to the database.

Below are examples of what would be returned by the **db2 get dbm cfg** command if run in the indicated order:

1. db2 get dbm cfg

Output of the command is displayed in the format shown in Figure 7-22.

2. db2 get dbm cfg show detail

SQL1427N An instance attachment does not exist

3. db2 attach to instance db2inst1

Instance Attachment Information
Instance server = DB2/6000 8.1.0
Authorization ID = DB2INST1
Local instance alias = DB2INST1

4. db2 get dbm cfg show detail

Output of the command is displayed in the format shown in Figure 7-23.

7.6 Improve availability through backup/recovery plan

This section provides a high-level description of what types of backup and restore options exist, and focuses on enhancements to the DB2 backup and restore utilities available in Version 7 and Version 8; specifically the new and faster table space restore in DB2 UDB V8.1.

Business requirements are the driving force that determine the backup and restore plan that is implemented at your site. To be able to develop the plan, it is crucial that the answers to many questions be known and taken into consideration, among them:

- ▶ Are there regulatory requirements for data retention and recoverability?
- ▶ Is a disaster recovery plan required?
- ▶ Is off-site archival needed?
- ▶ Can the data be re-generated from other sources?
- ▶ Is there a specific availability requirement for the database?
- ▶ What amount of time is acceptable to perform backups and restores?
- ▶ To what point must the data be able to be recovered? Always to current point in time? Midnight on the previous day? Last weekend?
- ▶ Does the database backup need to work in conjunction with an overall high availability structure in your environment?
- ▶ What is the cost and impact for implementing the plan? This would include items such as storage, off-site archival, database and system administration support, additional software such as TSM or backup and restore tools, and additional network traffic that would result.

Backup options for your DB2 UDB databases have increased dramatically over the last several years. Addressing all issues that affect the plan is beyond the scope of this book. There are multiple publications that do address these issues in detail. Among them are the DB2 UDB manual *Data Recovery and High Availability Guide and Reference Version 8*, and the redbooks *DB2 UDB Backup and Recovery with ESS Copy Services*, SG24-6557, and *DB2 Warehouse Management: High Availability and Problem Determination Guide*, SG24-6544.

7.6.1 Backup and restore options

The basic DB2 backup and restore utilities allow you to backup or restore your multi-partition database by partition or by table space(s).

A backup always degrades the performance of the production system. Especially if the data warehouse is big or in a 24x7 environment; it is hard to find a time

frame when to plan the backup to not interfere with the normal operation. To free the production system from the overhead of backup, a copy or mirror of the database would be nice to be available for backup, report, or other purposes. Some intelligent storage servers such as IBM Enterprise Storage Server (ESS), support the split mirror feature.

For example, the FlashCopy is the ESS's implementation of the split mirror feature. As far as DB2 UDB is concerned, FlashCopy is a point-in-time, non disruptive, almost instantaneous way to copy the data in the database. DB2 UDB is suspended for the moment of time it takes to initialize the FlashCopy, and from that point has complete access to all data. The actual I/O taking place on the ESS continues on in the background until the copy is complete. ESS Copy Services uses the storage system to copy the data and control the I/O thus removing the CPU usage and the I/O activity normally used by DB2 backup or utility. This has a positive impact on performance, and also improves availability of the database. FlashCopy is done within a single ESS and often, a DB2 backup is then taken of the *copied* database to be able to maintain multiple backup copies.

Peer-to-Peer Remote Copy (PPRC) performs dynamic synchronous mirroring to a remote ESS. This allows you to effectively store your data off-site immediately. To restore the database, these steps are basically reversed. These two types of services can also be combined in a variety various ways to achieve even higher availability and accessibility. The redbook *DB2 UDB Backup and Recovery with ESS Copy Services*, SG24-6557 gives some excellent examples of how to use these services to solve various business needs.

The DB2 Recovery Expert, a DB2 Multiplatform Tool, can be used to perform comprehensive automated recovery of your DB2 UDB data. Its extensive diagnostics and *Self Managing And Resource Tuning* (SMART) abilities also help to improve availability by minimizing the duration of the outage. This tool currently supports DB2 UDB V8.1 single partitioned databases, and will soon support multi-partitioned databases.

7.6.2 DB2 backup utility

The DB2 backup utility provides the following options to back up your DB2 UDB data by partition or by table space(s):

- ▶ Full database backup
- ▶ Incremental backup
- ▶ Split mirror

Full database backups

Full backups contain all information for the specified objects and are allowed to recover whole database and its partition to a specific point-in-time.

Offline backups (the default) can be performed on databases with circular logging, and also with log retention or user exit enabled. The partition containing your DB2 UDB system catalogs should be backed up first. All remaining partitions can then be backed up concurrently if desired.

When performing database backups partition by partition, the catalog partition must be backed up separately to keep a consistent backup between partitions, and the catalog before all remaining partitions may be backed up independently and in parallel using the **db2_a11** command in the following steps (example assumes the catalog partition is on partition 1):

1. Issue the backup command to the catalog partition only:

```
db2_a11 "<<+1<db2 backup db bigdb to ...
```

2. When complete, issue the backup command to all remaining partitions in parallel (except partition 1):

```
db2_a11 "||<<-1<db2 backup db bigdb to ...
```

A partition backup is valid only if it is synchronized with the catalog partition.

In the case of an offline backup, this means that no database activity (connections) should be allowed from the start of the catalog backup to the end of the last partition backup. If there is a connection then restore of that partition will fail.

Online backups require that log retention or user exit are enabled. Online backups allow applications to have complete access to the database throughout the backup process, so the use of them can increase the availability of your database. Online backups are very useful for a complete recovery of the database; using the logs and their low impact on the workload allows one to schedule them more often than the offline backups.

7.6.3 Incremental backups

In Version 7, incremental backups and the ability use the **db2inidb** utility to create split mirrors were introduced. A very high level overview is included in this section. Details on why, when, and how to implement these features in your environment are available in the documents referenced the introductory paragraph of this section.

There are two types of incremental backups as described below, and they are illustrated in Figure 7-24:

- Incremental backups are cumulative and contain all data that has changed since the last successful full backup was taken.
- Incremental delta backups are non-cumulative backups and contain data that has changed since the last backup was taken, no matter what type of backup it was.

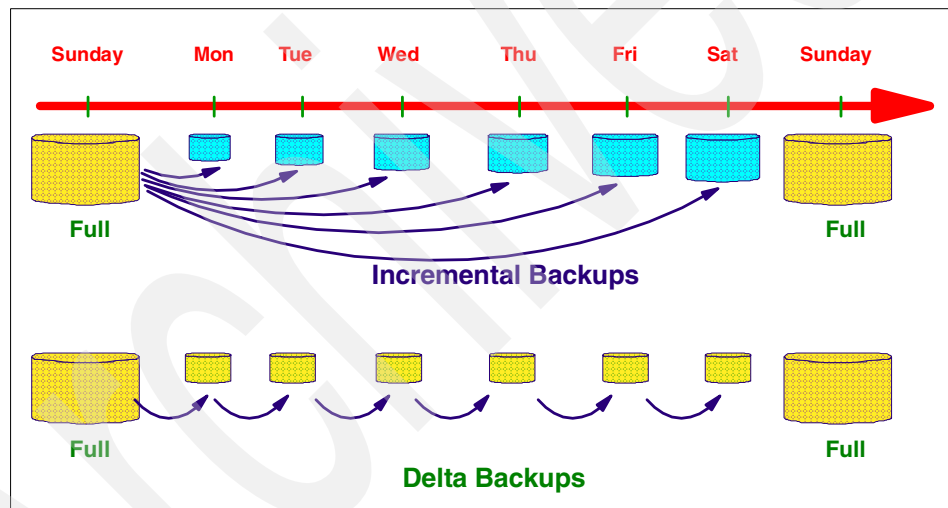


Figure 7-24 Types of incremental backups

Note: Incremental backups are not necessarily faster than full backups. It will depend on the number and size of the DB2 UDB table spaces that have had updates to them since the last full backup. However, incrementals do use fewer CPU and DB2 UDB resources than a full backup does, and the resulting backup file is much smaller. They are generally recommended when a small percentage of the data has been changed. They can be particularly advantageous to extremely large warehouse applications by reducing the time and hardware required for backups, and still be able to ensure complete recoverability.

7.6.4 Backup using split mirror in conjunction with DB2 and ESS

The *DB2 suspended I/O* feature and the *db2inidb* utility were introduced with DB2 UDB V7.2. The suspended I/O support is necessary to bring the database into a consistent state before establishing the split mirror. With the **write suspend** command, the database enters a well defined state where it can recover later using the **db2inidb** utility. During the suspend of the database, the DB2 UDB clients will see that their transactions will hang. As soon as the database is resumed again for writing (write resume), the transactions will proceed normally. The **db2inidb** utility is necessary in combination with the split mirror feature, to bring the target database into a running state again after the split mirror was established.

Split mirror is very useful in off loading the performance impact that backups can have. It uses the unobtrusive storage services to replicate the data. Then an actual DB2 backup can be taken from the secondary server with only minimal I/O impact on the active database's performance, and no impact on its CPU. Performing the DB2 backup utility on the secondary database allows multiple version of backups to be kept.

The scenario to take a split mirror on a target database that can be restored later on the source database is to set up a standby database. A standby database in this context is a database where the log files of the source database will be applied on the target (standby) database. The standby database will be in a permanent rollforward process as long as the source database produces log files (and as long as the source database is available). See Figure 4-19.

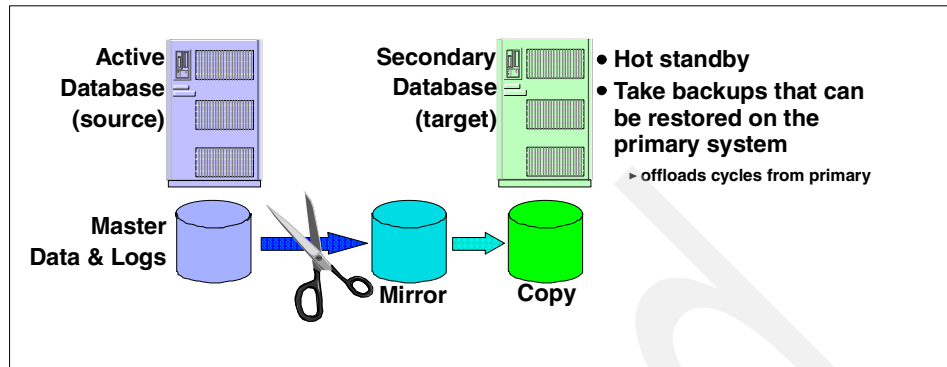


Figure 7-25 Using split mirror for backing up the database

The steps required to set up a standby database are:

1. Suspend I/O on source database. The following DB2 UDB command will suspend all write activities from DB2 UDB clients, but they will not be disconnected from the database: **db2 set write suspend for database**
2. Establish split mirror. This step is dependent on the vendor. For example, issue FlashCopy on the IBM ESS. The whole database must be copied to use it on the target machine.
3. Resume I/O on source database. After a few seconds the split mirror is established and the database can be made available again for writing. The transactions will proceed as normal: **db2 set write resume for database**
4. Make data visible on target machine. The target volumes will now contain all the data from the time the split mirror was established, but they may not yet be visible to the target operating system. The operating system on the target machine has to provide access to the data (import, mount, and so on). (For initial setup, the database needs to be cataloged at the target machine.)
5. Start the target database by issuing the **db2start** command at the instance at the target machine.
6. Set the database into rollforward mode: **db2inidb <target-database>as standby**

Use **db2_a11** command to run **db2inidb** on each partition before restoring or using the split mirror image.

Now the database can be restored on the source system.

The log files from the source database can also be used to rollforward the target database and maintain a hot standby database (To make the transfer of the log files easier, a user exit may be configured.):**db2 rollforward db <target-database>to end of logs**

As long as the database stays in rollforward pending mode, no database backups are allowed.

If the source database crashes, the standby database can be activated for access. (Make sure that all data has already been copied to the target volumes.) The rollforward must be stopped with the stop option of the DB2 UDB rollforward command: **db2 rollforward db <target-database>stop**

Then the users can switch over to the standby database to continue their work.

Note: Do not copy the log files using the split mirror feature. This will break your rollforward chain. The necessary logs for rollforward must all be acquired from the source database as soon as they are available (no longer used by the source database). If log files have been copied to the target machine using the split mirror feature, then they must be deleted.

The configuration used to test ESS and FlashCopy is described in “Configuration 2” on page 36. ESS was formatted as RAID 5 arrays and 8 GB LUNs. Half of the disk space was allocated to the database, the other half was allocated to FlashCopy.

ESS is attached to p690 by 8 Fiber channels (FC 6228) with a throughput of 2 Gb/s. The throughput performances results are shown in Table 7-1.

While a FlashCopy was running, elapsed time medium queries were measured before FlashCopy was started and when FlashCopy was running. Results show no obvious elapsed time increase and the impact of FlashCopy in performance was low. The FlashCopy was done at adapter level without using the ESS cache, and FlashCopy priority was positioned at low when the task is started. Positioning FlashCopy to high priority speeds up the copy of data, but may degrades queries.

Tip: Performance is not impacted using database containers striping along the same array or across arrays. This is also called *vertical* or *horizontal* striping.

Physical data placement is not crucial to performance (except for access contention, logs and data on the same disk).

A better solution is to stripe the containers across multiple ESS.

Note: ESS large sequential read performance (table scan 16 K pages):

- ▶ DB2/ESS F20 can sustain 290/320 MB/s bandwidth
- ▶ DB2/ESS 800 can sustain 550-660 MB/s bandwidth

Table 7-1 ESS highest performance

ESS open storage	ESS F20	ESS 800
100% read in cache (4K block)	45 000 IO/s	113 000 IO/s
100% read in cache (64K block)	384 MB/s	720 Mb/s

We may notice that DB2 UDB ESE is using the IO bandwidth of the ESS quite well.

7.6.5 DB2 full and incremental restore

The DB2 restore utility provides the following options to restore your DB2 UDB data by partition or by table space(s). Please note that all online restore options require that log retain or user exit have been enabled and the necessary logs are available:

- ▶ Restoring from an offline backup allows you to restore the exact image of the database at the time the backup was taken. The various flavors are:
 - If the database uses circular logging, offline incremental and incremental delta backups may be applied.
 - If log retain or user exit has been enabled:
 - Either online or offline incremental and delta incremental backup images may be used. If the online backups are used, a rollforward must be done.
 - The **rollforward** command can be used to recover the data to a point-in-time, or to the end of the logs. This option ensures the all available data is recovered.

- ▶ Restoring from online backups always requires that the database be rolled forward. The DB2 history file keeps track of what logs *must* be applied during the rollforward process. These are logs that were updated while the online backup was in processed, and are the minimum point-in-time that the data can be restored:
 - Offline incremental and delta incremental backup images may *not* be used.
 - The **rollforward** command can be used to recover the data to a point-in-time or to the end of the logs. The rollforward to end of logs ensures the all available data is recovered. The point-in-time option allows you to specify the time to roll forward to; an example of its use is if someone by mistake deleted all of the data from a table at 10:00 AM, you can restore and then rollforward till 9:59 AM, and all the deleted data will be recovered.
- ▶ Restoring using the files created by incremental backups

Additional items to be considered:

- ▶ When restoring the entire database, restore the partition with the DB2 UDB system catalog tables first. All remaining partitions may be restored concurrently if desired.
- ▶ In a multi-partitioned database, any and all rollforward operations *must* be invoked from the catalog partition.
- ▶ When restoring an entire partition, the DB2 restore utility requires exclusive access to the database while performing the restore and rollforward operations.
- ▶ At a minimum, to restore using incremental delta backups requires the original full backup, the last non-cumulative incremental (if taken) and *all* of the delta backup files taken since then. If online backups are being used, the logs needed to rollforward are also required.
- ▶ Table space restores can be done from either full backups or from table space backups.
- ▶ Table space(s) may be restored either offline or online. When they are restored online, only the data in the table space(s) being restored is unavailable. The data in all other table spaces is available throughout the restore processes. Many large warehouses can take advantage of this method to dramatically improve availability of the database by restoring the most accessed or important data first, allowing the applications to start up, and then continue on restoring data until all necessary restores are complete.

7.6.6 Restore using split mirror in conjunction with DB2 and ESS

Using the **db2inidb** utility you can effectively copy the data back from the split mirror image database to restore the source database, and then rollforward the logs of the source database. This can be much faster than performing a restore using the restore utility. However, if the storage copy that created the secondary database is for some reason lost, a DB2 restore utility will need to be performed to restore the secondary database, and then the restore of the split mirror would be done. (See Figure 7-26.)

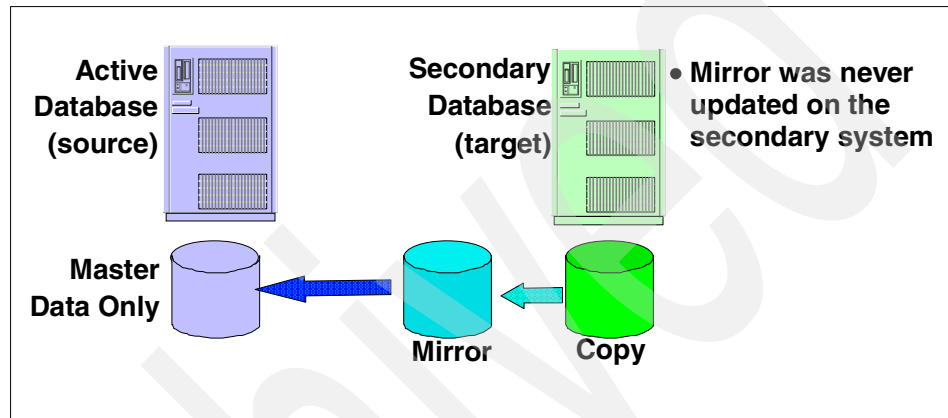


Figure 7-26 Using a split mirror copy to restore the primary database

The steps are:

1. Stop the source database: **db2stop**

On a partitioned database, this requires one to stop all partitions using **db2_a11**

2. Restore the database.

With operating system tools (for example, copy, xcopy, cp, tar and so on), copy the data files of the split mirror database over the original database. But do not copy the log files from the target database to the source database. (Make sure that all data has already been copied to the target database.). Another restore possibility would be to *reverse* the split mirror and copy the data on the target disk volumes back to the source disk volumes using the split mirror feature of the storage server again.

3. Start the source database again. **db2start**

On a partitioned database this requires to stop al partitions using **db2_a11**

4. Reestablish the mirrored copy on the source database: **db2inidb**
<database>as mirror

Note: No DB2 UDB operations are allowed on the target (split mirror) database after the split mirror was established in order to use the target database for restore on the source database. The database needs to stay in this frozen state.

5. Now a **roll forward** to end of logs of the database can be started. The logs from the original source database will be used.

7.6.7 Parallel recovery

Performance for crash and rollforward recovery was also significantly improved by enabling parallel recovery.

Prior to V7, crash and rollforward recovery was a serial process. A single database agent serially fixed each page requiring recovery, and performed the necessary changes on the page, before recovery completed.

Since V7 recovery processing is parallelized, it exploits all CPUs in an SMP as shown in Figure 7-27. This improves availability by returning the database to use sooner than what was possible previous to Version 7.

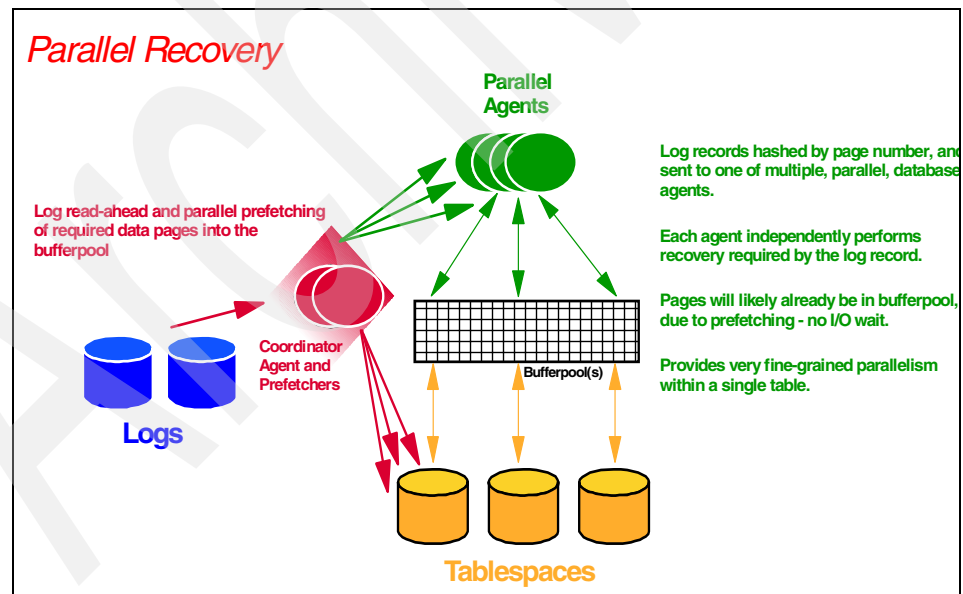


Figure 7-27 Parallel recovery

DB2 UDB ESE V8.1 has added further improvements to the restore utility, including performance improvements for table space restores, and the ability to use local time when specifying the ISO time to roll forward to.

The most significant of these improvements is the way table space restores are processed.

Table space recovery has been dramatically improved, because now only the log files required to recover the table space are processed. DB2 UDB keeps track of which log files are needed, and all log files that are not required are not processed, *and* the user exit does not retrieve them from the archive. In earlier versions of DB2 UDB, *all* log files since the table space backup was taken were required to be processed and retrieved by the user exit from the archive.

In particular, for large databases that have specific table spaces whose data is static or virtually static, only a few logs will need to be retrieved and processed instead of potentially thousands. This can provide a *huge* reduction in the amount of time required to restore a table space. It is worth reviewing your backup strategy to see if your environment may be able to take advantage of this new feature. For example, if a table space has data for the year 2000 that is never changed and you took a table space backup on 1/1/2001, you most likely are not relying on that table space backup to restore that data, since you would need all off the logs from 2001 and 2002. You either have to be taking much more frequent table space backups of that data so fewer logs are needed, or full backups so that the data is included. Now with proper planning, table space backups and restores can become much more important in your overall backup and recovery plans.

Rolling forward to a point-in-time has always required that the time to rollforward to be supplied in GMT time. To add to the confusion, in the DB2 history file, the backup timestamp is in local server time. This also requires that the database or system administrator performing the rollforward be aware of this requirement, because if they used local time instead of GMT time, the recovery point of database would not be where they intended it to be. This requirement could result in some very odd looking restore combinations such as these two steps taken on a database whose server's local time zone is USA Pacific Time. At noon on the 10th, a decision is made to restore the database, and you need to restore from the previous night's backup, and rollforward to 06:00 local time. To restore the backup taken at 23:00 on January 9th, the following command could be used on a DB2 command line:

```
db2 restore db bigdb taken at 20030109230000
```

Pacific Time is 8 hours behind GMT time. To effectively rollforward to 06:00 Pacific Time January 10th, the **rollforward** command with the translation of local time to GMT would be:

```
db2 rollforward db bigdb to 2003-01-09-20.00.00.000000 and stop
```

At first glance, it looks like we are rolling forward to a point in time before the backup was even taken! With v8, the **rollforward** command can use either GMT or local time, so now the **rollforward** command can look like:

```
db2 rollforward db bigdb to 2003-01-10-06.00.00.000000 using local time and stop
```

It is now much clearer that the point-in-time restore will leave the database as it was 8 hours after the backup was taken the previous night at 23:00.

7.7 Additional ways to improve availability

Two enhancements that have been made that may be overlooked when looking for ways to increase availability are: changes affecting DMS containers and a new online **INSPECT** command.

7.7.1 DMS container operations

Previous to DB2 UDB ESE V8.1, DB2 UDB did not have a way to remove containers from a DMS table space. If the disk that was no longer needed had to be recovered for use by some other DB2 object or any other applications, the only way to do it was to export or unload all of the data in the table space, and then drop the table space and rebuild it with the new smaller amount of disk needed, and put the data back in. These steps were often considered too time consuming to be worth doing, and therefore, would not be done at all. Over time this unused space can build up to quite a bit of disk that is there, just not usable.

Prior to Version 7, the only way to make more space available in a DMS table space (without redirected restore) was to add another disk to its *stripe set* or add containers. This always required a rebalancing of the data. The data in the table space is fully available to all applications throughout the rebalancing process, however, rebalancing is a very I/O intensive operation that could degrade performance on concurrent workloads while it was running in the background. Figure 7-28 shows this operation.

Adding space to a DMS tablespace pre V7

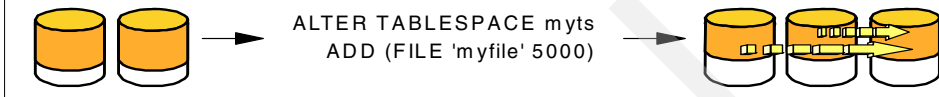


Figure 7-28 The data was ALWAYS redistributed prior to Version 7

Version 7 allowed DMS containers to be extended and resized as long as it was resized to a larger size, which would usually not require the containers to be rebalanced, but it was possible some rebalancing would be done. The ability to extend and resize with little or no impact on performance was a great improvement, and these options are used with great results in many environments with large databases. Figure 7-29 demonstrates how the *EXTEND* option is done. With *EXTEND* you tell DB2 UDB how much space to add, with in V7 *RESIZE* could only be used to add space to the containers. You specify to DB2 UDB what the size of the containers will be, and DB2 UDB does the calculation on how much space needs to be added. Removing space from the DMS table space was still not possible.

Adding space to a DMS tablespace in V7



Figure 7-29 Redistribution was no longer a necessity in Version 7

With DB2 UDB ESE V8.1 a database or system administrator can guarantee that no rebalancing will be performed by creating a new stripe set. This operation is shown in Figure 7-30.

Adding space to a DMS tablespace in V8



Figure 7-30 Adding additional stripe sets

For the first time, space can also be removed from a DMS table space by using the **DROP**, **REDUCE** or **RESIZE** commands:

- ▶ **DROP** will remove a container from a DMS table space. An example is shown in Figure 7-31.
- ▶ **REDUCE** the amount of space to be removed from the containers is specified.
- ▶ **RESIZE** can now remove space from the containers. You specify to DB2 UDB what the size of the containers will be, and DB2 UDB does the calculation on how much space needs to be removed.

Removing space takes more consideration than adding space does. In V8, dropping a container from a DMS table space will always require a very I/O intensive rebalance operation, which could degrade concurrent workloads.

You need to ensure that you leave enough space in the containers to hold all of the data in the table space, and if there is technically enough space, that there is also *logically* enough space to be able to perform online reorganizations, index reorganizations, and to accommodate any changes to the remaining data. If there is not enough free space, you will receive an error as soon as you execute the statement. The data will also have to be rebalanced. If there are concerns about the impact the large amount of additional I/O required for the duration of the rebalance may have on the performance of your production system, *when* to execute the command should be carefully determined.

Dropping a container from a DMS tablespace in V8



Figure 7-31 Dropping a container from a DMS table space

7.7.2 Online inspect command

Very rarely architectural integrity issues arise with a DB2 UDB database. An example would be a bad block on DASD. This makes the data written on that block unreadable and unavailable to DB2 UDB.

Previous to DB2 UDB ESE V8.1, the only way for architectural integrity to be checked was by using the **db2dart** command. In order to run **db2dart**, no connections to the database can exist, so a planned outage is required just to perform the integrity checking, which reduces the availability of the database.

There is now an additional way to verify the architectural integrity of your database that does not require an outage, increasing the overall availability of your environment. This is done by using the new command **INSPECT**. The syntax of the command is shown in Figure 7-32. This command will validate the architectural integrity of your tables and table spaces online and by default, and will run against all partitions specified in the `db2nodes.cfg` file.

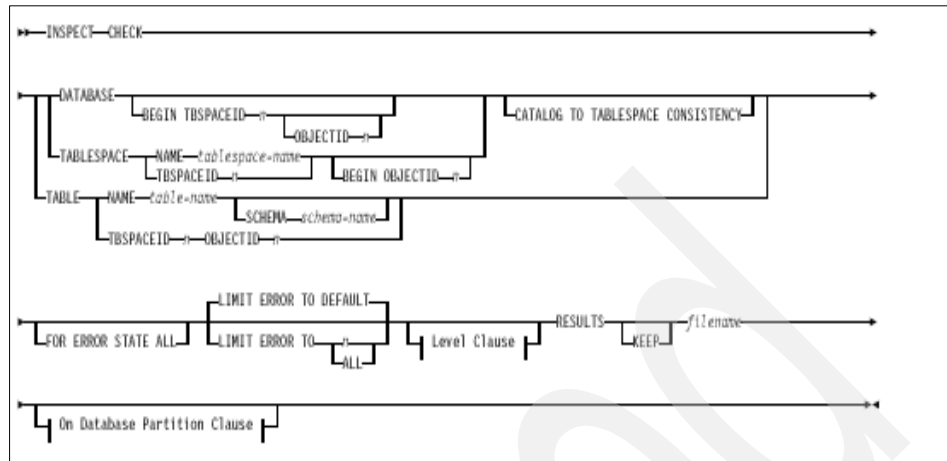


Figure 7-32 *INSPECT* command syntax

The unformatted results from **INSPECT** are written to a file in the diagnostics data directory path (DIAGPATH) specified in the database manager configuration file. If no errors are found, this file will be automatically as part of the processing performed by command. If any errors are found, the file will be left. To be able to access the information in the results file, the utility **DB2INSPF** needs to be run against the results files to format the data.

