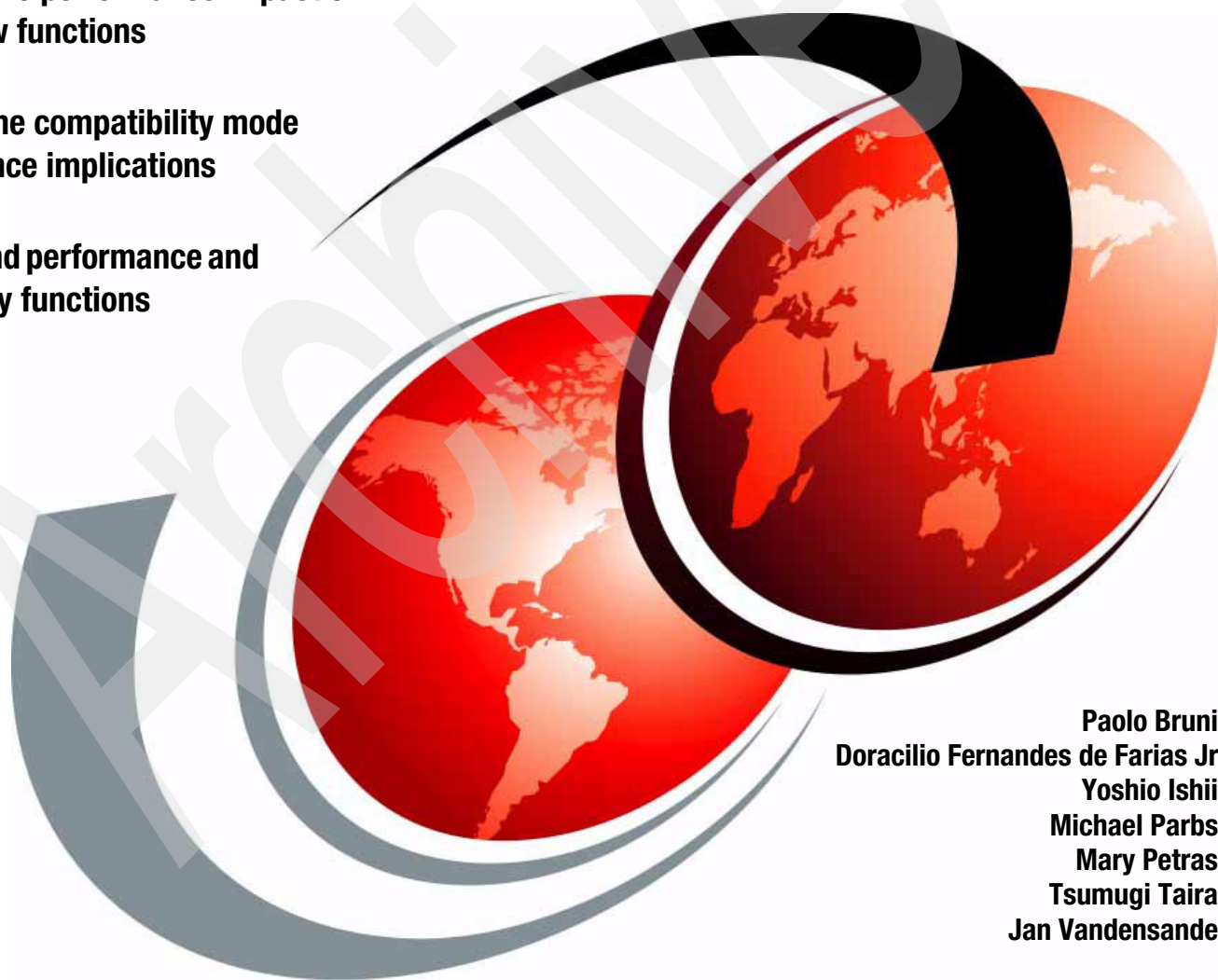


DB2 UDB for z/OS Version 8 Performance Topics

**Evaluate the performance impact of
major new functions**

**Find out the compatibility mode
performance implications**

**Understand performance and
availability functions**



**Paolo Bruni
Doracilio Fernandes de Farias Jr
Yoshio Ishii
Michael Parbs
Mary Petras
Tsumugi Taira
Jan Vandensande**

Redbooks



International Technical Support Organization

DB2 UDB for z/OS Version 8 Performance Topics

April 2005

Archived

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

First Edition (April 2005)

This edition applies to Version 8 of IBM Database 2 Universal Database for z/OS (program number 5625-DB2).

© Copyright International Business Machines Corporation 2005. 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	xi
Tables	xv
Examples	xvii
Notices	xix
Trademarks	xx
Summary of changes	xxi
April 2005, First Edition	xxi
April 2005, First Update	xxi
May 2005, Second Update	xxi
November 2005, Third Update	xxii
January 2006, Fourth Update	xxiii
June 2006, Fifth Update	xxiii
October 2006, Sixth Update	xxiii
August 2007, Seventh Update	xxiii
September 2007, Eighth Update	xxiv
Preface	xxv
The team that wrote this book	xxv
Become a published author	xxviii
Comments welcome	xxviii
Chapter 1. DB2 UDB for z/OS Version 8	1
1.1 Overview	2
1.1.1 Architecture	2
1.1.2 Usability, availability, and scalability	3
1.1.3 Data warehouse	6
1.1.4 Performance	6
1.1.5 Migration changes	8
Chapter 2. Performance overview	9
2.1 Overview of major impact items	11
2.1.1 DB2 catalog	11
2.1.2 CPU	11
2.1.3 I/O time	12
2.1.4 Virtual storage	12
2.1.5 Real storage	13
2.1.6 Compatibility mode	13
2.1.7 New-function mode	14
2.2 CPU usage	15
2.3 DBM1 virtual storage mapping	16
2.3.1 DB2 structures above the bar	17
2.3.2 What are the implications of more storage?	17
2.4 Real storage	17
2.5 Compatibility mode	18
2.5.1 64-bit addressing	18
2.5.2 No more hiperpools and data spaces	19

2.5.3	Increased number of deferred write, castout and GBP write engines.	20
2.5.4	EDM pool changes	20
2.5.5	Fast log apply	21
2.5.6	Multi-row operations with DRDA	21
2.5.7	IRLM V2.2	21
2.5.8	Page-fixed buffer pools	21
2.5.9	Unicode parser	22
2.5.10	Optimizer enhancements	22
2.5.11	CF request batching	22
2.6	New-function mode	22
2.6.1	Plans and packages	23
2.6.2	DB2 catalog	23
2.6.3	Precompiler NEWFUN option	23
2.6.4	SUBSTRING function	24
2.7	New installation default values	24
2.8	Recent z/OS releases	26
2.8.1	Multilevel security	27
2.8.2	DFSMSshm Fast Replication	27
Chapter 3.	SQL performance	29
3.1	Multi-row FETCH, INSERT, cursor UPDATE and DELETE	31
3.1.1	Example of usage of multi-row fetch in PL/I	32
3.1.2	Example of usage of multi-row insert in PL/I	32
3.1.3	Multi-row fetch/update and multi-row fetch/delete in local applications	33
3.1.4	Performance	34
3.1.5	Conclusion	37
3.1.6	Recommendation	38
3.2	Materialized query table	39
3.2.1	Creating MQTs	40
3.2.2	Populating MQTs	41
3.2.3	Controlling query rewrite	41
3.2.4	Performance	45
3.2.5	Conclusions	52
3.2.6	Recommendations	53
3.3	Star join processing enhancements	53
3.3.1	Star schema	53
3.3.2	In memory work file	56
3.3.3	Sparse index	57
3.3.4	Performance	60
3.3.5	Conclusion	64
3.3.6	Recommendations	65
3.4	Stage 1 and indexable predicates	65
3.4.1	New indexable predicates in V7	67
3.4.2	More stage 1 and indexable predicates in V8	67
3.4.3	Performance	69
3.4.4	Conclusions	71
3.4.5	Recommendations	71
3.4.6	Considerations on CASTing	71
3.5	Multi-predicate access path enhancement	71
3.5.1	Performance	74
3.5.2	Conclusions	76
3.5.3	Recommendations	76
3.6	Multi-column sort merge join	77

3.6.1 Performance	77
3.6.2 Conclusion	79
3.6.3 Recommendations	79
3.7 Parallel sort enhancement	79
3.7.1 Performance	80
3.7.2 Conclusions	82
3.7.3 Recommendations	83
3.8 Table UDF improvement	83
3.8.1 Description	83
3.8.2 Performance	84
3.8.3 Conclusion	86
3.8.4 Recommendations	86
3.9 ORDER BY in SELECT INTO with FETCH FIRST N ROWS	86
3.10 Trigger enhancements	86
3.10.1 Performance	87
3.10.2 Conclusion	90
3.10.3 Recommendation	90
3.11 REXX support improvements	90
3.11.1 Performance	91
3.11.2 Conclusion	91
3.11.3 Recommendations	91
3.12 Dynamic scrollable cursors	91
3.12.1 Static scrollable cursor description	92
3.12.2 Dynamic scrollable cursor description	94
3.12.3 Performance	97
3.12.4 Test description and study	98
3.12.5 Conclusion	106
3.12.6 Recommendations	106
3.13 Backward index scan	107
3.14 Multiple distinct	108
3.14.1 Performance	109
3.14.2 Conclusion	110
3.14.3 Recommendations	110
3.15 Visual Explain	110
3.15.1 Installation	112
3.15.2 Visual Explain consistency query analysis	112
3.15.3 Statistics Advisor	116
3.15.4 Visual Explain and Statistics Advisor on MQTs	123
Chapter 4. DB2 subsystem performance	127
4.1 DB2 CPU considerations	129
4.1.1 Performance	130
4.1.2 Conclusion	136
4.1.3 Recommendations	138
4.2 Virtual storage constraint relief	139
4.2.1 The DB2 pools	141
4.2.2 Performance	147
4.2.3 Conclusion	156
4.2.4 Recommendations	156
4.3 Monitoring DBM1 storage	157
4.4 Profiles do vary	166
4.5 Buffer pool long term page fixing	169
4.5.1 Performance	170

4.5.2 Conclusion	171
4.5.3 Recommendations	171
4.6 IRLM V2.2	171
4.6.1 Performance	172
4.6.2 Conclusion	174
4.6.3 Recommendations	174
4.7 Unicode	174
4.7.1 Performance	184
4.7.2 Conclusion	187
4.7.3 Recommendations	187
4.8 Data encryption	188
4.8.1 Performance	190
4.8.2 Conclusion	190
4.8.3 Recommendations	191
4.9 Row level security	191
4.9.1 Performance	195
4.9.2 Conclusion	198
4.9.3 Recommendations	198
4.10 New CI size	199
4.10.1 Performance	200
4.10.2 Conclusion	202
4.10.3 Striping	202
4.10.4 Recommendations	202
4.11 IBM System z9 and MIDAWs	202
4.12 Instrumentation enhancements	204
4.12.1 Unicode IFCIDs	205
4.12.2 Statistics enhancements	205
4.12.3 Lock escalation IFCID 337	207
4.12.4 Accounting enhancements	208
4.12.5 SQLCODE -905 new IFCID 173	209
4.12.6 Performance trace enhancements	210
4.12.7 Miscellaneous trace enhancements	212
4.13 Miscellaneous items	212
4.13.1 DB2 logging enhancements	212
4.13.2 Up to 65,041 open data sets	213
4.13.3 SMF type 89 record collection enhancements	213
4.13.4 Lock avoidance enhancements	214
Chapter 5. Availability and capacity enhancements	217
5.1 System level point-in-time recovery	218
5.1.1 Performance	221
5.1.2 Conclusion	226
5.1.3 Recommendations	226
5.2 Automated space management	226
5.2.1 Conclusion	229
5.2.2 Recommendations	229
5.3 Online schema	230
5.3.1 Changing attributes	230
5.3.2 Performance	233
5.4 Volatile table	246
5.4.1 Conclusion	246
5.4.2 Recommendations	246
5.5 Index enhancements	246

5.5.1	Partitioned table space without index	247
5.5.2	Clustering index	247
5.5.3	Clustering index always	247
5.5.4	DPSI and impact on utilities and queries	248
5.6	Other availability enhancements	256
5.6.1	More online DSNZPARMs	256
5.6.2	Monitor long running UR back out.	256
5.6.3	Monitor system checkpoints and log offload activity	256
5.6.4	Improved LPL recovery	257
5.6.5	Partitioning key update enhancement.	257
5.6.6	Lock holder can inherit WLM priority from lock waiter.	258
5.6.7	Detecting long readers	258
5.6.8	Explain a stored procedure	258
5.6.9	PLAN_TABLE access via a DB2 alias	259
5.6.10	Support for more than 245 secondary AUTHIDs	260
Chapter 6.	Utilities.	261
6.1	Online CHECK INDEX	262
6.1.1	Performance measurement.	264
6.1.2	Conclusion	267
6.1.3	Recommendations	267
6.2	LOAD and REORG	268
6.2.1	LOAD with default performance measurement.	268
6.2.2	Conclusion	269
6.2.3	Recommendations	269
6.3	LOAD and UNLOAD delimited input and output	270
6.3.1	Performance	271
6.3.2	Conclusion	272
6.4	RUNSTATS enhancements	272
6.4.1	General performance improvements.	272
6.4.2	DSTATS	273
6.4.3	Description of distribution statistics in V8	273
6.4.4	Performance	275
6.4.5	Conclusion	280
6.4.6	Recommendations	280
6.5	Cross Loader.	280
6.5.1	Performance APAR.	280
Chapter 7.	Networking and e-business.	281
7.1	DDF enhancements	283
7.1.1	Requester database alias	283
7.1.2	Server location alias	283
7.1.3	Member routing in a TCP/IP Network	283
7.1.4	DRDA allows larger query blocks	284
7.1.5	DB2 Universal Driver for SQLJ and JDBC	285
7.2	Multi-row FETCH and INSERT in DRDA.	285
7.2.1	Performance - Distributed host to host	286
7.2.2	Performance - Distributed client to Host	290
7.3	WebSphere and the DB2 Universal Driver	292
7.3.1	Performance measurement.	293
7.3.2	DB2 Universal Driver X/Open XA support.	303
7.3.3	Miscellaneous items	304
7.4	ODBC enhancements	306

7.4.1 ODBC Unicode support	306
7.4.2 ODBC long name support	306
7.4.3 ODBC connection authentication	306
7.4.4 ODBC support for 2 MB SQL	307
7.4.5 SQLcancel	307
7.5 XML support in DB2 for z/OS V8	307
7.5.1 XML extender	307
7.5.2 XML publishing functions	308
7.5.3 Conclusion	312
7.5.4 Recommendation	312
7.6 Enterprise Workload Manager	312
7.6.1 Introduction	312
7.6.2 Description	313
7.6.3 Performance	316
Chapter 8. Data sharing enhancements	319
8.1 CF request batching	321
8.1.1 Performance	321
8.1.2 Conclusion	325
8.1.3 Recommendations	325
8.2 Locking protocol level 2	325
8.2.1 Performance	328
8.2.2 Conclusion	331
8.2.3 Recommendations	331
8.3 Change to IMMEDIATE default BIND option	332
8.3.1 Performance	332
8.3.2 Conclusion	333
8.3.3 Recommendations	333
8.4 DB2 I/O CI limit removed	333
8.4.1 Performance	333
8.4.2 Conclusion	335
8.4.3 Recommendations	335
8.5 Miscellaneous items	336
8.5.1 Impact on coupling facility	336
8.5.2 Improved LPL recovery	336
8.5.3 Restart Light enhancements	337
8.5.4 Change to close processing for pseudo close	338
8.5.5 Buffer pool long term page fixing	339
8.5.6 Batched index page splits	339
Chapter 9. Installation and migration	341
9.1 Unicode catalog	342
9.1.1 Character conversion	342
9.1.2 Unicode parsing	342
9.1.3 DB2 catalog changes	344
9.2 IVP sample programs	346
9.3 Installation	350
9.4 Migration	353
9.4.1 Migration plan recommendations	354
9.4.2 Performance measurements	355
9.5 Catalog consistency query DSNTESQ	359
Chapter 10. Performance tools	363
10.1 DB2 Estimator	364

10.2 DB2 Performance Expert	364
10.3 Tivoli OMEGAMON XE for DB2 on z/OS	366
10.4 Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS	367
10.5 DB2 Archive Log Accelerator	367
10.6 DB2 Query Monitor	368
10.7 DB2 SQL Performance Analyzer for z/OS	369
10.8 Data Encryption for IMS and DB2 Databases	369
10.9 DB2 Data Archive Expert	370
10.10 DB2 Bind Manager	370
10.11 DB2 High Performance Unload	370
10.11.1 Performance measurements	371
Appendix A. Summary of performance maintenance	377
A.1 Maintenance for DB2 V8	378
A.2 Maintenance for DB2 V7	386
A.3 Maintenance for z/OS	390
Appendix B. The DBM1 storage map	393
B.1 DB2 PE Storage Statistics trace and report	394
Appendix C. SQL PA sample reports	399
C.1 DB2 SQL PA additional reports	400
Appendix D. EXPLAIN and its tables	407
D.1 DB2 EXPLAIN	408
D.1.1 DSN_PLAN_TABLE	409
D.1.2 DSN_STATEMNT_TABLE	415
D.1.3 DSN_FUNCTION_TABLE	416
D.1.4 DSN_STATEMNT_CACHE_TABLE	417
Abbreviations and acronyms	423
Related publications	425
IBM Redbooks	425
Other publications	426
Online resources	426
How to get IBM Redbooks	427
Help from IBM	427
Index	429

Archived

Figures

2-1	Virtual storage growth	19
3-1	Multi-row fetch reduces the trips across the API	31
3-2	Usage of multi-row fetch	32
3-3	Example of multi-row cursor update and delete.	34
3-4	CPU time (class 1) for single-row and multi-row fetch.	35
3-5	CPU time (class 1) for single-row and multi-row insert	36
3-6	Relationship between two special registers for MQTs.	42
3-7	Rules for query rewrite.	43
3-8	Query rewrite example.	44
3-9	MQTs and short running queries	46
3-10	Increase of BIND cost	47
3-11	Query rewrite for simple query.	48
3-12	Query rewrite for join	49
3-13	Regrouping on MQT result.	50
3-14	Expression® derivation example	51
3-15	Star schema.	54
3-16	In memory work file - Description.	57
3-17	Star join access path (before PQ61458)	58
3-18	Sparse index on work file.	59
3-19	Sparse index for star join	60
3-20	Sparse index performance.	61
3-21	Star join access path	62
3-22	In memory work file - CPU reduction.	63
3-23	Star join workload	64
3-24	Analysis of Stage 1 and Stage 2 predicates	66
3-25	Example of mismatched numeric types.	67
3-26	Example of mismatched string types.	68
3-27	Mismatched numeric data types	69
3-28	Different lengths for string data type	70
3-29	Access path graph in Visual Explain	73
3-30	Better filter factor estimation	75
3-31	Access path improvements - Multi-column statistics	76
3-32	Mismatched data types	78
3-33	Query parallelism in V8	79
3-34	EXPLAIN example - Parallel sort.	80
3-35	Multiple table group by sort	81
3-36	Multiple table order by sort.	82
3-37	Table UDF - Cardinality option.	85
3-38	Table UDF - Block fetch support	85
3-39	SQL and BPOOL activity - Trigger 100 times	88
3-40	CPU time when invoking the trigger 100 times	88
3-41	SQL and BPOOL activity - Trigger with WHEN true only once	89
3-42	CPU time to invoke trigger but WHEN condition true once	90
3-43	REXX support improvement	91
3-44	DECLARE CURSOR syntax	94
3-45	FETCH syntax	96
3-46	Multiple DISTINCT performance	110
3-47	Explain of the old consistency query 31.	114