When that data is reviewed, a plan can then be made to address the errors reported. For example, if it was an index object that returned errors, several options exist:

- ▶ Schedule a planned outage to run **db2dart** In repair mode mark the index as bad. This will result in the index being automatically rebuilt by DB2 UDB the first time the index is accessed. Even though this option does require an outage, it will greatly reduce the outage time, since using **db2dart** to do the initial checking would have required an additional outage.
- ▶ While the database is up and running, the database or system administrator would drop and recreate the index. This requires no outage, but may impact performance during the drop and create steps. This option would require no planned outages at all.

Manage the data warehouse easily

This chapter discusses some of the many new features provided in DB2 UDB ESE V8.1, which simplify the steps required to configure, manage, and monitor the data warehouse on a DB2 UDB ESE V8.1 system.

These improvements will be especially beneficial to customers running large data warehouses due to the sheer size of their environments, and will better enable them to stay competitive with DB2 administration support costs and effectiveness.

Features such as Storage Management and Memory Visualizer provide powerful new ways to automate the collection of data on DB2 resource use. The Health Monitor analyses the health of DB2 UDB and can provide suggestions to improve the health of the system, and will even make many of those changes if you wish it to. These tools can also help identify new ways that the warehouse may be being used.

Combining the intelligence and information provided by these tools with your DBAs database and business experience, and improving the performance of your workload using DB2 UDB ESE V8.1 is better than ever.

8.1 Self Managing And Resource Tuning

DB2 UDB greatly reduces the complexity of data management by eliminating, simplifying, and automating many tasks traditionally associated with maintaining an enterprise-class database. Some of these advances are the first implementation of the Self-Managing and Resource Tuning (SMART) project and the first steps towards making autonomic computing a reality for database implementations.

They include:

- ▶ Health Monitor and Health Center
- ▶ Several new wizards and GUI tools to provide guidance when creating objects, manipulating data, or configuring the environment. For example:
 - Memory Visualizer
 - Storage Management view
 - Design Advisor
- ▶ **AUTOCONFIGURE** command

8.2 Health Monitor

Everyone that runs DB2 UDB will benefit from the new features provided in the two new tools that assist in monitoring the health of your system. These management by exception tools are the Health Monitor and the Health Center. The Health Monitor is on by default.

The expensive, time consuming, old way

In the majority of customer environments, especially when availability and performance are a high priority, home-grown scripts were written to gather information about the health of the database and its components in order to forecast the areas of the database may need attention, and what preventative action should be taken. These home-grown scripts are expensive to maintain. Changes such as new versions of the database code, the level of the operating systems, or even a move to a new server, often require updates to these types of scripts.

In addition, as time goes on, often times the person that wrote the scripts leaves and the DB2 UDB knowledge is lost, so even if there is a scripter available, the expertise to know what to look for and how to interpret the output is no longer there, and the scripts do not maintain the level of functionality or usefulness that they once had. Additional or more detailed information is also often requested by the customer, again requiring work on the existing scripts or completely new

ones. In the past this has been a hard cycle to break out of. Often these scripts use the **db2 snapshot** tool, which requires one or more of the database monitor switches to be on, and which can degrade performance.

Even if your environment already has a system wide monitoring solution in place, you can benefit from this new feature, since both DB2 UDB commands and APIs may be used to integrate with your current environment.

The new way

The Health Monitor information, as well as the Health Indicator configurations, are available several ways:

- ▶ Health Center (and by extension Web or PDA Health Center)
- ▶ Command line
- ▶ APIs

Everybody wins with the way these tools have been implemented. Regardless of if your environment uses the graphical interface tool environment, or if you are restricted to using the traditional command line and API interface environment, these tools are rich in features and information that can free up your DBAs and system administrators from maintaining those old home-grown scripts, and quite possibly prevent you from spending hours fixing an avoidable problem if there had been any indication that your DB2 UDB system was starting to show some health issues.

The Health Monitor runs on the DB2 UDB server, constantly monitoring the health of the DB2 UDB system. Health Monitor does not require any of the snapshot monitor switches to be on, and does not have a performance impact like the previous methods did. It has indicators covering table space storage, sorting, database management system, database, logging, application concurrency, package, and catalog caches, workspace, and memory.

Health Center use

Health Indicator settings must first be configured as shown in Figure 8-1.

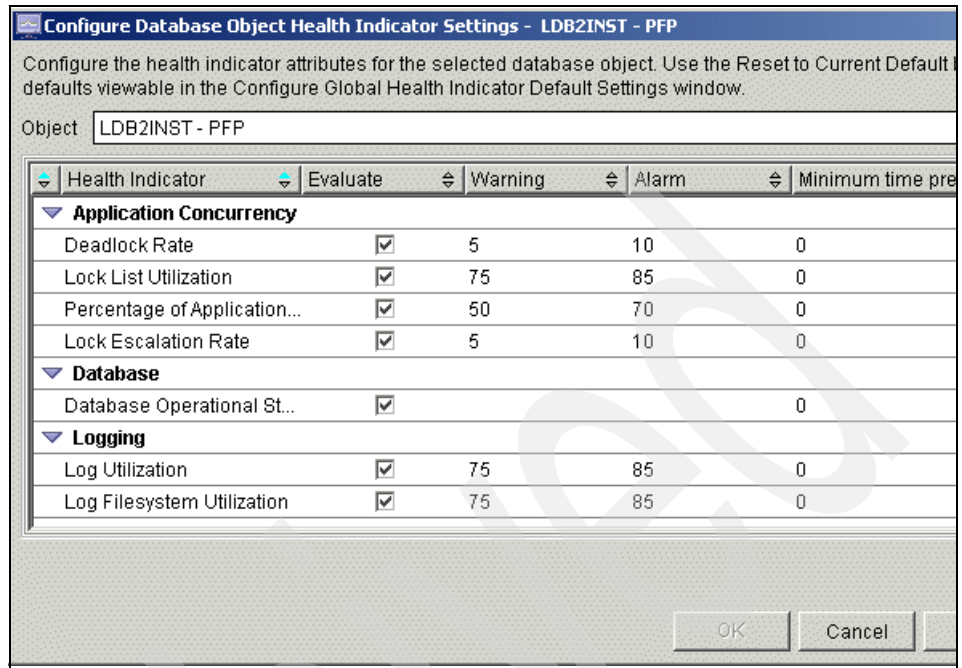


Figure 8-1 Configuring the Health Indicator settings

Health Monitor is a very easy to use tool and has clear alerts and graphics.

It would be especially by operations staff that monitor the environments around the clock and are responsible for paging the DBAs or system administrators when a problem occurs, because of all of the information available for the on call staff.

This information can also be accessed through the Web, making it even more valuable to DBAs providing around the clock support who can receive detailed information, and take steps to correct the issue, even from remote locations.

Command line use

The alerts are written to the new Administration Notification log as shown in Figure 8-2.

Diagnostic error capture level	<DIAGLEVEL> = 3
Notify Level	<NOTIFYLEVEL> = 3
Diagnostic data directory path	<DIAGPATH> = /db2dump

Figure 8-2 Excerpt from the database manager configuration

An entry from the log is shown in Figure 8-3. This log provides easily understood information concerning the level of the alert, what Health Indicator was breached, the database, (if applicable) the partition number, and the history of the previous alerts of this type.

```
2002-12-10-16.38.28.638997 Instance:db2inst1 Node:001
PID:7036932(db2sysc) TID:1 Appid:none
Health Monitor HealthIndicator::update Probe:500

ADM10501W Health indicator "Percentage of Sorts That Overflowed"
(<"db.spilled_sorts">) breached the "upper" warning threshold of "30" with value
"33" on "database" "db2inst1.1.PFP". Calculation:
"<"db.sort_overflows/db.total_sorts">*100;" = "<(4/12)*100;" = "33". History
<Timestamp, Value, Formula>: "<12-10-2002 16:33:28.816143,66,<(2/3)*100>,<12-10-2002 16:28:28.376662,100,<(2/2)*100>,<12-10-2002 16:23:28.1950654,100,<(2/2)*100>,<12-10-2002 16:18:28.1584307,100,<(2/2)*100>,<12-10-2002
16:13:35.624366,100,<(2/2)*100>,<12-10-2002 16:08:28.1334710,100,<(1/1)*100>)"
~
```

Figure 8-3 Health Indicator alert from the administration notification log

The command **db2 get recommendations for health indicator** will return suggested actions and instructions on how to perform those actions for each of the health indicators. Figure 8-4 shows what is returned from the command for the *bp.spilled_sorts* alert.

```
db2 get recommendations for health indicator db.spilled_sorts

RECOMMENDATIONS FOR db.spilled_sorts

Increase sort heap

From the Control Center:
1. Expand the object tree until you find the Databases folder and
   expand the folder until you find the database that you want.
2. Right-click the database, and click Configure... from the pop-up
   menu. The Configure Database dialog opens.
3. On the "Performance" tab, update the "sort heap" parameter as recommended
   above and click OK to apply the update.

From the Command Line Processor, type as shown in the following example:
CONNECT TO DATABASE name
UPDATE DATABASE CONFIGURATION FOR DATABASE name USING "SORTHEAP" size
CONNECT RESET

Tune workload
You can run the Design Advisor to tune the database performance for your workload
by adding indexes and materialized query tables. This can help reduce the need
for sorting. You will need to provide your query workload and database name.
The advisor will evaluate the existing indexes and materialized query tables in
terms of the workload and recommend any new objects required.

From the Control Center:
1. Expand the object tree until you find the Databases folder and
   expand the folder until you find the database that you want.
2. Right-click the database, and click Design Advisor... from the
   pop-up menu. The Design Advisor opens.
3. Follow the steps of the wizard to optimize query performance
   and apply the recommendations of the wizard.

This function is also available from the command line by typing:
db2adviz
```

Figure 8-4 Recommendations for spilled_sorts health alerts

Attention: If you noticed a delay on the sending of the alerts from the Health Monitor, it is recommended that both your current monitors and the health check monitors be run for critical monitors or alerts such as a full table space.

It is also recommended that you take multiple snapshots (2 or 3) before the alert is sent out.

8.3 New wizards

DB2 UDB provides a bunch of wizards to simplify the administration of DB2 UDB, which are available from the Control Center or by command. As examples, here are four useful wizards with partitioned databases:

- ▶ Memory Visualizer
- ▶ Storage Management tool
- ▶ Configuration Advisor
- ▶ Design Advisor

8.3.1 Memory Visualizer

The Memory Visualizer helps you to uncover and fix memory-related problems on a DB2 instance. It is started under DB2 Control Center by right-clicking on the instance and selecting **View memory usage** or (through the command line) using **db2mtrk** command.

It uses visual displays and plotted graphs to help you understand memory components and their relationships to one another. You can invoke it from a Health Center recommendation, or use it on its own as a monitoring tool.

High-level memory components are:

- ▶ Database manager shared memory
- ▶ Database global memory
- ▶ Application global memory
- ▶ Agent/Application shared memory
- ▶ Agent private memory

Historical values of alarm and warning thresholds are also displayed for each component.

Each high-level component is divided into lower-level components that determine how the memory is allocated and de-allocated. For example, memory is allocated and de-allocated when the database manager starts, when a database is activated, and when an application connects to a database.

A memory usage plot can also be displayed for the desired component or components. The usage plot shows the original value, the new value, and the time when the value changed.

Some of the tasks you can perform with the Memory Visualizer:

- ▶ View overall memory usage.
- ▶ Specify which memory information to display, and which information to hide for the DB2 instance and its databases.
- ▶ Update the configuration parameters for an individual memory component to prevent it from using too much or too little memory.
- ▶ Save the memory allocation data.
- ▶ Load memory allocation data from a file into a Memory Visualizer window.

8.3.2 Storage Management

A Storage Management tool (see Figure 8-5) is now available through the Control Center. With this tool you can access the Storage Management view that displays a snapshot of the storage for a particular database, database partition group, or table space. This is useful to monitor the storage state of a partitioned database.

Statistical information can be captured periodically and displayed depending on the object chosen:

- ▶ For table spaces, information is displayed from the system catalogs and database monitor for tables, indexes, and containers defined under the scope of the given table space.
- ▶ For databases or database partition groups, information is displayed for all the table spaces defined in the given database or database partition group.
- ▶ For databases, information is also collected for all the database partition groups within the database.

You can use the information displayed in this view to monitor various aspects of your storage, such as space usage for table spaces, data skew (database distribution) for database partition groups, and capture cluster ratio of indexes for database partition groups and table spaces.

From the Storage Management view you can also set thresholds for data skew, space usage, and index cluster ratio. A warning or alarm flag will let you know if a target object exceeds a specified threshold.

Name	Sche...	Partiti...	Column...	Number of Rows	Data Skew in Rows
CUSTOMER	TPCD			8 Average: 281 250	Maximum: 1 301
LINEITEM	TPCD			16 Average: 11 250 363	Maximum: 7 513
ORDERS	TPCD			9 Average: 2 812 500	Maximum: 2 049
PART	TPCD			9 Average: 375 000	Maximum: 1 620
PARTSUPP	TPCD			5 Average: 1 500 000	Maximum: 6 480
STMG_CUR...	SYSTOO...			6 Average: 2	Maximum: 1
STMG_HIST...	SYSTOO...			6 Average: 11	Maximum: 0
STMG_OBJ...	SYSTOO...			8 Average: 14	Maximum: 0
STMG_OBJ...	SYSTOO...			2 Average: 2	Maximum: 1
STMG_ROO...	SYSTOO...			3 Average: 1	Maximum: 0
STMG_TAB...	SYSTOO...			18 Average: 18	Maximum: 0
STMG_TBP...	SYSTOO...			7 Average: 203	Maximum: 0
STMG_THR...	SYSTOO...			3 Average: 2	Maximum: 1
SUPPLIER	TPCD			7 Average: 18 750	Maximum: 433

Figure 8-5 Storage Management view

8.3.3 Configuration Advisor

The Configuration Advisor provides an easy-to-use tool to start tuning a DB2 UDB system with minimal input. It is a very good starting point for achieving satisfactory database performance for most workloads.

The Configuration Advisor updates configuration parameters for database manager and the database named. Each instance of the database manager should only manage one database (see Figure 8-6).

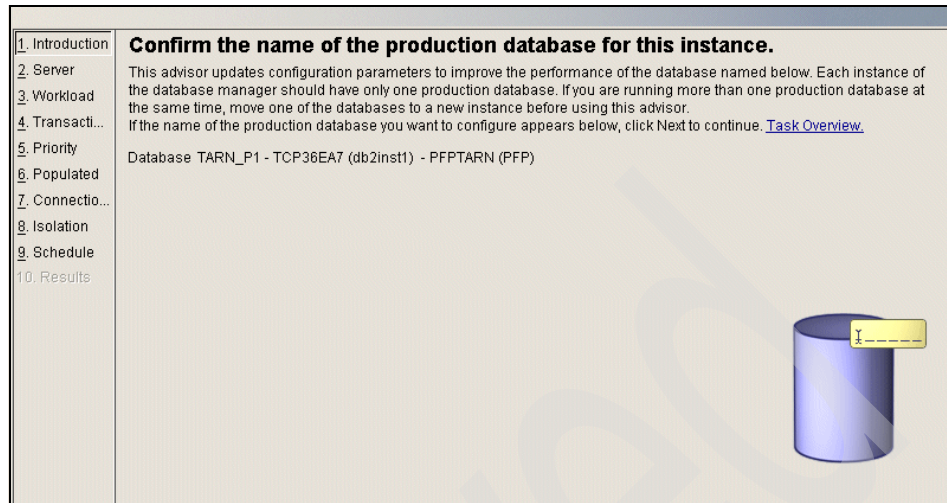


Figure 8-6 Configuration Advisor: specify the database

The specifications of the server has to be specified (see Figure 8-7).

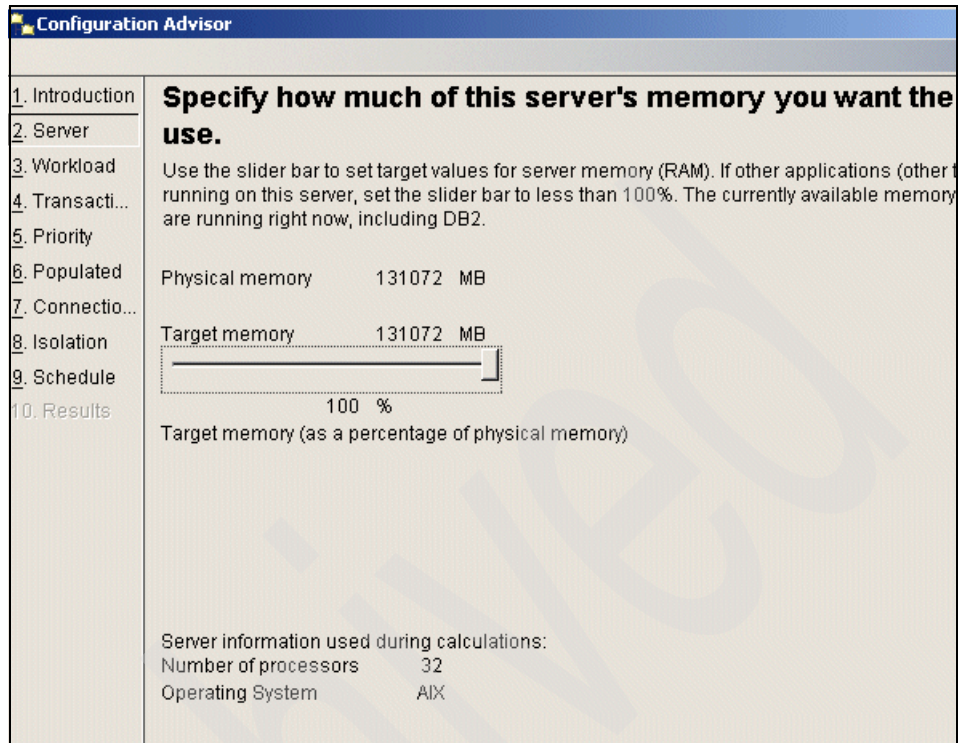


Figure 8-7 Configuration Advisor: specify server specifications

You can automatically set up configuration parameters, which are the most suitable to a specific workload chosen between three kinds of workloads (see Figure 8-8):

- ▶ Queries
- ▶ Transactions
- ▶ Mixed

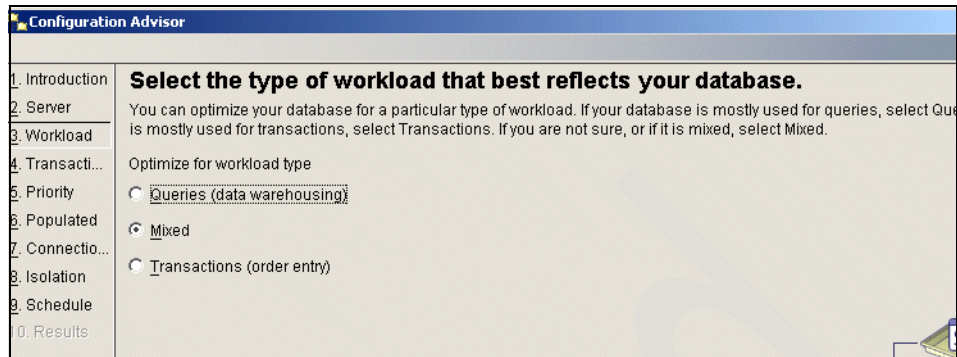


Figure 8-8 Configuration Advisor: specify the workload

Configuration Advisor will ask for:

- A transactions profile to get the number of SQL statements per commit, and the number of transactions per minute (see Figure 8-9).

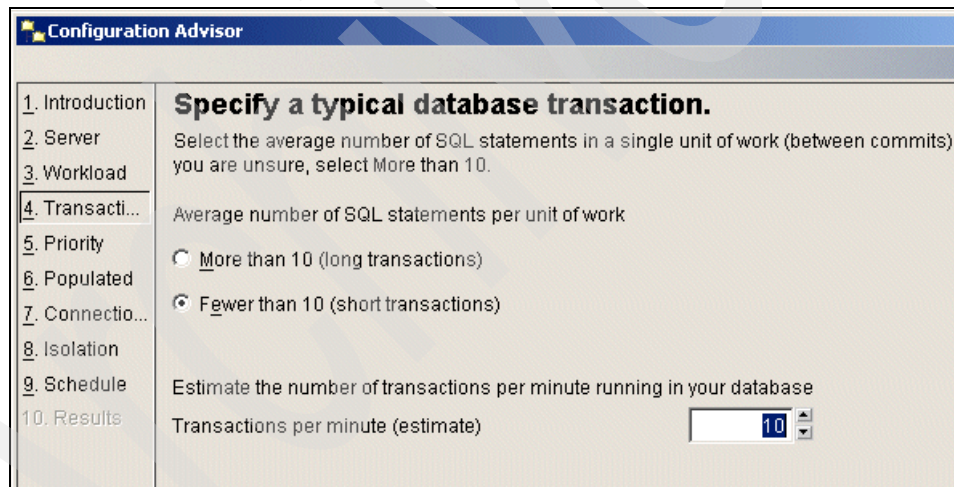


Figure 8-9 Configuration Advisor: Specify the typical database transaction

- Priority to know if priority is given to performances or recovery (see Figure 8-10).

Configuration Advisor	
<ul style="list-style-type: none"> 1. Introduction 2. Server 3. Workload 4. Transacti... 5. Priority 6. Populated 7. Connectio... 8. Isolation 9. Schedule 10. Results 	<p>Specify a database administration priority.</p> <p>Specify whether you want to optimize for transaction performance (more transactions per m... If you are not sure, or if both are equally important, select Both.</p> <p>Optimize for:</p> <p> <input type="radio"/> Faster transaction performance (slower recovery) <input checked="" type="radio"/> Both <input type="radio"/> Faster database recovery (slower transactions) </p>

Figure 8-10 Configuration Advisor: Specify the database administration priority

- Data population to know if the data volumes to load is increasing (see Figure 8-11).

Configuration Advisor	
<ul style="list-style-type: none"> 1. Introduction 2. Server 3. Workload 4. Transacti... 5. Priority 6. Populated 7. Connectio... 8. Isolation 9. Schedule 10. Results 	<p>Specify whether the database is populated with data.</p> <p>The volume of data in the database can affect the recommended values. Run this advisor database increases or decreases significantly.</p> <p>Is the database populated with data?</p> <p> <input checked="" type="radio"/> Yes <input type="radio"/> No </p>

Figure 8-11 Configuration Advisor: Specify database population

- Number of connections (see Figure 8-12).



Figure 8-12 Configuration Advisor: Specify the number of connections

- Isolation level (see Figure 8-13).

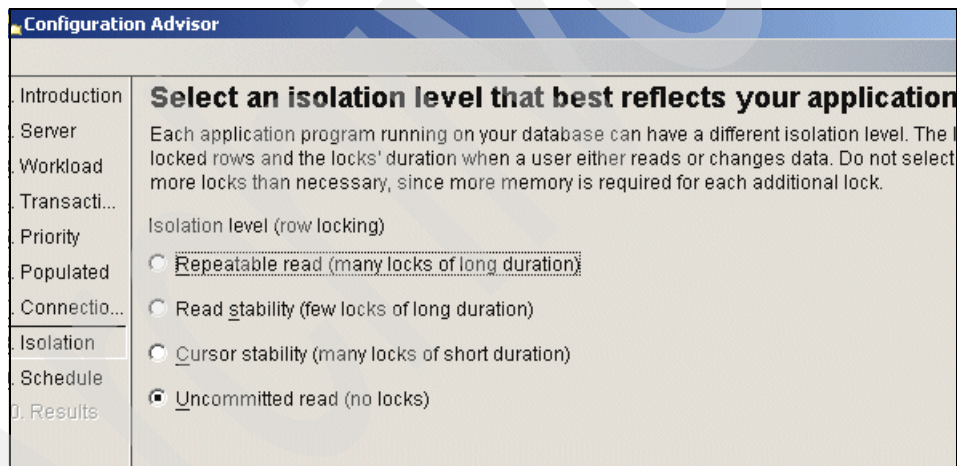


Figure 8-13 Configuration Advisor: Specify isolation

- Scheduling to define when to launch the automatic configuration run (see Figure 8-14).

Configuration Advisor

1. Introduction
2. Server
3. Workload
4. Transacti...
5. Priority
6. Populated
7. Connectio...
8. Isolation
9. **Schedule**
10. Results

Scheduling task execution...

You can select whether to execute the commands immediately or create a task in the Task Center to schedule task execution and maintain its history.

☐ Run now without saving task history

☒ Create this as a task in the Task Center

Run System: tarn_p1

Scheduler System: TARN_P1 Advanced...

Task name: Configuration Advisor - 20/12/02 14:3

☐ Save task only

☒ Save and run task now

☐ Schedule task execution

Details

Change...

Runtime authorization

User ID:

Password:

Figure 8-14 Configuration Advisor: specify scheduling

Based on your answers, Configuration Advisor displays configuration parameters recommendations, and it is up to you to accept them and set them in the database configuration parameters file.

The recommendations made by the Configuration Advisor wizard should be used as a starting point when you create a new database, or when an existing database has been changed significantly and you need to find better values for configuration parameters. Use the suggested values as a base on which to make further adjustments as you try to optimize the performance of your database.

8.3.4 Design Advisor

The Design Advisor can assist in designing and defining suitable indexes for the tables according to the workload you have, or help to verify if a virtual index is appropriate for the database system. It is particularly beneficial for designing indexes for very large tables, to know if the index to create is really helpful.

For tables with large volumes of data, creating an index is time consuming and requires disk space. Design Advisor helps to optimize workload performance by creating a set of indexes recommended by **db2adv**. It can work on virtual indexes with no need to create real indexes.

The Design Advisor replaces the Create Index wizard that was available in V7.

The Design Advisor wizard will determine the best set of indexes for a given workload. A workload contains a set of weighted SQL statements that can include queries as well as updates. The wizard will recommend which new indexes to create, which existing indexes to keep, and which existing indexes to drop. When the Design Advisor recommends indexes, these are not immediately created; instead they are stored in an explain table called **ADVISE_INDEX**. Information can also be inserted into this table using the **db2adv** tool or manually using SQL. The workload information which is considered by the Design Advisor wizard is stored in an additional explain table **ADVISE_WORKLOAD**. The recommendations provided by the Design Advisor are only as good as the input you provide it. To ensure accurate recommendations, the Design Advisor should only be run after your database is populated with data, and the SQL you provide it is an accurate representation of your workload. Before running the Design Advisor wizard against a database, ensure that the explain tables have been created for that database.

8.4 AUTOCONFIGURE command

AUTOCONFIGURE is a new command that recommends and optionally applies new values for bufferpool sizes, database configuration, and database manager configuration. This command provides initial tuning for a database, to which additional tuning can be applied.

AUTOCONFIGURE may also be used with the **CREATE DATABASE** command to configure databases as soon as they are created.

Upgrade to DB2 UDB ESE environment

In this chapter we provide an overview and guidelines for upgrading the database environment to DB2 UDB ESE V8.1.

The topics covered include:

- ▶ Upgrade considerations
- ▶ 64-bit considerations
- ▶ Upgrade procedure

9.1 Upgrade considerations

In this section, we describe:

- ▶ Upgrade recommendations
- ▶ Upgrade restrictions
- ▶ Space considerations for DB2 UDB upgrade
- ▶ Recording system configuration for DB2 UDB upgrade
- ▶ Changing the diagnostic error level before DB2 UDB upgrade
- ▶ Verifying that your databases are ready for an upgrade

9.1.1 Upgrade recommendations

In this topic we describe the main upgrade recommendations:

- ▶ Perform hardware and operating system upgrades separately from DB2 UDB upgrade

Performing hardware and operating system upgrades separately from DB2 UDB upgrade allows for easier problem determination should you encounter upgrade difficulties. If you upgrade your software or hardware prior to upgrading to DB2 UDB, ensure that your system is operating acceptably before attempting DB2 UDB upgrade.

Restriction: If you plan to upgrade the DB2 UDB EEE V7.2 32-bit on AIX V4.3.3 to DB2 UDB ESE V8.1 64-bit on AIX V5, you must upgrade the AIX from V4.3.3 to V5 before the DB2 UDB upgrade, because DB2 UDB ESE V8.1 64-bit requires AIX 5. There is an upgrade strategy example in “Example 2 upgrade strategy” on page 296.

- ▶ Down level server support

As you move your environment from DB2 UDB Version 7 to DB2 UDB Version 8 (if you are in a situation where you can upgrade your DB2 clients to Version 8.1 before you upgrade all of your DB2 UDB servers to Version 8) there are several restrictions and limitations. These restrictions and limitations are not associated with DB2 Connect; nor with zSeries, OS/390, or iSeries database servers. To avoid the known restrictions and limitations, you should upgrade all of your DB2 UDB servers to Version 8.1 before you upgrade any of your DB2 clients to Version 8.1.

- ▶ Benchmark DB2 UDB performance

Run a number of test queries before upgrading DB2 UDB. Record the exact environment conditions when queries are run. Also, keep a record of the `db2exp1n` command output for each test query. Compare the results before

and after upgrade. This practice may help to identify and correct any performance degradation.

- ▶ Devise a plan to back out of an upgrade.

There is no utility to reverse in an upgrade. If you must back out of an upgrade, you might have to remove DB2 UDB Version 8.1 code from your system, reinstall the previous version of DB2 UDB to recreate back-level instances, and restore your database backups. Should you have to back out of a upgrade, current database backups and a detailed record of database and database configuration settings are essential.

- ▶ Upgrade to DB2 UDB Version 8.1 in a test environment.

Upgrade to DB2 UDB Version 8.1 in a test environment before upgrading your production environment. This practice will allow you to address upgrade difficulties and make sure applications and tools work properly before committing your production environment to the upgrade process.

- ▶ Upgrading DataJoiner instances

Before migrating an instance of DataJoiner, DB2 UDB for UNIX, or DB2 UDB for Windows where you are running the Capture or Apply programs for DB2 DataPropagator, be sure to read the migration documentation for DB2 DataPropagator Version 8.1. You must prepare to migrate your replication environment before you migrate the DB2 UDB or DataJoiner instance. You must also perform steps immediately after the migration of your DB2 UDB or DataJoiner instance. Migration documentation for DB2 DataPropagator Version 8.1 can be found at:

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

9.1.2 Upgrade restrictions

In this section we describe a list of upgrade restrictions:

- ▶ Upgrade is only supported from:
 - DB2 UDB Version 6.x or Version 7.x (all platforms supported in Version 6.x and Version 7.x; Linux must be at Version 6 FixPak2.)
 - DB2 DataJoiner V2.1.1 32-bit (AIX, Windows NT, and Solaris operating environment)
- ▶ Issuing the migrate database command from a DB2 UDB Version 8.1 client to migrate a database to a DB2 UDB Version 8.1 server is supported; however, issuing the migration command from an DB2 UDB Version 6 or Version 7 client to migrate a database to a DB2 UDB Version 8.1 server is not supported.
- ▶ When upgrading from DB2 DataJoiner V2.1.1, DB2 Relational Connect is required to support non-IBM data sources.

- ▶ Upgrade between platforms is not supported. For example, you cannot migrate a database from a DB2 UDB server on Windows to a DB2 UDB server on UNIX.
- ▶ Upgrading a partitioned database system that has multiple computers requires that database migration be performed after DB2 UDB Version 8.1 is installed on all participating computers.
- ▶ Windows allows only one version of DB2 UDB to be installed on a computer. For example, if you have DB2 UDB Version 7 and install DB2 UDB Version 8, DB2 UDB Version 7 will be removed during the installation. All instances are migrated during DB2 UDB installation on Windows operating systems.
- ▶ User objects within your database cannot have DB2 UDB Version 8.1 reserved schema names as object qualifiers. These reserved schema names include: SYSCAT, SYSSTAT, and SYSFUN.
- ▶ User-defined distinct types using the names BIGINT, REAL, DATALINK, or REFERENCE must be renamed before migrating the database.
- ▶ You cannot migrate a database that is in one of the following states:
 - Backup pending
 - Roll-forward pending
 - One or more table spaces not in a normal state
 - Transaction inconsistent
- ▶ Restoration of down-level (DB2 UDB Version 6.x or Version 7.x) database backups is supported, but the rolling forward of down-level logs is not supported.
- ▶ Database transactions executed between database backup time and the time DB2 UDB Version 8.1 upgrade is completed, are not recoverable.
- ▶ To upgrade a DB2 UDB Version 7 AIX Version 4 64-bit instance:
 - Upgrade your AIX operating system to AIX Version 5:
 - i. Upgrade your operating system to AIX Version 5.
 - ii. Upgrade DB2 UDB Version 7 with DB2 UDB Version 7 FixPak 4 for AIX 5.
 - iii. Update your instances using the command:


```
/usr/lpp/db2_07_01/instance/db2iupdt
```
 - iv. Ensure your database continues to work. It is not recommended to proceed directly to the next step without confirming that your database works in AIX Version 5 on DB2 UDB Version 7.
 - v. Install DB2 UDB Version 8.1 for AIX Version 5.
 - vi. Migrate your instances using the command:

`/usr/opt/db2_08_01/instance/db2imigr`

- Remain with AIX Version 4:
 - i. Drop your instances.
 - ii. Recreate them as 32-bit instances. You may have to reconfigure your instance parameters.
 - iii. Install DB2 UDB Version 8.1 for AIX Version 4.
 - iv. Migrate your instances using the command:

`/usr/opt/db2_08_01/instance/db2imigr`

9.1.3 Backing up databases before DB2 UDB upgrade

Before you start the upgrade process, it is recommended that you perform an offline backup of your databases. If an error should occur during the upgrade process, database backups are required for recovery.

9.1.4 Space considerations for DB2 UDB upgrade

This section provides the following information about space requirements for DB2 UDB upgrade:

► Table spaces

Ensure that you have sufficient table space for databases you are migrating. System catalog table space is required for both old and new database catalogs during upgrade. The amount of space required will vary, depending on the complexity of the database, as well as the number and size of database objects. General recommendations are in Table 9-1.

Table 9-1 Catalog table space recommendations

Table space	Recommended space
system catalog table space (SYSCATSPACE)	2 x the space currently occupied
temporary table space (TEMPSPACE1 is the default name)	2 x the system catalog table space

To check the size of your table spaces you can use the following commands:

`db2 list database directory`

`db2 connect to database_alias`

`db2 list tablespaces show detail`

For the system catalog table space, free pages should be equal to or greater than used pages. Total pages for the temporary table space should be twice the amount of total pages for the system catalog table space. To increase the amount of space to a DMS table space, you can add additional containers.

► Log file space

You should consider increasing (doubling) the values of *logfilsiz*, *logprimary*, and *logsecond* to prevent log file space from running out. If your system catalog table space is an SMS type of table space, you should consider updating the database configuration parameters that are associated with the log files.

9.1.5 Recording system configuration for DB2 UDB upgrade

It is recommended that you record the database and database manager configuration settings before the DB2 UDB upgrade. Configuration records can be used to verify that an upgrade was successful, and may also be useful in problem determination, should you encounter post-upgrade difficulties.

After you have upgraded DB2 UDB, it is recommended that you compare these records with post-upgrade settings to ensure that the existing settings were upgraded successfully.

This topic lists several database commands. For references to complete command syntax, refer to the related reference section at the end of this section:

► Procedure

- Save a copy of your database configuration settings. Configuration parameters for a database should be the same on each computer in a partitioned database system. If not, save a copy of the database configuration settings for each partition. You can compare configuration settings before and after the upgrade to ensure that they have been upgraded properly. You can retrieve database configuration settings using the **db2 get database configuration for <database_alias>** command. Perform this task for each database you are migrating.
- Save a copy of your database manager configuration settings. You can retrieve database manager configuration settings using the **db2 get database manager configuration** command.
- Save a record of the table spaces for each database you are migrating. You can retrieve a list of table spaces using the **db2 list tablespaces** command.
- Save a list of packages for each database you are migrating. You can retrieve a list of packages using the **db2 list packages** command.

9.1.6 Changing the diagnostic error level before DB2 UDB upgrade

For the duration of upgrade activities, change the diagnostic error level to 4. The diagnostic error level 4 records all errors, warnings, and informational messages. This information can be used for problem determination should you encounter upgrade errors. The DIAGPATH configuration parameter is used to specify the directory that contains the error file, event log file (on Windows only), alert log file, and any dump files that may be generated based on the value of the *diaglevel* parameter.

9.1.7 Verifying that your databases are ready for migration

Ensure that the migration.log, found in the instance owner's home directory, contains the following text:

Version of DB2CKMIG being run:VERSION 8

Enter the **db2ckmig** command to verify that databases owned by the current instance are ready to be migrated. The **db2ckmig** command ensures that:

- ▶ A database is not in a inconsistent state.
- ▶ A database is not in backup pending state.
- ▶ A database is not in rollforward pending state.
- ▶ Table spaces are in a normal state.

9.1.8 Understanding DB2 V8.1 objects terminology

To understand the differences of terminology between DB2 UDB V7 and DB2 UDB V8.1, please refer to Table 9-2. Nodes and node groups are still valid in DB2 V8.1.

Table 9-2 Terminology differences

DB2 UDB V8.1 object name	DB2 UDB V7 object name
Database partition	Node
Database partition group	Node group
Coordinator database partition	Coordinator node
Large table space	Long table space

9.2 64-bit considerations

In this section, we describe:

- ▶ Why 64-bit support?
- ▶ 64-bit restrictions on DB2 UDB V8.1
- ▶ 64-bit client / server compatibilities
- ▶ 64-bit configuration on DB2 UDB V8.1
- ▶ How to check if the DB2 UDB is 32-bit or 64-bit

9.2.1 Why 64-bit support?

The advent of 64-bit computing platforms presents new possibilities for increased performance of database servers, as well as database applications. The 32-bit platforms have an inherent address space limitation of 4 gigabytes (GB) for the kernel, plus user text and data. Removal of this 4 GB limit on the address space of the database servers allows for the creation of larger bufferpools, SORTHEAP, package caches, and other resources that can consume large amounts of memory. This, in turn, leads to much improved performance, especially for sort and input/output (I/O) operations.

Other 32-bit limitations and problems can be removed by using a 64-bit implementation. For example, on AIX, there are approximately ten memory segments available for use in mmap or shmat calls. This directly limits the number of local database connections that a DB2 application can have, because one shared memory segment is required for each local connection. As well as eliminating restrictions on the amount of memory available both on the stack and the heap of an AIX executable, a 64-bit implementation is not subject to potential stack heap collisions.

The availability of both hardware and operating systems capable of using greater than 4 GB of memory means that a 4 GB barrier is a significant limitation to memory-intensive applications and the larger middleware vendors who require large database installations. With 64-bit DB2 UDB, you can utilize much larger bufferpools, SORTHEAP, package cache, and other resources that consume large amount of memory.

Some application developers are also migrating their applications to 64-bit platforms. The 64-bit applications are able to access files that are greater than 2 GB in size. To achieve this level of access with 32-bit applications requires special code or build modifications. For 64-bit applications to work with DB2 family databases, they must have access to 64-bit versions of the DB2 application libraries. Although 64-bit platforms allow both 32-bit and 64-bit processes to coexist, they do not allow intermixing of 32-bit and 64-bit executables and libraries within the same process.

Gain are where DB2 UDB or application can benefit from using more resources beyond the limits of 32-bit computing.

In small scale 32-bit DB2 to 64-bit DB2 comparisons, the performance gains are not visible.

In DB2 UDB ESE V8.1 the common client was implemented. With the common client, you get the best of both worlds; keep 32-bit application and use 64-bit DB2 UDB.

Most of the DB2 UDB V7 64-bit limitations and restrictions are gone in DB2 UDB V8.1. The DB2 UDB V8.1 64-bit restrictions are described in the next section.

9.2.2 64-bit restrictions on DB2 UDB V8.1

The following restrictions are related to updating a 32-bit instance to a 64-bit instance on DB2 UDB V8.1:

- ▶ Remote databases (TYPE != SQL_INDIRECT) that are cataloged at the 32-bit instance are skipped. The local database directories for these databases will not be migrated. But these cataloged remote DB2 UDB V8.1 32-bit databases will remain accessible after the instance migration to DB2 UDB V8.1 64-bit, because of the common client concept.
- ▶ All databases local to the instance that you intend to update must be cataloged before the instance is updated.
- ▶ A 64-bit database image cannot be restored into a 32-bit instance.
- ▶ In a DB2 UDB ESE partitioning environment, all partitions must be 64-bit partitions (not a mix of 32-bit and 64-bit partitions).
- ▶ The following elements are not supported in this release:
 - DB2 Extenders
 - Data Links Manager
- ▶ DB2 UDB ESE V8.1 64-bit requires AIX 5.

9.2.3 64-bit client/server compatibilities

Table 9-3 provides a 64-bit client/server compatibilities table.

Table 9-3 64-bit compatibilities table

Client	Server	Supported?
V6 32-bit (Unix and Windows)	V8 32-bit (Unix and Windows)	YES
V6 32-bit (Unix and Windows)	V8 64-bit (Windows)	YES

Client	Server	Supported?
V6 32-bit (Unix and Windows)	V8 64-bit (Unix)	NO
V7 32-bit (Unix and Windows)	V8 32-bit (Unix and Windows)	YES
V7 32-bit (Unix and Windows)	V8 64-bit (Windows)	YES
V7 32-bit (Unix and Windows)	V8 64-bit (Unix)	NO
V7 64-bit (Unix)	V8 64-bit (Unix)	YES
V7 64-bit (Unix)	V8 64-bit (Windows)	NO
V8 32-bit (Unix and Windows)	V8 32-bit (Unix and Windows)	YES
V8 32-bit (Unix and Windows)	V8 64-bit (Unix and Windows)	YES
V8 32-bit (Unix and Windows)	V7 64-bit (Unix)	NO
V8 64-bit (Unix and Windows)	V7 64-bit (Unix)	YES

Note: In DB2 UDB Server, 32-bit instance and 64-bit instances can coexist in the same environment, since the DB2 UDB and the operating system support 64-bits.

9.2.4 64-bit configuration on DB2 UDB V8.1

Some database manager and database configuration parameters now have higher upper limits, but only for 64-bit releases. To take full advantage of the larger address space available to 64-bit executables, you may need to tune memory-related configuration parameters. Table 9-4 and Table 9-5 denote previous and current upper limits for affected database and database manager configuration parameters.

Table 9-4 Memory-related database manager configuration parameters

Parameter	Previous upper limit	Current upper limit
AGENT_STACK_SZ	1000	Same
APPLHEAPSZ	128	256
ASLHEAPSZ	524288	Same
AUDIT_BUF_SZ	65000	Same
BACKBUFSZ	524288	Same
DOS_RQRIOLBK	65535	Same

Parameter	Previous upper limit	Current upper limit
DRDA_HEAP_SZ	60000	Same
FCM_NUM_ANCHORS	120000	524288
FCM_NUM_BUFFERS	65300	524288
FCM_NUM_CONNECT	120000	524288
FCM_NUM_RQB	120000	524288
JAVA_HEAP_SZ	4096	Same
MIN_PRIV_MEM	112000	Same
MON_HEAP_SZ	60000	Same
PRIV_MEM_THRESH	112000	Same
QUERY_HEAP_SZ	524288	Same
RESTBUFSZ	524288	Same
RQIOBLK	65535	Same
SHEAPTHRES	2097152	2147483647
STMTHEAP	2048	4096
UDF_MEM_SZ	60000	same

Note: *FCM_NUM_ANCHORS*, *FCM_NUM_CONNECT*, *FCM_NUM_RQB* are now self tuning.

Table 9-5 Memory-related database configuration parameters

Parameter	Previous upper limit	Current upper limit
APP_CTL_HEAP_SZ	64000	Same
APPLHEAPSZ	60000	Same
BUFFPAGE	524288	2147483647
CATALOGCACHE_SZ	60000	Same
DBHEAP	60000	524288 (for both 32-bit and 64-bit)
ESTORE_SEG_SZ*	1048575	Same
LOGBUFSZ	4096	65535

Parameter	Previous upper limit	Current upper limit
PCKCACHESZ	64000	524288
SORTHEAP	524288	same
STAT_HEAP_SZ	524288	same
STMTHEAP	60000	same
UTIL_HEAP_SZ	524288	same
Note: * This parameter has been disabled for all 64-bit platforms.		

9.2.5 How to check if the DB2 UDB is 32-bit or 64-bit

There are three things you have to consider:

- ▶ Operating system
- ▶ DB2 UDB product code given by **db2level** command
- ▶ Instance level

Operating system

In AIX operating system, you can check if it is running in 32-bit or 64-bit mode, executing the AIX command **bootinfo -y**.

Important: In a partitioned database environment, all partitions involved must be on the same mode (not a mix of 32-bit and 64-bit partitions).

DB2 UDB product code

In AIX operating system, you can check if the DB2 UDB product code installed is 32-bit or 64-bit support executing the **ls1pp -l | grep db2** AIX command. You can check the file sets level output with the Table 9-6 DB2 level information.

Table 9-6 DB2 UDB level information

DB2 UDB version	Signature
DB2 UDB V7 32-bit	7.1.0.xx
DB2 UDB V7 64-bit	7.1.1.xx
DB2 UDB V7 64-bit AIX V5	7.1.2.xx
DB2 UDB V8.1 32-bit	8.1.0.xx
DB2 UDB V8.1 64-bit	8.1.1.xx
DB2 UDB V8 64-bit AIX 5	8.1.2.xx

Instance level

To check what level your instance is, you can run the **db2level** command logged in with the instance owner user. You can check the instance information output with the Table 9-6 DB2 level information.

9.3 Upgrade procedure

In this section we describe upgrade procedure.

The topics covered include:

- ▶ Environment used in the examples
- ▶ From V7.2 32-bit to v8.1 32-bit on the same machine
- ▶ From V7.2 32-bit to V8.1 32-bit on different machine
- ▶ From V8.1 32-bit to v8.1 64-bit on the same machine
- ▶ Upgrade strategies examples

9.3.1 Environment used in the examples

For the upgrade examples we use two environments:

- ▶ SP environment example
- ▶ p690 environment example

SP environment example

We are using two SP nodes with the following configuration:

- ▶ Products installed:
 - AIX 5 ML3
 - DB2 UDB EEE V7.2 (FixPak7)
 - DB2 UDB ESE V8.1 GA

- ▶ Instance and database partitions definition

We defined six database partitions across 2 SP nodes, making three database partitions per SP node. The DP0 is the catalog partition.

Figure 9-1 shows the database partitions in the SP environment.

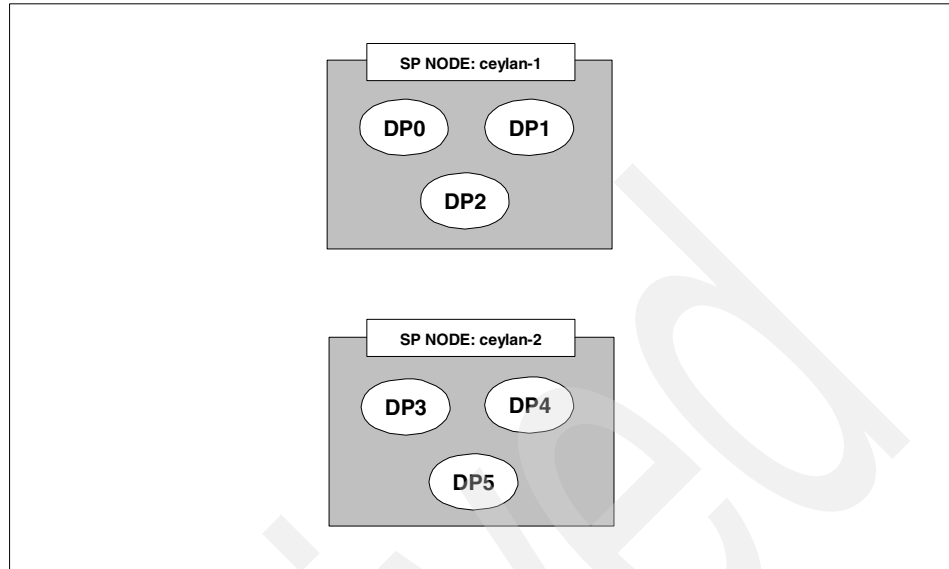


Figure 9-1 SP database partitions definition.

p690 Environment example

We are using one p690 with the following configuration:

- ▶ Products installed:
 - AIX 5 ML3
 - DB2 UDB ESE V8.1 GA
- ▶ Instance and database partitions

We defined six database partitions on a p690 machine where DP0 is the catalog partition. Figure 9-2 shows the p690 database partitions definition.

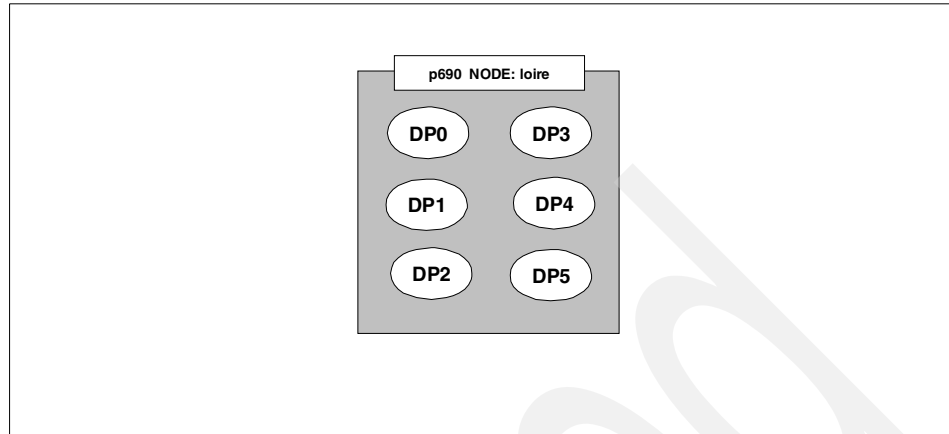


Figure 9-2 p690 database partitions definition.

9.3.2 From V7.2 32-bit to v8.1 32-bit on the same machine

In this section we provide the steps to migrate the database from DB2 UDB EEE V7.2 32-bit instance to DB2 UDB ESE V8.1 32-bit instance on the same environment as shown in Figure 9-3.

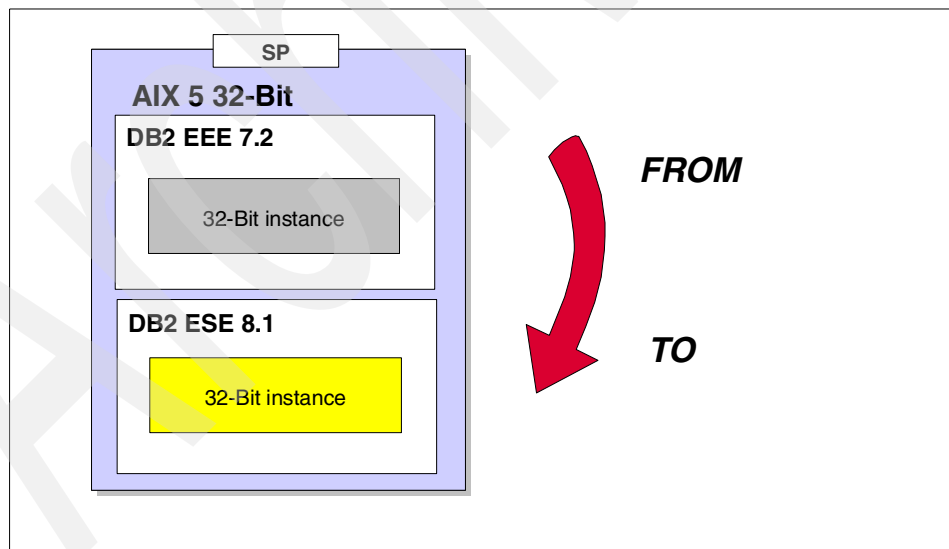


Figure 9-3 From V7.2 32-bit to V8.1 32-bit on the same machine

The topics covered in this section are:

- ▶ Upgrading DB2 UDB
- ▶ Migrating instances
- ▶ Migrating databases
- ▶ Upgrading the DB2 Administration Server

Upgrading DB2 UDB

This topic lists the steps for upgrading to DB2 UDB ESE V8.1 which includes:

- ▶ Prerequisites
- ▶ Procedure

Prerequisites:

Ensure that you review:

- ▶ 9.1.2, “Upgrade restrictions” on page 265
- ▶ 9.1.1, “Upgrade recommendations” on page 264
- ▶ 9.1.4, “Space considerations for DB2 UDB upgrade” on page 267

Procedure:

To upgrade DB2 UDB:

1. Record configuration settings before DB2 UDB upgrade.
2. Change the diagnostic error level.
3. Take the DB2 UDB server offline for DB2 UDB upgrade.
4. Back up your databases.
5. **Optional:** If you will be using replication, you must archive all of the DB2 log files.
6. Install the DB2 UDB ESE V8.1 product.
7. Migrate instances.
8. **Optional:** If you have created a DB2 UDB tools catalog, and want to use your existing pre-Version 8.1 scripts and schedules (for the Control Center), you must migrate the DB2 Administration Server.
9. Migrate databases.

Migrating instances

This topic includes:

- ▶ Overview
- ▶ Prerequisites
- ▶ Procedure
- ▶ Procedure example

Overview

This task is part of the main task of DB2 UDB upgrade.

You can migrate existing DB2 UDB Version 6 or DB2 UDB Version 7 instances using the **db2imigr** command. Migrating instances is done after installing DB2 UDB Version 8.1.

The **db2imigr** command does the following:

- ▶ Checks cataloged databases owned by the instance to make sure they are ready for upgrade.
- ▶ Runs the **db2icrt** command to create the DB2 UDB Version 8.1 instance.
- ▶ Updates system and local database directories to a Version 8.1 format.

Merges the pre-DB2 UDB Version 8.1 database manager configuration with DB2 UDB Version 8.1 database manager configuration.

Prerequisites

You must be logged in as a user with root authority.

Before running the **db2imigr** command, it is recommended:

- ▶ That /tmp have up to 70 percent free space. The instance migration trace file is written to /tmp.
- ▶ That you run manually and resolve any problems prior to running **db2imigr**. **db2imigr** will not migrate as long as **db2ckmig** finds problems.

Procedure

To migrate an instance:

1. Migrate instances using the **db2imigr** command:

```
/DB2DIR/instance/db2imigr [-u fencedID] InstName
```

where **DB2DIR** is `/usr/opt/db2_08_01` on AIX and **-u fenced ID** is the user under which the fenced user-defined functions (UDFs) and stored procedures will execute. This parameter is required only when migrating from a client instance to a server. **InstName** is the login name of the instance owner.

Note: The **db2imigr** utility renames the DB2 UDB V7 sqllib directory to `sqllib_v7` and creates a new one for DB2 UDB V8.1.

Important: If you have migrated from a non-partitioned version of DB2 UDB to a partitioned version of Enterprise Server Edition, you must update your instances to a partitioned format using the **db2iupdt** command.

Procedure example

This procedure example is based on the instance migration from DB2 UDB EEE V7.2 to DB2 UDB ESE V8.1 on “SP environment example” on page 275.

It is described in three steps:

1. Before instance migration

Before DB2 UDB V7 instance migration, the **db2level** output shows the DB2 UDB V7 information as in Example 9-1. And the **db2 list db directory** command output shows the database release level related to DB2 UDB V7 for local and remote DB2 UDB V7 databases, as shown in Example 9-1. The local database is DB2 UDB V7 and the remote database is TPCD.

Example 9-1 db2level output from DB2 UDB v7 instance before migration

```
$ db2level
DB21085I  Instance "db2v7" uses DB2 code release "SQL07024" with level
identifier "03050105" and informational tokens "DB2 v7.1.0.60", "s020313" and
"U481406"
```

Example 9-2 db2 list database directory output before migration

```
$ db2 list database directory
```

System Database Directory

Number of entries in the directory = 2

Database 1 entry:

Database alias	= DB2V7DB
Database name	= DB2V7DB
Local database directory	= /db2v7db
Database release level	= 9.00
Comment	=
Directory entry type	= Indirect
Catalog node number	= 0

Database 2 entry:

Database alias	= TPCD
Database name	= TPCD
Node name	= TPCD
Database release level	= 9.00
Comment	=
Directory entry type	= Remote
Catalog node number	= -1

2. Instance migration

The **db2imigr** utility executes **db2ckmig** first to verify that databases owned by the current instance are ready to be migrated, and if so, the instance migration process is started as shown in Example 9-3.

Example 9-3 db2imigr utility execution

```
205{ceylan-1:root}/usr/opt/db2_08_01/instance -> ./db2imigr db2v7
```

```
db2ckmig was successful. Database(s) can be migrated.
```

```
DBI1070I Program db2imigr completed successfully.
```

3. After migration

After the instance migration, the **db2level** output shows the new instance information for DB2 UDB V8.1 as shown in Example 9-4. The **list db directory** command output also shows the new database release level for DB2 UDB V8.1 as in Example 9-5.

Example 9-4 db2level output from DB2 UDB v7 instance after migration

```
$ db2level
DB21085I Instance "db2v7" uses "32" bits and DB2 code release "SQL08010" with
level identifier "01010106".
Informational tokens are "DB2 v8.1.1.0", "s021023", "", and FixPak "0".
Product is installed at "/usr/opt/db2_08_01".
```

Example 9-5 db2 list db directory output after DB2 UDB v7 instance migration

```
$ db2 list db directory
```

```
System Database Directory
```

```
Number of entries in the directory = 2
```

```
Database 1 entry:
```

Database alias	= DB2V7DB
Database name	= DB2V7DB
Local database directory	= /db2v7
Database release level	= a.00
Comment	=
Directory entry type	= Indirect
Catalog database partition number	= 0

```
Database 2 entry:
```

Database alias	= TPCD
----------------	--------

Database name	= TPCD
Node name	= TPCD
Database release level	= a.00
Comment	=
Directory entry type	= Remote
Catalog database partition number	= -1

The database release level shows the information for DB2 UDB V8.1, but the local database is still in DB2 UDB V7.2 level. The TPCD remote database is accessible, since it belongs to another instance as shown in Example 9-6, but the db2v7db local database is still inaccessible until you migrate it, as shown in the Example 9-7.

The database migration is the next step in this section.

Example 9-6 Remote DB2 UDB V7 database connection

```
$ db2 connect to tpcd user db2v7 using db2v7
```

Database Connection Information

Database server	= DB2/6000 7.2.4
SQL authorization ID	= DB2V7
Local database alias	= TPCD

Example 9-7 Local database connection before database migration

```
$ db2 connect to db2v7db
SQL5035N The database requires migration to the current release.
SQLSTATE=55001
```

Migrating databases

In this section, we describe:

- ▶ Prerequisites
- ▶ Procedure
- ▶ Procedure example

Prerequisites

- ▶ The DB2 instance must be started
- ▶ You require SYSADM authority

Procedure

To migrate a DB2 UDB database:

1. Migrate the database using the **db2 migrate database** command:

migrate database <database alias> **user** <username> **using** <pw>

where:

database database-alias

Specifies the alias of the database to migrated to the currently installed version of the database manager

user username

Identifies the user name under which the database is to be migrated

using password

The password used to authenticate the user name. If the password is omitted, but a user name was specified, the user is prompted to enter it.