3-48	Consistency query - 31 New	116
3-49	Starting to analyze a sample query	117
3-50	Suggested RUNSTATS statements.	118
3-51	Reason for RUNSTATS.	118
3-52	Saved queries	119
3-53	New suggested RUNSTATS	119
3-54	Statistical output.	120
3-55	Recollection of inconsistent statistics.	121
3-56	Conflicting statistics	121
3-57	Statistics Advisor Plug-integrated with Visual Explain	122
3-58	Predefined default values	122
3-59	Conflict statistics threshold setup.	123
3-60	Explain without using an MQT table by Visual Explain	124
3-61	Explain using an MQT table by Visual Explain	125
4-1	CPU workload summary	131
4-2	V7 vs. V8 Utility CPU measurements	132
4-3	DBM1 virtual storage constraint relief (no data spaces nor hiper pools)	140
4-4	DBM1 below 2 GB virtual storage comparison for different workloads	149
4-5	DBM1 user thread storage comparison DB2 V7 vs. V8.	152
4-6	Real storage comparison DB2 V7 vs. V8.	155
4-7	MVS storage overview.	161
4-8	Sample RMF Monitor III STORF Report	166
4-9	DBM1 storage by time	167
4-10	Thread-related storage by time	168
4-11	Storage per thread by time	169
4-12	CCSID conversion - Local application	176
4-13	CCSID conversion - Remote application - 1	177
4-14	CCSID conversion - Remote application - 2	178
4-15	CCSID conversion - Local z/OS Universal Java Type 2 Driver	180
4-16	CCSID conversion - Remote Prepare using Java Universal Type 4 Driver.	181
4-17	CCSID conversion - Remote Prepare of SQL statement.	182
4-18	CCSID conversion - Local Prepare using Universal Java Type 2 Driver.	182
4-19	Conversion performance overview	184
4-20	Larger CI size impact on DB2	201
4-21	IBM System z9.	203
4-22	Table space scan with MIDAWs	204
5-1	FRBACKUP PREPARE elapsed time vs. #volumes in the copy pool	222
5-2	FRBACKUP PREPARE elapsed time vs.#versions for 67 volume copy pool	222
5-3	BACKUP SYSTEM DATA ONLY measurement	223
5-4	BACKUP SYSTEM FULL measurement	224
5-5	RESTORE SYSTEM measurement.	225
5-6	Sliding scale for 64 GB data sets.	227
5-7	Sliding scale for 16 GB data set.	228
5-8	CPU time before ALTER, after ALTER and after REORG.	234
5-9	INSERT CPU time before ALTER, after ALTER and after REORG	235
5-10	SELECT elapsed and CPU time before and after ALTER INDEX add column	236
5-11	FETCH CPU time - Alter index padded	237
5-12	FETCH CPU time - Alter index not padded and rebind	238
5-13	Query CPU overhead using NOT PADDED vs. PADDED index	239
5-14	Average index leaf pages and index getpages	240
5-15	NOT PADDED index impact on utilities - Index VARCHAR(14)	241
5-16	NOT PADDED index impact on utilities - Index VARCHAR(1)	241
5-17	NOT PADDED index impact on utilities - Index 4 columns VARCHAR(1).	242

5-18	Rotate partition - Elapsed time comparison	243
5-19	Partition distribution before and after REORG REBALANCE	244
5-20	REORG REBALANCE elapsed time	245
5-21	Load with 1 NPI vs. 1 DPSI	249
5-22	Load with 5 NPIs vs. 5 DPSIs	250
5-23	REORG with 1 NPI vs. 1 DPSI	250
5-24	REORG with 5 NPIs vs. 5 DPSIs	251
5-25	Rebuild index partition of PI, NPI and DPSI	252
5-26	Rebuild index PI, NPI, and DPSI	252
5-27	Check index PI, NPI and DPSI	253
5-28	Runstats index PI, NPI and DPSI	253
6-1	CHECK INDEX SHRLEVEL REFERENCE and SHRLEVEL CHANGE	262
6-2	SHRLEVEL CHANGE vs. SHRLEVEL REFERENCE with FlashCopy	266
6-3	SHRLEVEL CHANGE vs. SHRLEVEL REFERENCE w/o FlashCopy	267
6-4	LOAD with default SORTKEYS	269
6-5	Distribution statistics performance comparison	277
7-1	Multi-row FETCH distributed DB2 V7 vs. DB2 V8	286
7-2	Multi-row FETCH - Distributed host to host	287
7-3	Multi-row INSERT - Distributed host to host	289
7-4	Multi-row fetch - DB2 client to host	290
7-5	Multi-row insert - DB2 client to host	291
7-6	DB2 Universal Type 2 Driver vs. legacy JDBC 2.0 Driver	294
7-7	DB2 Universal Type 2 Driver - Gateway DB2 vs. DB2 Universal Type 4 Driver	295
7-8	DB2 Universal Type 2 Driver vs. DB2 Universal Type 4 Driver	296
7-9	JDK improvements	297
7-10	JDBC .app/DB2 Connect vs. JCC Type 4 - CMP workload	297
7-11	DB2 Universal Type 4 Driver - JDBC vs. SQLJ - CMP workload	298
7-12	DB2 Universal Type 4 Driver - JDBC vs. SQLJ - Servlet workload	298
7-13	JCC Type 4 vs. JCC Type 4 with DB2 Connect	299
7-14	JCC T4 configuration for balancing options	300
7-15	JCC T2 and DB2 Connect configuration	301
7-16	Performance of JCC T4 vs. DB2 Connect	302
7-17	Servlet workload - Type 2 non-XA vs. Type 2 XA compliant driver	303
7-18	DB2 Universal Driver - KEEP DYNAMIC YES	305
7-19	XML extender decomposition	308
7-20	XML extender vs. SQL publishing functions	311
7-21	Composition by XML publishing functions	311
7-22	EWLM support with WAS/AIX and DB2 for z/OS V8	313
7-23	Enterprise Workload Manager Control Center - Managed servers	314
7-24	Enterprise Workload Manager Control Center - Server topology	315
7-25	EWLM control center - Service class statistics	315
7-26	EWLM implementation cost	316
9-1	Unicode conversion example	343
9-2	Multi-row fetch support of DSNTEP4	347
9-3	Multi-row fetch support of DSNTEP4	348
9-4	Multi-row fetch support of DSNTIAUL	349
9-5	Multi-row support of DSNTIAUL comparing V7 with V8	350
9-6	Migration times to CM	357
9-7	Migration from CM to NFM	358
9-8	Catalog consistency queries - Measurement results	361
9-9	Catalog consistency queries - Measurement results	362
10-1	DB2 HPU vs. UNLOAD utility Case 1 through Case 5	374
10-2	DB2 HPU vs. UNLOAD utility Case 6 through Parallel	374

Archived

Tables

2-1	Changed DSNZPARM default settings	25
2-2	z/OS release functionality	26
3-1	Class 1 CPU time (sec.) to process 100,000 rows	38
3-2	Measurements for mismatched numeric data types	70
3-3	Static cursor declare and fetch specifications	92
3-4	Read cursor - Static vs. dynamic	99
3-5	Update cursor static vs. dynamic	101
3-6	Multi-row FETCH static	104
3-7	Multi-row FETCH dynamic	104
3-8	Multi-row FETCH and UPDATE or DELETE static	105
3-9	Multi-row FETCH and UPDATE or DELETE dynamic	106
3-10	Access type and sort with backward index scan	108
3-11	Old consistency query 31	113
3-12	Consistency Query 31 New	115
3-13	Query using MQT table	123
4-1	CM vs. NFM - Non-data sharing	133
4-2	CM vs. NFM - Data sharing	134
4-3	Accounting class 2 CPU time increase	137
4-4	System storage configuration DB2 V7 vs. DB2 V8	148
4-5	Non-distributed DBM1 virtual storage comparison	149
4-6	Virtual storage comparison for CLI, JDBC, embedded SQL, and SQLJ workloads	150
4-7	DB2 storage messages	153
4-8	Real storage comparison for CLI, JDBC, embedded SQL, and SQLJ workloads	154
4-9	Long term page fix - non-data sharing	170
4-10	Long term page fix - data sharing	170
4-11	IRLM CPU - V2.1 vs. V2.2	173
4-12	DB2 V7 - Sizes in KB (usable track size 48 KB for 3390)	199
4-13	DB2 V8 - Sizes in KB (usable track size 48 KB for 3390)	200
4-14	I/O service time on ESS 800	201
4-15	Lock avoidance of overflow record	214
5-1	Concurrent execution of BACKUP SYSTEM and IRWW workload	224
5-2	LOGAPPLY phase measurement	225
5-3	NPI vs. DPSI SELECT COUNT query	254
5-4	NPI vs. DPSI SELECT COUNT query parallelism	254
5-5	DPSI overhead	255
5-6	DPSI access path change	255
6-1	Delimited Load and Unload CPU comparisons	272
6-2	Summary of RUNSTATS improvements	272
6-3	RUNSTATS with DSTATS measurement data in V7	278
6-4	RUNSTATS with DSTATS measurement data in V8	279
7-1	DB2 connect evaluation DB2 V7 vs. DB2 V8	285
7-2	DB2 V8 with implicit multi-row FETCH vs. V7	288
7-3	DB2 V8 with explicit multi-row FETCH vs. V7	288
7-4	Comparing the key points of the two solutions	302
8-1	CF Request Batching - DB2 PE extract (OLTP)	322
8-2	CF Request Batching - RMF extract (OLTP)	322
8-3	CF Request Batching - DB2 PE extract (batch)	323
8-4	CF Request Batching - RMF extract (batch)	324

8-5	CF Request Batching - DB2 CPU	324
8-6	Protocol Level 2 - Statistics Report extract	328
8-7	Protocol Level 2 - Accounting Report extract	328
8-8	Protocol Level 2 - RMF XCF Report extract	328
8-9	Protocol Level 2 - RMF CF Activity Report extract	329
8-10	Protocol Level 2 - DB2 PE Statistics Report extract	330
8-11	Protocol Level 2 - RELEASE(DEALLOCATE) vs. RELEASE(COMMIT)	330
8-12	DB2 I/O CI Limit Removal - Non-data sharing	334
8-13	DB2 I/O CI limit removal - Data sharing	334
8-14	DB2 I/O CI limit removal - Non-data sharing CPU	334
8-15	DB2 I/O CI limit removal - Data sharing CPU	335
9-1	DB2 catalog growth	344
9-2	Clist changes in the installation panels	351
9-3	Row counts for catalog tables - 698 MB catalog in V7	355
9-4	CATMAINT statistics in non-data sharing	356
9-5	Queries 5 and 18 rewritten for better performance	359
9-6	Catalog consistency - Query 31	360
9-7	Catalog consistency - Query 33	361
9-8	Catalog consistency - Queries 35 and 36 from SDSNSAMP	361
10-1	RECOVER and Archive Log Accelerator	368
10-2	DB2 HPU vs. UNLOAD utility	371
10-3	HPU output formats	371
10-4	Unload test cases	372
10-5	HPU V2.2 vs. DB2 V8 UNLOAD utility	372
10-6	HPU V2.2 vs. DB2 V8 DSNTIAUL (no multi-row)	375
10-7	HPU V2.2 vs. DB2 V8 DSNTIAUL (multi-row)	376
A-1	DB2 V8 performance-related APARs	378
A-2	DB2 V7 performance-related APARs	386
A-3	z/OS DB2 performance-related APARs	390
B-1	DBM1 AND MVS STORAGE BELOW 2 GB	394
B-2	DBM1 STORAGE ABOVE 2 GB	396
B-3	REAL AND AUXILIARY STORAGE	397
D-1	PLAN_TABLE columns by release	409
D-2	PLAN_TABLE contents and brief history	410
D-3	EXPLAIN enhancements in DSN_STATEMNT_TABLE	415
D-4	DSN_FUNCTION_TABLE extensions	416
D-5	Contents of DSN_STATEMNT_CACHE_TABLE	419

Examples

3-1	Insert 10 rows using host variable arrays for column values	32
3-2	COBOL array declaration	38
3-3	Define a table as MQT	40
3-4	Alter a table to MQT	41
3-5	Explain for Q662	50
3-6	SQL text for Visual Explain	72
3-7	RUNSTATS statement provided by the Statistics Advisor	73
3-8	Multiple table GROUP BY sort query	81
3-9	Multiple table ORDER BY sort query	82
3-10	User-defined table function (1/3)	84
3-11	User-defined table function (2/3)	84
3-12	User-defined table function (3/3)	84
3-13	Sample row trigger creation	86
3-14	Read program of static cursor	98
3-15	Read program of dynamic cursor	99
3-16	Update program of static cursor	100
3-17	Update program of dynamic cursor	101
3-18	Multi-row FETCH program of a static scrollable cursor	103
3-19	Multi-row FETCH program of a dynamic scrollable cursor	103
3-20	Multi-row FETCH program of a static scrollable cursor	104
3-21	Multi-row FETCH program of a dynamic scrollable cursor	105
4-1	DBM1 Storage Statistics for a DB2 V7 subsystem	158
4-2	DBM1 Storage Statistics for a DB2 Version 8 subsystem	163
4-3	DBM1 storage usage summary	165
4-4	DB2 PE enhanced subsystem services reporting	205
4-5	Enhances deadlock trace record	206
4-6	DSNI031I - Lock escalation	208
4-7	IFCID 173 DSECT	210
6-1	Sample JCL of CHECK INDEX SHRLEVEL CHANGE	263
6-2	Sample LOAD statement	270
6-3	LOAD DATA text format	271
6-4	LOAD DATA binary format	271
6-5	UNLOAD DATA in EBCDIC format	271
6-6	UNLOAD DATA in ASCII format	271
6-7	RUNSTATS statement in V7	276
6-8	DSTATS tool in V7	276
6-9	RUNSTATS with distribution statistics in V8	276
7-1	XML publishing example	309
8-1	Sample -DISPLAY GROUP output	327
9-1	DSNTIAUL with multi-row parameter	349
C-1	SQL PA query description	400
C-2	SQL PA index information	400
C-3	SQL PA query analysis	402
C-4	SQL PA predicate analysis	405
D-1	Creating DSN_STATEMENT_CACHE_TABLE	418

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:

Redbooks (logo)  ®	DS8000™	Parallel Sysplex®
Redbooks (logo)™	Enterprise Storage Server®	QMF™
eServer™	Enterprise Workload Manager™	Redbooks®
z/Architecture®	FlashCopy®	RACF®
z/OS®	Footprint®	RAMAC®
zSeries®	FICON®	RETAIN®
z9™	Geographically Dispersed Parallel Sysplex™	REXX™
AIX®	GDPS®	RMF™
CICS®	IBM®	S/360™
DB2 Connect™	IMST™	S/390®
DB2 Universal Database™	MQSeries®	System z™
DB2®	MVS™	System z9™
DFSMS™	MVS/ESA™	System/390®
DFSMSdss™	MVS/XA™	Tivoli®
DFSMSHsm™	OMEGAMON®	TotalStorage®
DFSORT™	OS/390®	WebSphere®
DRDA®		

The following terms are trademarks of other companies:

SAP R/3, SAP, and SAP logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries.

Oracle, JD Edwards, PeopleSoft, Siebel, and TopLink are registered trademarks of Oracle Corporation and/or its affiliates.

Java, Java Naming and Directory Interface, JDBC, JDK, JVM, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

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

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

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

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

Summary of changes

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

Summary of changes
for SG24-6465-00
for DB2 UDB for z/OS Version 8 Performance Topics
as created or updated on September 26, 2007.

April 2005, First Edition

The revisions of this First Edition, first published on April 11, 2005, reflect the changes and additions described below.

April 2005, First Update

Changed information

- ▶ Figure 4-2 on page 132, corrected text from reocover to recover
- ▶ Figure 4-16 on page 181, corrected the direction of arrows
- ▶ Removed the non supported message DSNB360I from “Detecting long-running readers” on page 207.
- ▶ Table 5-3 on page 254, corrected Elapsed % from +73% to +186%.
- ▶ Figure 5-7 on page 228, corrected title and caption from MB to GB.
- ▶ Removed an incorrect chart on Check Index on 10 partitions just before 6.1.2, “Conclusion” on page 267.
- ▶ Figure 7-3 on page 289, corrected text in the title from 10,000 to 100,000
- ▶ Figure 7-9 on page 297, corrected TPCIP to TCP/IP
- ▶ Figure 7-18 on page 305, corrected tops to tps

Added information

- ▶ Added ITR definition under Figure 7-17 on page 303.

May 2005, Second Update

Changed information

- ▶ Clarified the CPU % regression on DRDA workloads in 2.1.6, “Compatibility mode” on page 13.
- ▶ Clarified partition pruning in 5.5.1, “Partitioned table space without index” on page 247.
- ▶ Updated measurements and text for Table 5-3 and Table 5-4 on page 254 on NPI v. DPSI queries.
- ▶ Corrected some text in 5.2, “Automated space management” on page 226 and updated Figure 5-7 on page 228.
- ▶ Updated status of some APARs in Table A-1 on page 378 and Table A-2 on page 386.

Added information

The following APARS have been added to Table A-1 on page 378:

- ▶ PQ99658 (IFCID 225 in Class 1 Statistics)
- ▶ PK04076 (SORTKEYS not activated for LOAD with one index)
- ▶ PK05360 (hybrid join with local multi-row support)
- ▶ PK01510 and PK03469 for RUNSTATS

November 2005, Third Update

Changed information

- ▶ Updated the DB2 Estimator Web site at 10.1, “DB2 Estimator” on page 364.
- ▶ Updated status of several APARs in Appendix A, “Summary of performance maintenance” on page 377.

Added information

- ▶ Added PTF UK06848 for APAR PQ99482 for QMF for TSO/CICS 8.1 support of multi-row fetch and insert at the end of 3.1, “Multi-row FETCH, INSERT, cursor UPDATE and DELETE” on page 31.
- ▶ Added an important note at the end of 4.3, “Monitoring DBM1 storage” on page 157 to mention the changes provided by APAR PQ99658 to the availability of virtual storage diagnostic record IFCID 225.
- ▶ Added section 4.11, “IBM System z9 and MIDAWs” on page 202.
- ▶ Added a note at the end of 5.2, “Automated space management” on page 226 to mention the preformatting enhancement provided by APAR PK05644.
- ▶ Added a note at the end of 6.2, “LOAD and REORG” on page 268 to mention that with APAR PK04076 LOAD sort is avoided for LOAD SHRLEVEL NONE if data is sorted and only one index exists.
- ▶ Added the informational APAR II14047 f on the use of DFSORT by DB2 utilities at 6.2.3, “Recommendations” on page 269.
- ▶ Added JDBC .app/DB2 Connect vs. JCC Type 4 - Servlet workload and the measurements of Figure 7-10 on page 297.
- ▶ Added JCC Type 4 vs. JCC Type 4 with DB2 Connect measurements of Figure 7-13 on page 299.
- ▶ Added information on maintenance in 7.6, “Enterprise Workload Manager” on page 312.
- ▶ Added Figure 7-26 on page 316 with EWLM measurements.
- ▶ The following APARS have been added to Table A-1 on page 378:
 - PQ99707, performance improvement for JCC clients
 - PK05644, preformatting change
 - PK09268, column processing improvement
 - PK12389 SQL enhancements in compatibility mode
- ▶ The following APARS, related to EWLM support in z/OS, have been added to Table A-3 on page 390:
 - OA07196
 - OA07356
 - OA12005

January 2006, Fourth Update

Changed information

- ▶ Updated the APAR information “User thread storage comparison” on page 151.
- ▶ Updated status of several APARs in Appendix A, “Summary of performance maintenance” on page 377.

Added information

- ▶ Added PTF status.

June 2006, Fifth Update

Changed information

- ▶ Updated the APAR information “User thread storage comparison” on page 151.
- ▶ Updated 5.6.6, “Lock holder can inherit WLM priority from lock waiter” on page 258.
- ▶ Updated status of several APARs in Appendix A, “Summary of performance maintenance” on page 377.

Added information

- ▶ Added new APARs in Appendix A, “Summary of performance maintenance” on page 377.

October 2006, Sixth Update

Changed information

- ▶ Clarified the scope of improvements at 3.11, “REXX support improvements” on page 90.
- ▶ Updated the paragraph on FREQVAL on page 274 to show the correct parameter NUMCOLS instead of COLUMN.
- ▶ Updated status of several APARs in Appendix A, “Summary of performance maintenance” on page 377.

Added information

- ▶ Added a reference to data sharing information at Chapter 8, “Data sharing enhancements” on page 319.
- ▶ Added a reference to Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS in new section 10.4, “Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS” on page 367.
- ▶ Added new APARs in Appendix A, “Summary of performance maintenance” on page 377.

August 2007, Seventh Update

Changed information

- ▶ Updated status of several APARs in Appendix A, “Summary of performance maintenance” on page 377.

Added information

- ▶ Added a Note box to the RUNSTATS option on page 274 to show how to avoid collecting FREQVAL values.
- ▶ Added new APARs in Appendix A, “Summary of performance maintenance” on page 377.

- ▶ Added reference to June 15, 2007 DB2 for z/OS Version 8 SUP tape in Appendix A.1, “Maintenance for DB2 V8” on page 378.

September 2007, Eighth Update

This is likely to be the last update. Change bars identify the changed or added areas by this update. Change bars from previous updates have been removed.