2. **Optional:** Update statistics. When database migration is complete, old statistics that are used to optimize query performance are retained in the catalogs. However, DB2 UDB Version 8.1 has statistics that are modified or do not exist in DB2 UDB Version 6 or DB2 UDB Version 7. To take advantage of these statistics, you may want to execute the **RUNSTATS** command on tables, particularly those tables that are critical to the performance of your SQL queries.
3. **Optional:** Rebind packages. During database migration, all existing packages are invalidated. After the migration process, each package is rebuilt when it is used for the first time by the DB2 UDB Version 8.1 database manager. You can run the **db2rbind** command to rebuild all packages stored in the database.
4. **Optional:** Revoke EXECUTE privileges on external stored procedures that contain SQL data access from PUBLIC. During database migration, EXECUTE privileges are granted to PUBLIC for all existing functions, method, and external stored procedures. This will cause a security exposure for external stored procedures that contain SQL data access, which allow users to access SQL objects for which they would not otherwise have privileges. Revoke the privileges by entering the **db2undgp - r** command.
5. **Optional:** Migrate DB2 EXPLAIN tables.
6. **Optional:** If you recorded configuration settings before migration, you might want to compare pre-migration configuration settings to current configuration settings to verify successful migration. Verify:
 - Database configuration parameter settings
 - Database manager configuration parameter settings
 - Table space records
 - Packages records

Procedure examples

This procedure example is based on the database migration from DB2 UDB EEE V7.2 to DB2 UDB ESE V8.1 on “SP environment example” on page 275.

The db2v7db database migration is executed with **migrate database** command using a SYSADM user as shown in Example 9-8.

Example 9-8 Migrate database command execution

```
$ db2 migrate database db2v7db
```

```
DB20000I The MIGRATE DATABASE command completed successfully.
```

Now the database d2v7db is accessible and the new Database Server information is shown as in Example 9-9.

Example 9-9 Connect command

```
$ db2 connect to db2v7db
```

Database Connection Information

Database server	= DB2/6000 8.1.0
SQL authorization ID	= DB2V7DB
Local database alias	= DB2V7DB

Tip: Back up your databases after migration.

Upgrading the DB2 Administration Server (DAS)

This topic lists the steps for upgrading DB2 Administration Server (DAS) and it includes:

- Overview
- Prerequisites
- Procedure

Overview

If you created a DB2 tools catalog on your DB2 UDB Version 8.1 system and want to use your existing pre-Version 8.1 scripts and schedules (from the Control Center) that were created in the pre-Version 8.1 DB2 Administration Server (DAS), you must upgrade the DAS to Version 8.1.

On UNIX, this upgrade must be done manually after the DB2 tools catalog has been created, either during the installation or at a later time.

Prerequisites

You must have:

- ▶ An existing DB2 tools catalog.
- ▶ DAS ADM authority to upgrade the pre-Version 8.1 information into the DB2 tools catalog.

Procedure

To migrate a pre-Version 8.1 DAS to the DB2 tools catalog, enter the command:

```
dasmigr <previous_das_name> <new_das_name>
```

where `previous_das_name` represents the name of the pre-Version 8.1 DAS instance, and `new_das_name` represents the name of the new Version 8.1 DAS.

9.3.3 From V7.2 32-bit to V8.1 32-bit on different machine

In this section we describe the ways and how to upgrade a DB2 UDB EEE V7.2 32-bit instance to DB2 UDB ESE v8.1 32-bit instance on another machine, as shown in Figure 9-4.

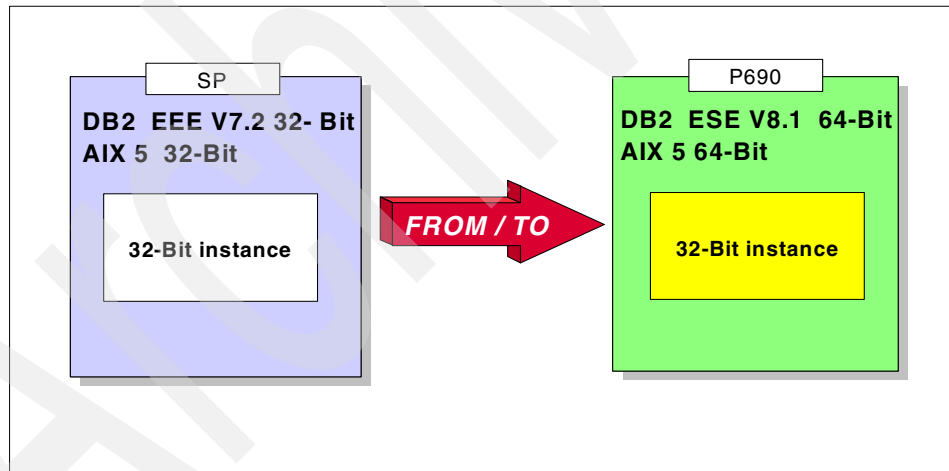


Figure 9-4 From V7.2 32-bit to V8.1 32-bit on different machine

The topics covered in this section are:

- ▶ Prerequisites
- ▶ Procedure

Prerequisites

Ensure that you review:

- ▶ 9.1.2, “Upgrade restrictions” on page 265
- ▶ 9.1.1, “Upgrade recommendations” on page 264

Procedure

There are two ways you can upgrade a DB2 UDB EEE V7.2 database to DB2 UDB ESE V8.1 on another machine:

- ▶ The backup and restore method
- ▶ Recreating new database method

Backup and restore method

The backup and restore method is based on the following tasks:

- ▶ **In DB2 UDB EEE V7.2 environment:**
 - a. Record all configuration settings.
 - b. Ensure that your database is consistent for the migration. You can use the **db2ckmig** utility.
 - c. Back up your databases in offline mode. You will use these offline backup image to restore it on DB2 UDB ESE V8.1 new environment.

Restriction: Restoration of down-level (DB2 UDB Version 6.x or Version 7.x) database backups is supported, but the rolling forward of down-level logs is not supported. Off-line backups must be used.

- d. Make the DB2 offline backup images from all partitions available to the DB2 UDB ESE V8.1 machine.
- ▶ **In DB2 UDB ESE V8.1 environment:**
 - a. Install your DB2 UDB ESE V8.1 server.
 - b. Create the instance where the migrated database will belong to.
 - c. Ensure that the number of database partitions definition is the same as it is defined on the previous environment.

Attention: The database partitions definition must be the same as it is defined in the DB2 UDB V7.2 environment.

If the number of database partitions in DB2 UDB V8.1 is smaller than in DB2 UDB V7.2, the missing database partitions will not be restored and the database will be damaged.

After the migration, you can add the extra database partitions to the new DB2 UDB V8.1 database.

- d. Ensure that the database directory where the migrated database will reside is created.
- e. Ensure that all containers paths are created to allow the database restoration.
- f. Execute the **restore** command on the catalog partition first, and then on the other partitions as shown in Example 9-10 (which is based on the database migration from DB2 UDB EEE V7.2 on “SP environment example” on page 275 to DB2 UDB ESE V8.1 on “p690 Environment example” on page 276.)

Example 9-10 Restore database example

```
$ db2_all "<<+0<db2 restore database DB2V7DB from /db2_backup/v7 taken at  
2002111516 to /DB2V8 into DB2V8DB"
```

```
SQL2519N The database was restored but the restored database was not migrated  
to the current release. Error "1088" with tokens  
"/db2v7home/db2v7/sqllib/bnd/db2schema.bnd" is returned.  
loire: db2 restore database ... completed rc=4
```

```
$ db2_all "<<-0<db2 restore database DB2V7DB from /db2_backup/v7 taken at  
2002111516 into DB2V8 without prompting" <
```

```
rah: omitting logical node 0
```

```
SQL2517W The restored database was migrated to the current release.  
loire: db2 restore database ... completed rc=2
```

```
SQL2517W The restored database was migrated to the current release.  
loire: db2 restore database ... completed rc=2
```

```
SQL2517W The restored database was migrated to the current release.  
loire: db2 restore database ... completed rc=2
```

```
SQL2517W The restored database was migrated to the current release.
```

loire: db2 restore database ... completed rc=2

SQL2517W The restored database was migrated to the current release.

loire: db2 restore database ... completed rc=2

Tip: The 1088 error means:

SQL1088W The database was created, but an error occurred while binding the utilities. The utilities are not bound to the database.

It means that **db2rbind** command must be executed after the database restoration.

Important: In order to migrate the database to DB2 UDB ESE V8.1 by restoring a backup, the partitions need to be restored serially, that is, the parallel option (; or ||) should not be used since the MIGRATE DATABASE requires an exclusive connection to the database.

By running this in parallel, all partitions are attempting to get an exclusive database connection across all partitions and receiving the message: SQL1035N A database is currently in use.

- g. Execute **db2rbind** utility against the new DB2 UDB ESE V8.1 database.
- h. Set the configuration settings if needed.

Recreating a new database method

If you decide to recreate the database on DB2 UDB ESE V8.1 environment, the based tasks are:

► **In DB2 UDB ESE V7.2 environment:**

- a. Record all configuration settings.
- b. Ensure that your database is consistent for the migration. You can use the **db2ckmig** utility.
- c. Back up your databases for safety.
- d. Execute **db2look** utility with **-e** option to get the database DDL.

Tip: You can use the command:

```
db2look -d <dbalias> -e -a -o <DDL output file> -l -xd -f
```

The **db2look** utility will not collect the statement for the three default table spaces creation: SYSCATSPACE, TEMPSPACE1 and USERSPACE1.

You can modify the DDL output file before running it on the new environment.

- e. Export data from all tables only in Non-delimited ASCII (ASC) and Delimited ASCII (DEL) files since only ASC and DEL format can be partitioned.

Restriction: PC/IXF files cannot be partitioned. To load a PC/IXF file into a multiple partition table, you can first load it into a single-partition table, you can first load it into a single-partition table, and then perform a load operation using the CURSOR file type to move the data into a multiple partition table.

► **In DB2 UDB ESE V8.1 environment:**

- a. Install your DB2 UDB ESE V8.1 server.
- b. Create the 32-bit instance where the new database will belong to.
- c. Create the new database.
- d. Ensure that all containers paths specified in DDL output file are created to allow the objects creation.
- e. Connect to the new database.
- f. Create the objects using the DDL output file from the DB2 UDB V7.2 database.

Tip: To execute the statements in DDL output file:

```
db2 -tvf <DDL output file> > <output file name>
```

- g. Set the configuration settings if needed.
- h. Load the tables. For more details about loading data, see Chapter 3., “LOAD and populate the data warehouse in parallel” on page 111.
- i. Back up the database.

9.3.4 From V8.1 32-bit to v8.1 64-bit on the same machine

In this section we provide the steps to upgrade a DB2 UDB ESE V8.1 32-bit instance to a DB2 UDB ESE V8.1 64-bit instance on the same machine as shown in Figure 9-5.

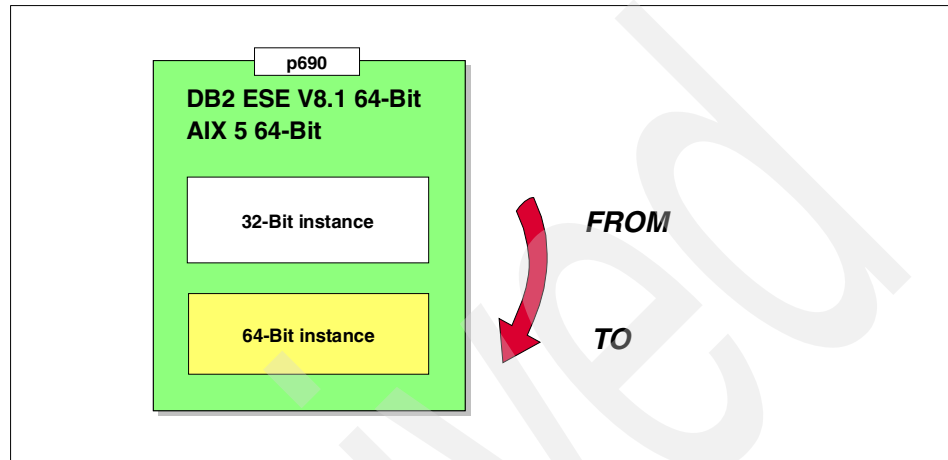


Figure 9-5 From V8.1 32-bit to V8.1 64-bit on the same machine

The topics included are:

- ▶ Prerequisites
- ▶ Procedure
- ▶ Procedure example

Prerequisites

Ensure that you review 9.2, “64-bit considerations” on page 270.

Attention: To have a 64-bit instance, both DB2 UDB V8.1 product and the operating system must support 64-bit.

Procedure

In this section we describe the 64-bit instance upgrade procedure.

1. Back up your databases.

Attention: There is no way to undo the word size upgrade of an instance, so back up all local databases before attempting a word size instance update.

2. You must be logged in as a user with root authority.
3. Update the 32-bit instance to 64-bit instance using the **db2iupdt** command:
`/DB2DIR/instance/db2iupdt -w 64 InstName`

where

DB2DIR is `/usr/opt/db2_08_01` on AIX.

InstName is the instance name that will be migrated.

Attention: You cannot revert to a 32-bit instance once you have migrated to a 64-bit instance. To once again work with 32-bit instances, you must drop your 64-bit instances and databases, recreate 32-bit instances, and restore your most recent 32-bit backup images.

4. Back up all your local databases after updating the word size of the instance to 64-bit.

Tip: Forward recoverable databases (those whose LOGRETAIN database configuration parameter is set to ON) are put in *backup pending* state, and you must back them up once the instance has updated.

Note: If you currently use DB2 UDB EEE 7.2 32-bit instance with many partitions to fully utilize a large machine with lots of memory, should you move to DB2 UDB ESE V8.1 single partition 64-bit instance? You must consider the following:

- ▶ Size of largest table
- ▶ Transaction logging
- ▶ Data movement
- ▶ Workload characteristics
- ▶ Other benefits of shared-nothing architecture

Procedure example

This procedure example is based on the DB2 UDB ESE V8.1 32-bit instance to DB2 UDB ESE V8.1 64-bit instance on “p690 Environment example” on page 276.

1. Before instance migration

Before the 32-bit instance db2v8 migration the **db2level** command output shows the DB2 UDB V8.1 32-bit information as shown in Example 9-11.

Example 9-11 db2level output before migration

```
$ db2level
DB21085I  Instance "db2v8" uses "32" bits and DB2 code release "SQL08010" with
level identifier "01010106".
Informational tokens are "DB2 v8.1.1.0", "s021023", "", and FixPak "0".
Product is installed at "/usr/opt/db2_08_01"
```

The **list db directory** output shows db2v8db local database and db32bit remote database also from a 32-bit instance. The **connect** command output shows the database server information as shown in Example 9-12.

Example 9-12 Connect output before migration

```
$ db2 connect to db2v8db

Database Connection Information

Database server          = DB2/6000 8.1.0
SQL authorization ID     = DB2V8DB
Local database alias     = DB2V8DB

$ db2 connect to db32bit

Database Connection Information

Database server          = DB2/6000 8.1.0
SQL authorization ID     = DB32BIT
Local database alias     = DB32BIT
```

2. Instance migration

We executed the **db2iupdt** command with **-w 64** option using root authority as shown in Example 9-13.

Example 9-13 db2iupdt command execution

```
{loire:root}/usr/opt/db2_08_01/instance -> ./db2iupdt -w 64 db2v8

DBI1070I Program db2iupdt completed successfully.
```

3. After the migration

After the 32-bit instance db2v8 migration, the **db2level** command output now shows the DB2 UDB V8.1 64-bit information as in Example 9-14.

Example 9-14 db2level output after migration

```
$ db2level
DB21085I  Instance "db2v8" uses "64" bits and DB2 code release "SQL08010" with
level identifier "01010106".
```

Informational tokens are "DB2 v8.1.1.0", "s021023", "", and FixPak "0".
Product is installed at "/usr/opt/db2_08_01".

Now the connection command output to the local db2v8db database shows a new database server information as in Example 9-15.

Example 9-15 Connect output after instance migration

```
$ db2 connect to db2v8db
```

Database Connection Information

```
Database server      = DB2/AIX64 8.1.0
SQL authorization ID = DB2V8DB
Local database alias = DB2V8DB
```

```
$ db2 connect to db32bit
```

Database Connection Information

```
Database server      = DB2/6000 8.1.0
SQL authorization ID = DB32BIT
Local database alias = DB32BIT
```

Tip: Back up your local databases after the instance migration.

9.3.5 Upgrade strategies examples

There are multiple scenarios possible for DB2 UDB upgrade from DB2 UDB EEE 7.2 32-bit to DB2 UDB ESE V8.1 64-bit. These depend on how your current environment is defined and what will be is the target environment.

In this section we describe some upgrade examples and procedure suggestions.

The topics included in this section are:

- ▶ Prerequisites
- ▶ Important considerations
- ▶ Example 1 upgrade strategy
- ▶ Example 2 upgrade strategy

Prerequisites

Ensure that you review:

- ▶ 9.1.2, "Upgrade restrictions" on page 265

- ▶ 9.1.1, “Upgrade recommendations” on page 264
- ▶ 9.2, “64-bit considerations” on page 270

Important considerations

You have to keep in mind the following important considerations:

- ▶ The p690 hardware requires AIX 5.
- ▶ Performing hardware and operating system upgrades separately from DB2 UDB upgrade allows for easier problem determination should you encounter upgrade difficulties. If you upgrade your software or hardware prior to migrating to DB2 UDB, ensure that your system is operating acceptably before attempting DB2 UDB upgrade.
- ▶ You should upgrade all of your DB2 UDB servers to Version 8.1 before you upgrade any of your DB2 clients to Version 8.1. As you move your environment from DB2 UDB Version 7 to DB2 UDB Version 8, if you are in a situation where you upgrade your DB2 clients to Version 8.1 before you upgrade all of your DB2 UDB servers to Version 8, there are several restrictions and limitations. These restrictions and limitations are not associated with DB2 Connect; nor with zSeries, OS/390, or iSeries database servers.
- ▶ There is no direct upgrade path from an existing 32-bit (pre-Version 8.1) instance to a 64-bit instance. You must first upgrade your pre-version 8.1 32-bit instance to Version 8.1, and then update it to a 64-bit instance.
- ▶ A 32-bit database image cannot be restored into a 64-bit instance.
- ▶ In a partitioned database environment, all partitions must be 64-bit partitions (not a mix of 32-bit and 64-bit partitions).
- ▶ For your DB2 UDB ESE V8.1 64-bit upgrade strategy, avoid upgrading the DB2 UDB V7.2 32-bit instance to 64-bit, because of DB2 UDB V7.2 32-bit clients connection incompatibility to databases on 64-bit servers.
- ▶ Test the DB2 UDB functionality every upgrade step.
- ▶ You can have 32 and 64-bit instances if the DB2 UDB and AIX is prepared to use 64-bit.
- ▶ Each upgrade step, always back up your databases.

Example1 upgrade strategy

In this section we provide upgrade strategy suggestions and include the following topics:

- ▶ Environment
- ▶ Upgrade strategy suggestion

Environment

The Example 1 upgrade environment is from DB2 UDB EEE V7.2 32-bit to DB2 UDB ESE V8.1 64-bit on the same machine as shown in Figure 9-6 Example 1 upgrade environment.

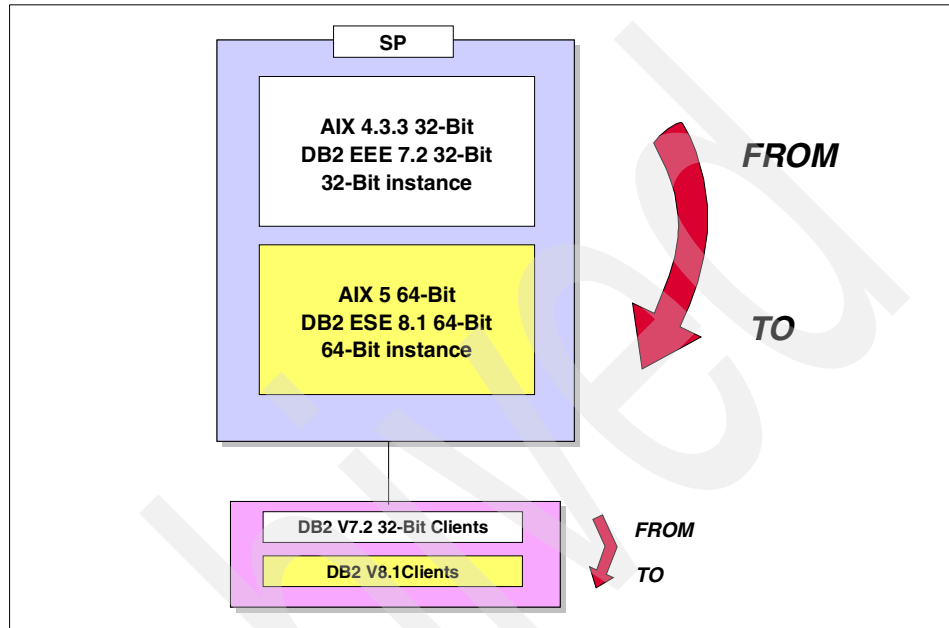


Figure 9-6 Example 1 upgrade environment

Upgrade strategy suggestion

The upgrade strategy suggestion is:

1. Upgrade AIX 4.3.3 32-bit to AIX 5 32-bit.

Attention: If you upgrade your software or hardware prior to upgrading DB2 UDB, ensure that your system is operating acceptably before attempting DB2 UDB upgrade.

2. Install DB2 UDB ESE V8.1 64-bit for AIX and migrate the DB2 UDB V7 32-bit instance to DB2 UDB V8.1 32-bit instance. The procedure for this step is described in 9.3.2, "From V7.2 32-bit to v8.1 32-bit on the same machine" on page 277.

Important: You should first migrate the DB2 UDB V7 32-bit instance to the DB2 UDB V8.1 32-bit instance, before migrating to DB2 UDB V8.1 64-bit. It is not recommended to migrate the DB2 UDB V7.2 32-bit instance to DB2 UDB V7.2 64-bit instance, because of the DB2 UDB V7 64-bit restrictions and incompatibilities.

3. Test DB2 UDB functionality and the applications.
4. Upgrade DB2 UDB V7 clients to DB2 UDB V8.1.

Restriction: You must upgrade your DB2 UDB V7 32-bit clients to DB2 UDB V8.1 clients before upgrading the DB2 UDB V8.1 32-bit instance to 64-bit, because of the incompatibility between DB2 UDB V7 32-bit clients and 64-bit databases.

5. Upgrade AIX 5 32-bit to AIX 5 64-bit.
6. Migrate the DB2 UDB V8.1 32-bit instance to DB2 UDB V8.1 64-bit instance. The procedure for this step is described in 9.3.4, “From V8.1 32-bit to v8.1 64-bit on the same machine” on page 290.
7. Test DB2 UDB functionality and all the applications.

Example 2 upgrade strategy

In this section we provide:

- ▶ Environment
- ▶ Upgrade strategy suggestion

Environment

The Example 2 upgrade environment is from DB2 UDB EEE 7.2 32-bit to DB2 UDB ESE V8.1 64-bit on a different machine as shown in Figure 9-7 Example 2 upgrade environment.

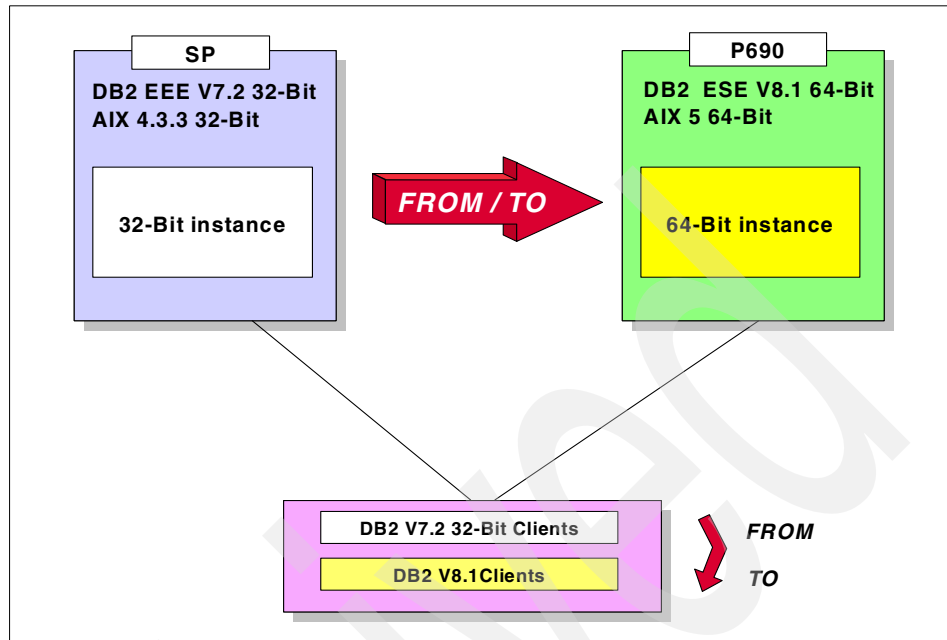


Figure 9-7 Example 2 upgrade environment

Upgrade strategy suggestion

The steps of the upgrade strategy suggestion are:

1. Back up your DB2 UDB EEE V7.2 32-bit databases on SP.
2. Install the DB2 UDB ESE V8.1 64-bit for AIX 5 64-bit on p690, and create a DB2 UDB V8.1 32-bit instance and the migrate the DB2 UDB V7 32-bit database by restoring the images on DB2 UDB V8.1 32-bit instance on p690. The procedure for these steps is described in 9.3.3, "From V7.2 32-bit to V8.1 32-bit on different machine" on page 285.

Important: You should first migrate the DB2 UDB V7 32-bit instance to the DB2 UDB V8.1 32-bit instance, before migrating to DB2 UDB V8.1 64-bit. It is not recommended that you migrate the DB2 UDB V7.2 32-bit instance to DB2 UDB V7.2 64-bit instance because of the DB2 UDB V7 64-bit restrictions and incompatibilities.

3. Test the DB2 UDB functionality and all applications.
4. Upgrade DB2 UDB V7 clients to DB2 UDB V8.1.

Restriction: You must upgrade your DB2 UDB V7 32-bit clients to DB2 UDB V8.1 clients before migrating the DB2 UDB V8.1 32-bit instance to 64-bit because of the incompatibility between DB2 UDB V7 32-bit clients and 64-bit databases.

5. Test the applications.
6. Migrate the DB2 UDB V8.1 32-bit instance to DB2 UDB V8.1 64-bit instance. The procedure for this step is described in 9.3.4, “From V8.1 32-bit to v8.1 64-bit on the same machine” on page 290.
7. Test DB2 UDB functionality and all the applications.

Using DB2 LOAD for the case study

To explore the new functionalities of the **LOAD** utility, we used the database schema and the data generation tool from the TPC-H, as mentioned in 1.4.1, “The case study based on TPC-H” on page 31.

The TPC-H data generation tool, dbgen, was used to create the data to be loaded into the tables. The output of dbgen is a delimited ASCII source file; the vertical bar (|) delimits the data columns.

Planning the load

Before loading the eight tables in the case study, some work has to be done.

The six large multipartition tables (customer, part, supplier, partsupp, orders, and lineitem) will be loaded using **DB2 LOAD**. The two small tables (nation and region) will be stored in a single partition table space and loaded using DB2 INSERT.

The loading sequence will be as follows:

1. The two small tables will be created from input files, which will be generated by dbgen and stored on disk. Those input files will in turn be used by DB2 IMPORT to populate the tables.
2. Because there is insufficient disk space to house the generated input files which would normally be used to load the six large tables, we decided to create named pipes to which dbgen would write the generated load data and from which DB2 LOAD would populate the tables. To facilitate this activity, script **gen_and_load_one.ksh** was created.
3. Those tables on which foreign key and primary key constraints were created are in *CHECK PENDING* state after DB2 LOAD. So, we will need to use DB2 **SET INTEGRITY** to get the tables into normal state.

Load the multipartition tables

The same procedure was used to load each of the six multipartition tables: customer, part, supplier, partsupp, orders, and lineitem. The input was generated by dbgen using a scaling factor of 60 and using 4 input processes for all tables except order and lineitem which used 12. The procedure shown here documents the load of the lineitem table, the largest table of the set.

Script **gen_and_load_one.ksh**, which is documented in Example A-1, was used to load the multipartition tables. The new partitioned load support was employed. For details about the DB2 UDB Version 8.1 partitioned load enhancements, see 3.1.4, “Multipartition LOAD support” on page 119.

Example: A-1 The load script for multipartition tables, gen_and_load_one.ksh

```
#!/usr/bin/ksh
# -----
# Script:   gen_and_load_one.ksh
# Author:   George Latimer, Fourth Millennium Technologies
# Created:  6 November 2002
#
# Description: This script loads the large, multipartition tables. It
#              creates the required named pipes and invokes dbgen and
```

```

#           DB2 LOAD in the background.
#
# Program 'dbgen' is from the TPC Benchmark (c) H specification,
# revision 1.5.0. TPC is "Transaction Processing Performance Council"
# at URL 'www.tpc.org'.
# -----
#

# set -x

export DSS_PATH=/db2work/dbgen_data
export DSS_CONFIG=$DSS_PATH/config           # Location of $DSS_DIST file
export DSS_DIST=dists.dss
export DSS_QUERY=$DSS_PATH/query

DEFAULT_SCHEMA=tpcd

PATH=$PATH:/usr/sys/inst.images/DBGEN # Location of dbgen program

function usage {
    MESSAGE="

    Usage: $SCRIPT_NAME table_id [scaling_factor] [process_count]

    Table_id      Identifies the table to generate and load. Valid values
                  are L, c, O, P, S, and s. (Nation and region will be in
                  single-partition group. The will be generated to and
                  loaded from disk.)

    scaling_factor The TPC-H (c) scaling factor. Default value is '1'.

    process_count  The number of dbgen processes used to generate the
                  input data. Default value is '2'.

    "
    print "$MESSAGE"
    exit 5
    return
}

# Edit input parameters

if [[ $# -eq 0 ]]; then
    print "\nMust specify at least 1 parameter -- TABLE ID."
    usage
elif [[ $# -gt 3 ]]; then
    print "\nToo many parameters specified."
    usage
fi

```

```

TABLE_ID=${1}
SCALE=${2:-1}
COUNT=${3:-2}
SCHEMA=${DEFAULT_SCHEMA:-tpcd}

# Input parameters not supported at this time:
# 'T p' - Generates 2 tables -- parts and partsupp
# 'T l' - Generates 2 tables -- nation and region
# 'T n' - Table too small for autoloading
# 'T r' - Table too small for autoloading
#
case $TABLE_ID in
    L) TABLE=lineitem
        ;;
    c) TABLE=customer
        ;;
    O) TABLE=orders
        ;;
    P) TABLE=part
        ;;
    S) TABLE=partsupp
        ;;
    s) TABLE=supplier
        ;;
    *) print "\nTable ID '$TABLE_ID' is either not supported or invalid."
        usage
        ;;
esac

cd $DSS_PATH

DBGEN_COMMAND="dbgen -s$SCALE -C$COUNT -T$TABLE_ID "
print "\nTPC-H (c) dbgen command is='$DBGEN_COMMAND'."

typeset -i I=1

COMMA=""
while [[ $I -le $COUNT ]]; do
    FIFO_FILE=$DSS_PATH/$TABLE.tbl.$I
    if [[ -e $FIFO_FILE ]]; then
        print "Removing pre-existing FIFO file, $FIFO_FILE."
        /bin/rm -f $FIFO_FILE
        RC=$?
        if [[ $RC -ne 0 ]]; then
            print "Unable to remove '$FIFO_FILE'. ----- EXITING -----"
            exit 10
        fi
    fi
    mkfifo $FIFO_FILE

```



```

        INPUT_LIST="$INPUT_LIST$COMMA $FIFO_FILE"
        COMMA=","
        I=I+1
done

DB2LOAD_COMMAND=$(print "db2 load from $INPUT_LIST of del \"\
    \"modified by coldel| fastparse anyorder \"\
    \"messages /db2home/db2inst1/messages/$TABLE \"\
    \"replace into $SCHEMA.$TABLE nonrecoverable\")
print "\nDB2 LOAD command is '$DB2LOAD_COMMAND'."

OUT1=/tmp/out1.$$
($DBGEN_COMMAND 2>&1; echo "DBGEN_RC=$?") > $OUT1 &

OUT2=/tmp/out2.$$
($DB2LOAD_COMMAND 2>&1; echo "DB2LOAD_RC=$?";db2 terminate) > $OUT2 &

print "\n\nWaiting for all processes to complete...."

wait

print "\n\nDBGEN output follows:\n$(cat $OUT1)\n\n"
print "DB2LOAD output follows:\n$(cat $OUT2)\n\n"

for FILE in $(echo "$INPUT_LIST" | tr ',' ' ');do
    /bin/rm $FILE
done
/bin/rm $OUT1 $OUT2

exit

```

Script **gen_and_load_one.ksh** was executed for each large table. All tables were created using four dbgen processes each, except for orders and line items, which used 12 processes. The commands to run **script gen_and_load_one.ksh** are documented as Example A-2.

Example: A-2 Loading the multipartition tables using gen_and_load_one.ksh

```

{loire:db2inst1}/db2home/db2inst1 -> gen_and_load_one.ksh c 60 4    # customer
{loire:db2inst1}/db2home/db2inst1 -> gen_and_load_one.ksh P 60 4    # part
{loire:db2inst1}/db2home/db2inst1 -> gen_and_load_one.ksh S 60 4    # partsupp
{loire:db2inst1}/db2home/db2inst1 -> gen_and_load_one.ksh s 60 4    # supplier
{loire:db2inst1}/db2home/db2inst1 -> gen_and_load_one.ksh O 60 12   # orders
{loire:db2inst1}/db2home/db2inst1 -> gen_and_load_one.ksh L 60 12   # lineitem

```

The number of rows loaded into each table is documented in Table A-3.

Example: A-3 Number of rows loaded for each table

TABLE NAME	ROW COUNT
customer	9,000,000
part	12,000,000
partsupp	48,000,000
supplier	6,000,000
orders	90,000,000
lineitem	360,011,594
nation	5
region	25

An example of the output from script `gen_and_load_one.ksh` is included as Example A-4.

This is from the `lineitem` table load. Twelve `dbgen` processes create input files. While this load was running, the 32-processor `p690` machine was running at 79% utilization. Looking at the output, you can see that all multipartition options were allowed to default. That is, no **PARTITIONED DB CONFIG** list was specified. Consequently, **LOAD** was allowed to select the partitioning partitions (it picked partitions 1 through 9), the number of output partitions (all partitions where the `lineitem` table is defined), and so forth.

Example: A-4 Output from `gen_and_load_one.ksh` for `lineitem` table

```
{loire:db2inst1}/db2home/db2inst1/scripts -> gen_and_load_one.ksh L 60 12
```

TPC-H (c) `dbgen` command is '`dbgen -s60 -C12 -TL '`.

DB2 LOAD command is '`db2 load from /db2work/dbgen_data/lineitem.tbl.1, /db2work/dbgen_data/lineitem.tbl.2, /db2work/dbgen_data/lineitem.tbl.3, /db2work/dbgen_data/lineitem.tbl.4, /db2work/dbgen_data/lineitem.tbl.5, /db2work/dbgen_data/lineitem.tbl.6, /db2work/dbgen_data/lineitem.tbl.7, /db2work/dbgen_data/lineitem.tbl.8, /db2work/dbgen_data/lineitem.tbl.9, /db2work/dbgen_data/lineitem.tbl.10, /db2work/dbgen_data/lineitem.tbl.11, /db2work/dbgen_data/lineitem.tbl.12 of del modified by coldel| fastparse anyorder messages /db2home/db2inst1/messages/lineitem replace into tpcd.lineitem nonrecoverable'`.

Waiting for all processes to complete....

DBGEN output follows:

TPC-H Population Generator (Version 1.3.0)

Copyright Transaction Processing Performance Council 1994 - 2000

DBGEN_RC=0

DB2LOAD output follows:

Agent Type	Node	SQL Code	Result
LOAD	001	+00000000	Success.
LOAD	002	+00000000	Success.
LOAD	003	+00000000	Success.
LOAD	004	+00000000	Success.
LOAD	005	+00000000	Success.
LOAD	006	+00000000	Success.
LOAD	007	+00000000	Success.
LOAD	008	+00000000	Success.
LOAD	009	+00000000	Success.
LOAD	010	+00000000	Success.
LOAD	011	+00000000	Success.
LOAD	012	+00000000	Success.
LOAD	013	+00000000	Success.
LOAD	014	+00000000	Success.
LOAD	015	+00000000	Success.
LOAD	016	+00000000	Success.
LOAD	017	+00000000	Success.
LOAD	018	+00000000	Success.

LOAD	019	+00000000	Success.
LOAD	020	+00000000	Success.
LOAD	021	+00000000	Success.
LOAD	022	+00000000	Success.
LOAD	023	+00000000	Success.
LOAD	024	+00000000	Success.
LOAD	025	+00000000	Success.
LOAD	026	+00000000	Success.
LOAD	027	+00000000	Success.
LOAD	028	+00000000	Success.
LOAD	029	+00000000	Success.
LOAD	030	+00000000	Success.
LOAD	031	+00000000	Success.
LOAD	032	+00000000	Success.
PARTITION	001	+00000000	Success.
PARTITION	002	+00000000	Success.
PARTITION	003	+00000000	Success.
PARTITION	004	+00000000	Success.
PARTITION	005	+00000000	Success.
PARTITION	006	+00000000	Success.
PARTITION	007	+00000000	Success.
PARTITION	008	+00000000	Success.
PARTITION	009	+00000000	Success.
PRE_PARTITION	000	+00000000	Success.
RESULTS:	32 of 32 LOADs completed successfully.		

Summary of Partitioning Agents:
Rows Read = 360011594
Rows Rejected = 0
Rows Partitioned = 360011594

Summary of LOAD Agents:
Number of rows read = 360011594
Number of rows skipped = 0
Number of rows loaded = 360011594
Number of rows rejected = 0
Number of rows deleted = 0
Number of rows committed = 360011594

DB2LOAD_RC=0
DB20000I The TERMINATE command completed successfully.

During the load of lineitem, the **DB2 LOAD QUERY** was run several times. A partial capture of the output from one such query is shown in Example A-5. Partitions 5 through 32 have been deleted from what is displayed in the example, but the output is similar to the other partitions. Notice that the output shows Check Pending and Load in Progress for the table state (tablestate:).

Example: A-5 Example of LOAD QUERY during load of lineitem table

```
{loire:db2inst1}/db2home/db2inst1/scripts ->db2_all "\|" echo DB2PART=##; db2  
load query table tpcd.lineitem" # The LOAD QUERY command
```

```
loire: DB2PART=0  
loire: SQL6024E Table or index "LINEITEM" is not defined on node "0".  
loire: db2 load query table ... completed ok  
  
loire: DB2PART=1  
loire: SQL3530I The Load Query utility is monitoring "LOAD" progress on  
partition  
loire: "1".  
loire:  
loire: SQL3109N The utility is beginning to load data from file  
loire: "/db2work/dbgen_data/lineitem.tbl.1".  
loire:  
loire: SQL3500W The utility is beginning the "LOAD" phase at time "12-16-2002  
loire: 12:40:03.780421".  
loire:  
loire: SQL3519W Begin Load Consistency Point. Input record count = "0".  
loire:
```

```

loire: SQL3520W Load Consistency Point was successful.
loire:
loire: SQL3532I The Load utility is currently in the "LOAD" phase.
loire:
loire:
loire: Number of rows read          = 1802974
loire: Number of rows skipped       = 0
loire: Number of rows loaded        = 1802974
loire: Number of rows rejected      = 0
loire: Number of rows deleted       = 0
loire: Number of rows committed    = 0
loire: Number of warnings          = 0
loire:
loire: Tablestate:
loire: Check Pending
loire: Load in Progress
loire: db2 load query table ... completed ok

loire: DB2PART=2
loire: SQL3530I The Load Query utility is monitoring "LOAD" progress on
partition
loire: "2".
loire:
loire: SQL3109N The utility is beginning to load data from file
loire: "/db2work/dbgen_data/lineitem.tbl.1".
loire:
loire: SQL3500W The utility is beginning the "LOAD" phase at time "12-16-2002
loire: 12:39:59.920275".
loire:
loire: SQL3519W Begin Load Consistency Point. Input record count = "0".
loire:
loire: SQL3520W Load Consistency Point was successful.
loire:
loire: SQL3532I The Load utility is currently in the "LOAD" phase.
loire:
loire:
loire: Number of rows read          = 1762780
loire: Number of rows skipped       = 0
loire: Number of rows loaded        = 1762780
loire: Number of rows rejected      = 0
loire: Number of rows deleted       = 0
loire: Number of rows committed    = 0
loire: Number of warnings          = 0
loire:
loire: Tablestate:
loire: Check Pending
loire: Load in Progress
loire: db2 load query table ... completed ok

```

```

loire: DB2PART=3
loire: SQL3530I The Load Query utility is monitoring "LOAD" progress on
partition
loire: "3".
loire:
loire: SQL3109N The utility is beginning to load data from file
loire: "/db2work/dbgen_data/lineitem.tbl.1".
loire:
loire: SQL3500W The utility is beginning the "LOAD" phase at time "12-16-2002
loire: 12:40:04.995437".
loire:
loire: SQL3519W Begin Load Consistency Point. Input record count = "0".
loire:
loire: SQL3520W Load Consistency Point was successful.
loire:
loire: SQL3532I The Load utility is currently in the "LOAD" phase.
loire:
loire:
loire: Number of rows read          = 1756374
loire: Number of rows skipped       = 0
loire: Number of rows loaded        = 1756374
loire: Number of rows rejected      = 0
loire: Number of rows deleted       = 0
loire: Number of rows committed    = 0
loire: Number of warnings          = 0
loire:
loire: Tablestate:
loire: Check Pending
loire: Load in Progress
loire: db2 load query table ... completed ok

loire: DB2PART=4
loire: SQL3530I The Load Query utility is monitoring "LOAD" progress on
partition
loire: "4".
loire:
loire: SQL3109N The utility is beginning to load data from file
loire: "/db2work/dbgen_data/lineitem.tbl.1".
loire:
loire: SQL3500W The utility is beginning the "LOAD" phase at time "12-16-2002
loire: 12:40:02.818759".
loire:
loire: SQL3519W Begin Load Consistency Point. Input record count = "0".
loire:
loire: SQL3520W Load Consistency Point was successful.
loire:
loire: SQL3532I The Load utility is currently in the "LOAD" phase.
loire:
loire:
loire:

```

```

loire: Number of rows read      = 1772485
loire: Number of rows skipped  = 0
loire: Number of rows loaded   = 1772485
loire: Number of rows rejected = 0
loire: Number of rows deleted  = 0
loire: Number of rows committed = 0
loire: Number of warnings      = 0
loire:
loire: Tablestate:
loire: Check Pending
loire: Load in Progress
loire: db2 load query table ... completed ok

```

===== Output truncated by author =====

After the lineitem table load completed, **LOAD QUERY** was run again with the abbreviated results documented in Example A-6. Notice that the table state no longer shows Load in Progress but it still shows Check Pending. This is due to the referential integrity constraints on this (and on other) tables.

Example: A-6 Example of LOAD QUERY after load of lineitem table completes

```

{loire:db2inst1}/db2home/db2inst1/scripts ->db2_all "|" echo DB2PART=##; db2
load query table tpcd.lineitem"          # The LOAD QUERY command #2

```

```

loire: DB2PART=12
loire: Tablestate:
loire: Check Pending
loire: db2 load query table ... completed ok

```

```

loire: DB2PART=13
loire: Tablestate:
loire: Check Pending
loire: db2 load query table ... completed ok

```

```

loire: DB2PART=14
loire: Tablestate:
loire: Check Pending
loire: db2 load query table ... completed ok

```

```

loire: DB2PART=15
loire: Tablestate:
loire: Check Pending
loire: db2 load query table ... completed ok

```

```

loire: DB2PART=16
loire: Tablestate:

```



```

loire: Check Pending
loire: db2 load query table ... completed ok

loire: DB2PART=17
loire: Tablestate:
loire: Check Pending
loire: db2 load query table ... completed ok

===== Output truncated by author =====

```

Load the single partition tables, region, and nation

Only two tables remain: nation and region. For those, dbgen creates the small input files on disk. Then DB2 IMPORT is used to load the files. The commands to do this and the results are listed in Example A-7.

Example: A-7 Loading nation and region tables with IMPORT

```

{loire:db2inst1}/db2home/db2inst1/scripts ->db2 "import from
/db2work/dbgen_data/nation.tbl mo
dified by coldel| replace into tpcd.nation"

```

```

SQL3109N The utility is beginning to load data from file
"/db2work/dbgen_data/nation.tbl".

```

```

SQL3110N The utility has completed processing. "25" rows were read from the
input file.

```

```

SQL3221W ...Begin COMMIT WORK. Input Record Count = "25".

```

```

SQL3222W ...COMMIT of any database changes was successful.

```

```

SQL3149N "25" rows were processed from the input file. "25" rows were
successfully inserted into the table. "0" rows were rejected.

```

```

Number of rows read      = 25
Number of rows skipped   = 0
Number of rows inserted  = 25
Number of rows updated   = 0
Number of rows rejected  = 0
Number of rows committed = 25

```

```

{loire:db2inst1}/db2home/db2inst1/scripts ->db2 "import from
/db2work/dbgen_data/nation.tbl of
del modified by coldel| replace into tpcd.nation"

```

```

SQL3109N The utility is beginning to load data from file
"/db2work/dbgen_data/region.tbl".

SQL3110N The utility has completed processing. "5" rows were read from the
input file.

SQL3221W ...Begin COMMIT WORK. Input Record Count = "5".

SQL3222W ...COMMIT of any database changes was successful.

SQL3149N "5" rows were processed from the input file. "5" rows were
successfully inserted into the table. "0" rows were rejected.

Number of rows read          = 5
Number of rows skipped       = 0
Number of rows inserted      = 5
Number of rows updated       = 0
Number of rows rejected      = 0
Number of rows committed     = 5

{loire:db2inst1}/db2home/db2inst1/scripts ->

```

Run SET INTEGRITY

After all loads are done, special action must be taken to get the tables in Check Pending back to normal and usable status. To determine what, if anything, is in Check Pending state, run a DB2 query against SYSCAT.TABLES where status is C. Run SET INTEGRITY for each table found. The set of commands to do this is captured as Example A-8.

Example: A-8 Using SET INTEGRITY to return tables to normal state

```

{loire:db2inst1}/db2home/db2inst1 ->db2 "select cast(tabschema as char(10)) as
schema, cast(tabname as char(10)) as table, status from syscat.tables where
status='C'"

```

SCHEMA	TABLE	STATUS
-----	-----	-----
TPCD	PART	C
TPCD	SUPPLIER	C
TPCD	PARTSUPP	C
TPCD	CUSTOMER	C
TPCD	ORDERS	C
TPCD	LINEITEM	C

6 record(s) selected.

```
{loire:db2inst1}/db2home/db2inst1 ->db2 set integrity for tpcd.part immediate
checked
DB20000I The SQL command completed successfully.
{loire:db2inst1}/db2home/db2inst1 ->db2 set integrity for tpcd.supplier
immediate checked
DB20000I The SQL command completed successfully.
{loire:db2inst1}/db2home/db2inst1 ->db2 set integrity for tpcd.partsupp
immediate checked
DB20000I The SQL command completed successfully.
{loire:db2inst1}/db2home/db2inst1 ->db2 set integrity for tpcd.customer
immediate checked
DB20000I The SQL command completed successfully.
{loire:db2inst1}/db2home/db2inst1 ->db2 set integrity for tpcd.orders immediate
checked
DB20000I The SQL command completed successfully.
{loire:db2inst1}/db2home/db2inst1 ->db2 set integrity for tpcd.lineitem
immediate checked
DB20000I The SQL command completed successfully.
{loire:db2inst1}/db2home/db2inst1 ->db2 "select cast(tabschema as char(10)) as
schema, cast(tabname as char(10)) as table, status from syscat.tables where
status='C'"
```

SCHEMA	TABLE	STATUS
-----	-----	-----

0 record(s) selected.

```
{loire:db2inst1}/db2home/db2inst1 ->
```

The initial database load is complete.

MDC dimension analyzer

This section had its genesis in the following statement: It would be nice if there was a tool that would let you see how well (or poorly) extents would be utilized if you converted an existing table to a MDC table using a particular set of dimension columns. Such a tool could be used to model how much space would be required to create an MDC table using a specific set of dimensions. It would avoid the trial and error process of actually creating the MDC table.

As part of this residency we created a prototype tool consisting of a KornShell script and a skeleton SQL file used by that script.

This appendix documents the tool and provides examples of its use.

Important: The normal caveat applies: this script is to be used at your own risk. Your mileage may, and likely will, vary.

Overview of the dimension analyzer

The dimension analyzer is a design tool to help select dimensions (and in simple cases, dimension granularity) for an MDC table. The prototype tool only projects disk space requirements. It does not provide performance information.

Furthermore, the tool does not consider space used for:

- ▶ Dimension block indexes and composite block index
- ▶ Block 0 of MDC table which is reserved
- ▶ Block map file (SMS) or block map object (DMS)
- ▶ Free space control record which occurs in the first page of every MDC table block

Except for the indexes, the items listed above are quite small and ignoring them will not greatly detract from the accuracy of the estimated utilization. We feel that the block indexes, while not necessarily small, will likely replace RID indexes which currently exist on the non-MDC table. The space required for a RID index is much larger than that required for a block index on the same column. An exception to this assumption occurs when generated columns are added to increase dimension granularity. In such cases, the RID indexes would likely remain and the block indexes would require additional storage.

The dimension analyzer tool consists of two primary parts: a korn shell script and a skeleton SQL file.

The script exists only to submit a customized SQL query. Two positional parameters are required: a table name and a set of dimension columns. Script `evaluate_dimensions.ksh` allows the user to direct the operation of the script by specifying options and arguments. Among the items that can be modified are page size, extent size, average row length, and database partition (or all partitions). The script takes all these parameters and generates a SQL query to be submitted to DB2 UDB. Just prior to submitting the query, the script displays a summary of the values and asks the user whether or not execution should continue.

The query examines table rows and determines every unique set of dimension column values. Each unique value combination will be a cell in the MDC table. The query counts the number of rows that will be in each cell. With the number of rows, page size, and extent size, the script estimates how many extents (or blocks) will be needed to contain each cell and how well (or poorly) these extents will be utilized.

The extent utilization information is output as a set of buckets, which provides a kind of visual picture of the extent utilization. The buckets are defined as follows.

Table B-1 Description of buckets

bucket	description
0	Utilization greater than 0 and less than 5 percent
10	Utilization greater than or equal 5 percent and less than 15 percent
20	15 <= utilization < 25
30	25 <= utilization < 35
40	35 <= utilization < 45
50	45 <= utilization < 55
60	55 <= utilization < 65
70	65 <= utilization < 75
80	75 <= utilization < 85
90	85 <= utilization < 95
100	95 <= utilization <= 100 and all multi-extent cells

Associated with each bucket is a count of cells and a count of extents. For cells which can be stored in a single extent, the cell utilization is the same as the extent utilization. Cells which have enough rows to fill more than one extent are counted in bucket 100, as are single-extent cells with utilizations from 95% to 100%. So, if all rows of a cell can be stored in a single extent and the extent utilization is 16%, then the count of cells and the count of extents for bucket 20 will each be incremented by 1.

If a cell contains more rows than can fit in a single extent, the cell count for bucket 100 will be increased by 1. The extent count of bucket 100 will be increased by the count of full extents. If the last extent is only partially full, 1 will be added to the extent count of the bucket representing the utilization of this partial block. For instance, assume that a cell requires three extents to store all of its rows, and that the last extent is 80% full. Then the cell count for bucket 100 would be increased by 1; the extent count for bucket 100 would be increased by 2 (two of three extents are full); and the extent count for bucket 80 would be increased by 1 (the last or three extents is only partially full).

Hopefully, a couple of examples will clarify.

The first example (Example B-1) indicates that for the dimensions selected, the table when converted to MDC would fill most of the extents allocated to the table. Potentially, only eight cells will contain a single extent and have a utilization of less than 25%. Most of the cells (166) either have more than one extent or have a

single extent that is 95% full or better. Most extents (4741) are full. The extents counted in buckets 90 and below are those currently filling for 136 (= 35 + 26 + 7 + 2 +1 + 2 + 5 +7 + 28 + 23) of the 166 cells in bucket 100. The last three numbers (7, 28, and 23) reflect the subtraction of the single-extent cell counts from the associated extent count.

Example: B-1 Output showing good utilization of extents

BUCKET	CELL_COUNT	EXTENT_COUNT
100	166	4741
90	0	35
80	0	26
70	0	7
60	0	2
50	0	1
40	0	2
30	0	5
20	5	12
10	1	29
0	2	25

11 record(s) selected.

The next example (Example B-2) shows what happens when each of the cells has so few rows that its containing extent is hardly used at all. This is a good example of sparse blocks. No cell is more than 15% full. In this case, it is time to reconsider the selection of dimension columns. Perhaps the extent size and/or page size could be reduced.

Example: B-2 Output showing poor utilization of extents

BUCKET	CELL_COUNT	EXTENT_COUNT
10	70327	70327
0	5427	5427

2 record(s) selected.

Unless overridden by the user, the script will modify the SQL to evaluate the impact of MDC on a single database partition. No attempt is made to see if a dimension key is the same as a partitioning key for a multipartition table. See 4.5 “MDC and multipartition database” on page 150 for a more detailed discussion.

The script `evaluate_dimensions.ksh`

The tool for estimating the utilization of MDC table extents consists of two items: a script and a skeleton SQL file. This section describes the script; the following section, “The skeleton SQL file, `evaluate_dimensions.sql`” on page 322, describes the SQL.

The sole purpose of the script (listed in Example B-9 on page 332) is to generate a set of variables with which to modify the skeletal SQL and then to call DB2 UDB using the generated SQL file.

Tip: The script `evaluate_dimensions.ksh` is written to run on a DB2 UDB Version 8.1 database. To run the script on a Version 7 database, make the changes which are documented in the heading of the script just below the examples (`db2partitiongroupdef` to `nodegroupdef`, and so on).

The syntax of script `evaluate_dimensions.ksh` follows:

Example: B-3 Syntax for `evaluate_dimensions.ksh`

```
#      evaluate_dimensions.ksh [-D] [-d database_alias ]          --->
#      ---> [-p pagesize] [-e extentsize]                        --->
#      ---> [-P partition_number|all]                            --->
#      ---> [-r nnn|+nn]                                          --->
#      ---> [-o outfile]                                          --->
#      ---> Schema.Table 'DimensionCol1, ... ,DimensionColN'
```

The script requires two positional parameters. The first provides the fully qualified name (schema.table) of an existing table to be modelled for conversion to an MDC table. The second parameter is a comma-separated set of candidate dimensions, and the entire set is enclosed in single quotation marks. These dimensions can be simple like `1_shipdate`, `1_receiptdate`, or a little more complicated like `integer(1_shipdate/100)`, `integer(1_receiptdate)/100`, which models generated columns.

Options can be specified as described in Table B-2 to request help or to override default values.

Table B-2 Options for *evaluate_dimensions.ksh*

Option	Description
-h	Request usage (help) information.
-d database_alias	<p>Specify the database alias.</p> <p>The default value is obtained from the output of db2set DB2DBDFT or from variable DB2DBDFT. If neither of these are set, the script terminates with an error message.</p> <p>The script requires a connection to the database. In this residency the default database name was established with db2set DB2DBDFT=pfp where pfp is the name and the alias of our residency database. Since some installations choose not to set a default database, those installations may need to make the appropriate modifications to connect to the database. The existing -d option can be used to input the alias name, but DB2 CONNECT statements may need to be added where appropriate.</p>
-D	<p>Option -D which takes no arguments turns on the debug flag which gathers additional information mostly by activating set -x commands in the mainline and in most of the functions. Also, if DEBUG is active, the generated SQL file will not be deleted. (The SQL file is deleted if the script ends normally; the SQL file will not be deleted if the command ends with a non-zero return code regardless of the setting of the DEBUG variable.)</p> <p>The default is to gather no debug information.</p> <p>As an option to using the -D option, the DEBUG variable can be set prior to invoking the evaluate_dimensions.ksh script, for example, export DEBUG=1.</p>
-e extentsize	Use to override extentsize (number of pages). The default value is obtained from the row of SYSCAT. TABLESPACES , which describes the table space in which the evaluated table is stored.
-p pagesize	Use to override pagesize. The default value is obtained from the row of SYSCAT. TABLESPACES which describes the table space in which the evaluated table is stored.

Option	Description
-P [partition_number all]	<p>One of the problems with converting a table to MDC has to do with the situation where the dimension keys differ from the partitioning key. For a detailed discussion refer back to Example 4.5 on page 150.</p> <p>By default, script evaluate_dimensions.ksh assumes that the partitioning key will match none of the dimension keys, and uses the first partition on which the table exists to perform the evaluation. This means that because the extent utilization report will be for a single partition only, the reported extent count will need to be multiplied by the total number of partitions in the table's database partition group to determine the total storage needed to contain the MDC table.</p> <p>The -P flag can override the default behavior. If you know that one of the dimension keys is a partitioning key, you can specify -P all option to direct the script to provide a total for the entire new MDC table (across all partitions). Or, if you would prefer to use a partition other than the first one, you can specify that partition as the argument to the -P flag.</p> <p>The script assumes that the table rows are virtually evenly distributed across the partitions over which the table is defined. In other words, the script assumes no data skew.</p>
-o output_filename	<p>By default, the script writes its output to a date stamped file in /tmp. Use the -o option to specify an alternate filename in which the script can record its output.</p>
-r [nnn +nnn -nnn]	<p>The default value for the average row length is determined as described in "How average row size is obtained" on page 322.</p> <p>The user can override the average row size by specifying the -r flag on the command line, for example, -r 123. The -r flag can also be used to specify a value to be added to (or subtracted from) the computed row length. That the value is an adjustment to the length and not a complete override is made known to the script by prefixing the value with a '+' (plus) sign. This provision is to cover the addition of generated dimension columns to the new MDC table. For example, if two new integer dimension columns are evaluated (integer(l_shipdate)/100 and integer(l_receiptdate)/100), you would specify -r +8 as an option to the command. If you want to do both, specify an alternate row length and an increment, it is up to you to do the math.</p>

How average row size is obtained

The script tries two methods to make a best guess at the length of an average row in the table.

- ▶ Method 1. The *CARD* divided by the *NPAGES* values for the table from SYSCAT.TABLES estimates the rows per page. Assuming that the table has been recently reorganized and that **RUNSTATS** has been done, the average row length can be approximated by dividing the rows per page into the page size of the table space containing the table.
- ▶ Method 2. The average row length is computed as the sum of the following three values derived from SYSCAT.COLUMNS for the table to be evaluated: the sum of *AVGCOLLEN* (average column length) for each of the columns, the sum of the number of columns that can include null values, and the sum of the lengths of the length fields for each VARCHAR column. The latter value is the sum of the number of VARCHAR columns multiplied by four.

If both method1 and method2 return acceptable values, the script will calculate the difference between the two values. If the difference is within a specific tolerance value, the larger of the two values will be used. This will provide a more conservative evaluation (the predicted MDC table size will be larger). On the other hand, if the difference exceeds that tolerance, the average row size computed by method2 is preferred. If only one of the methods returns an estimated row size, that estimate will be used.