Changed information

- ▶ Updated text on JCC T4 concentrator in the section JCC Type 4 vs. JCC Type 4 with DB2 Connect - IRWW workload at page 301.
- ▶ Updated status of several APARs in Appendix A, “Summary of performance maintenance” on page 377.

Added information

- ▶ Added information on new APARs in Appendix A, “Summary of performance maintenance” on page 377.

Preface

IBM® DATABASE 2 Universal Database Server for z/OS Version 8 (DB2 V8 or V8 throughout this IBM Redbooks publication) is the twelfth and largest release of DB2 for MVS™. It brings synergy with the zSeries® hardware and exploits the z/OS 64-bit virtual addressing capabilities. DB2 V8 offers data support, application development, and query functionality enhancements for e-business, while building upon the traditional characteristics of availability, exceptional scalability, and performance for the enterprise of choice.

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

This book introduces the major performance and availability changes as well as the performance characteristics of many new functions. It helps you understand the performance implications of migrating to DB2 V8 with considerations based on laboratory measurements. It provides the type of information needed to start evaluating the performance impact of DB2 V8 and its capacity planning needs.

Notice that in this document DB2 refers to the z/OS member of the DB2 family of products. We have specified the full name of the products whenever the reference was ambiguous.

This book concentrates on the performance aspects of DB2 V8, for a complete picture of V8 functions and more details on them, refer to the documentation listed at the Web site:

<http://www.ibm.com/software/data/db2/zos/index.html>

The team that wrote this book

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

Paolo Bruni is a DB2 Information Management Project Leader at the International Technical Support Organization, San Jose Center. He has authored several Redbooks® about DB2 for z/OS and related tools, and has conducted workshops and seminars worldwide. During Paolo's many years with IBM, in development, and in the field, his work has been mostly related to database systems.

Doracilio Fernandes de Farias Jr is a Systems Analyst with Bank of Brazil since 1999. He has provided DB2 for z/OS support on installation, customization, migration, and performance in a very large data sharing complex. Doracilio has extensive experience with performance analysis, both at system and subsystem level, and logical and physical database design.

Yoshio Ishii is a DB2 for z/OS Consultant at Lawrence Consulting, an IBM business partner company providing strategic IT consulting services, education and mentoring for IBM DB2 and WebSphere® software. Yoshio has been working as DBA, performance analyst for data warehouse, and instructor for "DB2 Performance for Applications" since 2003. Before becoming a Consultant in 1999, he had been working for IBM for 30 years. He has previously co-authored the book *DB2 Version 4 Data Sharing Performance Topics*, SG24-4611.

Michael Parbs is a DB2 Specialist with IBM Global Services A/NZ, from Canberra, Australia. He has over 15 years experience with DB2, primarily on the OS/390® platform. Before joining IBM Global Services A/NZ he worked in the public sector in Australia, as a DBA and a DB2 Systems Programmer. Michael's main areas of expertise are data sharing and distributed processing, but his skills include database administration and DB2 performance and tuning. Michael has co-authored the redbooks: *DB2 for MVS/ESA Version 4 Data Sharing Implementation*, SG24-4791, *DB2 UDB Server for OS/390 and z/OS Version 7 Presentation Guide*, SG24-6121, and *DB2 UDB for z/OS Version 8: Everything You Ever Wanted to Know, ... and More*, SG24-6079.

Mary Petras is a Senior Software Engineer at the Silicon Valley Lab, where she is currently a DB2 Tools Technical Support Specialist working with DB2 Tool customers. She studied Mathematics at Pratt Institute, School of Engineering and Science, in Brooklyn, New York, and the University of Connecticut in Storrs. Her areas of expertise include DB2 data sharing and performance and tuning. Mary has previously co-authored the redbooks: *DB2 UDB for OS/390 Version 6 Performance Topics*, SG24-5351, and *DB2 Performance Expert for z/OS Version 2*, SG24-6867-01.

Tsumugi Taira is a Project Leader with NIWS Co, Ltd. in Japan, a company specialized in advanced services in the Web/Server Business System Field, joint venture between IBM Japan and Nomura Research Institute. Tsumugi has six years of experience in DB2 for AIX® administration, and is currently on a one year assignment to the IBM Silicon Valley Lab, DB2 for z/OS Performance Department, where she has worked on system set up, tuning and recovery utilities. Her areas of expertise include ESS operations, Tivoli Storage Manager, fault tolerant systems, and backup and recovery solutions. Tsumugi has co-authored the recent book *Disaster Recovery with DB2 UDB for z/OS*, SG24-6370.

Jan Vandensande is an IT consultant and teacher at Jeeves Consulting Belgium, an IBM business partner. He has over 20 years of experience in the database management field. Before joining Jeeves Consulting in 1995, Jan worked as an IMS™ and DB2 system administrator in the financial sector. His areas of expertise include backup and recovery, data sharing, and performance. Jan has previously co-authored the book *DB2 for z/OS and OS/390 Version 7 Performance Topics*, SG24-6129.

A photo of the team is in Figure 1.

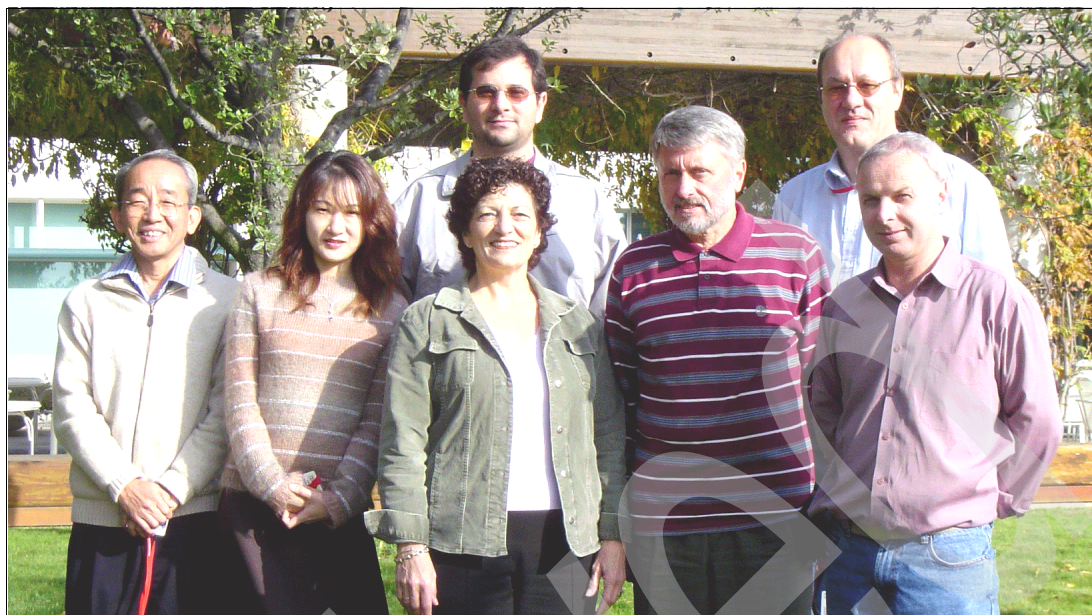


Figure 1 Left to right: Yoshio, Tsumugi, Mary, Doracilio, Paolo, Jan and Michael

Special thanks to the members of the DB2 Performance Department, Silicon Valley Lab, and their manager Catherine Cox, for providing measurements data and continued support during this project.

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

Rich Conway
 Emma Jacobs
 Bob Haimowitz
 Sangam Racherla
 International Technical Support Organization

Jeff Berger
 Patrick Bossman
 John Campbell
 Gene Fuh
 James Guo
 Akiko Hoshikawa
 Koshy John
 Gopal Krishnam
 Ching Lee
 Dave Levish
 Chan-hua Liu
 Roger Miller
 Manvendra Mishra
 Todd Munk
 Ka C Ng
 Mai Nguyen
 Jim Pickel
 Terry Purcell
 Jim Ruddy
 Akira Shibamiya
 Don Smith

Hugh Smith
Jim Teng
John Tobler
Yoichi Tsuji
Steve Turnbaugh
Frank Vitro
Jay Yothers
Maryela Weihrauch
Dan Weis
Chung Wu
David Zhang
IBM Silicon Valley Laboratory

Bart Steegmans
IBM Belgium

Norbert Heck
IBM Germany

Martin Packer
IBM UK

Richard Corrihons
IBM France

Become a published author

Join us for a two- to seven-week residency program! Help write an IBM Redbooks publication 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 will 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 book form found at:

ibm.com/redbooks

- Send your comments in an e-mail to:

redbook@us.ibm.com

- Mail your comments to:

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

Archived

Archived

DB2 UDB for z/OS Version 8

Version 8 of DB2 UDB for z/OS is the largest release ever shipped, including Version 1, which became generally available in 1985. In this chapter we briefly describe the major enhancements available with DB2 UDB for z/OS Version 8. This represents an introduction to all functions, not just those related to performance.

For a more inclusive list, more details, and current information always refer to the Web site:

<http://www.ibm.com/software/data/db2/zos/index.html>

1.1 Overview

IBM DB2 UDB for z/OS Version 8 (DB2 V8 or V8 throughout this publication) includes dozens of changes in SQL, improving DB2 family consistency in many cases, and leading the way in others. Many limits our customers faced are no longer a restriction at all: Using 64-bit virtual memory addressing, providing longer table and column names, allowing 2 MB SQL statements, 4,096 partitions, and three times the log space.

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

Major improvements include:

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

In this section we group the enhancements into categories based on the area of impact on your applications. These categories are:

- ▶ Architecture
- ▶ Usability, availability, and scalability
- ▶ Data warehouse
- ▶ Performance
- ▶ System level point-in-time backup and recovery

1.1.1 Architecture

The most important enhancements to the DB2 architecture are:

- ▶ 64-bit virtual storage

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

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

- Buffer pools and buffer pool control blocks
- EDM pool (DBDs and dynamic statement cache)
- Sort pool
- RID pool
- Castout buffers
- Compression dictionaries

A large address space of up to 2^{64} bytes (16 exabytes) can now be defined for DBM1 and IRLM address spaces. DBM1 replaces hiperspaces and data spaces with buffer pools. As a result, managing virtual storage becomes simpler, and the scalability, availability, and performance improves as your real storage requirements and number of concurrent users increase. IRLM can support large numbers of locks.

► **Schema evolution**

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

► **DB2 Universal Driver for SQLJ and JDBC™**

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

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

► **Unicode support**

Architectural changes to DB2 V8 enhance the DB2 support for Unicode with SQL in Unicode, Unicode in the DB2 catalog and the ability to join Unicode, ASCII and EBCDIC tables. This means that you can manage data from around the world in the same SQL statement. DB2 now converts any SQL statement to Unicode before parsing, and as a result, all characters parse correctly.

► **DB2 Connect and DRDA**

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

1.1.2 Usability, availability, and scalability

The main enhancements related to usability, availability, and scalability are:

► **Online schema evolution**

You can change data and indexes with minimal disruption.

- Column data type changes can be done without losing data availability to data and indexes. In V5 you could increase the size of VARCHAR columns, the changes in V8 allow you to extend numeric and character columns and to change between CHAR and VARCHAR.
- Indexes can have columns added, and VARCHAR columns changed from padded to non padded.
- Partitioning

Several partitioning enhancements are included in DB2 V8 that are useful in almost all real life environments.

 - Online partitioning changes

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

This enhancement increases the maximum number of partitions in a partitioned table space and index space past the current maximum of 254. The new maximum number of partitions is 4,096. The DSSIZE and page size determine the maximum number of possible partitions.
 - Data-partitioned secondary indexes

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

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

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

The system level point-in-time recovery enhancement provides the capability to recover the DB2 system to any point-in-time, irrespective of the presence of uncommitted units of work, in the shortest amount of time. This is accomplished by identifying the minimum number of objects that should be involved in the recovery process, which in turn reduces the time needed to restore the data and minimizes the amount of log data that needs to be applied.

For larger DB2 systems with more than 30,000 tables, this enhancement significantly improves data recovery time, which in turn results in considerably shorter system downtime.
- BACKUP and RESTORE SYSTEM

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

- REORG utility enhancements

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

- Online Check Index utility

The new SHRLEVEL CHANGE option specifies that applications can read from and write to the index, table space, or partition that is to be checked. This option uses techniques which allow access to the index during almost all of the CHECK process. The processing is performed on a copy of the data and can use FlashCopy® to obtain a copy in a matter of seconds.

- Create index dynamic statement invalidation

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

- Automatic space management

Many IT people in DB installations consider this a very valuable feature. It potentially eliminates one of the main causes of failure where rapid growth was not anticipated.

- Minimize impact of creating deferred indexes

Indexes created as deferred are ignored by the DB2 optimizer.

- LOB ROWID transparency

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

- Longer table and column names

Architectural changes to DB2 V8 expand the DB2 catalog and SQL with support for long names (tables and other names up to 128 bytes, column names up to 30 bytes). Support for longer string constants (up to 32,704 bytes), longer index keys (up to 2,000 bytes), and longer predicates (up to 32,704 bytes) makes DB2 UDB for z/OS compatible with other members of the DB2 family.

- SQL statements extended to 2 MB

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

- Multiple DISTINCT clauses in SQL statements

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

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

- More open data sets

With DB2 V8 and z/OS 1.5, a DB2 subsystem can have up to 65,041 open data sets, instead of 32,767 in V7. This enhancement also reduces the amount of virtual storage used by DB2 below the 16 MB line, as MVS control blocks related to dynamically allocated

data sets are requested above the 16 MB line. When those open data sets (or partitions) are compressed, the compression dictionary is loaded above the 2 GB bar.

- More log data sets

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

- CI size larger than 4 KB

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

1.1.3 Data warehouse

The most important enhancements specifically affecting the data warehouse applications are:

- More tables in joins

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

- Sparse index and in-memory work files for star join

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

- Common table expression and recursive SQL

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

- Materialized query tables

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

However several other changes to improve optimization techniques, improve statistics for optimization, and better analysis of the access path, listed under performance, also benefit warehousing applications. Even more additions that add to warehousing applications are enhancements to the ability to use indexes, the new DSTATS option of RUNSTATS, the new Visual Explain, and the online schema changes.

1.1.4 Performance

The main DB2 V8 improvements related to performance are:

- Multi-row INSERT and FETCH

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

► RUNSTATS improvements

The improvements for RUNSTATS are related to the following areas:

– Code optimization

The changes have produced up to a 40% reduction in the execution time of a RUNSTATS TABLESPACE.

– Distribution statistics

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

– Fast-cached SQL statement invalidation

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

► Host variables' impact on access paths

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

► Index only access for VARCHAR

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

► Backward index scan

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

► Local SQL cache issues and short prepare

This enhancement reduces the cost of a short prepare.

► Multiple IN values

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

► Dynamic statement cache statement ID in EXPLAIN

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

► Locking improvements

The most important improvements in locking:

– Reduced lock contention on volatile tables

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

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

- CF lock propagation reduction

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

- Several other lock enhancements, including overflow lock avoidance, skip uncommitted inserts, and drain lock on partition key update.

- Instrumentation enhancements

DB2 V8 comes with numerous enhancements to the Instrumentation Facility Interface. The most important are:

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

1.1.5 Migration changes

Migration is allowed exclusively from DB2 V7 subsystems. The migration SPE must have been applied and started. The migration process has been changed and now consists of three distinct steps or phases:

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

Performance overview

Scalability, availability, and performance are key reasons to choose DB2 as your premier enterprise database server. These reasons are mandatory criteria for DB2 in order to allow growth and be able to exploit (in a balanced manner) increasing processor speed, larger central storage and faster I/O devices. However, two major inhibitors slowed down growth for some leading edge users. These inhibitors were the size of virtual storage available per address space and the persistent difference in speed between processors and disks' service time.

The 2 GB virtual memory size, dictated by the 31-bit MVS architecture, was an inhibitor to growth for the DBM1 address space where many DB2 throughput-related pools were acquired. This limited addressing capability in turn reduced the size of the buffer pools that could be assigned for accessing data. The theoretical limit for the virtual buffer pools is 1.6 GB, but realistically, for most high-end customers, it is less than 800 MB, regardless of the availability of real storage. DB2 V7 customers, needing a larger buffer pool, likely use data space buffer pools, and are able to use 10 to 20 GB.

The architectural solution DB2 V8 provides is to adopt the 64-bit z/OS architecture and remove the major storage inhibitors. Both the DBM1 and IRLM address spaces now run in 64-bit addressing and take advantage of new 64-bit instructions of the zSeries architecture. This allows DB2 to move several storage pools above the 2 GB bar and provides customers virtual storage constraint relief. In turn, this allows exploitation of the processor speed and usage of the real storage size of the largest systems.

Savvy users are cautious about changing long-established capacity rules and will ask you pertinent questions similar to the following ones:

- ▶ What is the impact of the new architecture in terms of CPU and I/O utilization?
- ▶ How is the virtual storage layout changing? What is still below the 2 GB bar, and what has moved above the bar?
- ▶ How can I make sure I have enough real storage to support my workload today, and tomorrow?
- ▶ What else needs to be done to take advantage of the architecture?
- ▶ What new DSNZPARMs affect my real storage consumption?
- ▶ What old DSNZPARM default settings may have changed to affect storage and CPU considerations?

In this chapter we anticipate the answers to these questions. The details are spread across the remainder of this book. More complete answers will be provided when available from the experiences of a larger population of V8 users.

This chapter contains:

- ▶ Overview of major impact items
- ▶ CPU usage
- ▶ DBM1 virtual storage mapping
- ▶ Real storage
- ▶ Compatibility mode
- ▶ New-function mode
- ▶ New installation default values
- ▶ Recent z/OS releases

2.1 Overview of major impact items

There is a tremendous amount of new function available in DB2 V8. Like any release, there are things you really like and things you like less. And the things you like come at some cost. In this chapter we present a realistic perspective of DB2 V8 for you: The good and the not so good. Certainly, adding new function implies adding instructions, and it becomes more and more difficult to contain the increase in path length. The performance department at Silicon Valley Lab, along with DB2 development, tries to minimize the overhead you can incur with each new release. This is especially true with V8 because of the large amount of additional code.

It is important that you are aware of the effect DB2 V8 has on components of DB2, such as DB2 catalog, CPU growth, virtual storage, real storage, compatibility mode, and new-function mode. The more educated and informed you are before migrating to DB2 V8, the easier the planning becomes. The migration is simplified by avoiding known pitfalls. The migration goal is a smooth and successful implementation and you can concentrate on exploiting the new functions.

Important: Remember that you can only migrate to DB2 V8 from DB2 V7. It is possible to migrate to DB2 V7 from DB2 V5 by a *skip release migration*. However, skip release migration is not generally supported, including migration to V8.

We now begin to discuss the components of DB2 V8.

2.1.1 DB2 catalog

In DB2 V8, much of the character data in the DB2 catalog changes to the Unicode encoding scheme. This does not mean you have to change all your EBCDIC or ASCII data to Unicode. The catalog conversion to Unicode allows you to support Unicode object names and literals. But it does require that you, before migrating to DB2 V8, implement z/OS Conversion Services for your environment, and this requires an IPL.

Most character columns in the DB2 catalog are converted to Unicode VARCHAR(128), and there is typically a 1 to 10% increase in the space used. If you have any user-defined indexes defined on the catalog, you may want to drop them before V8. Afterwards, you can redefine these indexes and specify them as NOT PADDED. If you do not drop these indexes, the default is PADDED and these user-defined indexes grow considerably, since many columns in the index grow often from 10 to 128 bytes. For this reason, DB2 V8 allows variable length keys (VARCHAR(128)) in an index.

We anticipate that there may be some growth in the size of the DB2 catalog once you migrate. However, it really depends on a number of factors. Some installations never or infrequently perform a reorganization of the DB2 catalog. For them, the difference in size after migration may be negligible. For those who REORG on a regular basis, they can see up to a 10% increase in the size of the DB2 catalog.

For details regarding the effect of Unicode on the DB2 catalog and directory, refer to 9.1.2, "Unicode parsing" on page 342.

2.1.2 CPU

In the past, as you migrated from one release of DB2 to the next, the typical increase was not more than a 5% CPU cost. In fact, staying under 5% has been the target of DB2 development. We estimate that there is four times more code in V8 than in V7, and some CPU increase is

experienced. But this is not the main reason, since the developers have worked hard to keep the same path length when executing the DB2 functions. The main reasons are that the instructions now deal with possibly large variable length names and that 64-bit mode instructions often take more time to execute than in 31-bit mode, and there is a need for a lot of expansion of instructions from the 24 and 31-bit type.