Tip: The script `evaluate_dimensions.ksh` is written to run on a DB2 UDB Version 8.1 database. To run the script on a Version 7 database, make the changes that are documented in the heading of the script, just below the examples (`db2partitiongroupdef` to `nodegroupdef`, and so on).

The skeleton SQL file, `evaluate_dimensions.sql`

The file `evaluate_dimensions.sql`, shown as Example B-4, is input to an AIX `sed` command within script `evaluate_dimensions.ksh`. The `sed` command generates a new file by replacing special keywords in the skeleton SQL. The keywords are:

- ▶ *YOUR_SCHEMA* and *YOUR_TABLE* are updated using information provided in the first positional parameter of `evaluate_dimensions.ksh`.
- ▶ *YOUR_DIMENSIONS* is replaced with the second positional parameter passed to the script.
- ▶ *YOUR_DIMENSIONS2* is virtually the same as *YOUR_DIMENSIONS* except column names are included.

- ▶ *YOUR_DBPARTITION_CLAUSE* is either removed (if script is called with option **-p all**) or replaced by a where clause to select only rows from a specific database partition.
- ▶ *YOUR_ROWS_PER_EXTENT* keyword is replaced with an estimate of the maximum number of rows that can be stored in one extent. The replacement value is a calculation based upon average row size, page size, and extent size any or all of which can be overridden by the user.

The generated sql is submitted to DB2 UDB to generate the extent utilization information.

Example: B-4 The skeleton SQL file, evaluate_dimensions.sql

```
-- Name:  evaluate_dimensions.sql
-- This skeleton SQL is used by script evaluate_dimensions.ksh.
-- It attempts to estimate how many extents will be allocated
-- and their utilization (as a set of buckets 0% <= 0 < 5%,
-- 5% <= 10 < 15%, ..., 95% <= 100 <= 100.
--
with
  cell_tab as (
    select distinct YOUR_DIMENSIONS2
           , count(*) as rowcount
    from YOUR_SCHEMA.YOUR_TABLE
      YOUR_DBPARTITION_CLAUSE
    group by YOUR_DIMENSIONS
  )
,
  utilization_tab as (
    select rowcount as rowcount_instance,
           count(rowcount) as cell_count,
           rowcount/YOUR_ROWS_PER_EXTENT.0 as utilization,
           integer(rowcount/YOUR_ROWS_PER_EXTENT.0) as full_extents,
           10*(integer((rowcount/YOUR_ROWS_PER_EXTENT.0
             - integer(rowcount/YOUR_ROWS_PER_EXTENT.0) + .05) * 10))
           as bucket
    from cell_tab group by rowcount
  )
,
  bucket_tab as (
    select 100 as bucket, cell_count as cell_count,
           cell_count*full_extents as extent_count
    from utilization_tab
    where full_extents>0
    union all
    select bucket,
           case when full_extents>0 then 0
                else cell_count
    end as extent_count
```

```

        end as cell_count,
        cell_count as extent_count
    from utilization_tab
)

select bucket, sum(cell_count) as cell_count,
       sum(extent_count) as extent_count
    from bucket_tab
   group by bucket
   order by bucket desc
;

```

Example: B-5 Example of a generated SQL file

```

-- Name:  evaluate_dimensions.sql
-- This skeleton SQL is used by script evaluate_dimensions.ksh.
-- It attempts to estimate how many extents will be allocated
-- and their utilization (as a set of buckets 0% <= 0 < 5%,
-- 5% <= 10 < 15%, ..., 95% <= 100 <= 100.
--
with
  cell_tab as (
    select distinct  integer(l_shipdate)/100 as dim1,
    integer(l_receiptdate)/10
  0 as dim2
    , count(*) as rowcount
    from TPCD.LINEITEM
    where dbpartitionnum(L_ORDERKEY) = 1
    group by integer(l_shipdate)/100, integer(l_receiptdate)/100
  )
,
  utilization_tab as (
    select rowcount as rowcount_instance,
    count(rowcount) as cell_count,
    rowcount/1888.0 as utilization,
    integer(rowcount/1888) as full_extents,
    10*(integer((rowcount/1888.0
    - integer(rowcount/1888.0) + .05) * 10))
    as bucket
    from cell_tab group by rowcount
  )
,
  bucket_tab as (
    select 100 as bucket, cell_count as cell_count,
    cell_count*full_extents as extent_count
    from utilization_tab
    where full_extents>0
    union all

```

```

select bucket,
       case when full_extents>0 then 0
           else cell_count
       end as cell_count,
       cell_count as extent_count
from utilization_tab
)

select bucket, sum(cell_count) as cell_count,
       sum(extent_count) as extent_count
       from bucket_tab
       group by bucket
       order by bucket desc
;

```

A real example of use

The lineitem table was one the tables used to test the accuracy of the script **evaluate_dimensions.ksh** and its generated SQL. The goal was to predict the storage which would be required store an MDC version of the table, to create the new MDC version, and then to compare the actual results with the prediction. This prediction was for the base table only. All indexes were written to a different table space than the one that contained the base table.

The table space containing the lineitem table has a page size of 16 K. The extent size is 16 pages.

Create a duplicate of the lineitem table

First, we established a baseline for validation. We determined how much disk space was required to store the lineitem base table (indexes were created in a separate table space). Before creating the lineitem table, **RUNSTATS** was performed. Table space utilization and catalog statistics were gathered. After the table was created, **RUNSTATS** was run again. A second set of table space utilization and catalog statistics was gathered. The differences between the second set of statistics, and the first set were calculated to determine the storage requirements for table lineitem.

First cut

The first evaluation of the lineitem table was made using columns **I_shipdate** and **I_receiptdate** for dimension columns. The script was called with only the required positional parameters: table name (tpcd.lineitem) and a list of dimension columns

for evaluation ('l_shipdate, l_receiptdate'). No option flags were used to override default values.

The initial script command and the output to the terminal are displayed in Example B-6. The interesting information is at the end of the example. So, for just database partition 1, the tool has determined that the choice of dimension columns will require 75,754 cells, one cell for each unique pair of columns values. Furthermore, each cell maps requires a single extent. The bad news is that every single one of those extents is under 15% utilized. It looks like most of the extents will probably average around 10% full.

Example: B-6 Evaluate l_shipdate and l_receiptdate as dimensions

```
{loire:db2inst1}/db2home/db2inst1/scripts ->evaluate_dimensions.ksh  
tpcd.lineitem 'l_shipdate,l_receiptdate'
```

```
[evaluate_dimensions.ksh 12/18/02 12:15:32]:
```

```
Running with parameters: tpcd.lineitem l_shipdate,l_receiptdate
```

```
[evaluate_dimensions.ksh 12/18/02 12:15:34]:
```

```
Creating MDC sql using the following parameters:
```

```
Database Alias: pfp  
Table Name: TPCD.LINEITEM  
Row Length Adjustment:  
Average Row Length: 138  
Page Size: 16384  
Extent Size: 16  
Rows/Page: 118  
Rows/Extent: 1888  
DB Partition: 1  
Dimensions: l_shipdate,l_receiptdate
```

```
Comments:
```

```
Average row length calculated from average column lengths plus  
NULL indicators and VARCHAR length fields is '131'.  
Average row length calculated from CARD and NPAGES is '138'.
```

```
[evaluate_dimensions.ksh 12/18/02 12:15:34]:
```

```
Do you wish to continue? (y or n) --> y
```

```
[evaluate_dimensions.ksh 12/18/02 12:15:42]:
```

```
Submitting generated SQL file, '/tmp/MDC_471666.sql'
```

```
[evaluate_dimensions.ksh 12/18/02 12:18:13]:
```



```
Output follows:
with cell_tab as ( select distinct l_shipdate as dim1, l_receiptdate as dim2 ,
count(*) as rowcount from TPCD.LINEITEM where dbpartitionnum(L_ORDERKEY) = 1
group by l_shipdate,l_receiptdate ) , utilization_tab as ( select rowcount as
rowcount_instance, count(rowcount) as cell_count, rowcount/1888.0 as
utilization, integer(rowcount/1888) as full_extents,
10*(integer((rowcount/1888.0 - integer(rowcount/1888.0) + .05) * 10)) as bucket
from cell_tab group by rowcount ) , bucket_tab as ( select 100 as bucket,
cell_count as cell_count, cell_count*full_extents as extent_count from
utilization_tab where full_extents>0 union all select bucket, case when
full_extents>0 then 0 else cell_count end as cell_count, cell_count as
extent_count from utilization_tab ) select bucket, sum(cell_count) as
cell_count, sum(extent_count) as extent_count from bucket_tab group by bucket
order by bucket desc
```

BUCKET	CELL_COUNT	EXTENT_COUNT
10	71389	71389
0	4365	4365

2 record(s) selected.

```
[evaluate_dimensions.ksh 12/18/02 12:18:13]:
Script completed successfully. Highest return code: 0.
```

Second cut, generated columns

The first attempt did not turn out well. So, something had to change. Using script options, you could evaluate the impact of changing the page size (-p option), the extent size (-e option), or a combination of the two. Our decision, however, was to try using prototyping with generated columns.

Using generated columns, our goal was to see if we would get fewer unique combinations of dimension column values. Fewer combinations means fewer cells. With fewer cells there should be more rows per cell. This in turn would translate to better utilization of disk storage. We decided to generate two new columns using the INTEGER(date) function to create, say, an l_shipYrMo column using integer (l_shipdate)/100. A similar action was imagined for l_receiptdate. So, for the second evaluation, we ran script **MSC_prototype.ksh** to model the use of two new generated dimension columns. If the evaluation produced a positive outcome, we would create a new multidimensionally clustered lineitem table. The new rows would be identical to the old lineitem rows, except that two new integer columns would be added to the end of the row.

The `INTEGER(date)/100` computation is a monotonic function. Consequently, DB2 UDB can rewrite a query to apply additional predicates, so that the generated columns can be used to resolve range predicates specifying `l_shipdate` and `l_receiptdate`.

Example B-7 on page 328 shows the command that submitted the prototype script and the output displayed on the terminal. Note that `-r` flag was used to pass an adjustment to increase the estimated row length. Specifically, 8 bytes were added to compensate for the two four-byte generated columns, which would be added to each row.

Notice that for the dimensions to be evaluated, we entered the functions that would be actually used to generate the new (and additional) columns. The table against which the evaluation runs is still `tpcd.lineitem` except the row length to be modelled has been increased by 8 bytes to cover the two new columns. Also notice that the evaluation is still modelling a single database partition, partition 1. The reason is that none of the new columns is anticipated to be used as a partitioning key for the new table. (If one of the dimension keys was to also be a partitioning key, you could model the entire table (all partitions) by including option `-P all` when `evaluate_dimensions.ksh` is run.)

As you can see, the script estimated that by using generated columns instead of the base columns, the number of cells needed was reduced to 172. Quite a drop from the first attempt. Note, too, that for most extents the predicted utilization is in the 95 to 100% range. The extents in the other buckets are (except for the 8 cells in the 30, 20, 10 and 0 buckets) are partially filled additional extents for those cells (a lot) which have at least one other full extent assigned. One more consideration is that, on average, each cell requires 37.6 (6241 extents divided by 166 cells) extents to contain all of its row. If a larger page size and/or extent size was used, the number of extents needed to contain a cell would be decreased. This might translate to slightly improved performance as DB2 UDB would perform fewer input operations to read all of a cell's rows.

Example: B-7 Evaluate two new generated columns for MDC table

```
{loire:db2inst1}/db2home/db2inst1/scripts ->evaluate_dimensions.ksh
tpcd.lineitem -r +8 'integer(l_shipdate)/100,integer(l_receiptdate)/100'
```

```
[evaluate_dimensions.ksh 12/18/02 12:12:19]:
Running with parameters: -r +8 tpcd.lineitem
integer(l_shipdate)/100,integer(l_receiptdate)/100
```

```
[evaluate_dimensions.ksh 12/18/02 12:12:22]:
```

```
Creating MDC sql using the following parameters:
```

```

Database Alias: pfp
Table Name: TPCD.LINEITEM
Row Length Adjustment: +8
Average Row Length: 146
Page Size: 16384
Extent Size: 16
Rows/Page: 111
Rows/Extent: 1776
DB Partition: 1
Dimensions: integer(l_shipdate)/100,integer(l_receiptdate)/100
Comments:
Average row length calculated from average column lengths plus
NULL indicators and VARCHAR length fields is '131'.
Average row length calculated from CARD and NPAGES is '138'.
Row length was adjusted by user specified amount, +8

```

```

[evaluate_dimensions.ksh 12/18/02 12:12:22]:
Do you wish to continue? (y or n) --> y

```

```

[evaluate_dimensions.ksh 12/18/02 12:12:23]:
Submitting generated SQL file, '/tmp/MDC_433936.sql'

```

```

[evaluate_dimensions.ksh 12/18/02 12:14:53]:

```

Output follows:

```

with cell_tab as ( select distinct integer(l_shipdate)/100 as dim1,
integer(l_receiptdate)/100 as dim2 , count(*) as rowcount from TPCD.LINEITEM
where dbpartitionnum(L_ORDERKEY) = 1 group by
integer(l_shipdate)/100,integer(l_receiptdate)/100 ) , utilization_tab as (
select rowcount as rowcount_instance, count(rowcount) as cell_count,
rowcount/1776.0 as utilization, integer(rowcount/1776) as full_extents,
10*(integer((rowcount/1776.0 - integer(rowcount/1776.0) + .05) * 10)) as bucket
from cell_tab group by rowcount ) , bucket_tab as ( select 100 as bucket,
cell_count as cell_count, cell_count*full_extents as extent_count from
utilization_tab where full_extents>0 union all select bucket, case when
full_extents>0 then 0 else cell_count end as cell_count, cell_count as
extent_count from utilization_tab ) select bucket, sum(cell_count) as
cell_count, sum(extent_count) as extent_count from bucket_tab group by bucket
order by bucket desc

```

BUCKET	CELL_COUNT	EXTENT_COUNT
100	166	6241
90	0	24
80	0	37
70	0	21
60	0	26

50	0	6
40	0	2
30	3	12
20	2	15
10	1	12
0	2	8

11 record(s) selected.

[evaluate_dimensions.ksh 12/18/02 12:14:53]:

Script completed successfully. Highest return code: 0.

Creating lineitem as MDC table

Satisfied with the second result, we created a new table, `tpcd.lineitem_mdc`, as a multidimensionally clustered table using the generated columns. Example B-8 shows the DDL and SQL, which was used to create the new MDC version of `lineitem`.

Example: B-8 DDL and SQL to create `tpcd.lineitem_mdc` as MDC table

```
-- DDL to create an lineitem as an MDC table with generated columns for
-- comparison to the original non-MDC version (and without the generated
-- columns).
-- This creates and populates the tpcd.lineitem_mdc table.
-- To execute, turn off autocommit:
--
--      db2 +c -tvf gcl_create_lineitem_mdc2.ddl
--
```

```
CONNECT TO pfp;
```

```
-- DROP TABLE TPCD.LINEITEM_MDC ;
```

```
CREATE TABLE TPCD.LINEITEM_MDC (
    L_ORDERKEY    INTEGER NOT NULL
    , L_PARTKEY    INTEGER NOT NULL
    , L_SUPPKEY    INTEGER NOT NULL
    , L_LINENUMBER INTEGER NOT NULL
    , L_QUANTITY   DECIMAL(15,2) NOT NULL
    , L_EXTENDEDPRICE DECIMAL(15,2) NOT NULL
    , L_DISCOUNT  DECIMAL(15,2) NOT NULL
    , L_TAX        DECIMAL(15,2) NOT NULL
    , L_RETURNFLAG CHAR(1) NOT NULL
    , L_LINESTATUS CHAR(1) NOT NULL
    , L_SHIPDATE   DATE NOT NULL
    , L_COMMITDATE DATE NOT NULL
)
```

```

, L_RECEIPTDATE DATE NOT NULL
, L_SHIPINSTRUCT CHAR(25) NOT NULL
, L_SHIPMODE CHAR(10) NOT NULL
, L_COMMENT VARCHAR(44) NOT NULL
, L_shipyrmo generated always
  as ((integer(l_shipdate)/100))
, L_rcptyrmo generated always
  as ((integer(l_receiptdate)/100))
)
organize by dimensions ( l_shipyrmo, l_rcptyrmo )
in TBS1
index in MDC
not logged initially
;

INSERT into TPCD.LINEITEM_MDC (L_ORDERKEY, L_PARTKEY, L_SUPPKEY, L_LINENUMBER,
L_QUANTITY, L_EXTENDEDPRICE, L_DISCOUNT, L_TAX, L_RETURNFLAG,
L_LINESTATUS,L_SHIPDATE, L_COMMITDATE, L_RECEIPTDATE, L_SHIPINSTRUCT,
L_SHIPMODE, L_COMMENT) select * from TPCD.LINEITEM
;

COMMIT WORK;

TERMINATE;

```

After the MDC table was created, **RUNSTATS** was run, **DB2 LIST TABLESPACES** was run on all partitions, and system catalog was queried to get CARD, NPAGES, FPAGES, and ACTIVE_BLOCKS. These values were compared to what was predicted. The results are summarized in Table B-3 on page 332.

Summary of results

Table B-3 summarizes the difference between values estimated by the **evaluate_dimensions.ksh** script and the actual values measured after the MDC table was created. As you can see, the prediction was fairly close to the actual outcome. In all of our tests so far, the highest variance between predicted and actual has been that the prototype undershot the actual by 5%. But, you might experience worse results.

Important: These evaluations were made in a controlled environment. There was no data skew. Your results may differ, your mileage may vary.

Table B-3 Comparison of MDC prediction versus MDC actual

Measurement	MDC predicted	MDC actual	Difference (col 2 - col3)	Percent
NPAGES	--	3222206		
Partition 1 pages	102464	102016	448	0.4%
All pages	3278848	3266032	12816	0.4%
Partition 1 extents	6404	6376	28	0.4%
All extents	204928	204127	801	0.4%
ACTIVE_BLOCKS	--	203904	--	--

The summary table does not show how the MDC version of the table compared with the non-MDC version. The non-MDC table (base table only, no indexes) required 3222976 16 K pages; for MDC (also base table, no indexes) the page count grew to 3266032, an increase of 43046 pages (1.3%).

Listing of evaluate_dimensions.ksh

Example B-9 is a complete listing of script **evaluate_dimensions.ksh**. The script and the skeleton SQL file, **evaluate_dimensions.sql**, are available as part of the additional material associated with this redbook. For information about how to obtain these items, see Appendix E, “Additional material” on page 365.

Tip: The script **evaluate_dimensions.ksh** is written to run on a DB2 UDB Version 8.1 database. To run the script on a Version 7 database, make the changes that are documented in the heading of the script just below the examples (db2partitiongroupdef to nodegroupdef, and so on).

Example: B-9 Script *evaluate_dimensions.ksh*

```
#!/usr/bin/ksh
# -----
# Script:  evaluate_dimensions.ksh
# Version: 1.0.3
# Created: 13 December 2002
# Author:  George Latimer, Fourth Millennium Technologies
#
# Description: This script was developed to help answer the questions:
#              “If an existing table is converted to MDC, what space will
#              be required?” “How will that space be utilized?” It lets
#              a user play “what if...” games: page and extent sizes can
```

```

#           be changed, dimension columns can be changed (even simple
#           generated columns can be specified).
#
# SYNTAX:
#
#           evaluate_dimensions.ksh [-D] [-d database_alias ]           --->
#           ---> [-p pagesize] [-e extentsize]           --->
#           ---> [-P partition_number|all]           --->
#           ---> [-r nnn|+nn]           --->
#           ---> [-o outfile]           --->
#           ---> Schema.Table 'DimensionCol1, ... ,DimensionColN'
#
#           The positional parameters "Schema.Table" and "dimension column
#           list" are required. Note that the table name must be qualified
#           with the schema name. The dimension columns can include both
#           existing columns and simple generated columns. This parameter must
#           be enclosed in single quotes. See EXAMPLE 2 below for a sample
#           using generated columns.
#
#           -h           Request help
#           -D           Turn on debug information (set -x, etc)
#           -d           Specify the database alias name
#           -e           Override extentsize
#           -p           Override pagesize
#           -r           Override row size or specify an amount to be added to
#                       account for generated columns that might be added.
#                       If an increment, prefix numeric value with '+'.
#           -P           DB partition number or 'all'.
#           -o outfile   Specify the output file where this script will write
#                       its messages. Default is /tmp/$SCRIPT_NAME.out.\$\$
#                       where '\$\$' will be replaced by the current process
#                       ID.
#
# EXAMPLE 1: Convert an existing table to MDC.
#
#           prompt> evaluate_dimensions.ksh tpcd.lineitem \
#           cont> 'l_shipdate, l_receiptdate'
#
# EXAMPLE 2: Convert an existing table to MDC with generated columns. Because
#           we plan to add two new integer generated columns as dimensions,
#           we specify an 8-byte addition to the row length with -r flag.
#
#           prompt> evaluate_dimensions.ksh -r +8 tpcd.lineitem \
#           cont> 'integer(l_shipdate)/100,integer(l_receiptdate)/100'
#
# To run this script on a DB2 Version 7 database, make the following changes:
#
#           -----CHANGE FROM-----           -----CHANGE TO-----
#           dbpartitiongroupdef           nodegroupdef

```

```

#      dbpartitionnum($COLNAME)      nodenumber($COLNAME)
#      dbpartitionnum                nodenum
#      dbpgname                      ngname
#
# (C) Copyright Fourth Millennium Technologies 2002 All Rights Reserved.
# End_of_Prolog
#
# History/Modifications:
# 12/13/2002 V1.0.0 Initial release (G. Latimer)
# 12/19/2002 V1.0.1 Extended edit check through partition check (G. Latimer)
# 01/02/2003 V1.0.2 Terminate script if cannot determine or connect to
#                  database (G. Latimer)
# 02/23/2003 V1.0.3 Changed script name (G. Latimer)
# -----

SCRIPT_NAME=$(basename $0)
FULL_PATH=$0
typeset -i RC=0
typeset -i MAX_RC=0
LOGFILE=${LOGFILE:-"/tmp/${SCRIPT_NAME}.out.${date +%m%d%Y}"}

EDIT_MESSAGES=""          # Used to batch editing messages
VALID_PAGE_SIZES="4096 8192 16384 32768"
typeset -i CARD
typeset -i NPAGES
typeset -i FPAGES
typeset -i PGSZ
typeset -i EXT SZ
typeset -i TOLERANCE=6
typeset -i MAX_ROWS_PER_PAGE=255 # Maximum number of rows in DB2 page
COMMENTS=""

trap 'all_done' 0,1,2,3,6,9,11,15

function all_done {
    [ $DEBUG ] && set -x

    if [[ ${MAX_RC} -ne 0 ]]; then
        if [[ ${MAX_RC} -ne 555 ]]; then # No error message if "usage"
            MESSAGE="Script failed. Highest return code: $MAX_RC."
            print_message "$MESSAGE"
        fi
    else
        MESSAGE="Script completed successfully. Highest return code: $MAX_RC."
        print_message "$MESSAGE"
        if [[ ! "$DEBUG" = 1 ]]; then
            /bin/rm -f $SQL_FILE
        fi
    fi
}

```



```

fi

## Any cleanup?
db2 terminate >/dev/null 2>&1
exit ${MAX_RC}
}

function usage {
    end_of_prolog=$( awk '/End_of_Prolog/ {print NR-1 ; exit }' $FULL_PATH )
    head -n $end_of_prolog $FULL_PATH | tee $TTY
    RC=555
    check_max_rc $RC
    exit
}

function print_message {
    [ $DEBUG ] && set -x
    print "${SCRIPT_NAME $(date +"%D %T")}:\n\t${*}\n" | tee $TTY
    return
}

function check_arg {
    [ $DEBUG ] && set -x
    typeset -i RC=0
    ## Checks that an option requiring an argument has one. So, check
    ## that an option needing an argument is not immediately followed by
    ## another option flag. (Unfortunately, getoptopts cannot be directed
    ## to evaluate the following option flag as a flag -- say by subtracting
    ## 1 from OPTIND. getoptopts regards the following flag as an argument
    ## for the preceeding flag and nothing you can do (short of modifying
    ## the entire option string and restarting getoptopts) will cause getoptopts
    ## to re-evaluate this flag. For example, say you enter command
    ##     db2gov_alert -n -d -l gov_log_stem
    ## getoptopts has been told that '-n' flag requires an argument. So,
    ## '-d' flag is determined to be that argument. Although the following
    ## check will determine that '-d' is an option flag, getoptopts cannot be
    ## instructed that the argument for '-n' is missing and so '-d' should
    ## be re-evaluated as an option flag. (The script will end because
    ## the following check flags a parse error.)

    if [[ $(echo $OPTARG | egrep -c "[+-]") -gt 0 ]]; then
        MESSAGE="Option '$OPTION' requires an argument."
        add_edit_message "$MESSAGE"
        RC=19
    fi
    return $RC
}

```

```

function check_max_rc {
    typeset -i RC=$1
    if [[ $RC > $MAX_RC ]]; then
        MAX_RC=$RC
    fi
    return
}

function test_signed_numeric {
    # Valid numbers are at least 1 digit; may be prefixed by '+' or '-'
    [ $DEBUG ] && set -x
    typeset -i RC=0
    if [[ "$1" != ?([+-])+([0-9]) ]]; then
        RC=1
    fi
    return $RC
}

function test_numeric {
    # Valid numbers are at least 1 digit
    [ $DEBUG ] && set -x
    typeset -i RC=0
    if [[ "$1" != +([0-9]) ]]; then
        RC=1
    fi
    return $RC
}

function db2_command {
    [ $DEBUG ] && set -x
    typeset -i RC=0 # Local variable

    OUT=$(db2 +n $1)
    RC=$?
    # if [[ $RC -ne 0 ]]; then
    #     print "Output follows:\n$OUT"
    # fi
    return $RC
}

function edit_extent_size {
    [ $DEBUG ] && set -x
    typeset -i RC=0 # Local variable

    test_numeric $EXTENT_SIZE
    RC=$?
    if [[ $RC -ne 0 ]]; then
        MESSAGE="Specified extent size '$EXTENT_SIZE' is not numeric."
        add_edit_message "$MESSAGE"
    fi
}

```

```

        check_max_rc $RC
    else
        if [[ $EXTENT_SIZE -lt 2 || $EXTENT_SIZE -gt 256 ]]; then
            MESSAGE="Specified extent size '$EXTENT_SIZE' is invalid value."
            add_edit_message "$MESSAGE"
        fi
    fi
    return $RC
}

function edit_page_size {
    [ $DEBUG ] && set -x
    typeset -i RC=0                # Local variable

    test_numeric $PAGE_SIZE
    RC=$?
    if [[ $RC -ne 0 ]]; then
        MESSAGE="Specified page size '$PAGE_SIZE' is not numeric."
        add_edit_message "$MESSAGE"
        check_max_rc $RC
    else
        if [[ $(print "$VALID_PAGE_SIZES" | grep -c -w $PAGE_SIZE) -ne 1 ]]; then
            MESSAGE="Specified page size '$PAGE_SIZE' is invalid value."
            add_edit_message "$MESSAGE"
        fi
    fi
    return $RC
}

function edit_row_option {
    ## This will either provide an override to average row length or it will
    ## provide a value by which to increase the row length to account for the
    ## possible addition of generated columns.
    [ $DEBUG ] && set -x
    typeset -i RC=0                # Local variable

    test_signed_numeric $ROW_OPTION
    RC=$?
    if [[ $RC -eq 0 ]]; then
        typeset -RL1 X=$ROW_OPTION
        if [[ "$X" = '+' || "$X" = '-' ]]; then
            ROW_ADJUST=$ROW_OPTION
            continue
        else
            # ROW_SPEC must be greater than 0
            if [[ $ROW_OPTION -eq 0 ]]; then
                MESSAGE="Row length, '$ROW_OPTION', must be greater than zero."
                add_edit_message "$MESSAGE"
            fi
        fi
    fi
}

```

```

        USER_ROW_LENGTH=$ROW_OPTION
        ROW_ADJUST=0
    fi
else
    MESSAGE="Row length or adjustment, '$ROW_OPTION', is not numeric."
    add_edit_message "$MESSAGE"
fi
return $RC
}

function edit_partlist {
    [ $DEBUG ] && set -x
    typeset -i RC=0                                # Local variable

    if [ [ "$DBPARTNUM" = 'ALL' ] ]; then
        continue                                # no where clause needed
    else
        test_numeric $DBPARTNUM
        RC=$?
        if [ [ $RC -ne 0 ] ]; then
            MESSAGE="DB partition number, $DBPARTNUM, must be numeric."
            add_edit_message "$MESSAGE"
            check_max_rc $RC
        else
            DB2_CMD="select 'xyxyz',c.colname, d.dbpartitionnum
                from syscat.tables a, syscat.tablespace b,
                syscat.columns c, syscat.dbpartitiongroupdef d
                where a.tabschema='$SCHEMA' and a.tabname='$TABLE'
                and d.dbpartitionnum=$DBPARTNUM
                and a.tbpaceid=b.tbpaceid and b.dbpgname=d.dbpgname
                and a.tabname=c.tabname and a.tabschema=c.tabschema
                and c.colno=0 and d.in_use='Y'
            "
            db2_command "$DB2_CMD"
            RC=$?
            if [ [ $RC -ne 0 ] ]; then
                MESSAGE=$(print "DB partition number, $DBPARTNUM, is not valid." \
                    "Output follows:\n$OUT")
                add_edit_message "$MESSAGE"
                check_max_rc $RC
            else
                print "$OUT" | egrep -w "^[ ]*xyxyz" \
                    | read junk COLNAME DBPARTNUM
                if [ [ -n "$COLNAME" ] && -n "$DBPARTNUM" ] ]; then
                    PARTITION_CLAUSE="where dbpartitionnum($COLNAME) =
$DBPARTNUM"
                else
                    add_edit_message "Unable to determine DB partition number."
                fi
            fi
        fi
    fi
}

```

```

        fi
    fi
    return $RC
}

function add_edit_message {
    [ $DEBUG ] && set -x
    EDIT_MESSAGES="$EDIT_MESSAGES$1\n\t"
    return
}

function check_edit_messages {
    [ $DEBUG ] && set -x
    if [[ ! -z "$EDIT_MESSAGES" ]]; then
        RC=99 # Make sure exit code is non-zero
        check_max_rc $RC
        print_message "Edit check errors:\n\t$EDIT_MESSAGES"
        MESSAGE="Due to the above errors, script is EXITING, RC=$RC."
        print_message "$MESSAGE"
        usage # Usage will force an exit.
    fi
    return
}

function update_comment {
    [ $DEBUG ] && set -x
    COMMENTS="$COMMENTS$*\n"
    return
}

function check_tolerance {
    [ $DEBUG ] && set -x
    typeset -i RC=0

    if [[ $1 -gt $2 ]]; then
        LARGER=$1
        SMALLER=$2
    else
        LARGER=$2
        SMALLER=$1
    fi

    typeset -i CHECK=$(( ($LARGER - $SMALLER) * 100) / $SMALLER )
    if [[ $CHECK -gt $TOLERANCE ]]; then
        RC=1
    fi
    return $RC
}

```

```

# =====
# Mainline begins here.....
# =====

typeset -u SCHEMA=
typeset -u TABLE=
PARTITION_CLAUSE=""

exec 1>> $LOGFILE
exec 2>&1

if [[ "$DEBUG" != 1 ]]; then # DEBUG=1 activates debug statements. Set
    unset DEBUG           # ...to NULL, 'unset DEBUG', to turn off.
fi

[ $DEBUG ] && set -x

VARS="$@"                # Save input parameters

TTY=$(tty 2>/dev/null)   # Determine whether this script was called from
if [[ $? -ne 0 ]]; then  # ...a terminal session. If so, messages
generated               # ...by this script will be written to the
    TTY=/dev/null       # terminal
terminal                # ...as well as the script output file.
fi

DIRNAME=$(dirname $(whence $0))
SQL_SKELETON_FILE=$DIRNAME/evaluate_dimensions.sql # <----- verify!
SQL_FILE=/tmp/MDC_$.sql

print_message "Running with parameters: $VARS"

while getopts :hDd:e:p:P:r:o: OPTION ; do

    [ $DEBUG ] && print "OPTION=$OPTION"

    case $OPTION in

        h) usage                # Help!
            ;;
        D) export DEBUG=1        # Turn on debugging information
            [ $DEBUG ] && set -x
            ;;
        d) ## Check database alias
            check_arg
    esac
done

```

```

RC=$?
if [[ $RC -ne 0 ]]; then
    set -- $OPTARG $*
else
    DB_ALIAS=$OPTARG          # Save database alias
fi
;;
e) ## Override extentsize
check_arg
RC=$?
if [[ $RC -ne 0 ]]; then
    set -- $OPTARG $*
else
    EXTENT_SIZE=$OPTARG      # Save extentsize
    edit_extent_size
fi
;;
p) ## Override pagesize
check_arg
RC=$?
if [[ $RC -ne 0 ]]; then
    set -- $OPTARG $*
else
    PAGE_SIZE=$OPTARG        # Save pagesize
    edit_page_size
fi
;;
P) ## Override partition
check_arg
RC=$?
if [[ $RC -ne 0 ]]; then
    set -- $OPTARG $*
else
    typeset -u DBPARTNUM=$OPTARG # Save DB partition number
    ## If specified, will check later
fi
;;
r) ## Override or adjust row length
ROW_OPTION=$OPTARG          # Override or adjust row length
edit_row_option
;;
o) ## If output file already exists, make sure that it's not a
## directory. Make sure that we can write to file.
check_arg
RC=$?
if [[ $RC -ne 0 ]]; then
    set -- $OPTARG $*
else
    if [[ ! -d $OPTARG ]]; then

```

```

        if [[ ! -e "$OPTARG" ]]; then
            touch "$OPTARG"
        fi
        if [[ -w "$OPTARG" ]]; then
            cat $LOGFILE >> $OPTARG # Append!
            if [[ $? -eq 0 ]]; then
                /bin/rm $LOGFILE # Old file no longer needed
            fi
            LOGFILE=$OPTARG # Change to new log
            exec 1>> $LOGFILE
            exec 2>&1
        else
            MESSAGE="Unable to write to $OPTARG."
            add_edit_message "$MESSAGE"
        fi
    else # Oh oh. It's a directory!
        MESSAGE="Cannot write to a directory!"
        add_edit_message "$MESSAGE"
    fi
fi
;;
*) ## Unrecognized option.
MESSAGE="Option $(eval echo \${(( OPTIND - 1 ))}) is invalid."
add_edit_message "$MESSAGE"
esac
done

shift $(( OPTIND - 1 ))

if [[ $# -ne 2 ]]; then
    RC=20
    check_max_rc $RC
    MESSAGE="'$#' is an incorrect number of input parameters."
    add_edit_message "$MESSAGE"
    check_edit_messages # This is serious. Print and quit.
fi

if [[ -z $DB_ALIAS ]]; then
    DB_ALIAS=$(db2set db2dbdft)
    RC=$?
    if [[ $RC -ne 0 ]]; then
        DB_ALIAS=$DB2DBDFT
        if [[ -z $DB_ALIAS ]]; then
            MESSAGE="Unable to determine database name."
            add_edit_message "$MESSAGE"
            check_max_rc $RC
            check_edit_messages # This is serious. Print and quit.
        fi
    fi
fi

```



```

        fi
    fi
    export DB_ALIAS

    typeset -u SCHEMA
    typeset -u TABLE
    echo "$1" | tr '.' ' ' | read SCHEMA TABLE

    DIMENSIONS="$2"
    saveIFS="$IFS"
    IFS=","
    set -- $(echo "$DIMENSIONS")
    IFS="$saveIFS"
    typeset -i i=1
    COMMA=""
    while [[ $# -gt 0 ]]; do
        DIMENSIONS2="$DIMENSIONS2$COMMA $1 as dim$i"
        i=i+1
        COMMA=","
        shift
    done

    SQLCODE=$(db2 -ec +o connect to $DB_ALIAS)
    if [[ "$SQLCODE" -ne 0 ]]; then
        RC=21
        check_max_rc $RC
        MESSAGE=$(print "*** Unable to connect to database alias '$DB_ALIAS',"
            "sqlcode=$SQLCODE.")
        add_edit_message "$MESSAGE"
        check_edit_messages # This is serious. Print and quit.
    fi

    ## Check for -P: if 'all|ALL' then no partition clause. If -P specifies
    ## a number, make sure that number is valid for table. If no -P option,
    ## default to first partition on which tablespace is defined.

    ## (Possible enhancements: If '-P all', should script verify that
    ## dimension column is one of partitioning cols? What if table has
    ## a small total row count -- but small tables probably should not
    ## be partitioned anyway. Would a better default be to sum rows on
    ## each partition and use partition with largest row count? check for skew?)

    if [[ ! -z $DBPARTNUM ]]; then # -P option was coded
        edit_partlist
    else
        DB2_CMD="select 'xyyyz',c.colname, d.dbpartitionnum
            from syscat.tables a, syscat.tablespaces b,
            syscat.columns c, syscat.dbpartitiongroupdef d"
    fi

```

```

        where a.tabschema='$SCHEMA' and a.tabname='$TABLE'
              and a.tbpaceid=b.tbpaceid and b.dbpgname=d.dbpgname
              and a.tabname=c.tabname and a.tabschema=c.tabschema
              and c.colno=0 and d.in_use='Y'
              order by d.dbpartitionnum
              fetch first 1 row only
    "

db2_command "$DB2_CMD"
RC=$?
if [[ $RC -ne 0 ]]; then
    check_max_rc $RC
    MESSAGE=$(print "Unable to determine DBPARTITIONNUM for"\
        "$SCHEMA.$TABLE"\
        "\n\tOutput follows:\n$OUT")
    add_edit_message "$MESSAGE"
else
    print "$OUT" | egrep -w "^[ ]*xyyzz" \
        | read junk COLNAME DBPARTNUM
    if [[ -n "$COLNAME" && -n "$DBPARTNUM" ]]; then
        PARTITION_CLAUSE="where dbpartitionnum($COLNAME) = $DBPARTNUM"
    else
        add_edit_message "Unable to determine DB partition number."
    fi
fi

check_edit_messages          # If any messages, print them and exit

## Compute average row length using average length of table columns
## +1 for each NULL indicator field +4 for each VARCHAR length field.
##
typeset -i BIG_NUMBER=999999
DB2_CMD="select 'xyyzz'
        ,sum(case when avgcollen = -1 then $BIG_NUMBER
                  else avgcollen
                end)
        ,sum(case when typename = 'VARCHAR' then 4
                  else 0
                end)
        ,sum(case when nulls = 'Y' then 1
                  else 0
                end)
        from syscat.columns
        where tabschema='$SCHEMA' and tabname='$TABLE'
    "
    # End of db2 command

db2_command "$DB2_CMD"
RC=$?

```

```

if [[ $RC -eq 0 ]]; then
    print "$OUT" | egrep -w "^[*xxyyzz]" \
    | read junk COLUMN_LENGTH LENGTH_FIELD_SUM NULL_SUM
    AVG_ROW_LENGTH_1=$(( COLUMN_LENGTH + LENGTH_FIELD_SUM + NULL_SUM ))
else
    check_max_rc $RC
    print "DB2 command failed, RC=$RC. Output follows:\n"
    print_message "$MESSAGE"
    if [[ ! $USER_ROW_LENGTH -gt 0 ]]; then
        exit # If no user override, exit
    fi
fi

if [[ $AVG_ROW_LENGTH_1 -gt $BIG_NUMBER ]]; then
    MESSAGE=$(print "Computed average row length is excessively high." \
    "\n\tApparently RUNSTATS has not been run for table" \
    "$SCHEMA.$TABLE'.")
    print_message "$MESSAGE"
    AVG_ROW_LENGTH_1=0
fi

DB2_CMD="-v select 'xxyyzz', a.card, a.npages, a.fpages,
    b.pagesize, b.extentsize
    from syscat.tables a, syscat.tablespaces b
    where a.tabname='$TABLE' and a.tabschema='$SCHEMA'
    and a.tbpaceid=b.tbpaceid"
db2_command "$DB2_CMD"
RC=$?
if [[ $RC -eq 0 ]]; then
    print "$OUT" | egrep -w "^[*xxyyzz]" \
    | read junk CARD NPAGES FPAGES PGSZ EXTSZ
else
    check_max_rc $RC
    MESSAGE=$(print "DB2 command failed for table '$SCHEMA.$TABLE'." \
    "\n\t\tPossible reasons are:" \
    "\n\t\t1. Table and/or schema invalid." \
    "\n\t\t2. Table is inaccessible." \
    "\n\n\tDB2 output follows:\n$OUT")
    print_message "$MESSAGE"
    exit
fi

PAGE_SIZE=${PAGE_SIZE:-$PGSZ}

EXTENT_SIZE=${EXTENT_SIZE:-$EXTSZ}

## Try to determine average row length from CARD and NPAGES
if [[ ! -z $CARD && $CARD -gt 0 ]] \
    && [[ ! -z $NPAGES && $NPAGES -gt 0 ]]; then

```

```

typeset -i ROWS_PER_PAGE_2=$(( CARD / NPAGES ))
if [[ $ROWS_PER_PAGE_2 -le 0 ]]; then
    AVG_ROW_LENGTH_2=0
else
    # Use page size from syscat.tablespace
    AVG_ROW_LENGTH_2=$(( $PGSZ / $ROWS_PER_PAGE_2 ))
fi
fi

## Select ROWS_PER_PAGE as minimum of from CARD or from AVGCOLLEN
## if row length is input, use that. No need to calculate.

## Which row length to use?
## determined from CARD and NPAGES.
if [[ $AVG_ROW_LENGTH_1 -gt 0 ]]; then
    MESSAGE=$(print "Average row length calculated from average"\
        "column lengths plus"\
        "\\n\\tNULL indicators and VARCHAR length fields is"\
        "'$AVG_ROW_LENGTH_1'.")
    update_comment "$MESSAGE"
    if [[ $AVG_ROW_LENGTH_2 -gt 0 ]]; then
        update_comment $(print "Average row length calculated from CARD and"\
            "NPAGES is '$AVG_ROW_LENGTH_2'.")
        check_tolerance $AVG_ROW_LENGTH_1 $AVG_ROW_LENGTH_2
        RC=$?
        if [[ $RC -eq 0 ]]; then
            if [[ $AVG_ROW_LENGTH_2 -gt $AVG_ROW_LENGTH_1 ]]; then
                MDC_AVG_ROW_LENGTH=$AVG_ROW_LENGTH_2
            else
                MDC_AVG_ROW_LENGTH=$AVG_ROW_LENGTH_1
            fi
        else
            MESSAGE=$(print "Difference between the row lengths exceeds"\
                "tolerance, $TOLERANCE%."\
                "\\n\\t**** Check ****")
            update_comment "$MESSAGE"
            MDC_AVG_ROW_LENGTH=$AVG_ROW_LENGTH_1
        fi
    fi
else
    if [[ $AVG_ROW_LENGTH_2 -gt 0 ]]; then
        update_comment $(print "Average row length calculated from CARD and"\
            "NPAGES is '$AVG_ROW_LENGTH_2'.")
        MDC_AVG_ROW_LENGTH=$AVG_ROW_LENGTH_2
    fi
fi

if [[ $USER_ROW_LENGTH -gt 0 ]]; then
    update_comment "Average row length specified by user."

```

```

MDC_AVG_ROW_LENGTH=$USER_ROW_LENGTH
if [[ $MDC_AVG_ROW_LENGTH = 0 ]]; then      # Houston, we have a problem..
    RC=22                                # Pseudo code for this error
    check_max_rc $RC
    MESSAGE=$(print "Unable to determine row length and none was"\
        "provided by user."\
        "\n\t----- EXITING -----")
    print_message "MESSAGE"
    exit
fi
else
    if [[ $ROW_ADJUST -ne 0 ]]; then
        update_comment $(print "Row length was adjusted by user"\
            "specified amount, $ROW_ADJUST")
        MDC_AVG_ROW_LENGTH=$(( $MDC_AVG_ROW_LENGTH + $ROW_ADJUST ))
    fi
fi

if [[ $MDC_AVG_ROW_LENGTH -le 0 ]]; then      # Bad news
    RC=23                                # Pseudo code for this error
    check_max_rc $RC
    MESSAGE=$(print "Average row length must be greater than 0."\
        "Current value is '$MDC_AVG_ROW_LENGTH'."\
        "\n\t----- EXITING -----")
    print_message "$MESSAGE"
    exit
fi

typeset -i USEABLE_PAGE_SPACE=$(( PAGE_SIZE - 91 ))
if [[ $MDC_AVG_ROW_LENGTH -gt $USEABLE_PAGE_SPACE ]]; then      # Bad news, too
    RC=24                                # Pseudo code for this error
    check_max_rc $RC
    MESSAGE=$(print "Average row length, '$MDC_AVG_ROW_LENGTH', exceeds"\
        "maximum row size"\
        "\n\tfor page, '$USEABLE_PAGE_SPACE'."\
        "\n\t----- EXITING -----")
    print_message "$MESSAGE"
    exit
fi

typeset -i ROWS_PER_PAGE=$(( USEABLE_PAGE_SPACE / MDC_AVG_ROW_LENGTH ))
if [[ $ROWS_PER_PAGE -gt 255 ]]; then
    update_comment $(print "Rows per page calculated as '$ROWS_PER_PAGE'."\
        "Reducing to '$MAX_ROWS_PER_PAGE'")
    ROWS_PER_PAGE=$MAX_ROWS_PER_PAGE
fi

typeset -i ROWS_PER_EXTENT=$(( ROWS_PER_PAGE * EXTENT_SIZE ))

```

PARAMETER_MESSAGE=""

Creating MDC sql using the following parameters:

Database Alias: \$DB_ALIAS
Table Name: \$SCHEMA.\$TABLE
Row Length Adjustment: \$ROW_ADJUST
Average Row Length: \$MDC_AVG_ROW_LENGTH
Page Size: \$PAGE_SIZE
Extent Size: \$EXTENT_SIZE
Rows/Page: \$ROWS_PER_PAGE
Rows/Extent: \$ROWS_PER_EXTENT
DB Partition: \$DBPARTNUM
Dimensions: \$DIMENSIONS

Comments:

\$COMMENTS

```
“
typeset -u ANSWER="ask"
while [[ $ANSWER = 'ASK' ]]; do
    print_message "$PARAMETER_MESSAGE"
    print_message "Do you wish to continue? (y or n) -->\c"
    read ANSWER
    print "\n\n" | tee $TTY # Skip a couple of lines
    case "$ANSWER" in
        Y) break
        ;;
        N) MESSAGE="Discontinuing at user request."
            print_message "$MESSAGE"
            RC=33 # Set a dummy return code
            check_max_rc $RC
            exit
            ;;
        *) MESSAGE="'$ANSWER' is an invalid response."
            print_message "$MESSAGE"
            ANSWER="ask" # Try again
            ;;
    esac
done

sed < $SQL_SKELETON_FILE \
    > $SQL_FILE \
    -e's?YOUR_DIMENSIONS2?'"$DIMENSIONS2"'?g' \
    -e's?YOUR_DIMENSIONS?'"$DIMENSIONS"'?g' \
    -e's?YOUR_SCHEMA?'$SCHEMA'?g' \
    -e's?YOUR_TABLE?'$TABLE'?g' \
    -e's?YOUR_ROWS_PER_EXTENT?'$ROWS_PER_EXTENT'?g' \
    -e's?YOUR_DBPARTITION_CLAUSE?'"$PARTITION_CLAUSE"'?g' \
RC=$?
if [[ $RC -ne 0 ]]; then
```

```
    print_message "sed command failed, RC=$RC."
    check_max_rc $RC
    exit
fi

print_message "Submitting generated SQL file, '$SQL_FILE'"
db2_command "-tvf $SQL_FILE"
RC=$?
if [[ $RC -eq 0 ]]; then
    print_message "Output follows:\n$OUT"
else
    check_max_rc $RC
    MESSAGE="MDC design sql failed, RC=$RC. Output_follows:\n$OUT"
    print_message "$MESSAGE"
fi

exit $MAX_RC
```

Additional 64-bit migration considerations

This appendix provides additional 64-bit migration considerations and contains the following topics:

- ▶ Environments that should not be migrated to Version 8.1
- ▶ Discontinued and deprecated functions

Environments that should not be migrated to Version 8.1

The following environments should not be migrated to DB2 UDB V8.1:

- ▶ DB2 Relational Connect and DB2 Life Sciences Data Connect

IBM is restructuring and enhancing its offerings to focus on information integration. These activities will include introducing new functionality that replaces and extends federated functionality previously available with DB2 Relational Connect and DB2 Life Sciences Data Connect. Details will be announced later.

Customers accessing federated data sources using DB2 Relational Connect Version 7 or DB2 Life Sciences Data Connect Version 7 should wait until this new functionality is available before upgrading to DB2 UDB Version 8.1.

DB2 UDB Version 8.1 has built-in capability to federate relational data across the IBM Family of databases, including DB2 UDB and Informix IDS.

Customers who only want to use federated data from DB2 UDB and Informix IDS can begin upgrading to Version 8.1.

- ▶ DB2 Query Patroller environments

IBM plans to release Version 8 of DB2 Query Patroller, which is intended for use with DB2 UDB Version 8.1 databases.

However, DB2 Query Patroller Version 8 is not being made available at this time. Customers using Version 7.2 of the DB2 Query Patroller Version 7.2, which shipped with DB2 Warehouse Manager Version 7.2, should not upgrade to DB2 UDB Version 8.1 until Version 8.1 of DB2 Query Patroller is available. DB2 Query Patroller Version 8 will deliver enhanced functionality to better manage and control all aspects of query submission.

Discontinued and deprecated functions

This topic covers the discontinued and deprecated functions, ones that are considered obsolete and have been phased out in DB2 UDB V8.1:

- ▶ The following communication protocols are no longer supported:
 - IPX/SPX as a DB2 client-server protocol. This means that DB2 UDB Version 8 servers cannot accept IPX/SPX connections, and DB2 UDB Version 8 clients cannot be configured to use IPX/SPX.
 - SUNLINK SNA because SUN has announced that protocol's discontinuation.
- ▶ The following operating system environments are not supported in Version 8:
 - OS/2
 - PTX or NUMA-Q

– Windows 95

- ▶ The Generalize Replication Subscription function of the Satellite Administration Center is not supported.
- ▶ The asnmobility function and the asnjnet function are removed from Replication.
- ▶ The db2alert.log (alerts) facility has been removed. Users of the db2alerts.log should use the administration notification log as a replacement.
- ▶ The performance monitor capability of the Control Center has been removed. Users of the performance monitor should examine the functions of the Health Center (which is part of the Control Center) and the DB2 Performance Expert for Multiplatforms, Version 1 (a separate add-on tool) to replace the performance monitoring function they are using.
- ▶ The type 3 JDBC driver (also called the 'net' driver or applet driver) is deprecated as of Version 8. This means that, although it will be fully supported in this release, you should migrate existing DB2 JDBC applets to the new type 4 driver, because the type 3 driver will be phased out entirely in a future version.
- ▶ Distributed Computing Environment (DCE) changes

Support for DCE security has been removed in response to the industry move towards Kerberos as the mechanism for secure network authentication and single sign-on. Windows 2000 customers should consider moving to Kerberos as a replacement for DCE Security prior to migrating to DB2 UDB V8.1. Kerberos support for Windows 2000 was made available in Version 7.2. In a future delivery of DB2 UDB V8.1 will extend Kerberos support to UNIX and Linux servers and clients.

Lightweight Directory Access Protocol (LDAP) has become the industry standard for the enterprise class directory implementation. Customers should consider moving to LDAP as a replacement for DCE Directory support prior to migrating to DB2 UDB V8.1. LDAP support is available on all DB2 UDB Version 8 supported platforms with the exception of HP-UX and Linux. LDAP support for HP-UX and Linux is planned for future delivery of DB2 UDB V8.1.

- ▶ Authentication at the DB2 Connect gateway is no longer possible.

Version 8 removes the ability to authenticate a user at the DB2 Connect gateway. Authentication can only be done at the client (using CLIENT authentication) or at the server (using SERVER or SERVER_ENCRYPT). These options must be cataloged at the client in the database directory, or left as NOT_SPEC.

DCS and DCS_ENCRYPT now have exactly the same meaning as SERVER and SERVER_ENCRYPT. Any authentication entries in the database manager configuration file, or entries in the database catalog with authentication DCS or DCS_ENCRYPT will be migrated to SERVER and

SERVER_ENCRYPT, respectively. Any attempt to specify DCS or DCS_ENCRYPT will result in these types being mapped to SERVER or SERVER_ENCRYPT.

If there is a DCS entry for a given database catalog entry (signifying that DB2 Connect is being used), and the authentication type is SERVER or SERVER_ENCRYPT, the type will be migrated to CLIENT to preserve existing behavior. If there is no DCS catalog entry, migration does not take place.

There is a case where a value of DCS in the database manager configuration file at the server had a different meaning than a value of SERVER. For federated systems, a value of DCS or DCS_ENCRYPT meant that no authentication would take place at the federated gateway; it was expected that authentication would take place at the final data source. A new database manager configuration parameter was added to accommodate this: FED_NOAUTH. If this parameter is set to YES, and the authentication type is SERVER or SERVER_ENCRYPT, there will be no authentication at the federated gateway, preserving the old behavior. This value is properly set during migration: If the authentication type is DCS or DCS_ENCRYPT in the database manager configuration file, FED_NOAUTH will be set to YES.

DB2 UDB configuration parameters

This appendix contains a copy of information concerning the database manager and database configuration parameters summary from *DB2 UDB Administration Guide: Performance*, SC09-4821, and is included here as reference material for 2.3.11, “Recommended parameters for performance” on page 82, and 7.5.2, “Online changes to configuration parameters” on page 223.

Database manager configuration parameters summary

Table D-1 lists the parameters in the database manager (DBM) configuration file for database servers. When changing the database manager configuration parameters, consider the detailed information for each parameter. Specific operating environment information including defaults is part of each parameter description.

The column *Automatic* in the following table indicates whether the parameter supports the **AUTOMATIC** keyword on the **DB2 UPDATE DBM CFG** command. If you set a parameter to **AUTOMATIC**, DB2 UDB will automatically adjust the parameter to reflect current resource requirements.

The column *Online* indicates whether the parameter is configurable online or not.

The column *Impact* provides an indication of the relative importance of each parameter as it relates to system performance. It is impossible for this column to apply accurately to all environments; you should view this information as a generalization.

- ▶ High: Indicates the parameter can have a significant impact on performance. You should consciously decide the values of these parameters; which, in some cases, will mean that you accept the default provided.
- ▶ Medium: Indicates the parameter can have some impact on performance. Your specific environment and needs will determine how much tuning effort should be focused on these parameters.
- ▶ Low: Indicates that the parameter has a less general or less significant impact on performance.
- ▶ None: Indicates that the parameter does not directly impact performance. While you do not have to tune these parameters for performance, they can be very important for other aspects of your system configuration, such as enabling communication support.

Note: The *cpuspeed* parameter can have a significant impact on performance, but you should use the default value, except in very specific circumstances, as documented in the parameter description.

Table D-1 Configurable database manager configuration parameters

Parameter	Online	Automatic	Impact	Additional information
agentpri	No	No	High	Priority of Agents
agent_stack_sz	No	No	Low	Agent Stack Size
aslheapsz	No	No	High	Application Support Layer Heap Size
audit_buf_sz	No	No	High	Audit Buffer Size
authentication	No	No	Low	Authentication Type
catalog_noauth	Yes	No	None	Cataloging Allowed without Authority
comm_bandwidth	Yes	No	Medium	Communications Bandwidth
conn_elapse	Yes	No	Medium	Connection Elapse Time
cpuspeed	Yes	No	Low	CPU Speed
datalinks	No	No	Low	Enable Data Links Support
dft_account_str	Yes	No	None	Default Charge-Back Account

dft_monswitches dft_mon_bufpool dft_mon_lock dft_mon_sort dft_mon_stmt dft_mon_table dft_mon_timestam p dft_mon_uow	Yes	No	Medium	Default Database System Monitor Switches
dftdbpath	Yes	No	None	Default Database Path
diaglevel	Yes	No	Low	Diagnostic Error Capture Level
diagpath	Yes	No	None	Diagnostic Data Directory Path
dir_cache	No	No	Medium	Directory Cache Support
discover	No	No	Medium	Discovery Mode
discover_comm	No	No	Low	Search Discovery Communications Protocols
discover_inst	Yes	No	Low	Discover Server Instance
fcm_num_buffers	Yes	No	Medium	Number of FCM Buffers
fed_noauth	Yes	No	None	Bypass Federated Authentication
federated	No	No	Medium	Federated Database System Support
fenced_pool	No	No	Medium	Maximum Number of Fenced Processes
health_mon	Yes	No	Low	Health Monitoring
indexrec	Yes	No	Medium	Index Re-creation Time
instance_memory	No	Yes	Medium	Instance Memory
intra_parallel	No	No	High	Enable Intra-Partition Parallelism
java_heap_sz	No	No	High	Maximum Java Interpreter Heap Size
jdk_path	No	No	None	Java Development Kit Installation Path
keepfenced	No	No	Medium	Keep Fenced Process
maxagents	No	No	Medium	Maximum Number of Agents
maxcagents	No	No	Medium	Maximum Number of Concurrent Agents
max_connections	No	No	Medium	Maximum Number of Client Connections
max_connretries	Yes	No	Medium	Node Connection Retries configuration

max_coordagents	No	No	Medium	Maximum Number of Coordinating Agents
max_querydegree	Yes	No	High	Maximum Query Degree of Parallelism
max_time_diff	No	No	Medium	Maximum Time Difference Among Nodes
maxtotfilop	No	No	Medium	Maximum Total Files Open
min_priv_mem	No	No	Medium	Minimum Committed Private Memory
mon_heap_sz	No	No	Low	Database System Monitor Heap Size
nname	No	No	None	NetBIOS Workstation Name
notifylevel	Yes	No	Low	Notify Level
numdb	No	No	Low	Maximum Number of Concurrently Active Databases
num_initagents	No	No	Medium	Initial Number of Agents in Pool
num_initfenced	No	No	Medium	Initial Number of Fenced Processes
num_poolagents	No	No	High	Agent Pool Size
priv_mem_thresh	No	No	Medium	Private Memory Threshold
query_heap_sz	No	No	Medium	Query Heap Size
resync_interval	No	No	None	Transaction Resync Interval
rqrioblk	No	No	High	Client I/O Block Size
sheapthres	No	No	High	Sortheap Threshold
spm_log_file_sz	No	No	Low	Sync Point Manager Log File Size
spm_log_path	No	No	Medium	Sync Point Manager Log File Path
spm_max_resync	No	No	Low	Sync Point Manager Resync Agent Limit
spm_name	No	No	None	Sync Point Manager Name
start_stop_time	Yes	No	Low	Start and Stop Timeout
svcname	No	No	None	TCP/IP Service Name
sysadm_group	No	No	None	System Administration Authority Group Name
sysctrl_group	No	No	None	System Control Authority Group Name
sysmaint_group	No	No	None	System Maintenance Authority Group Name

tm_database	No	No	None	Transaction Manager Database Name
tp_mon_name	No	No	None	Transaction Processor Monitor Name
tpname	No	No	None	APPC Transaction Program Name
trust_allclnts	No	No	None	Trust All Clients
trust_clntauth	No	No	None	Trusted Clients Authentication
use_sna_auth	Yes	No	None	Use SNA Authentication

Table D-2 indicates the database manager informational parameters.