DB2 V8 contains many new features in the areas of availability, scalability, and portability from other platforms, such as family consistency, user productivity, and other features. For those applications not taking advantage of DB2 V8 performance features, an increase in CPU time is anticipated to support all of these new features.

As you examine the different measurements in this book reported in 4.1, “DB2 CPU considerations” on page 129, you see that there is fluctuation in CPU cost depending on the different types of workloads tested. The CPU ranges vary but generally the additional CPU cost is between 0 and 10%. Of course, we think that eventually you make changes to your applications and subsystem to take advantage of those new features that ultimately reduce the CPU cost and hopefully result even in a CPU reduction.

2.1.3 I/O time

There is only good news here:

- ▶ No change in I/O times for 4 KB pages.
- ▶ Significant sequential I/O time improvement for those table spaces with greater than 4 KB page sizes, as soon as the larger CI sizes are utilized in new-function mode. See 4.10, “New CI size” on page 199.
- ▶ DB2 V8 removes the limit of 180 CIs per I/O for list prefetch and castout I/O requests. See 8.4, “DB2 I/O CI limit removed” on page 333.
- ▶ Compression, with the dictionaries now above the bar, is even more widely applicable, and can help in reducing I/O.

2.1.4 Virtual storage

By far the biggest impact in DB2 V8 for performance is related to the introduction of the 64-bit architecture. The reason for moving to 64-bit architecture is that leading edge customers were running into virtual storage constraint problems. For more information on this problem, see the article on DB2 UDB for z/OS Storage Management by John Campbell and Mary Petras at:

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

The DB2 V8 implementation of 64-bit virtual storage is good news for installations that had virtual storage constraint problems in the past. Many of them had to rearrange and reduce the main DB2 storage consumers to avoid running into out-of-storage conditions; this new configuration may not have been as optimal from a performance point of view. However, you had no choice but to reduce storage utilization to avoid these out-of-storage conditions.

DB2 has moved several critical pools above the 2 GB bar, and left some below. Now in DB2 V8, with many main storage areas moving above the 2 GB bar, a possibility exists to enlarge these storage areas, which in turn, can result in much better performance. However, be careful when looking at thread storage. Most of the thread storage still remains below the bar, including the application’s local dynamic statement cache, so carefully monitor thread-related storage.

One observation to note is that both the DBM1 and IRLM address spaces exploit 64-bit virtual addressing. The other DB2 address spaces, DIST, MSTR and SPAS, do not yet exploit this technology.

DB2 V8 extends upon its improvements in storage management and has also increased some defaults. All of these changes do not eliminate completely short-on-storage conditions. You still need to proactively monitor your storage utilization; keep running your DB2 PE storage statistics reports and watch your storage utilization over time.

2.1.5 Real storage

You can experience some real storage growth usage when you migrate from DB2 V7 to V8. The growth is due to a number of factors, the most important is 64-bit addressability, Unicode, long names support and long key support. The 64-bit instructions are inherently longer than 31-bit counterparts; this effect can increase the size of control blocks that store pointers by a factor of two. Longer names require larger definitions and space.

You need to make sure that the buffer pools are 100% backed by real storage. DB2 uses an LRU or FIFO scheme to reclaim pages. As a result, if the oldest pages are paged out to disk, the paging impact on the system would be significant. In the zSeries architecture, if the buffer pool is not backed by main storage, the pages would be stored in and out of auxiliary storage (that is, disks), since there is no configurable expanded storage to provide some hierarchy of buffering. Expanded storage is not exploited by z/OS in the z/Architecture® model. It is wise to prevent excessive paging to auxiliary storage. Use RFM Monitor II and III to monitor the real storage frames and auxiliary frames used by the DB2 address spaces. It is very important to keep in mind that your DB2 subsystem should not page.

Important: Ensure the buffer pools are backed up by sufficient real storage.

2.1.6 Compatibility mode

Once you commit to migrating to DB2 V8, the first stop you make is compatibility mode (CM). As soon as you are in CM, there is no new function. Well, there is actually some new function; to start with, DB2 is running in 64-bit mode already, however, there is *no new function which can preclude a fall back to V7*. It is not officially documented what is available in CM, and it is subject to change with maintenance. During this phase, you can test the functionality of existing applications to ensure they run without problems. Only during this phase can you revert back to DB2 V7, so once you make the commitment to proceed to the next two stops, enabling-new-function mode (ENFM) and new-function mode (NFM), there is no turning back. You will probably remain in CM for a short period of time until you are comfortable that it is safe to move forward. If you have several DB2-interconnected subsystems, DRDA or otherwise, migrate all systems to CM before migrating any system to NFM.

Keep in mind that transition among the modes in a data sharing group is an instantaneous group wide event; it is not possible to have some members in a data sharing group in one mode while others are in a different mode. All members of a data sharing group are in the same mode.

Note: Customers typically stay in CM mode for a period of several weeks to a few months.

What we have measured in CM:

- ▶ The measurements of the CPU time per commit for non-distributed IRWW workload have shown a slight increase of 4% in accounting CPU time between DB2 V7 and DB2 V8 CM. The IRWW workload is described in *DB2 for MVS/ESA Version 4 Data Sharing Performance Topics*, SG24-4611. However, there is a considerable decrease in total CPU time for the DBM1 address space (about 20%).
- ▶ Other workloads have a different behavior.

- ▶ The performance benchmarks for non-data sharing and data sharing environments have shown slight differences.
- ▶ For the online transaction distributed workload, the increase in CPU can be up to 20% with an ITR reduction up to 12%. Batch DRDA can take advantage of multi-row fetch, so it can even show an improvement of 20%.

For details on these results, see Chapter 4, “DB2 subsystem performance” on page 127.

2.1.7 New-function mode

After you have run in CM for the appropriate length of time determined by your needs, you decide when it is time to get to the next step which is enabling-new-function mode (ENFM). You run a job to do so, and now you are ready to enable new-function mode (NFM). ENFM affects the changes to the catalog, including adding long name support to the catalog, converting to Unicode, changing system-defined indexes to not-padded, and changing the page size for some of the catalog table spaces to 8 KB, 16 KB, and so on. Once this job is started, you cannot fallback to DB2 V7 or to DB2 V8 CM; the capability to fallback is removed and enforced once you enable new function mode.

While you are in ENFM, you cannot begin to use any new function yet, but you are no longer in CM either.

We recommend you consider a minimum amount of changes as you flow through the migration path to NFM. Be sure to start with the same DSNZPARM settings, and then slowly begin making changes to your environment one parameter at a time. Similarly, when you finally reach NFM, slowly begin exploiting some of the new features. This slow path is the best path for risk mitigation and to evaluate the performance impact of the new features.

A measurement of the IRWW non-data sharing workload has shown no significant difference in CPU time between CM and *no new function* NFM.

Coexistence with data sharing

Data sharing coexistence exists only in CM. Coexistence means having both V7 and V8 CM subsystems in a single data sharing group. ENFM is a group wide event since it affects a single catalog. As soon as you enable new-function mode on one of the members, it occurs for all of the members in the group.

DB2 catalog and directory

Once the DB2 catalog and directory are converted to V8 mode, the size of these structures can be slightly affected. Many existing VARCHAR columns are altered to VARCHAR(128). Since most of the columns that are being changed are already VARCHAR, the effect should be minimal. The maximum length is changing, and not the data that is already there. Some columns are changed from CHAR(8) to VARCHAR(24). The columns that used to be CHAR and are changed to VARCHAR get bigger by two bytes. The system-defined catalog indexes, since they are NOT PADDED, may actually get smaller.

Some catalogs rarely get reorganized, so for those that do not, there may be a reclaiming of space. As a result, it is possible when you migrate to NFM that the size of the catalog may be slightly reduced.

Typically, we experienced a small amount of DB2 catalog and directory growth, but not more than 10%.

2.2 CPU usage

The most common question is “What did you experience in terms of CPU performance impact?”. In any release of DB2, what the new features cost is a concern. This release contains a dramatic amount of new function, and because of that, you may experience an increase in CPU cost. Since DB2 V8 deals with much longer names, and exploits 64-bit addressing, CPU utilization can increase even if instruction path length does not change. The number of instructions may not change, but because each instruction is now a 64-bit instruction instead of a 31-bit instruction, it can be more expensive to execute.

This release also has a number of new SQL functions which usually increases path lengths. Normally, the target is less than 5% CPU increase from release to release. In V8 the target is less than 10%.

The assumption, when measuring the CPU regression, is that you do not change the application to take advantage of any new function. Obviously, once you do rewrite the application to take advantage of new function, you may even experience a decrease in CPU time.

When looking at the percent increase in CPU cost, there can be a lot of deviation depending on a number of different factors, such as:

- ▶ Type of workload
 - Online
 - Batch
 - DRDA transaction
 - Batch running in DRDA
 - Data warehouse
- ▶ Characteristics of the workload
 - Static or dynamic
 - Fetch intensive
 - Update intensive
 - A combination of the two
 - Simple SQL
 - Complex query
- ▶ Data sharing or a non-data sharing environment
- ▶ JDBC or CLI ODBC
- ▶ Utilities execution
- ▶ Commit frequency and options

There are actions that you can take to mitigate any CPU overhead and we discuss these actions in the book. One obvious task is to rebind all plans and packages after you migrate.

Once you are in CM, if you rebind, DB2 re-establishes fast column processing and your application also exploits more efficient access paths, already available under CM, that can possibly improve CPU costs.

Once you are running in NFM, you can take advantage of more optimization enhancements after rebind.

For more about new access paths, refer to Chapter 3, “SQL performance” on page 29.

Another optional item of interest to reduce CPU time is the long term page fixing which can be specified at the buffer pool level; this feature can be enabled in compatibility mode. An

application that can have great potential benefit is one using a buffer pool with a poor read hit ratio and lots of I/O. Writes also benefit from page fixing.

We explore this topic in more detail in Chapter 4, “DB2 subsystem performance” on page 127.

There are new functions in DB2 V8 where we experienced a decrease in CPU cost. We use two new functions at the SQL level to minimize the CPU impact of migrating to DB2 V8: *Multi-row insert* and *multi-row fetch*. These new functions require application changes, but can offset an increase in CPU cost of an insert intensive or fetch intensive application and are worth exploring for those applications that apply. You will find a discussion of multi-row insert and multi-row fetch in Chapter 3, “SQL performance” on page 29.

For those applications that were virtual storage-constrained with V7, there are a number of areas for possible CPU improvements. One is, if you are using data space buffer pools or hiperpools with V7, once you migrate to DB2 V8, DB2 replaces these pools with buffer pools residing above the 2 GB bar, and fewer instructions are needed for their management. If your system can use real storage for bigger buffer pools, I/O can be reduced and this results in CPU savings, especially for those applications using a lot of synchronous I/O for critical batch processes.

If you are using DSNZPARMs MINSTOR and/or CONTSTOR to contract and minimize thread storage consumption, and thread storage consumption is not the major consumer below the 2 GB DBM1 storage, you can reset them to free up much needed virtual storage for other use and gain back the CPU overhead of running these processes.

You can find more information on buffer pools, MINSTOR and CONTSTOR in Chapter 4, “DB2 subsystem performance” on page 127.

Warning: When doing performance measurements between different releases of DB2, make sure you take a look at the whole picture. An increase in class 2 CPU time may be offset by a reduction in SRB or TCB time in the DB2 address spaces. We recommend you take both sets of numbers into account when doing comparisons since the CPU time may now be charged to the application owning TCB and less to one of the DB2 address spaces. The information needed to do a proper analysis would be spread across accounting and statistics reports.

2.3 DBM1 virtual storage mapping

The limitation of virtual storage in the DBM1 address space prior to V8 is a serious problem and one of the primary reasons customers cite as their motivation for moving to DB2 V8. Virtual storage is not something you can buy; it is inherent in the software product's architecture. Prior to DB2 V8, the amount of virtual storage available is limited by the 31-bit addressing scheme which equates to an address space capable of addressing limited to 2 GB.

If you have been virtual storage-constrained, you have most likely done one or more of the following to get some storage relief:

- ▶ Moved your virtual pools to data spaces
- ▶ Used the data space extension of the EDM pool for the (global) dynamic statement cache
- ▶ Used DSNZPARMs CONTSTOR and MINSTOR if using long running threads to minimize thread-related storage

- Reduced the number of objects using compression to minimize the number of compression dictionaries

2.3.1 DB2 structures above the bar

In DB2 V8 the data space buffer pool, data space lookaside pool, and hiper pools have been eliminated. Many elements have moved above the 2 GB bar, including the virtual buffer pools and their control blocks. Here are some areas now located above the 2 GB bar:

- Buffer pools
- Buffer pool control blocks
- Compression dictionaries
- EDM pool components for DBDs and dynamic statement cache
- Sort pool
- RID lists

What does this mean for the application? You need to examine the extra storage gained and decide what to do with it. The application may improve with larger buffer pools and since these reside above the 2 GB bar, it may be the vehicle that reduces execution time to the application. Or you may decide to enlarge the EDM pool or Sort pool.

If you were using the DSNZPARM CONTSTOR and MINSTOR to contract thread storage, and thread storage is not critical, you can now consider turning this off, eliminating the CPU overhead usage attributed to storage contraction.

If you are a heavy user of compression, you benefit from making more storage available below the bar.

2.3.2 What are the implications of more storage?

Are you able to expand the number of active threads? It depends on the storage footprint for threads. How many additional threads can be supported varies by workload and depends on various factors such as:

- Duration of the thread
- SQL workload
- BIND parameters RELEASE

If you have already exploited the virtual storage constraint relief recommendations above, the added benefit when you migrate to DB2 V8 can possibly result in additional virtual storage relief. If you have not done anything, the added benefit should be higher. To summarize, this added virtual storage relief can help customers currently experiencing virtual storage constraint problems in DB2 V7. But, if you are currently monitoring storage closely, then plan to continue monitoring in V8.

If IRLM is a significant consumer of ECSA, you can consider decreasing ECSA, since the IRLM no longer has its locks in ECSA.

2.4 Real storage

A key message is you must have the real storage to fully back increased storage use; otherwise, you risk paging that adversely affects performance. Remember that expanded storage no longer exists in this environment, and therefore, any paging goes directly to disk; the effect can be immediately significant.

Because you can significantly increase the size of the buffer pools; theoretically, you can dramatically reduce the number of synchronous I/Os and therefore reduce the elapsed time for most transactions. This can significantly improve batch performance, especially for those workloads where the number of synchronous I/Os is high. As a result, DB2 V8 can provide a good real memory versus I/O trade-off. However, you need to monitor real storage usage.

In general, we have experienced an increase of up to 10-20% in real storage going to DB2 V8 when keeping constant all the possible real storage requirements. If you are close to fully exploiting your real storage, it is safer to verify the exact usage with RMF™ reports and review the virtual storage definitions, including the new default values, in order to contain the demand on real storage.

2.5 Compatibility mode

Once you migrate to CM, presumably there is no new V8 function available. Well, actually there is no new function that would preclude a fallback. But new function is introduced. There are several changes you have absolutely no control over that automatically occur behind the scenes. The list below, compiled during this project, gives you an idea of what is available, but by no means can be considered precise or constant, because it is subject to change with maintenance:

- ▶ 64-bit addressing for DBM1 and IRLM address spaces
- ▶ Increased number of deferred write, castout and global buffer pool write engines
- ▶ EDM pool changes
- ▶ Fast log apply default
- ▶ Multi-row operations with DRDA
- ▶ IRLM locks in private storage
- ▶ Unicode parser
- ▶ Most of the optimizer enhancements
- ▶ Lock avoidance in singleton SELECT with CURRENT DATA YES
- ▶ 180 CI limit removal for list prefetch and castout read
- ▶ Ability to page fix buffer pools for long term
- ▶ SMF 89 improvement
- ▶ Data sharing IMMEDIATE WRITE default change
- ▶ All the enhancements provided by utilities, except for BACKUP and RECOVER SYSTEM

These situations and their performance implications are discussed in this section.

2.5.1 64-bit addressing

When you migrate to DB2 V8 in CM, the DBM1 and IRLM address spaces immediately start to use 64-bit addressing. You no longer have the option to allow either address space to run using 31-bit addressing. Applications on the other hand happily run as before; there is absolutely no need to make changes to the application, unless of course, you want to take advantage of new features.

Figure 2-1 shows the virtual storage layout for the different address spaces. Notice the IRLM and DBM1 address spaces use 64-bit virtual addressing, where DDF, MSTR, SPAS and the user-defined application program remain as before.

Also, it is important to realize that the storage for 64-bit is not drawn to scale; the DBM1 and IRLM address space have a huge amount of virtual storage, 8 billion times the other address spaces.

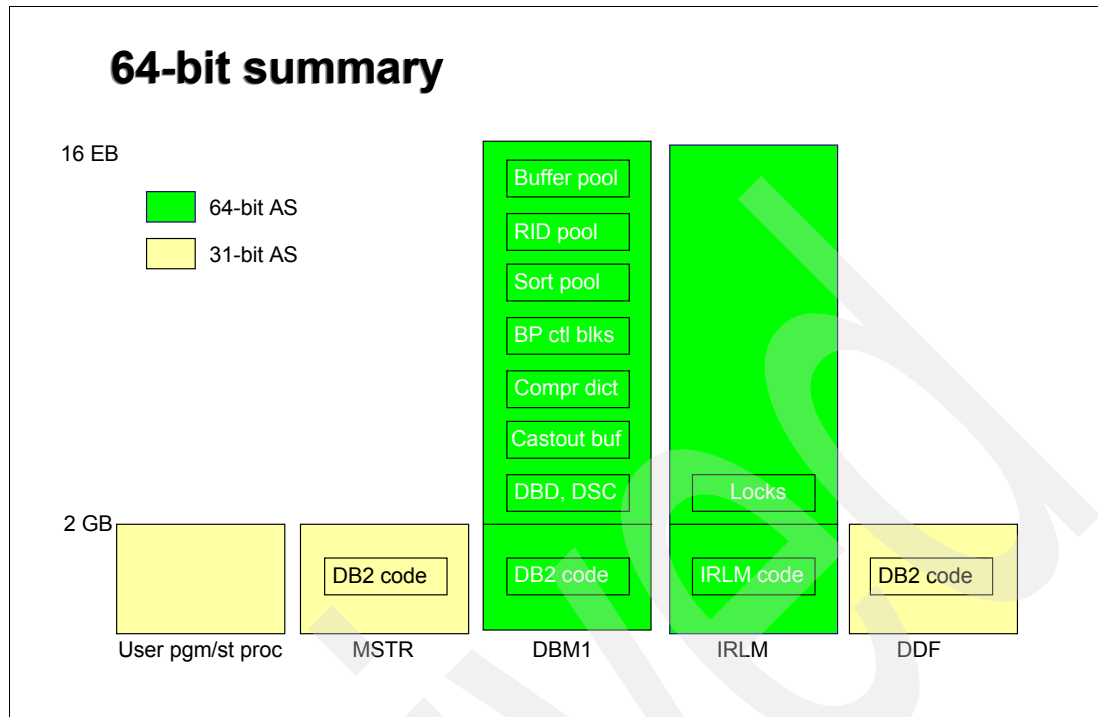


Figure 2-1 Virtual storage growth

2.5.2 No more hiperpools and data spaces

In DB2 V8 CM, there are no longer hiperpools and buffer pools in data spaces; if they existed prior to the migration, they automatically revert to buffer pools. DB2 automatically reallocates the buffer pool sizes based on the original virtual buffer pools and the hiperpools or buffer pools in data spaces. For example, if BP4 consisted of 4,000 virtual buffer pool pages and 20,000 hiperpool pages, then after the migration, BP4 contains 24,000 pages in a buffer pool.

These buffer pools are automatically allocated and reside above the 2 GB bar. DB2 no longer uses the terminology virtual pool and instead uses buffer pool. The total buffer pool storage must never exceed the real storage available for buffers. There are also reasonableness checks for 1 terabyte of storage for a single buffer pool and for the sum of all buffer pools. Not only do the buffer pools reside above the bar, but so do the corresponding buffer pool control blocks.

Important: If you are using a large number of hiper pools with V7, review the buffer pool definitions before your DB2 V8 migration. Make sure you have adequate real storage, and also adjust the thresholds of the new buffer pools, especially if they are not defined in percentages. When in doubt, use the V8 defaults.

An important performance consideration here is to make sure you have enough real storage to back these entities. If you do not, you can encounter severe performance penalties for paging; since expanded storage no longer exists, you page to DASD.

One immediate benefit of having one type of buffer pool is simplification, and you also have immediate virtual storage constraint relief. This factor alone can move you toward increasing the number of threads, but since the thread-related storage stays below the bar, we do not recommend this type of change so soon after migrating to CM without verification.

The possible increase in thread footprint in migrating to V8 is currently estimated to be between 30% and 70%. This is based on the laboratory measurements thus far.

For more information on the buffer pools in DB2 V8 and related performance implications, refer to Chapter 4, “DB2 subsystem performance” on page 127.

2.5.3 Increased number of deferred write, castout and GBP write engines

As the storage constraints are lifted, the workload may increase. As a result, DB2 increases the maximum number of deferred write, castout and GBP write engines from 300 to 600. This increase should alleviate *engine not available* conditions.

For more information on these write engines and their performance implications, refer to Chapter 4, “DB2 subsystem performance” on page 127.

2.5.4 EDM pool changes

One of the largest virtual storage consumers is usually the EDM pool. With DB2 V7 some virtual storage relief was provided by the capability to move part of the EDM pool to a data space. Further enhancements are provided by DB2 V8.

In this section we discuss:

- ▶ Dynamic statement cache
- ▶ DBD cache
- ▶ Plans, packages and DBDs

For more information on the EDM pool and its performance implications, refer to Chapter 4, “DB2 subsystem performance” on page 127.

Dynamic statement cache

In DB2 V7 dynamic statements are cached in the dynamic statement cache only if the DSNZPARM CACHEDYN is turned on. In DB2 V8 caching is enabled by default (new install) and the dynamic statement cache is moved above the 2 GB bar. This can provide additional storage relief to the DBM1 address space in case a data space was not used.

DBD cache

In addition, a new DBD cache is created for the storage of DBDs and also allocated above the 2 GB bar. Segregating the DBDs from the EDM pool relieves contention from other objects in the EDM pool. DB2 V8 can support more and larger DBDs adding to DB2 V8’s scalability.

Plans, packages, and DBDs

Plans, packages, and DBDs are all stored in the directory. There is minimal performance overhead running in CM because DB2 plans, packages, and DBDs are different between V7 and V8. This overhead can be alleviated for plans and packages by rebinding them. We recommend rebinding anyway to take advantage of V8 access path improvements, avoid conversions and enable faster processing.

Important: Rebind plans and packages once you are comfortable with CM. If you can live with the overhead during CM and cannot rebind twice, then rebind once you are in NFM.

2.5.5 Fast log apply

Once you migrate to DB2 V8 CM, the new default for fast log apply (FLA) is 100 MB. This should have a minimal effect on DBM1 storage. FLA is used by the RECOVER utility, during restart of a failed DB2, or when RESTORE SYSTEM is used. The storage is allocated when needed and released when the FLA phase is complete.

When multiple RECOVER utility jobs are run in parallel, DB2 can take advantage of FLA for up to 10 concurrent jobs, each utilizing 10 MB.

The default value for FLA at restart is always 100 MB, even if you explicitly specify 0 in the DSN TIPL panel or in the DSNZPARM LOGAPSTG.

During RESTORE SYSTEM, a new utility with DB2 V8, the default value for FLA (if LOGAPSTG is not zero) is 500 MB.

For more information on FLA and its performance implications, refer to Chapter 4, “DB2 subsystem performance” on page 127 and Chapter 5, “Availability and capacity enhancements” on page 217.

2.5.6 Multi-row operations with DRDA

DB2 V8 CM automatically exploits multi-row fetch when you use a remote client that uses block-fetching in a distributed application. The DDF address space uses multi-row fetch to build the blocks of data to be sent to the client. This implies you do not have to change the application to take advantage of this tremendous performance enhancement. We have experienced performance improvements in these client applications in DB2 V8 CM.

For more information on the multi-row fetch operations with DRDA, refer to Chapter 7, “Networking and e-business” on page 281.

2.5.7 IRLM V2.2

The IRLM address space also changes with DB2 V8 and supports 64-bit virtual addressing. The IRLM is able to support more concurrent locks. PC=YES is now enforced when you migrate to DB2 V8. The ECSA usage (PC=NO) capability can still be specified, but is no longer used by V8. Therefore, the ECSA previously used for IRLM, if you used PC=NO, could now be available for use by other applications.

Important: If you used PC=NO, review your ECSA usage, since potentially there can be less of a need for ECSA with IRLM V2.2.

Real storage for IRLM locks increases significantly, since they have doubled in size. For more information on the IRLM in 64-bit mode, refer to Chapter 4, “DB2 subsystem performance” on page 127.

2.5.8 Page-fixed buffer pools

DB2 V8 introduces a new buffer pool parameter called PGFIX which allows you to choose to fix buffer pools in real storage. Before reading or writing a page, DB2 must fix the page in real storage in order to perform the I/O. In DB2 V8, with this new buffer pool option, you can decide to permanently page fix an entire buffer pool. The option to page fix a buffer pool is available in CM, but is not automatically enabled.

Important: Consider setting the page fixing option for buffer pools with a poor read hit ratio and lots of I/O. Balance this setting against the need for real storage in the rest of the system.

For more information on page fixing and its performance implications, refer to Chapter 4, “DB2 subsystem performance” on page 127.

2.5.9 Unicode parser

DB2 must convert all SQL to Unicode since all SQL statements are now parsed in Unicode UTF-8 format. This conversion is generally performed by DB2 invoking z/OS Conversion Services. In addition, the EBCDIC metadata stored in the catalog is converted to Unicode in CM. After you enable NFM, the metadata is stored in Unicode and the conversion is not necessary.

If you are using an SBCS code page, the conversion is done directly by DB2. If you have a more complicated code page, then DB2 invokes z/OS Conversion Services.

For more information on the Unicode parser and its performance implications, refer to Chapter 4, “DB2 subsystem performance” on page 127.

2.5.10 Optimizer enhancements

Once you migrate to DB2 V8 CM, you can choose to rebind your application plans and packages to possibly get improvements based on the optimizer enhancements to access paths. For example, fast column processing is re-enabled if you rebind, and your application can take advantage of the new stage 1 and indexable predicates.

For more information on application rebinds, refer to Chapter 4, “DB2 subsystem performance” on page 127.

2.5.11 CF request batching

A new function in z/OS V1.4 and CF level 12, called *CF request batching*, is the ability to allow multiple pages to be registered to the CF with a single operation. DB2 V8 takes advantage of this feature in DB2 V8 during the actions of registering and writing multiple pages to a GBP and reading multiple pages from a GBP during castout processing. This new feature really allows for more efficient traffic across the links to and from the CF, and can result in a reduction of CF processing times and costs associated with data sharing for some workloads.

For more information on CF request batching, refer to Chapter 8, “Data sharing enhancements” on page 319.

2.6 New-function mode

The performance comparisons are generally dependent on the particular workload.

Preliminary measurements in non-data sharing and non-DRDA environments indicate no significant difference in CPU time overhead in CM versus NFM.

However, in data sharing, the story is much improved. If you are interested in the performance impact of a DB2 V7 application migrated to V8 without any application change and significant configuration change, there can be a noticeable reduction in V8 overhead going from CM to

NFM in data sharing because some data sharing-related enhancements are automatically taken advantage of in NFM. Most of the improvement can be attributed to the new locking protocol level 2. For more information on improvements in NFM for data sharing environments, see Chapter 8, “Data sharing enhancements” on page 319.

The real performance difference in NFM comes from the ability to exploit new functions not available in CM, for example, materialized query tables.

2.6.1 Plans and packages

There is overhead in NFM for plans, packages, and DBDs. The representations of these objects are different between DB2 V7 and V8. The costs come from having to reformat control structures to match the release and in converting between EBCDIC and Unicode.

Once you are in NFM, DB2 BIND and REBIND write plans and packages to the DB2 catalog and directory in Unicode.

Whenever DB2 accesses plans, packages, or DBDs that are not in DB2 V8 format, DB2 must expend some extra CPU to convert them into DB2 V8 format. This is always true in CM, but continues to be true even in NFM. In order for the plan or package to reside in the catalog in DB2 V8 format, but most important, to enable all DB2 functions, you just do a rebind or a bind.

Important: To get around any reformatting costs and enable DB2 V8 functions (such as NOT PADDED and index only access, the additional ability to use indexes, MQTs, IS NOT DISTINCT, and more), rebind your plans and packages in V8 NFM.

2.6.2 DB2 catalog

When you migrate to NFM, the catalog changes from EBCDIC to Unicode. The conversions happen in ENFM, and it is an important difference for a few customers. A few customers stay in ENFM, even though all catalog tables are in Unicode. Some may fall back from NFM to ENFM. The other significant difference is the capability of running during ENFM.

Important: Once you are in NFM, the collating sequence of the DB2 catalog changes since the catalog is now converted to Unicode. You may get different results when using range predicates against the DB2 catalog.

Other incompatibilities are described in the *Premigration activities* section of the *DB2 UDB for z/OS Version 8 Installation Guide*, GC18-7418-02.

For instance, In NFM, an identifier that was valid in DB2 V7 might be flagged as too long in DB2 V8 and you receive message DSNH107I (during precompilation) or SQLCODE -107 (otherwise).

The first 128 code points of Unicode match the ASCII code page, so the ordering is different when you use an ORDER BY in an SELECT against the catalog.

2.6.3 Precompiler NEWFUN option

Whenever a plan or package, whose last BIND was prior to DB2 V8 NFM, is rebound, DB2 converts the EBCDIC to Unicode before proceeding. This is always true in CM, and continues to be true even in NFM. To alleviate this cost, the plan or package must be bound with a DBRM produced by the precompiler using NEWFUN(YES). The default for NEWFUN switches to YES from NO for the precompiler once you switch to NFM.

You can keep the DBRMs in EBCDIC format if you use NEWFUN(NO), but you cannot use any new functions.

2.6.4 SUBSTRING function

Once you go into ENFM and beyond, you can use the new V8 SUBSTRING built-in function, which is character-based. The old V7 SUBSTR function continues to be byte-based. DB2 V8, with APAR PQ88784, introduces a number of new “character-based” functions rather than the older “byte-based” functions. Some code pages have special characters that can now result in more than one byte in Unicode, so you may get different results using SUBSTRING. Make sure you use the right substring function to get the correct results.

Several new functions are added and several existing functions are enhanced to allow the processing of character data in more of a “character-based” than “byte-based” manner. The new functions are:

- ▶ CHARACTER_LENGTH
- ▶ POSITION
- ▶ SUBSTRING

The changed functions are:

- ▶ CHAR
- ▶ CLOB
- ▶ DBCLOB
- ▶ GRAPHIC
- ▶ INSERT
- ▶ LEFT
- ▶ LOCATE
- ▶ RIGHT
- ▶ VARCHAR
- ▶ VARGRAPHIC

The new and changed functions allow you to specify the code unit in which the strings should be processed. That is, CODEUNITS determines how to interpret a character string.

2.7 New installation default values

In this section, we list the changes that have taken place to the default values of several buffer pool threshold settings in DB2 V8. These new values are the result of system performance tuning work done by the DB2 Development Performance Department in SVL, and considerations related to a more general applicability of those values. They are applied to your system if you are doing a new install of DB2 and select not to specify the values yourself.

If you are migrating to V8, and your parameter values are specified as the old defaults, then you should consider changing to the new values for improved performance and availability, particularly CACHEDYN and CHKFREQ.

We also list the new DSNZPARM default values. These are changes to the defaults in the installation CLIST for DB2 V8 which affect your definitions if you are just letting the panels assume defaults. If this is the case, and you want to compare performance across V7 and V8 in compatibility mode, you might want to review the new values and set them back to the old defaults, at least for the initial period.

New buffer pool threshold values

For the ALTER BUFFERPOOL statement, the initial default for the deferred write thresholds are:

- ▶ DWQT — buffer pool deferred write threshold as a percentage of the total virtual pool size.
The new value is 30%. The default is decreased from the old value of 50%.
- ▶ VDWQT — buffer pool vertical deferred write threshold as a percentage of the total virtual pool size.
The new value is 5%. The default is decreased from the old value of 10%.

For the ALTER GROUPBUFFERPOOL statement the initial default for the deferred write thresholds are:

- ▶ CLASST — threshold for the class castout to disk as a percentage of the size of the data entries in the group pool.
The new value is 30%. The default is decreased from the old value of 50%.
- ▶ GBPOOLT — threshold for the class castout as a percentage of the size percentage of the total virtual pool size.
The new value is 5%. The default is decreased from the old value of 10%.
- ▶ GBPCHKPT — default for the GBP checkpoint interval in minutes.
The new value is 4 minutes. The default is decreased from the old value of 8 minutes.

Changes to default parameter values

Table 2-1 summarizes the old and new DSNZPARM default values, sorted by PANEL ID.

Table 2-1 Changed DSNZPARM default settings

Macro	Parameter	Old value	New value	Panel	Description
DSN6FAC	TCPKPALV	ENABLE	120 sec.	DSNTIP5	TCP/IP keepalive
DSN6SYSP	LOBVALA	2048 KB	10240 KB	DSNTIP7	Storage value by user
DSN6SPRM	CACHEDYN	NO	YES	DSNTIP8	Cache for dynamic SQL
DSN6ARV	BLKSIZE	28672	24576	DSNTIPA	Archive log block size
DSN6FAC	IDTHTOIN	0	120	DSNTIPB	Idle thread timeout
DSN6SPRC	SPRMINT	180	120	DSNTIPC	DDF idle thread interval time out
DSN6SPRM	EDMDBDC	N/A	102400 KB	DSNTIPC	EDM DBD cache
DSN6SPRM	EDMPOOL	7312 KB	32768 KB	DSNTIPC	EDM pool size
DSN6SPRM	EDMSTMTC	N/A	102400 KB	DSNTIPC	EDM statement cache
DSN6SYSP	CONDBAT	64	10000	DSNTIPE	Max number of remote connected users
DSN6SYSP	CTHREAD	70	200	DSNTIPE	Max number of users
DSN6SYSP	IDBACK	20	50	DSNTIPE	Max number of batch connections
DSN6SYSP	IDFORE	40	50	DSNTIPE	Max number of TSO connections
DSN6SYSP	MAXDBAT	64	200	DSNTIPE	Max number of remote active users

Macro	Parameter	Old value	New value	Panel	Description
DSN6SYSP	CHKFREQ	50000	500000	DSNTIPL	Checkpoint log records frequency
DSN6SYSP	SYSPLOGD	NO	5	DSNTIPL	BACKODUR multiplier
DSN6SYSP	ACCUMACC	N/A	10	DSNTIPN	New with V8 Accounting accumulation for DDF
DSN6SYSP	SMFSTST	YES(1,3,4,5)	YES(1,3,4,5,6)	DSNTIPN	SMF classes for STATISTICS
DSN6SPRM	AUTHCACH	1024	3072	DSNTIPP	Cache for plan authorization
DSN6SYSP	LOGAPSTG	0	100 MB	DSNTIPL	Storage for log apply
DSN6FAC	CMTSTAT	ACTIVE	INACTIVE	DSNTIPR	DDF threads
DSN6SPRM	DSMAX	3000	10000	DSNTIPR	Maximum open data sets
DSN6SYSP	EXTSEQ	NO	YES	DSNTIPR	Extended security
N/A Function not available in DB2 V7.					

2.8 Recent z/OS releases

DB2's growing synergy with z/OS continues and DB2 V8 is no exception. New z/OS versions provide additional functionality and performance improvements. The table below illustrates some of the new features and improvements made to z/OS as it pertains to DB2.

Table 2-2 z/OS release functionality

z/OS version	Functionality
1.3	Minimum required level for DB2 V8. Ongoing support for 64-bit virtual addressing Interface to RMF monitor III for obtaining performance data on capacity consumption End of service March 31, 2005
1.4	z990 compatibility CF request batching WLM-managed batch initiator enhancements z/OS Conversion Services enhancements Support of Debug Tool for DB2 stored procedures RACROUTE REQUEST=VERIFY abends allow DB2 applications to tell RACF® to issue a return and reason code instead of an abend for certain types of situations
1.5	64-bit virtual storage enhancements Multilevel security Improved backup and recovery of DB2 data Improved performance for DFSORT Extended self-optimization of WebSphere applications WLM virtual 64-bit support SMF buffer constraint relief I/O management enhancements Number of open data sets increased DB2 nested stored procedures WLM enhancement Scrollable and multiline panel fields in ISPF Sysplex performance enhancements High level assembler V1R5 RACF DB2 V8 support

z/OS version	Functionality
1.6	64-bit virtual storage enhancements WLM virtual 64-bit support Support for zSeries application assist processor Requires a zSeries server (z800, z890, z900, z990) Increased scale up to 24 engines in a single z/OS image Improved availability of TCP/IP networks across a sysplex Sysplex performance enhancements Sysplex autonomies - automated recovery enhancements Improved multilevel security
1.7	Security Server (RACF) improvements: Support is planned for mixed-case passwords. Scalability enhancements to include: <ul style="list-style-type: none"> ▶ Sequential and EXCP data sets larger than 64K tracks, in addition to existing support for extended format sequential data sets larger than 64K tracks. ▶ More than 255 extents per VSAM component. ▶ XRC and Geographically Dispersed Parallel Sysplex™ (GDPS®) to allow log data managed by the System Logger to be replicated to a remote location.

2.8.1 Multilevel security

z/OS V1.5 adds (and z/OS V1.6 improves) support for multilevel security on the zSeries. Many organizations such as government agencies and financial institutions have stringent security requirements and only allow access to information based on the person's need to know, or clearance level. Multilevel security is designed to prevent individuals from accessing unauthorized information and declassifying information. DB2 V8 is the first version of DB2 that supports multilevel security.

For more information on multilevel security, see Chapter 4, "DB2 subsystem performance" on page 127 and the recent book, *z/OS Multilevel Security and DB2 Row-level Security Revealed*, SG24-6480.

2.8.2 DFSMSHsm Fast Replication

The DFSMSHsm Fast Replication in z/OS 1.5 also provides a fast, easy to use backup and recovery solution specifically designed for DB2 V8. It is designed to allow fast, non-disruptive backups to be taken at appropriate events when there is minimum activity at the application level or when a fast point-in-time backup is desired. See 5.1, "System level point-in-time recovery" on page 218, and the book *Disaster Recovery with DB2 UDB for z/OS*, SG24-6370.

Archived

SQL performance

In this chapter we discuss the major performance enhancements that affect SQL. The chapter contains the following sections:

- ▶ Multi-row FETCH, INSERT, cursor UPDATE and DELETE

Multi-row operations can be executed in a single SQL statement explicitly to process a rowset. It is used implicitly by default in the distributed environment as described in Chapter 7, “Networking and e-business” on page 281. The DSNTEP4 sample is similar to DSNTEP2 and exploits multi-row fetch as described in Chapter 9, “Installation and migration” on page 341.

- ▶ Materialized query table

Materialized query tables (MQT) can be created in NFM as a pre-computed join, sort or aggregation and re-used by DB2 query rewrite in dynamic SQL to improve the execution of long running queries in the Data Warehouse type environment.

- ▶ Star join processing enhancements

The execution of star join is improved by: Access path enhancements, better estimates of the filtering effect of dimensions, the use of sparse index in cache, and the use of in-memory work file.

- ▶ Stage 1 and indexable predicates

The predicates comparing mismatched string data types or comparing mismatched numeric data types that are Stage 2 in DB2 V7 can become Stage 1 and indexable in DB2 Version 8. This benefit can be obtained by rebinding the plan or package.

- ▶ Multi-predicate access path enhancement

Queries which contain two or more predicates referencing a table may get a better access path with statistics for a group of columns (non-indexed) collected by RUNSTATS.

- ▶ Multi-column sort merge join

Access path selection allows multi-column predicates for sort merge join (SMJ) with mismatched string data types; decides to sort new table based on cost and enables query parallelism for multi-column sort merge join (MCSMJ).

- ▶ Parallel sort enhancement

Parallel sort is enabled for multiple-table sort (not restricted for single-table sort) for long running queries resulting in reduction of elapsed time. Parallel sort may need bigger work file buffer pool size, more work file or use Parallel Access Volume.

- ▶ Table UDF improvement

DB2 allows you to specify the cardinality option when you reference a user-defined table function in an SQL statement. With this option, users have the capability to better tune the performance of queries that contain user-defined table functions.

- ▶ ORDER BY in SELECT INTO with FETCH FIRST N ROWS

It enables SELECT INTO to get the top rows based on a user-specified ordering, thereby reducing the need for a number of FETCHes.

- ▶ Trigger enhancements

This enhancement saves work file allocation if WHEN condition evaluates to false or transition file fits into working storage. This avoids work file creation at each trigger invocation.