Table D-2 Database manager informational parameters

Parameter	Additional information
nodetype	Machine Node Type
release	Configuration File Release Level

Database configuration parameters summary

Table D-3 lists the parameters in the database configuration file. When changing the database configuration parameters, consider the detailed information for the parameter.

The column *Automatic* in the following table indicates whether the parameter supports the **AUTOMATIC** keyword on the **db2 update db cfg** command. If you set a parameter to **AUTOMATIC**, DB2 UDB will automatically adjust the parameter to reflect current resource requirements.

The column *Impact* provides an indication of the relative importance of each parameter as it relates to system performance. It is impossible for this column to apply accurately to all environments; you should view this information as a generalization.

- ▶ **High:** Indicates the parameter can have a significant impact on performance. You should consciously decide the values of these parameters; which, in some cases, will mean that you accept the default provided.
- ▶ **Medium:** Indicates the parameter can have some impact on performance. Your specific environment and needs will determine how much tuning effort should be focused on these parameters.
- ▶ **Low:** Indicates that the parameter has a less general or less significant impact on performance.

- None: Indicates that the parameter does not directly impact performance. While you do not have to tune these parameters for performance, they can be very important for other aspects of your system configuration, such as enabling communication support.

Note: Changing the *indexsort* parameter to a value other than the default can have a negative impact on the performance of creating indexes. You should always try to use the default for this parameter.

Table D-3 Database configuration parameters

Parameter	Automatic	Impact	Additional Information
app_ctl_heap_sz	No	Medium	Application Control Heap Size
appgroup_mem_sz	No	Medium	Maximum Size of Application Group Memory Set
applheapsz	No	Medium	Application Heap Size
autorestart	No	Low	Auto Restart Enable
avg_appls	No	High	Average Number of Active Applications
blk_log_dsk_ful	No	None	Block on Log Disk Full
catalogcache_sz	No	High	Catalog Cache Size
chnpggs_thresh	No	High	Changed Pages Threshold
database_memory	Yes	Medium	Database Shared Memory Size
dbheap	No	Medium	Database Heap
dft_degree	No	High	Default Degree
dft_extent_sz	No	Medium	Default Extent Size of Table Spaces
dft_loadrec_ses	No	Medium	Default Number of Load Recovery Sessions
dft_prefetch_sz	No	Medium	Default Prefetch Size
dft_queryopt	No	Medium	Default Query Optimization Class
dft_refresh_age	No	Medium	Default Refresh Age
dft_sqlmathwarn	No	None	Continue upon Arithmetic Exceptions
discover_db	No	Medium	Discover Database
dlchktime	No	Medium	Time Interval for Checking Deadlock

dl_expint	No	None	Data Links Access Token Expiry Interval
dl_num_copies	No	None	Data Links Number of Copies
dl_time_drop	No	None	Data Links Time After Drop
dl_token	No	Low	Data Links Token Algorithm
dl_upper	No	None	Data Links Token in Upper Case
dl_wt_iexpint	No	None	Data Links Write Token Initial Expiry Interval
dyn_query_mgmt	No	Low	Dynamic SQL Query Management
estore_seg_sz	No	Medium	Extended Storage Memory Segment Size
groupheap_ratio	No	Medium	Percent of Memory for Application Group Heap
indexrec	No	Medium	Index Re-creation Time
locklist	No	High when it affects escalation	Maximum Storage for Lock List
locktimeout	No	Medium	Lock Timeout
logbufsz	No	High	Log Buffer Size
logfilsiz	No	Medium	Size of Log Files
logprimary	No	Medium	Number of Primary Log Files
logretain	No	Low	Log Retain Enable
logsecond	No	Medium	Number of Secondary Log Files
maxappls	Yes	Medium	Maximum Number of Active Applications
maxfilop	No	Medium	Maximum Database Files Open per Application
maxlocks	No	High when it affects escalation	Maximum Percent of Lock List Before Escalation
mincommit	No	High	Number of Commits to Group
min_dec_div_3	No	High	Decimal Division Scale to 3
newlogpath	No	Low	Change the Database Log Path
mirrorlogpath	No	Low	Mirror Log Path
num_db_backups	No	None	Number of Database Backups

num_estore_segs	No	Medium	Number of Extended Storage Memory Segments
num_freqvalues	No	Low	Number of Frequent Values Retained
num_iocleaners	No	High	Number of Asynchronous Page Cleaners
num_ioservers	No	High	Number of I/O Servers
num_quantiles	No	Low	Number of Quantiles for Columns
overflowlogpath	No	Medium	Overflow Log Path
pckcachesz	No	High	Package Cache Size
rec_his_retentn	No	None	Recovery History Retention Period
seqdetect	No	High	Sequential Detection Flag
sheapthres_shr	No	High	Sortheap Threshold for Shared Sorts
softmax	No	Medium	Recovery Range and Soft Checkpoint Interval
SORTHEAP	No	High	Sortheap Size
stat_heap_sz	No	Low	Statistics Heap Size
stmthep	No	Medium	Statement Heap Size
trackmod	No	Low	Track Modified Pages Enable
tsm_mgmtclass	No	None	Tivoli Storage Manager Management Class
tsm_nodename	No	None	Tivoli Storage Manager Node Name
tsm_owner	No	None	Tivoli Storage Manager Owner Name
tsm_password	No	None	Tivoli Storage Manager Password
userexit	No	Low	User Exit Enable
util_heap_sz	No	Low	Utility Heap Size

Table D-4 indicates the database informational parameters.

Table D-4 Database configuration informational parameters

Parameter	Additional Information
backup_pending	Backup Pending Indicator
codepage	Code Page for the Database
codeset	Codeset for the Database

collate_info	Collating Information
country	Database Territory Code
database_consistent	Database is Consistent
database_level	Database Release Level
log_retain_status	Log Retain Status Indicator
loghead	First Active Log File
logpath	Location of Log Files
multipage_alloc	Multipage File Allocation Enabled
numsegs	Default Number of SMS Containers
release	Configuration File Release Level
restore_pending	Restore Pending
rollfwd_pending	Roll Forward Pending Indicator
territory	Database Territory
user_exit_status	User Exit Status Indicator

Additional material

This redbook refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this redbook is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

<ftp://www.redbooks.ibm.com/redbooks/SG246917>

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the redbook form number, SG24-6917.

Using the Web material

The additional Web material that accompanies this redbook includes the following file:

<i>File name</i>	<i>Description</i>
pgm.zip	Zipped code samples including MDC dimension analyzer script, dbgen and tables load

System requirements for downloading the Web material

The following system configuration is recommended:

Hard disk space: 100 KB minimum
Operating System: AIX5L

How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material zip file into this folder before sending the different files by FTP to your AIX5L system to run them.

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 368.

- ▶ *IBM eServer pSeries 670 and pSeries p690*, SG24-7040
- ▶ *AIX5L Workload Manager (WLM)*, SG24-5977
- ▶ *DB2 UDB Backup and Recovery with ESS Copy Services*, SG24-6557
- ▶ *IBM ESS and IBM DB2 UDB Working Together*, SC24-6262
- ▶ *DB2 UDB V7.1 Performance Tuning Guide*, SG24-6012
- ▶ *DB2 UDB's High Function Business Intelligence in e-business*, SG24-6546
- ▶ *Managing VLDB Using DB2 UDB EEE*, SG24-5105
- ▶ *Enhance Your Business Applications: Simple integration of Advanced Data Mining Functions*, SG24-6879

Other resources

These publications are also relevant as further information sources:

- ▶ *DB2 High Performance Unload for Multiplatforms User's Guide*, SC27-1623
- ▶ *DB2 Administration Guide: Planning Version 8*, SC09-4822
- ▶ *DB2 Administration Guide: Implementation Version 8*, SC09-4820
- ▶ *DB2 Administration Guide: Performance Version 8*, SC09-4821
- ▶ *DB2 Quick Beginnings for DB2 Servers Version 8*, GC09-4836
- ▶ *DB2 SQL Reference Volume 2 Version 8*, SC09-4844
- ▶ *DB2 SQL Reference Volume 2 Version 8*, SC09-4845
- ▶ *DB2 Command Reference Version 8*, SC09-4828
- ▶ *DB2 What's New Version 8*, SC09-4848
- ▶ *DB2 System Monitor Guide and Reference Version 8*, SC09-4947

Referenced Web sites

These Web sites are also relevant as further information sources:

- ▶ IBM Software Homepage
<http://www.software.ibm.com/>
- ▶ IBM Database Management Homepage
<http://www.software.ibm.com/data/>
- ▶ TPC-H Web site
<http://www.tpc.org/tpch/>

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.

Index

Numerics

24x7 2, 77
32-Bit 264, 270
64-Bit 7, 15, 81, 264, 270

A

access plan 178, 181, 186, 190
administration support 247
aggregation 10, 16, 172, 177
 final 186
 intermediate 186
 parallelization 186
 partial 186
AIX 5L 21
AIX LVM options 40
AIX parameters 92, 95
AIX striping 40
alerts 18, 250
allocate space 60
ALLOW READ ACCESS 112, 123–124, 129
ALTER TABLESPACE 58, 61
ANALYZE 123, 125
ANYORDER 118, 122
Archival Logging 43, 76
archival logging 77
array 56, 61
Ascential Data Stage 130
ASCII 113, 125, 299
 delimited 119
 non-delimited 119
AST 10, 167, 169–170, 194
asynchronous I/O 91
asynchronous IO 92
AUTOCONFIGURE 221, 261
autoloader 113, 119, 124, 193
automate 2
Automatic Summary Table 167
automation 4, 248
availability xix, 5, 21, 41, 43, 76, 132, 168
 24x7 191, 229

B

backup 229
 database 118, 231
 incremental 232
 offline 77, 231
 online 77
 table space 118
balanced disk layout 48
BEGIN NEW STRIPE SET 58
BID 149, 161
BLK_LOG_DSK_FUL 8, 19
block 137–138
block identifier 149
block index 9, 130, 141, 148–149
 composite 150
broadcast join 182, 184
BTQ 189
buffered insert 130
bufferpool 30, 79, 168
bufferpool hit ratio 80
bulk load 112
business cycle 3, 131
business processes 2

C

cache data 168
catalog 67
 caching 11
 partition 49, 66
 table space 65
cell 140, 143–144, 149, 151
CHECK PENDING 114, 300
CHECK PENDING NO ACCESS 170
CHECK_TRUNCATION 125
Circular Logging 76
circular logging 43
CLIENT 119, 121
client connections 12
Cluster 1600 23
clustering index 9, 136, 165
clusters 6, 23
COLLECT STATISTICS 216
collocated joins 68, 181

- collocation 69, 72, 172, 174, 180–181
- colony switch 23
- commit point 119
- COMMITCOUNT 130
- commits 130
- concurrency 130
- concurrent activity 42
- concurrent load 131
- Configuration Advisor 221, 254
- configuration parameters 8, 25, 178, 272, 355
 - AUTOMATIC 355
 - online 355
- connection concentrator 12
- connections
 - monitor 30
- containers 43, 53, 56, 61, 242
 - configuration 52
 - rebalancing 242
 - size 57
- continuous feed 131
- continuous parallel load 131
- coordinator 122, 127, 178, 186
- COPY 118
- cordinator
 - node 269
- correlation 180
- CPU shares 31
- CPU_PARALLELISM 118
- CPUSPEED 86
- CREATE BUFFERPOOL 81
- CREATE DATABASE PARTITION GROUP 50
- create index
 - COLLECT STATISTICS 216
- CREATE TABLE 68, 141, 147
- cubes 3
- CuOD 24–25
- CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION 173
- CURSOR 119, 121
- customer
 - table 113
- customer-centered infrastructure 2
- D**
- DAS 284
- data availability 3
- data generation 113
- Data Links Manager 271
- data maintenance 8, 20
- data partitioning 44
- data warehouse 2
- database
 - containers striping 236
 - migration 269
- database configuration parameters 272, 355
- database design 44
- Database Managed Storage 54, 156
- database manager configuration parameters 84, 272, 355
- database partition group 49, 72, 269
- Database Partitioning Feature 5
- database partitions 6, 47
- databases
 - migration 282
- DataJoiner 265
- date 148
- db2 create event monitor 30
- DB2 event monitor 30
- DB2 explain 30
- DB2 Extenders 271
- db2 get recommendations for health indicator 251
- db2 get snapshot 30
- DB2 governor 30
- DB2 home file system 96
- DB2 I/O parallelism 53
- DB2 instance
 - create 101
 - start 110
- db2 migrate database 282
- DB2 query parallelism 53
- DB2 Query Patroller 142
- DB2 query rewrite 19
- DB2 Recovery Expert 230
- db2 set event monitor 30
- DB2 Setup wizard 97
- db2 snapshot 30, 249
- DB2 suspended I/O 233
- DB2 UDB configuration 75
- db2 update monitor switches 30
- DB2 variables 82
- DB2 Warehouse Manager 130
- db2_all 231, 238
- DB2_FORCE_FCM_BP 84
- DB2_InstanceName 106
- DB2_InstanceName_END 106
- DB2_PARALLEL_IO 83
- DB2_PARTITIONEDLOAD_DEFAULT 119

- DB2_STRIPED_CONTAINERS 83
- db2advis 261
- db2agent 127
- db2atld 113, 119, 124
- DB2ATLD_PORTS 124, 126
- db2batch 30
- db2ckmig 269, 281, 286
- db2dart 18, 244
- db2empfa 91, 165
- db2eva 30
- db2evmon 30
- db2exfmt 187
- db2expln 30, 264
- db2gov 30
- db2gpmmap 125
- db2home 96
- db2icrt 101, 279
- db2imigr 267, 279
- db2inidb 232–233
- DB2INSPF 245
- db2iupdt 266, 279, 292
- db2lcata 127
- db2level 280, 291–292
- db2linit 127
- db2lload 127
- db2lmibm 127
- db2load 193
- db2LoadQuery 193
- db2look 289
- db2lpart 127
- db2lpprt 127
- db2lurex 127
- db2mtrk 252
- db2nodes.cfg 46, 107
 - DB Partition Number 108
 - hostname 108
 - logical port 108
 - netname 108
- db2profile 105
- db2rbind 283
- db2setup 97
- db2start 110
- db2undgp 283
- dbgen 113, 299
- DBHEAP 81
- DBPARTITIONNUM 17
- deadlocks
 - monitor 30
- defect 25

- deferred refresh 171
- delete obsolete rows 130
- design 51
 - recommendations 51
- Design Advisor 260
- DFT_DEGREE 86
- DFT_QUERYOPT 173
- diaglevel 269
- DIAGPATH 81, 245
- DIMENSION 147
- dimension 139
 - analyzer 316
 - multicolumn 149
- dimension block index 141
- dimension key 150, 155
- dimension tables 175
- directed join 182
- disk layout 48, 60
- disk space 42
- DISTFILE 125
- distribution statistics 14
- DLPAR 24
- DMS 54, 57, 60, 78, 87, 156, 216, 241
 - container 83, 192, 241, 243
 - file tablespaces 56
 - raw tablespaces 55
- DPF 5, 177
- DROP 243
- drop table 131
- DUMPFIL 117
- dynamic CPU sparing 25
- Dynamic LPAR 24

E

- e-analytics 2–3
- ease of use xix
- easy-to-use 6
- e-BI 1
- e-Business Intelligence 1, 5
- Enterprise Storage Server 230
- equijoin 71–72, 182–183
- errors
 - partition-level 124
- ESS 36, 41, 57, 64, 83, 230, 234
 - Copy Services 230
- ETL 130
- evaluate_dimensions.ksh 319
- evaluate_dimensions.sql 322

- even distribution 68, 71
- event monitor 90, 142
- exception table 117
- EXPLAIN 187
 - tables 283
- EXTEND 242
- extent 60, 141, 143
 - allocation 156
 - size 57, 59–60, 152, 165
 - utilization 323
- EXTENTSIZ 61
- Extract-Transform-Load 130
- extshm 93

F

- fact table 142
- FASTPARSE 118
- FCM daemons 84
- FCM parameters 87
- FCM_NUM_ANCHORS 273
- FCM_NUM_BUFFERS 86
- FCM_NUM_CONNECT 273
- FCM_NUM_RQB 273
- federated database 168
- FILE_TRANSFER_CMD 120, 127
- filesystem 78
- FlashCopy 230, 234
- FOR EXCEPTION 117
- FORCE YES 124
- FPAGES 145
- frame 23
- Free Space Control Record 149
- FSCR 149

G

- get db 223
- get dbm 223
- GET SNAPSHOT FOR TABLES 211
- get_row_partition () 180
- get_table_partition () 180
- gigabyte ethernet 94
- granularity 150, 153
- GROUP BY 71, 142, 170, 186
- groups
 - create 97
- GUI tools 18

H

- hardware partitioning 45
- hash join 7
- hash partitioning 68
- Health Center 18, 249
- Health Monitor 247, 249
- Heath Monitor 18
- HOSTNAME 120
- HSJOIN 189

I

- IBM Data Replication products 130
- IBMCATGROUP 50
- IBMDEFAULTGROUP 50
- IBMTMPGROUP 50
- immediate refresh 171
- IMPORT 114, 129
- IN predicate 142
- INCREMENTAL 170
- incremental load 112
- incremental updates 129
- indexes 178, 216, 260
 - create 8
 - maintenance 131
 - rebuild 8
 - type-1 218
 - type-2 218
- initial population 112
- INSERT through UNION ALL view 8, 20
- INSPECT 18, 192, 241, 244
- installation 95, 97
 - checklist 95
- instance
 - migration 267, 278, 292
- integrated tool 3
- internal communications 124
- intra database replication 173
- INTRA_PARALLEL 86
- intra-partition parallelism 179
- ISOLATE_PART_ERRORS 123
- isolation 173

J

- join 174, 177
 - planning 183

K

KornShell 130

L

L2 cache 168

large table space 269

latency 131, 171

lineitem 151

table 113

Linux 21

LIST TABLESPACE SHOW DETAIL 65

LOAD 7, 112, 193, 299

DELETE phase 117

multipartition 113, 119, 124, 300

partitioned 126

subagents 126–127

wizard 113

load

wizard 195

load append 16

LOAD INSERT 129

LOAD QUERY 117, 129

LOAD RESTART 165

LOAD_ERRS_ONLY 123–124

LOAD_ONLY 121–122

LOAD_ONLY_VERIFY_PART 119, 121–122

LOB 65, 71, 121, 215

local bypass 180

lock 218

monitor 30

table space 112, 125

LOCK WITH FORCE 113

log retain 236, 291

logging 19, 42, 130

logical database partition 45–46

logical partition 22, 28

logical partitioning 132

LOGPRIMARY 79

logs 76, 235, 240

extents 79

mirroring 78

performance 41

size 79

long table space 269

long-running query 30

LPAR 22, 45, 132

M

manageability xix, 5, 18

MAP_FILE_INPUT 125

MAP_FILE_OUTPUT 123

mass inserts 130

mass load 112

Materialized Query Table 7, 165, 167

materialized view 168

MAX_CONNECTIONS 13

MAX_COORDAGENTS 13

MAX_NUM_PART_AGENTS 123

MAX_QUERYDEGREE 86

maximum logical nodes 106

maxperm 92

maxservers 91, 93

maxuproc 92, 95

MDC 6, 9, 130, 135, 167

create a MDC table 147

delete processing 150

design 142

prototype 165

dimension analyzer 315

extent size 146

full table scan 159

index ANDing 161

index ORing 162

insert processing 149

join 164

MQT 176

nested 163

page size 146

performance 155

point query 157

query on a cell 159

range query 157, 165

two dimensions 158

referential integrity 165

scalability 155

size 75

space calculation 155

storage requirements 143

table scan 160

tuning spac 146

views 165

Memory Visualizer 18, 247, 252

message queueing 131

minperm 92

minservers 91, 93

mirrored copy 40

- mirroring 40, 230
- MIRRORLOGPATH 19
- MODE 122
- monitor switches 30
- monitoring tool 252
- monotonic 148
- MPP 125
- MQT 7, 10, 132, 167
 - MAINTAINED BY SYSTEM 170
 - MAINTAINED BY USER 170
 - MDC 176
 - refresh 172
 - REFRESH DEFERRED 173
 - REFRESH IMMEDIATE 173
 - replication 175
 - size 75
 - system 172
 - tuning 170
 - USER MAINTAINED 172
- multicolumn dimension indexes 165
- MultiDimensional Clustering 6, 135–136
- multidimensional keys 9
- multi-page file allocation 91
- multipage_alloc 92
- multipartition 150
 - tables 113
- multiple dimensions 3

N

- nation 114
- near real time 2
- network parameters 93, 95
- NEWLINE 125
- NFS 96
- NO_ISOLATION 123–124
- node 23, 269
- nodegroup 269
- NODENUMBER() 179
- NON INCREMENTAL 170
- NONRECOVERABLE 118
- NUM_FREQVALUES 14
- NUM_IOCLEANERS 91
- NUM_IOSERVERS 62, 90
- NUM_QUANTILES 14
- number of disks 42

O

- OLAP 9

- OLTP 132
- OMIT_HEADER 126
- online 194
 - buffer pool 223
 - configuration 220
 - buffer pools 220
 - database checking 18
 - index reorganization 17
 - inspect 244
 - INSPECT command 241
 - LOAD 112
 - reorganization 17, 208
 - table load 16
 - utilities 8
- operational systems 131
- optimizer 7, 141, 168, 173, 178, 181, 183, 186
- ORDER BY 142
- orders
 - table 113
- ORGANIZE BY DIMENSIONS 141, 147, 165
- outage 191, 245
 - window 208
- outer join 169
- OUTPUT_DBPARTNUMS 121–122, 126
- overhead 63

P

- p690 20, 45
- page cleaner IO 91
- page size 57–58, 152
- parallel bulk loader 3
- Parallel System Support Program 23
- parallelism 3
- part
 - table 113
- PART_FILE_LOCATION 121
- partition
 - compatibility 72
 - coordinator 126
 - distribution file 125
 - loading 126
 - number 51, 121
 - partitioning
 - 126
- partition group 47
- PARTITION_AND_LOAD 122, 127
- partition_list 121
- PARTITION_ONLY 121–122

- partitioned database 45
- PARTITIONED DB CONFIG 114, 119–120, 126
- partitioning 178
- partitioning key 68, 70, 150, 155, 180
- partitioning map 67, 72, 125, 181
 - header 126
- PARTITIONING_DBPARTNUMS 121
- partitions
 - communication 178
- partsupp
 - table 113
- PC/IXF 119
- Peer-to-Peer Remote Copy 230
- performance xix, 4, 8, 10, 132, 172
- Perl 130
- physical disk layout 41
- physical partition 23
- placing tables 65
- pmap 67, 72
- point-in-time 231, 237, 240
- pool_async_data_writes 91
- pool_async_index_writes 91
- pool_data_writes 91
- PORT_RANGE 124, 126
- ports 105, 125, 128
- PPAR 23
- PPRC 230
- prefetch 8, 83
- prefetch size 57, 61
- prefetch_wait_time 63, 90
- PREFETCHSIZE 61
- primary key 71
- profile entries 105
- pSeries 21
- PSSP 23

Q

- query
 - optimization 173
 - performance 177
 - rewrite 7, 168, 178

R

- RAID 39, 53, 56, 60–61, 83
- RAID 0 39
- RAID 1 40, 43
- RAID 10 41
- RAID 5 41, 43, 235

- range
 - scan 148
- RAS 21, 25
- raw data 42
- raw logical volume 78, 87
- read/write queries 131
- real time 2, 131
- rebind packages 283
- RecLen 125
- recovery 229
- Redbooks Web site 368
 - Contact us xxiii
- redistribution 176
- REDUCE 243
- refresh 3
- REFRESH DEFERRED 169–170
- REFRESH IMMEDIATE 169
- region 114
- reliability xix, 21
- REORG 67, 150
 - availability 215
 - control 212
 - DBPARTITIONNUM 208, 210
 - history file 213
 - INDEXES ALL FOR TABLE 208, 211
 - INPLACE 165, 208
 - monitor 211
 - NOTRUCATE TABLE 209
 - PAUSE 212
- repartitioned join 183
- replica 11
- replicate 130
- REPLICATED 175
- replicated
 - tables 184
- replicated summary tables 167
- replicated tables 10
- RESIZE 58, 242–243
- restore 229, 236
- retention logging 118
- reverse scans 141
- rewrite 178
- rfc1323 94
- RID 150
- RID index 141, 157
- rollforward 118, 235, 239
- round robin 127
- row ID 137
- row-id 150

rpoolsz 94
RUN_STAT_DBPARTNUM 126
RUNSTATS 8, 13, 145, 168, 170, 216–217, 283

S

SAVECOUNT 165
scalability xix, 4, 6, 132
scan
 range 148
scheduling 77
schema 178
scoring 132
self tuning 273
Self-Managing and Resource Tuning 248
sequential detection 83
serviceability xix, 21
services 105
SET INTEGRITY 113–114, 300
set write resume 234
set write suspend 234
SETUP_AND_LOAD_ERRS 124
SETUP_ERRS_ONLY 123
shared nothing 44
SHEAPTHRES 80, 84, 88, 148
simplification 4
single database partition table 114
single partition group 51
single-partition 51, 71
size 57
 memory 223
slice 139
SMART 230, 248
smart xix
smit installp 97
SMP 6, 86, 111, 179
SMS 54–56, 65–66, 91
snapshot xix, 87, 90, 142, 211
sockets 124, 128
sort
 monitor 30
 non-overflowed 88
 non-piped 89
 overflowed 88
 parallelization 185
 performance 89
 piped 89
SORTHEAP 80, 84, 87–88, 148, 165, 186
SORTHEAPTHRESH 165

SP switch 23, 94
space calculation 150
sparse blocks 143, 165
sparse extents 143, 165
split mirror 230, 234, 238
SPLIT_NODES 118
splitters 121
spoolsz 94
SSA disks 40
staging tables 112, 169
standby
 database 235
star schema 3, 142, 164, 175, 181
statement
 monitor 30
statistics 13, 126, 145, 178, 283
STATISTICS YES 126
STATUS_INTERVAL 124
Storage Management 19, 247, 253
stored procedures 130
stripe size 60
striping 43, 236
subagent 178
 catalog 127
 initialize 127
 load 127
 partitioning 127
 prepartitioning 127
 userexit 127
subclasses 31
summaries 10, 168, 172
summary data 10
summary tables
 replicated 51
superclasses 31
supplier 113
switch 23
system catalog 231, 237
system health 18
System Managed Storage 54

T

table
 monitor 30
table load 8
table queue 178–179
 asynchronous 179
 broadcast 179

- directed 179
- listener 180
- local 180
- merging 179
- synchronous 179
- table spaces 52
 - capacity 65
- TCP ports
 - range 124
- tcp_recvspace 94
- tcp_sendspace 94
- temporary table spaces 66
- terabytes 3
- terminology 269
- thresholds 253
- time 148
- timestamp
 - monitor 30
- TPC-H 31, 113
- TRACE 125
- transactions
 - monitor 30
- transfer rates 63
- TRANSFERRATE 64
- transparent 20
- TSM 77
- two-phase commit 180
- type-2 index 17, 192, 210, 216

U

- uneven data distribution 69
- UNION ALL view 19
- unit of work
 - monitor 30
- upgrade 3–4, 263
 - AIX 264
 - backup 267
 - DB2 Administration Server 284
 - DB2 UDB 264
 - diagnostic error level 269
 - logs 268
 - procedure 275
 - space 267
 - strategy 294, 296
 - system catalog 267
 - to DB2 UDB ESE 278
- user exit 236
- users

- create 97
- UTILHEAPSZ 150

V

- vmstat 30
- vmtune 93

W

- wealth 6
- Web clients 12
- Web personalization 132
- Web portal xix
- wizards 18, 252
 - Add Partitions 18
 - Alter Database Partition Group 18
 - Backup 18
 - Configuration Advisor 19
 - Db2 setup 97
 - Design Advisor 19
 - Redistribute Data 18
 - Restore 18
- WLM 26
 - wlm_assign 31
- workload 3, 111, 142
 - query 172
- WorkLoad Manager 26
- write resume 234
- write suspend 233
- write to the table 112

X

- XBSA 192



Up and Running with DB2 UDB ESE: Partitioning for Performance in an e-Business Intelligence World

(0.5" spine)
0.475" <-> 0.875"
250 <-> 459 pages



Redbooks

Up and Running with DB2 UDB ESE

Partitioning for Performance in an e-Business Intelligence World

Simplify parallel data loading with multipartition load

Enhance performances using MultiDimensional Clustering

Improve 24x7 operations with new online utilities

Data warehouses in the 1990s were for the privileged few business analysts. Business Intelligence is now being democratized by being shared with the rank and file employee demanding higher levels of RDBMS scalability and ease of use, being delivered through Web portals.

To support this emerging e-Business Intelligence world, the challenges that face the enterprises for their centralized data warehouse RDBMS technology are scalability, performance, availability, and smart manageability.

This IBM Redbook focuses on the innovative technical functionalities of DB2 UDB ESE V8.1 and discusses:

- Guidelines on building the large database and determining the number of partitions
- Bulk load using the new multipartition load
- Performance enhancements using MultiDimensional Clustering and Materialized Query Tables
- Availability through the new online utilities
- Self Managing and Resource Tuning features
- Migration scenarios

This redbook positions the new functionalities, so you can understand and evaluate their applicability in your own enterprise data warehouse environment, and get started prioritizing and implementing them.

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