- ▶ REXX support improvements

The performance of REXX™ programs accessing DB2 tables is improved in DB2 V8 if the program issues large numbers of SQL statements by avoiding loading DSNREXX at each REXX API invocation and optimizing REXX API initialization.

- ▶ Dynamic scrollable cursors

Dynamic scrollable cursor is a new option which allows the application to see the updated/inserted rows and goes against the base table. It does not create the temporary table needed for static scrollable cursors.

- ▶ Backward index scan

Potential performance enhancements through better access paths and potentially fewer required indexes.

- ▶ Multiple distinct

With DB2 V7 only one DISTINCT keyword per SELECT/HAVING clause is allowed. If we need to aggregate (COUNT, SUM) by distinct values in more than one column, we must submit multiple queries. With DB2 V8, a query with multiple DISTINCTs reduces CPU and elapsed time when compared to the time necessary in DB2 V7 to execute multiple queries.

- ▶ Visual Explain

We provide an overview of the new Visual Explain and show examples of using the new Statistic Advisor function. Visual Explain was rewritten to provide better, faster analysis of problem queries. It provides much more detailed information about the access path. It formats the information into an XML document which can be sent to the lab for analysis. The EXPLAIN tables have been enhanced for DB2 V8, and a summary of these changes is provided in Appendix D, "EXPLAIN and its tables" on page 407.

3.1 Multi-row FETCH, INSERT, cursor UPDATE and DELETE

When using multi-row fetch operations, available in NFM, you normally fetch a set of rows, or a rowset in a single fetch operation, as shown here in the case of 10 rows:

```
FETCH NEXT ROWSET FROM my-cursor FOR 10 ROWS INTO :hva1, :hva2, :hva3
```

Figure 3-1 shows the advantage of dealing with rowsets.

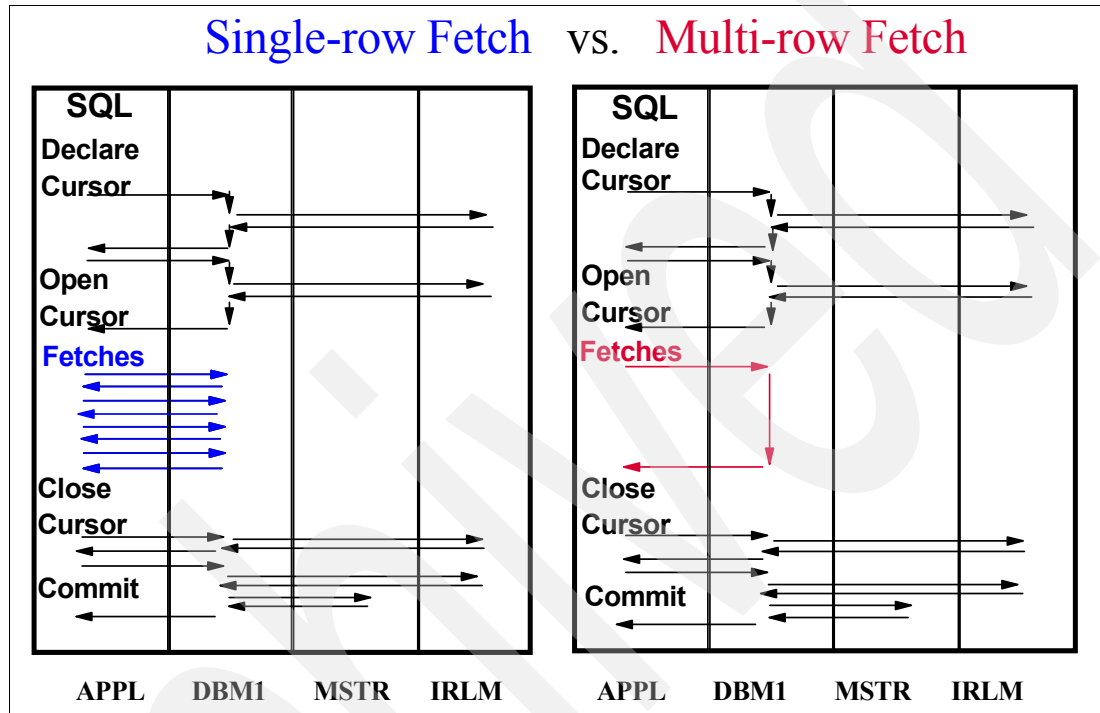


Figure 3-1 Multi-row fetch reduces the trips across the API

Since DB2 now fetches 10 rows in a single Application Program Interface (API) crossing (going from the application to DB2 and back), instead of 10 API crossings, one for each row fetched without rowsets, multi-row fetch reduces the multiple trips between the application and the database engine. This reduction produces even more benefits in distributed applications, see 7.2, "Multi-row FETCH and INSERT in DRDA" on page 285.

Characteristics of multi-row fetch:

- ▶ Multi-row fetch eliminates multiple trips between the application and the database engine
- ▶ Multi-row fetch enhances the usability and power of SQL
- ▶ A single fetch statement using multi-row fetch can retrieve multiple rows of data from the result table of a query as a rowset
- ▶ Rowset is the set of rows that is retrieved through a multi-row fetch

Characteristics of a rowset cursor:

- ▶ A rowset cursor is a cursor which returns one or more rows for a single fetch statement
- ▶ Each row in a rowset can be referenced in subsequent cursor-positioned UPDATE and DELETE statements

Characteristics of the host variable array (HVA):

- ▶ The HVA is an array in which each element of the array contains a value for the same column (one HVA per column)
- ▶ The HVA can only be referenced in multi-row FETCH or INSERT
- ▶ The HVA is used to receive multiple values for a column on FETCH, or to provide multiple values for a column on INSERT
- ▶ The HVA is supported in COBOL, PL/I, C and C++

3.1.1 Example of usage of multi-row fetch in PL/I

Figure 3-2 shows the declaration of Host Variable Arrays, the declaration of the cursor with rowset positioning, open cursor and fetch of a rowset of 10 rows in a single SQL statement.

Declare HVAs with 10 elements for each column

```
DCL COL1 (10) CHAR(8);
DCL COL2 (10) CHAR(8);
DCL COL3 (10) BIN FIXED(31);
```

Each element of the HOST VARIABLE ARRAY contains a value for the same column

Declare a CURSOR C1 and fetch 10 rows using a Multi-row FETCH

```
EXEC SQL
  DECLARE C1 CURSOR WITH ROWSET POSITIONING FOR
  SELECT * FROM TABLE1;

EXEC SQL OPEN C1;

EXEC SQL
  FETCH NEXT ROWSET FROM C1 FOR 10 ROWS
  INTO :COL1, :COL2, :COL3;
.....
```

WITH ROWSET POSITIONING specifies whether multiple rows of data can be accessed as a rowset on a single FETCH statement

The size of the rowset is not specified on the DECLARE CURSOR statement, it is done at FETCH time

Figure 3-2 Usage of multi-row fetch

3.1.2 Example of usage of multi-row insert in PL/I

Example 3-1 shows the statement to insert a rowset of 10 rows into a table with a single SQL statement.

Example 3-1 Insert 10 rows using host variable arrays for column values

```
EXEC SQL
  INSERT INTO TABLE2
  VALUES (:COL1, :COL2, :COL3)
  FOR 10 ROWS ;
.....
```

When we increase the number of rows processed per multi-row fetch or insert, the size of the host variable array is increased in the application program and it is processed row by row in DB2.

In an application, multi-row inserts, positioned updates, and positioned deletes have the potential of **expanding the unit of work**, for instance by deleting a rowset. This can affect the concurrency of other users accessing the data. Minimize contention by adjusting the size of the host variable array, committing between inserts, updates, and preventing lock escalation.

GET DIAGNOSTICS can be used in conjunction with and instead of the SQLCA to interrogate the results of all SQL statements. It is especially important when dealing with non-atomic multi-row insert statements, and objects with long names, which potentially no longer fit into the SQLCA message area. See *DB2 UDB for z/OS Version 8 SQL Reference*, SC18-7426-01 for details.

3.1.3 Multi-row fetch/update and multi-row fetch/delete in local applications

When the cursor is positioned on a rowset, the rows belonging to the rowset can be updated or deleted:

- ▶ Multi-row FETCH/UPDATE
 - Positioned UPDATE of multi-row FETCH
 - The rows of the current rowset are updated if the cursor is positioned on a rowset
- ▶ Multi-row FETCH/DELETE
 - Positioned DELETE of multi-row FETCH
 - The rows of the current rowset are deleted if the cursor is positioned on a rowset

If cursor CS1 is positioned on a rowset consisting of 10 rows of a table and we want to delete the fourth row of the rowset:

```
EXEC SQL DELETE FROM T1 WHERE CURRENT OF CS1 FOR ROW 4 OF ROWSET;
```

The same is applicable to UPDATE.

Example of usage of multi-row fetch/update and fetch/delete in PL/I

The example in Figure 3-3 shows the statements to update and delete rowsets of 10 rows from a table positioned by a rowset cursor.

Update 10 rows using a positioned Multi-row FETCH/UPDATE on a rowset cursor

```
EXEC SQL
  UPDATE TABLE1
  SET COL3=COL3+1
  WHERE CURRENT OF C1;
```

Delete 10 rows using a positioned Multi-row FETCH/DELETE

```
EXEC SQL
  DELETE FROM TABLE1
  WHERE CURRENT OF C1;
```

Figure 3-3 Example of multi-row cursor update and delete

To update in SQL, if only the nth row in the rowset instead of the whole rowset needs to be updated, which might be more common, you specify 'FOR ROW n OF ROWSET', as shown here:

```
EXEC SQL
  UPDATE TABLE1
  SET COL3=COL3+1
  WHERE CURRENT OF C1 FOR ROW n OF ROWSET;
```

All constraint checks are effectively made at the end of the statement. In the case of a multi-row update, this validation occurs after all the rows are updated. The only place it can be different is in AFTER UPDATE FOR EACH ROW triggers that have an external action. In all other cases, the whole UPDATE is an atomic operation. Either all changes are made or no changes are made.

3.1.4 Performance

In this section we discuss the environment and measurements related to multi-row operations when executed locally.

Performance measurement scenario

- ▶ Configuration
 - DB2 for z/OS V8 and V7
 - z/OS Release 1.4.0
 - 2-way processors 2084 z990
 - ESS 800 DASD with FICON® channels
 - Non-data sharing
- ▶ Non-partitioned table space, 100,000 rows table, 26 columns, 1 index

- Static SQL with table space scan
- Fetch, insert, fetch/update and fetch/delete, each for a total of 100k rows
 - V7 and V8 single row
 - V8 multi-row (explicit)

Multi-row fetch performance

Figure 3-4 shows the CPU time (class 1) measurement for single-row fetch and multi-row fetch for all 100,000 rows from a non-partitioned table space. For the multi-row fetch cases we fetch the 100k rows varying the rowset from 2, 10, 100, 1k, and 10k rows.

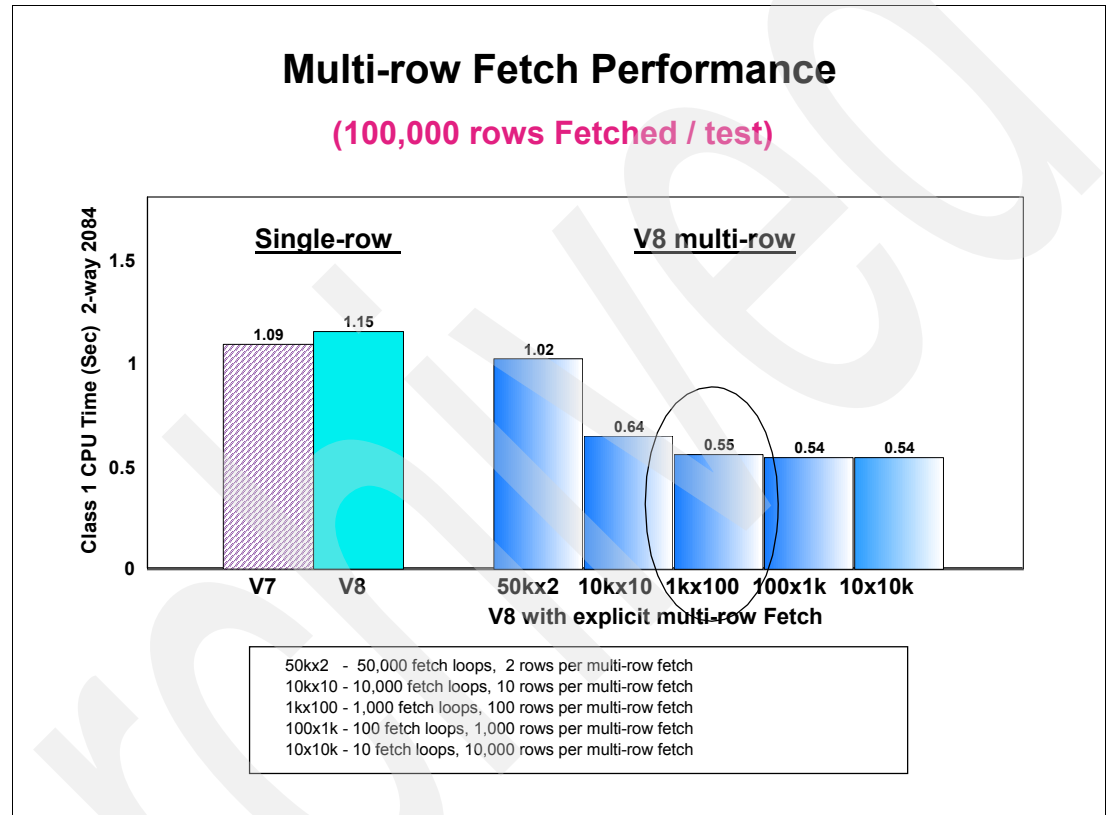


Figure 3-4 CPU time (class 1) for single-row and multi-row fetch

These measurements show:

- For single-row fetch there is a 5% increase in CPU time when we compare the measurements in V7 to V8. This increase is primarily due to the overhead when operating in 64-bit addressing as compared to 31-bit addressing and normal release to release overhead.
- 40% CPU time improvement with MR=10 rows
- 50% CPU time improvement with MR=100+ rows

The performance improvement using multi-row fetch in general depends on:

- Number of rows fetched in one fetch
- Number of columns fetched (more improvement with fewer columns), data type and size of the columns

- Complexity of the fetch. The fixed overhead saved for not having to go between the database engine and the application program has a lower percentage impact for complex SQL that has longer path lengths.

If the multi-row fetch reads more rows per statement, it results in CPU time improvement, but after 10 to 100 rows per multi-row fetch, the benefit is decreased. The benefit decreases because, if the cost of one API overhead per row is 100% in a single row statement, it gets divided by the number of rows processed in one SQL statement. So it becomes 10% with 10 rows, 1% with 100 rows, 0.1% for 1000 rows, and then the benefit becomes negligible.

It turns out that the percentage improvement is larger when class 2 accounting is turned on, because the cost of class 2 accounting which was encountered for each row fetched now is encountered only for each multi-row fetch SQL statement. So the cost of class 2 accounting would be dramatically reduced, for example, 100 times if fetching 100 rows in one multi-row fetch SQL statement.

Multi-row insert performance

Figure 3-5 shows the CPU time (class 1) measurements for single-row insert and multi-row insert of 100,000 rows into a non-partitioned table space.

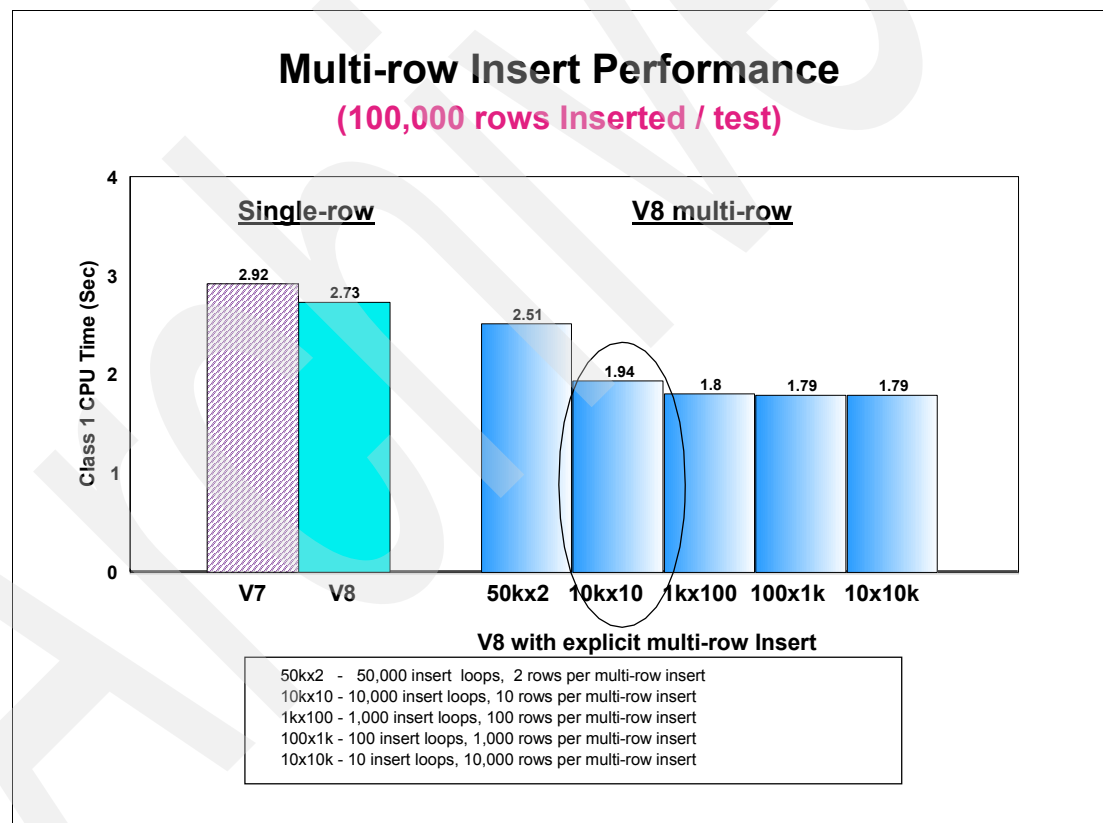


Figure 3-5 CPU time (class 1) for single-row and multi-row insert

The measurements, compared to V7, show:

- For single-row fetch there is a reduction in CPU time when we compare the measurements in V7 and V8 and insert requests.
- 35% CPU time improvement with MR=10 rows
- 40% CPU time improvement with MR=100+ rows

The performance improvement with multi-row insert is not as high as with multi-row fetch since a row insert is more complex and costly than a row fetch.

The performance improvement using multi-row insert depends on:

- ▶ Number of rows inserted in one insert SQL statement
- ▶ Number of columns inserted (more improvement with fewer columns), data type and size of columns.
- ▶ Number of indexes on the table
- ▶ For single-row insert, the measured CPU cost in V8 is the same as V7.

Multi-row fetch/update performance

Multi-row fetch/update measurements, where the whole rowset was updated, show a behavior similar to multi-row fetch. The measurement values are in Table 3-1.

- ▶ Measurements, compared to V7, show
 - 25% CPU time improvement with MR=10 rows
 - 40% CPU time improvement with MR=100+ rows

The performance improvement is not as high as MR fetch since a row cursor update is more complex and costly than a row fetch.

- ▶ Performance improvement depends on
 - Number of rows fetched/updated in one SQL statement
 - Number of columns fetched/updated
 - Complexity of the fetch/update SQL statement
 - Number of indexes updated

Multi-row fetch/delete performance

Multi-row fetch/delete measurements show a behavior similar to multi-row fetch. The measurement values are in Table 3-1.

Measurements, compared to V7, show

- ▶ 25% CPU time improvement with MR=10 rows
- ▶ 35% CPU time improvement with MR=100+ rows

Performance improvement is not as high as MR fetch since a row cursor delete is more complex and costly than a row fetch.

The performance improvement depends on:

- ▶ Number of rows fetched/deleted in one SQL statement
- ▶ Number of columns fetched
- ▶ Complexity of the fetch/delete SQL statement
- ▶ Number of indexes on the table

3.1.5 Conclusion

Multi-row operations allow DB2 V8 to reduce the traffic between the application program and DB2 when compared to single-row operations. The measurements show better performance improvement in the case of statements with:

- ▶ Less complex SQL and data types
- ▶ Fewer columns processed

- ▶ More rows per one SQL statement (10 rows per multi-row statement results in significant improvement)
- ▶ Fewer indexes processed in table

Table 3-1 provides a summary of the measurements discussed for multi-row operations processing 100,000 rows in a non-partitioned table space with a 100,000 row table, 26 columns, and 1 index.

Table 3-1 Class 1 CPU time (sec.) to process 100,000 rows

Type of MR operation loopsxrows per SQL	FETCH	INSERT	FETCH/UPDATE	FETCH/DELETE
V7 single-row	1.09	2.92	2.5	3.68
V8 single-row	1.15	2.73	2.79	3.81
50kx2	1.02	2.51	2.8	3.71
10kx10	0.64	1.94	1.88	2.76
1kx100	0.55	1.8	1.57	2.42
100x1k	0.54	1.79	1.51	2.37
10x10k	0.54	1.79	1.51	2.35

In general, for single-row processing DB2 V8 uses more CPU time, with the exception of the insert operation.

3.1.6 Recommendation

Explicit multi-row FETCH, INSERT, cursor UPDATE, and cursor DELETE in local environments can improve performance with a reduction of CPU time (between 25% and 40% in the measurements processing 10 rows per SQL statement) if a program processes a considerable number of rows. Depending on each particular case, a number between 10 rows and 100 rows per SQL can be a good starting point for multi-row operations.

This option requires definition of host variable array declaration of the cursor with rowset positioning and the text of the SQL to specify the rowset size.

You also have to change your coding on how to do cursor processing. You have to look at how many rows are returned on a FETCH via SQERRD3 flag in the SQLCA (or the equivalent ROW_COUNT when using GET DIAGNOSTICS).

For existing programs it is worth evaluating the benefits provided vs. the cost of recoding the programs. The extra storage needed for multi-row operations is allocated in the allied address space, or stored procedure, or DDF.

A variable array has to be explicitly declared. Example 3-2, extracted from the *DB2 Application Programming and SQL Guide*, SC18-7415, shows declarations of a fixed-length character array and a varying-length character array in COBOL.

Example 3-2 COBOL array declaration

```
01 OUTPUT-VARS.
   05 NAME OCCURS 10 TIMES.
       49 NAME-LEN PIC S9(4) COMP-4 SYNC.
       49 NAME-DATA PIC X(40).
```

Tip: Verify that the PTFs for APARs PQ91898, PQ93620, PQ89181, and PQ87969 are applied:

- ▶ PTF UQ92546 for performance APAR PQ91898 reduces excessive lock and getpage requests in multi-row insert.
- ▶ PTF UQ96567 for performance APAR PQ93620 reduces CPU overhead during multi-row cursor delete.
- ▶ PTFs UQ90464 and UQ92692, respectively for APAR PQ89181 and PQ87969, reduce CPU usage by providing improved lock management.

PTF UK06848 for APAR PQ99482 provides QMF for TSO/CICS 8.1 with the optional support of multi-row fetch and insert.

3.2 Materialized query table

For many years, the optimization for decision-support queries has been a difficult task, because such queries typically operate over huge amounts of data (1 to 10 terabytes), performing multiple joins and complex aggregation.

When working with the performance of a long running query executed via dynamic SQL, especially in Data Warehousing, involving joins of many tables, DBAs often created summary tables manually in order to:

- ▶ Improve the response time
- ▶ Avoid the redundant work of scanning (sometimes in the DB2 work file), aggregation and joins of the detailed base tables (history)
- ▶ Simplify SQL to be coded

However, you need to be aware of the existence of the DBA's summary tables and make a decision whether to use them or the base tables depending on the query. Another issue is setting up the synchronization of these summary tables with base tables.

A materialized query table (MQT) is a table containing materialized data derived from one or more source tables specified by a fullselect to satisfy long running queries requiring good response time (minutes or even seconds).

The characteristics of MQTs are:

- ▶ Source tables for MQTs can be base tables, views, table expressions or user-defined table expressions.
- ▶ MQTs can be chosen by the optimizer to satisfy dynamic query (through automatic query rewrite) when a base table or view is referenced. No query rewrite is done for static SQL, you need to address the MQT directly.
- ▶ MQTs are used to precompute once expensive joins, sorts or aggregations. They can be reused many times to improve query performance.
- ▶ MQTs can be accessed directly by static and dynamic SQL.

Materialized query tables are also known as *automatic summary tables* in other platforms.

MQTs compared to views

- ▶ MQTs occupy storage space
- ▶ MQTs are not usually referenced in a user query, but typically referenced via query rewrite by DB2.

MQTs compared to indexes

- ▶ MQTs can be associated with multiple tables. An index can be created for one table only.
- ▶ Exploitation of MQTs and indexes is transparent to the applications.

3.2.1 Creating MQTs

We use the extended CREATE TABLE SQL statement to define an MQT and we:

- ▶ Specify a fullselect associated with a table
- ▶ Specify the mechanisms that are used to refresh the MQT
- ▶ Enable/disable an MQT for automatic query rewrite

The privilege set needed to define the MQT must include at least one of the following:

- ▶ The CREATETAB privilege for the database, implicitly or explicitly specified by the IN clause
- ▶ DBADM, DBCTRL, or DBMAINT authority for the database in which the table is being created
- ▶ SYSADM or SYSCTRL authority

See Example 3-3 for the DDL of a materialized query table.

Example 3-3 Define a table as MQT

```
CREATE TABLE TRANSCNT (ACCTID, LOCID, YEAR, CNT) AS (  
    SELECT ACCTID, LOCID, YEAR, COUNT(*)  
    FROM TRANS  
    GROUP BY ACCTID, LOCID, YEAR )  
DATA INITIALLY DEFERRED  
REFRESH DEFERRED  
MAINTAINED BY SYSTEM  
ENABLE QUERY OPTIMIZATION;
```

Options creating MQTs

The options are:

- ▶ DATA INITIALLY DEFERRED
 - The MQTs are not populated at creation
 - Use REFRESH TABLE to populate a system-maintained MQT
 - Use the INSERT statement to insert data into a user-maintained MQT or use LOAD
- ▶ REFRESH DEFERRED
 - Data in the MQT is not refreshed when its base tables are updated
 - Can be refreshed any time using the REFRESH TABLE statement
- ▶ MAINTAINED BY SYSTEM/USER
- ▶ ENABLE /DISABLE QUERY OPTIMIZATION

Referential constraints between base tables are also an important factor in determining whether a materialized query table can be used for a query.

For instance, in a data warehouse environment, data is usually extracted from other sources, transformed, and loaded into data warehouse tables. In this case the referential integrity constraints can be maintained and enforced by means other than the database manager to avoid the serialization and overhead of enforcing them by DB2.

ALTER a base table to MQT

We use the extended **ALTER TABLE** SQL statement to change an existing base table into an MQT. With this statement we:

- ▶ Specify a fullselect associated with a table
- ▶ Specify one of the mechanisms to refresh the table
- ▶ Enable/disable an MQT for automatic query rewrite
- ▶ Switch between system-maintained and user-maintained types

See Example 3-4.

Example 3-4 Alter a table to MQT

```
ALTER TABLE TRANSCNT ADD MATERIALIZED QUERY AS (  
  SELECT ACCTID, LOCID, YEAR, COUNT(*) AS CNT  
  FROM TRANS  
  GROUP BY ACCTID, LOCID, YEAR )  
DATA INITIALLY DEFERRED  
REFRESH DEFERRED  
MAINTAINED BY USER;
```

3.2.2 Populating MQTs

We have mentioned that an MQT can be maintained by system or user.

- ▶ System-maintained MQTs (default) are populated with the new **REFRESH TABLE** option.

This option:

- Deletes all the rows in the MQT
- Executes the fullselect associated with the MQT to recalculate the data
- Inserts the calculated result into the MQT
- Updates the catalog for the refresh timestamp and cardinality of the MQT
- Cannot be updated by load, insert, update and delete

You need to refresh materialized query tables periodically to maintain data currency with base tables. However, realize that refreshing materialized query tables can be an expensive process.

- ▶ User-maintained MQTs can be populated by:
 - The **REFRESH TABLE** option
 - Load, insert, update and delete

3.2.3 Controlling query rewrite

Query rewrite is the process that can use an MQT if the query being optimized by DB2 has common source tables and all the predicates of the fullselect that make up the MQT.

DB2 uses two special registers for dynamic SQL to control query rewrite:

- ▶ **CURRENT REFRESH AGE**
- ▶ **CURRENT MAINTAINED TABLE TYPES**

Special register CURRENT REFRESH AGE

The value in this special register represents a refresh age. The refresh age of an MQT is the timestamp duration between the current timestamp and the timestamp of the last REFRESH TABLE statement for the MQT.

This special register controls whether an MQT can be considered in query rewrite as follows:

- ▶ Value “0” means no MQTs are considered in query rewrite
- ▶ Value “ANY” means all MQTs are considered in rewrite

The default value of this special register can be specified in the CURRENT REFRESH AGE field on panel DSNTIP4 at installation time.

The data type for CURRENT REFRESH AGE is DEC(20,6).

Note: The default value disables query rewrite. Add *SET CURRENT REFRESH AGE ANY* to make sure of the use of MQT. You can verify this with Explain.

Setting the CURRENT REFRESH AGE special register to a value other than zero should be done with caution. Allowing a materialized query table that may not represent the values of the underlying base table to be used to optimize the processing of a query may produce results that do not accurately represent the data in the underlying table.

Note: You need to add the new DSNZPARM with REFSHAGE=ANY to enable query rewrite *as a default*.

Special register CURRENT MAINTAINED TABLE TYPES

This special register specifies a VARCHAR(255) value. The name of the dynamic ZPARM is MAINTYPE. The value identifies the types of MQT that can be considered in query rewrite:

- ▶ Value 'SYSTEM' means all system-maintained, query optimization enabled MQTs
- ▶ Value 'USER' means all user-maintained, query optimization enabled MQTs
- ▶ Value 'ALL' means all query optimization enabled MQTs

The initial value of this special register is determined by the value of field CURRENT MAINT TYPES on installation panel DSNTIP6. The default of this field is SYSTEM.

Figure 3-6 shows the relationship between the two special registers for MQTs.

		CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION			
		SYSTEM	USER	ALL	NONE
CURRENT REFRESH AGE	ANY	All system-maintained query optimization enabled MQTs	All user-maintained query optimization enabled MQTs	All query optimization enabled MQTs	None
	0	None	None	None	None
default					

Figure 3-6 Relationship between two special registers for MQTs

Rules for query rewrite

If the query references tables that are also referenced in MQTs, the optimizer can decide to select the qualified MQT that gives the best performance for the query.

Figure 3-7 summarizes the checking that is performed for a query, if automatic query rewrite is generally enabled in your DB2 subsystem at installation time.

The automatic query rewrite is supported for dynamically prepared queries that are read-only and have some other restrictions.

The automatic query rewrite is a general process that can use a materialized query table M if the fullselect F that makes up the MQT has common source tables as the submitted query Q that is being optimized by DB2.

Other conditions are checked during *table mapping* to verify if the result of the submitted query can be derived from or can directly use the result of one or more materialized query tables.

During a *predicate matching* process, the predicates in the materialized query table fullselect are compared, one by one, to the predicates in the query Q.

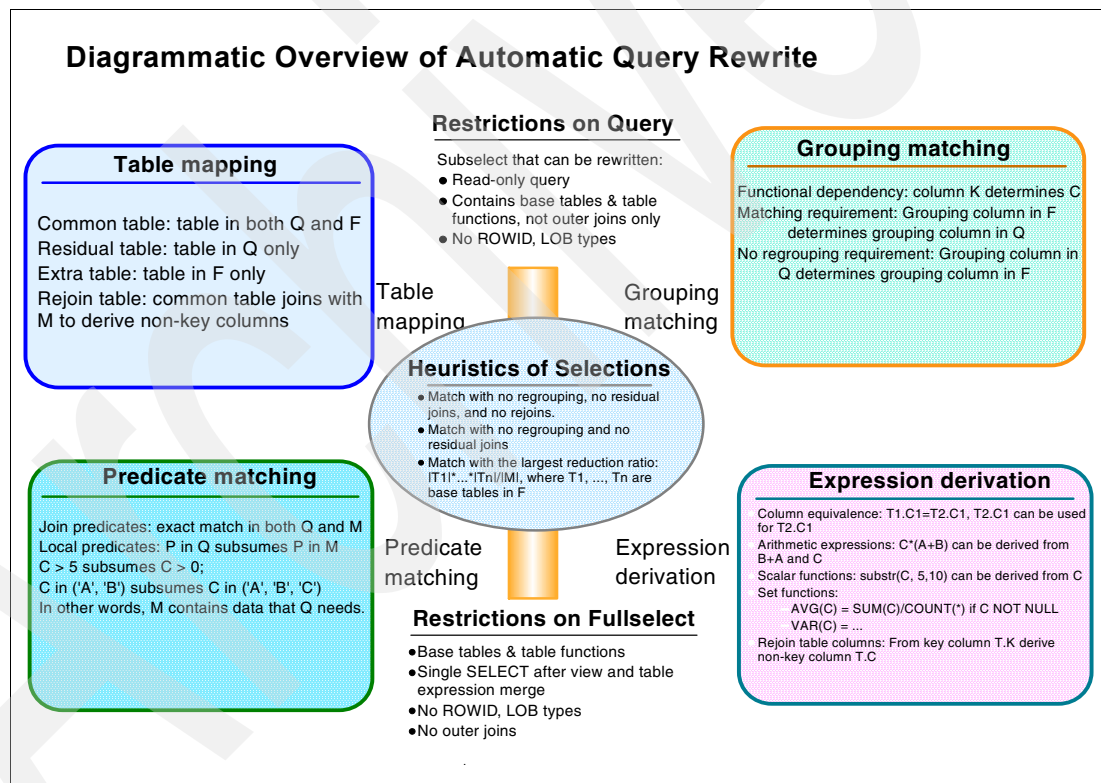


Figure 3-7 Rules for query rewrite

If there are any predicates that are in the materialized query table subselect, but are not in the query, then it can be assumed that these predicates may have resulted in discarded rows as the materialized query table was refreshed, and that any rewritten query that makes use of the materialized query table does not give the correct results.

The GROUP BY clauses, if any, are compared during *grouping matching*, to determine if a query GROUP BY clause results in a subset (not necessarily proper) of the rows that are in

the materialized query table. If it is, then the materialized query table remains a candidate for query rewrite.

If the query Q contains predicates that are not identical to predicates in the materialized query table subselect, they are examined to determine if column references to base table columns can derive values from materialized query table columns instead. This takes place during the *expression derivation* step.

If the contents of a materialized query table overlap with the contents of a query, the query and the materialized query table are said to match.

If there are multiple materialized query tables that match the query but cannot be used simultaneously, *heuristic* rules are used to select one of them. If the query rewrite process is successful, DB2 determines the cost and access path of the new query. Only if the cost of the new query is less than the cost of the original query is the new query substituted.

Query rewrite example

A simple example of query rewrite is depicted in Figure 3-8. If MQT1 is selected by automatic query rewrite for query RefQ, this means that:

- ▶ MQT1 is enabled for query optimization by CURRENT REFRESH AGE and CURRENT MAINTAINED TABLE TYPES
- ▶ MQT1 is populated as discussed before
- ▶ Query optimization is enabled
- ▶ Tables referenced in the query RefQ match (same tables) the source tables in MQT1
- ▶ The predicates in MQT1 match (same predicates) with predicates in the query RefQ
- ▶ The GROUP BY clause in the query RefQ is a subset (same GROUP BY) of the rows that are in MQT.

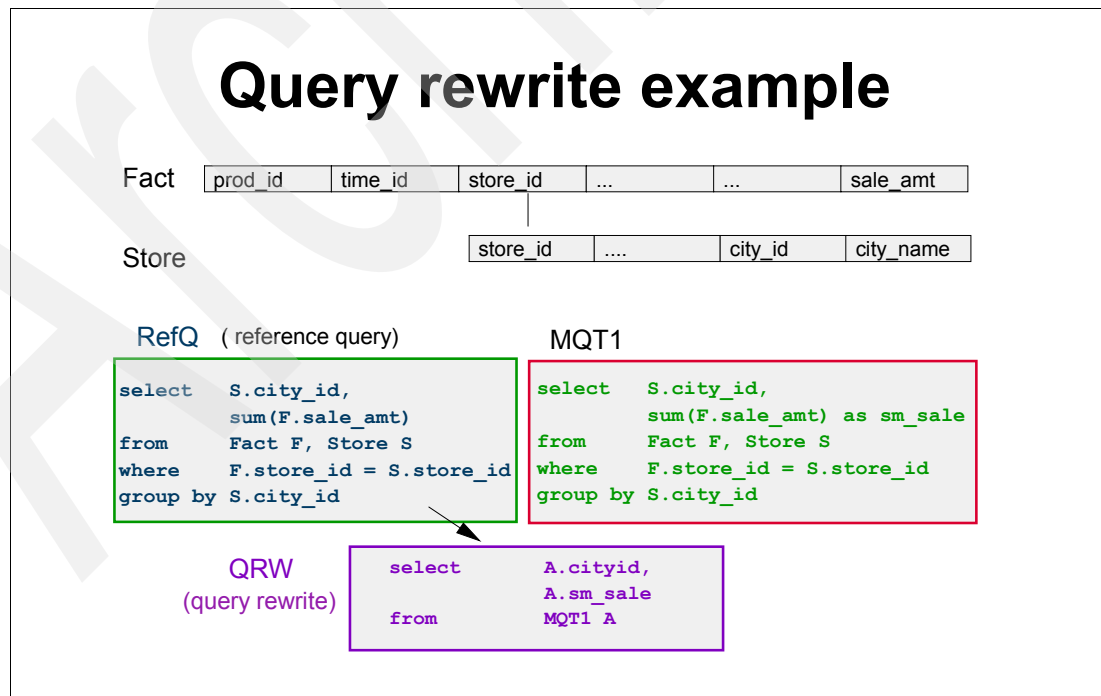


Figure 3-8 Query rewrite example

The SQL statement used by query rewrite is simpler and has a lower cost than running on the source tables because all calculations have been done when MQT was populated.

If the final query plan comes from a rewritten query:

- ▶ PLAN_TABLE shows the MQT used and its access path
- ▶ TABLE_TYPE column shows the value 'M' for an MQT

Informational referential constraints are introduced to allow users to declare a referential constraint, avoiding the overhead of enforcing the referential constraints by DB2. This allows DB2 to take advantage of the information provided by the referential constraints (data associations) in query rewrite and some utilities.

3.2.4 Performance

We now show some performance measurements using the functions supported in query rewrite using MQTs.

Performance evaluation of MQTs

The evaluation of performance of MQTs shows:

- ▶ MQTs are not used for short running queries.
- ▶ Query rewrite time grows reasonably when the number of eligible MQTs increases.
- ▶ Basic query rewrite works. Results can be obtained directly from MQTs or derived from MQTs.
- ▶ Query rewrite can select the best MQT from a set of qualified MQTs.

The environment for MQT measurements

- ▶ DB2 for z/OS V7 and V8 NFM
- ▶ z/OS Release 1.4.0
- ▶ z900 Turbo with 3 CPs
- ▶ ESS 800 DASD
- ▶ Three workloads with different profiles

MQTs and short running queries

Figure 3-9 shows the CPU time for a short running query.

MQTs and short running queries

- CPU time for SELECT COUNT(*) from tables with various numbers of 4K pages
- MQTs are not used for short running query scanning 815 page table (0.019s CPU time)
- MQTs are used for query scanning 1032 page table (0.015s cpu, 0.037s w/o MQT)

CPU Time for SELECT COUNT(*)

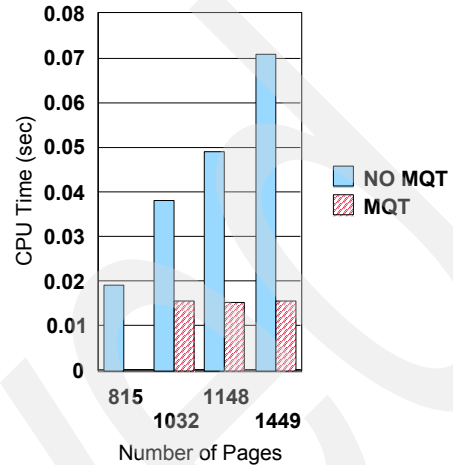


Figure 3-9 MQTs and short running queries

Observations on MQTs and short running queries:

- Create several MQTs which scan various numbers of 4 KB pages to count the number of rows in the table
- Enable/disable MQTs for query rewrite for queries in MQT fullselect
- Compare query CPU time with and without MQT
- MQTs are not used for short running query scanning 815 page table (0.019 sec. CPU time)
- MQTs are used for query scanning 1032 page table (0.015 sec. CPU using MQT, 0.037 sec. without MQT)

If the estimated cost for a given query is below the DB2 threshold, the automatic query rewrite is not attempted. The value is similar to the threshold to prevent parallelism for short running queries.

Increase of bind cost

Figure 3-10 shows the overhead introduced by the number of MQTs qualified in BIND cost.

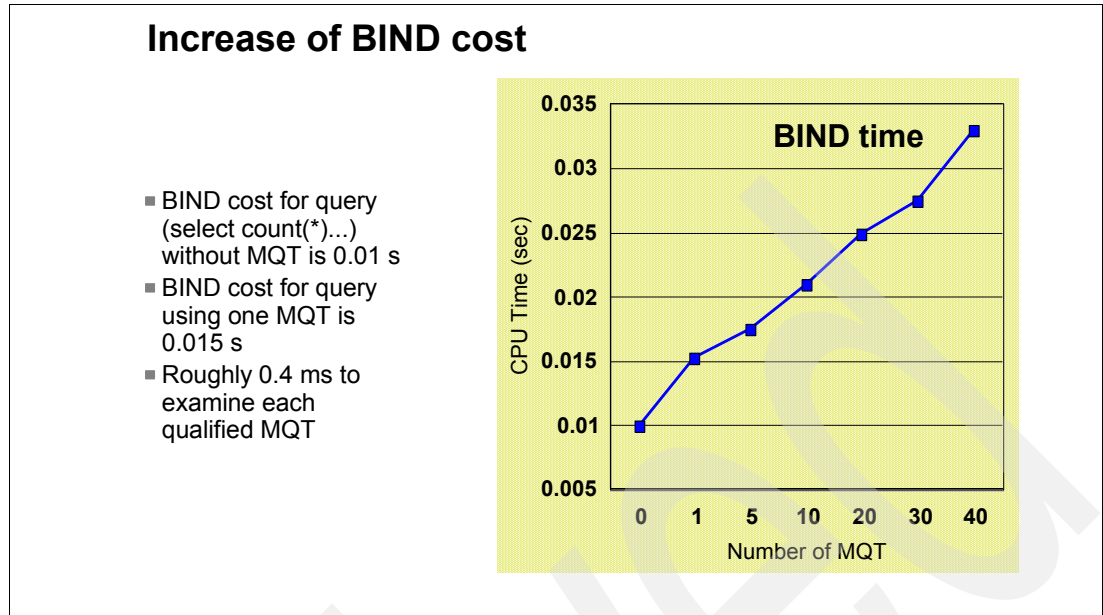


Figure 3-10 Increase of BIND cost

We have run Explain for a simple query with varying number of MQTs (0, 1, 5, 10, 20, 30, and 40) eligible for query rewrite. All MQTs are created the same except for the literal value in predicates. All MQTs are evaluated as possible candidates for query rewrite.

We see that bind cost in CPU increases almost linearly as more and more MQTs are eligible for query rewrite.

Query rewrite for a simple query

Figure 3-11 reports the performance of a query rewrite for a simple query.

This query selects the count of the number of rows grouped by column BRN for those values of BRNs that have more than 50 rows. In V7, DB2 has to scan all rows from the source table (one million rows). The plan table indicates a table space scan for table EVE. DB2 V7 could decrease the elapsed time if parallelism is enabled. With parallelism the elapsed time is smaller but there is a small increase in CPU time due to the parallelism overhead.

When executing the same query in V8, the automatic query rewrite is done by the optimizer selecting the MQT named MQT091. This fact is reflected in the plan table showing a table scan in a smaller table space.

The summary of measurements of query rewrite for a simple query:

- ▶ V7 DEGREE(1)
 - 24.77 sec. elapsed time and 4.06 sec. CPU time
- ▶ V7 DEGREE(ANY)
 - 6.11 sec. elapsed time and 4.19 sec. CPU time
- ▶ V8 using MQT
 - 1.1 sec. elapsed time and 0.06 sec. CPU time

The measurements show that:

- The elapsed time is 22 times smaller (about 95%) and CPU time 68 times smaller (about 98%) when we compare V7 DEGREE(1) with V8 using the MQT.

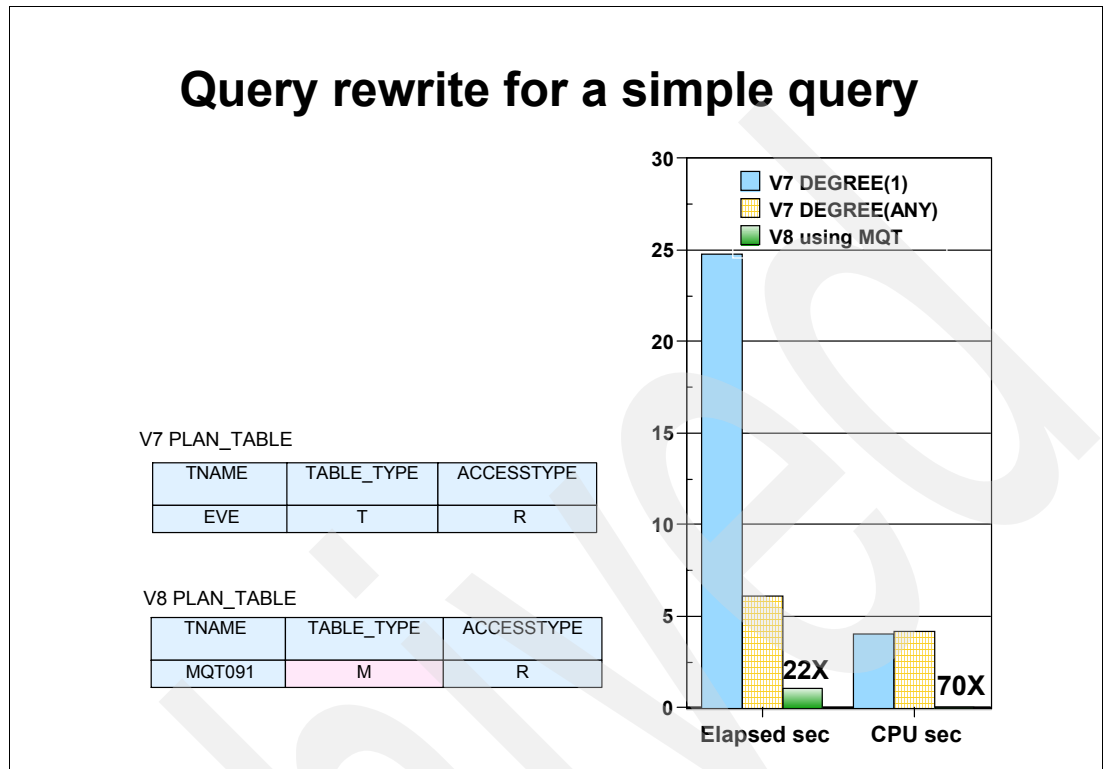


Figure 3-11 Query rewrite for simple query

Query rewrite for join

Figure 3-12 reports the performance of a query rewrite for a join.

The query rewrite for join is similar to the simple query. The plan table for V7 shows the join of the three tables (633k, 1.5 M and 1.3 M rows) using nested loop join. The execution in DB2 V8 uses the MQT201 that reads 171 rows with a large reduction in the elapsed and CPU time. Summary of the measurements of query rewrite for join:

- V7 DEGREE(1)
28.02 sec. elapsed, 16.52 sec. CPU
- V7 DEGREE(ANY)
14.25 sec. elapsed, 17.9 sec. CPU
- V8 using MQT
0.9 sec. elapsed, 0.07 sec. CPU

In this example, there is a reduction of 31 times (about 96%) in elapsed time and 236 times (about 99%) in CPU time when we compare V7 DEGREE(1) with V8 using the MQT.

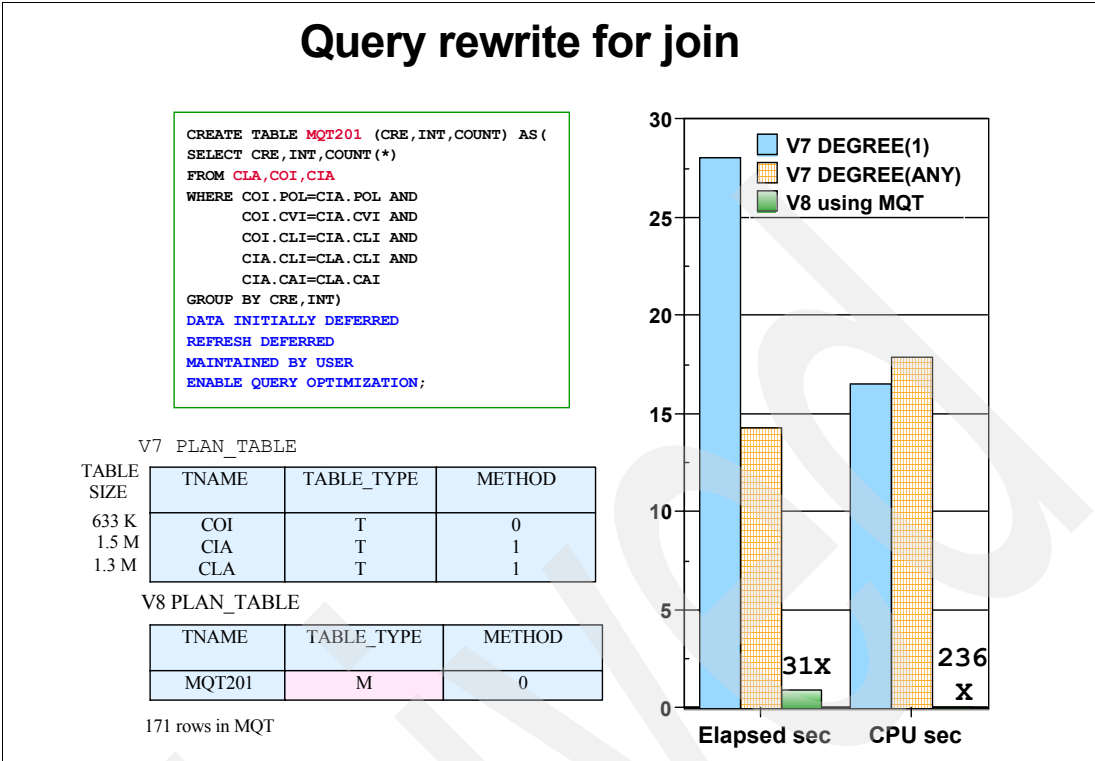


Figure 3-12 Query rewrite for join

GROUP BY and predicate matching

The MQT remains a candidate for query rewrite if the query GROUP BY clause results in a subset of the rows that are in the MQT

The predicate in the query subsumes the predicate in the MQT. Example: $C1 > 8$ subsumes $C1 > 0$.

The predicates in the query should be coded as those in the MQT subselect. Otherwise, matching may fail on some complex predicates.

Figure 3-13 shows the impact of regrouping the MQT data when DB2 obtains the final answer set in the query.

Re-grouping on MQT results

```
CREATE TABLE MQT661 (BRN,PST,COUNT,SUM) AS (
SELECT BRN,PST,COUNT(DISTINCT PLC.POL),
SUM(COV.OFC)
FROM COV, PLC
WHERE COV.POL=PLC.POL
AND COV.CVT= 'B'
GROUP BY BRN,PST)
DATA INITIALLY DEFERRED
REFRESH DEFERRED
MAINTAINED BY USER
ENABLE QUERY OPTIMIZATION;
```

V7 PLAN_TABLE

TABLE SIZE	TNAME	TABLE_TYPE	METHOD
1.7 M	COV	T	0
624 K	PLC	T	1

V8 PLAN_TABLE

TNAME	TABLE_TYPE	METHOD
MQT661	M	0

1808 rows in MQT

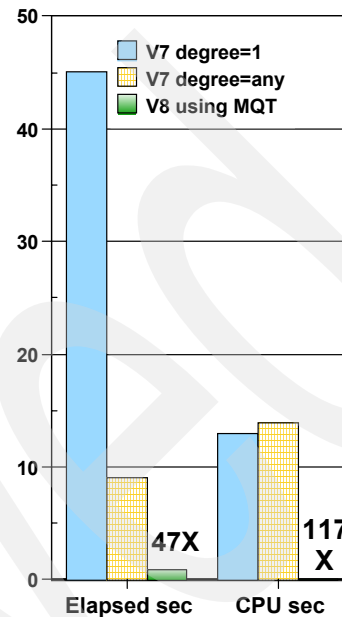


Figure 3-13 Regrouping on MQT result

In this example the populated MQT661 has 1808 rows as the result of joining two tables and grouping by **BRN, PST**. Explain was executed for query number 662 as shown in Example 3-5.

Example 3-5 Explain for Q662

```
** EXPLAIN ALL SET QUERYNO=662 FOR
SELECT BRN, SUM(OFC),COUNT(DISTINCT PLC.POL),
COUNT(COV.OFC)
FROM COV,PLC
WHERE
COV.POL = PLC.POL AND COV.CVT='B'
GROUP BY BRN
ORDER BY BRN;
```

The PLAN_TABLE for V8 in Figure 3-13 shows that the DB2 Optimizer has selected the MQT. The result is a reduction in elapsed time from 45.1 sec. to 0.92 sec. We see a significant reduction even if we compare it with V7 with parallelism.

The new access path using MQT regroups 1808 rows avoiding the nested loop join of one table having 1.7 million rows with a table with 624 thousand rows.

Summary of measurements of a query rewrite for simple query:

- V7 DEGREE(1)
 - 45.1 sec. elapsed time and 12.92 sec. CPU time
- V7 DEGREE(ANY)
 - 9.04 sec. elapsed time and 13.87 sec. CPU time

► V8 using MQT

0.92 sec. elapsed time and 0.11 sec. CPU time

The measurements show that the elapsed time was 49 times smaller (about 97%) and CPU time 117 times smaller (about 99%) when we compare V7 DEGREE(1) with V8 using an MQT.

Expression derivation

- Arithmetic expressions: $(A+B)*C$ can be derived from $B+A$ and C
- Column equivalence: If $T1.C1=T2.C1$, $T2.C1$ can be used for $T1.C1$
- Scalar functions: $SUBSTR(A,3,5)$ can be derived from A
- Set functions: $AVG(C)$ can be derived from $SUM(C)$ and $COUNT(*)$ if C NOT NULL

Figure 3-14 shows one example of expression derivation using an MQT.

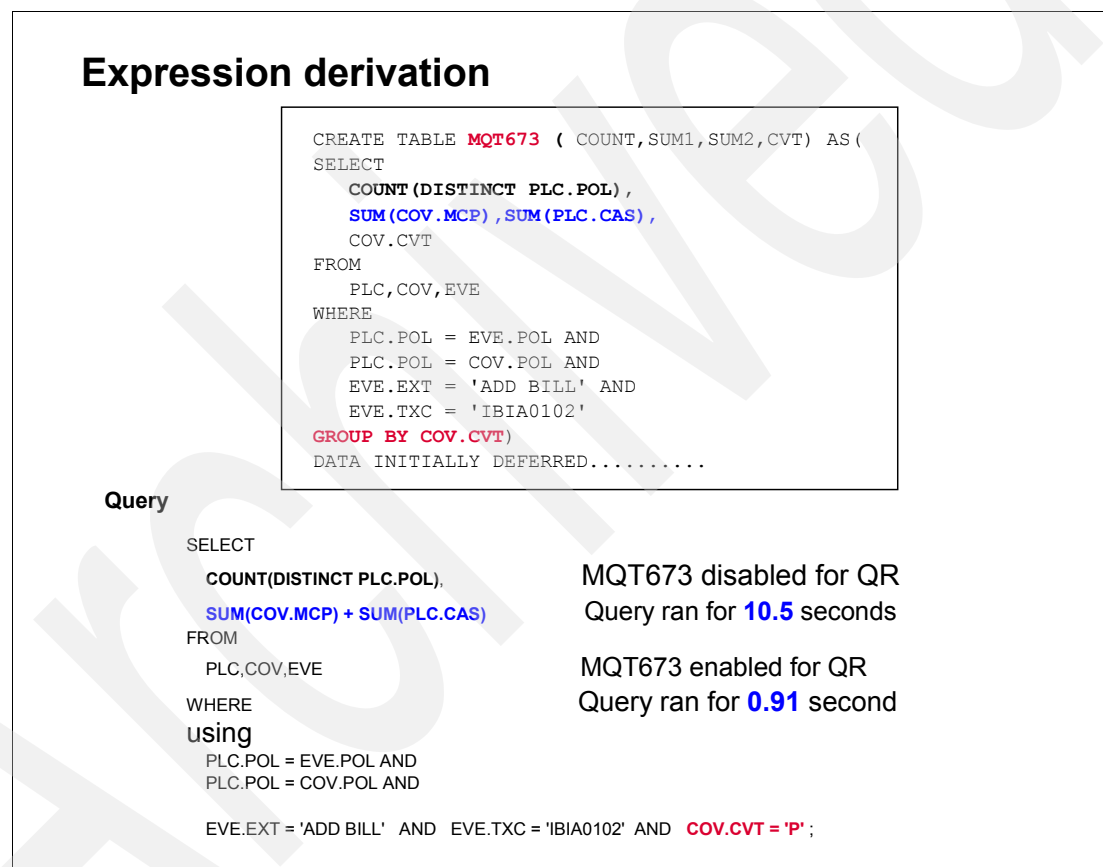


Figure 3-14 Expression@ derivation example

In the example the populated MQT673 is the result of joining three tables and two aggregations:

$SUM(COV.MCP), SUM(PLC.CAS)$

The measured query selects

$SUM(COV.MCP) + SUM(PLC.CAS)$

This can be derived from the two aggregated columns from MQT.

The predicates

EVE.EXT = 'ADD BILL' AND EVE.TXC = 'IBIA0102'

In the query match with the predicates

EVE.EXT = 'ADD BILL' AND EVE.TXC = 'IBIA0102'

In the MQT due to join predicates.

The predicate

COV.CVT = 'P'

filters to one aggregation in the MQT because of GROUP BY COV.CVT.

The measurements show a performance improvement in elapsed time from 10.5 sec. to 0.91 sec. (about 91%) when using the MQT.

Note that an expression like $(A+B)*C$ cannot be derived from $A*C+B*C$.

MQT considerations

- ▶ All MQTs and their indexes are dropped if their associated base tables are dropped.
- ▶ No primary keys, unique indexes or triggers are allowed on MQTs.
- ▶ Design issues between a few generic MQTs and many more specialized MQTs:
 - Trade off between query performance and MQT maintenance.
 - Identify a set of queries which consume a lot of resource. Design MQTs for the set of queries as a starting point. Example: Top queries that consume a large amount of resources.
- ▶ Automatic query rewrite is done at query block level by the DB2 optimizer.
- ▶ MQTs are not used for short running queries.
- ▶ Query rewrite works for both star join and non-star join queries.
- ▶ Code the predicates in the query in exactly the same way as they are in the MQT fullselect.

Exception: The order of the items in the IN list predicate does not need to be in exactly the same order.

3.2.5 Conclusions

MQTs can be created as a precomputed join, sort or aggregation, and reused by DB2 query rewrite in dynamic SQL to improve the execution of long running queries in the Data Warehouse environment, providing largely reduced elapsed time and CPU time.

- ▶ Query rewrite time seems reasonable for simple queries.
- ▶ Query rewrite time grows reasonably when the number of eligible MQTs increases.
- ▶ Query rewrite works. The query result can be obtained directly from the MQT or derived from the MQT.
- ▶ Query rewrite can select the best MQT from a set of qualified MQTs.
- ▶ Up to 100 times query performance improvement is observed; simple examples with tables that are not so large, show improvement of 22x, 31x, and 47x in elapsed time and 70x, 236x and 117x in CPU time.

The most important factor in the return on investment on MQTs remains the frequency of utilization, and, therefore, a balance of how specialized the MQT is and how well it performs versus the range of applicability.

3.2.6 Recommendations

MQTs allow more flexibility for the physical data base design of tables at the detailed level (sometimes known as *atomic level*) in a corporate data warehouse. The base tables can be more normalized, closer to the conceptual model, and the performance issues of long running queries involving joins, summarizations, and subqueries can be resolved when materialized as MQTs.

Plan MQTs for the top consuming queries, not for all queries. Use MQTs to aggregate results based on the most used predicates joining large tables and allow the DB2 optimizer to use regrouping, predicate matching and expression derivation.

Multi-dimensional applications can gain benefits by using MQTs, when queries have opportunities for regrouping.

If you can identify these cases, the same MQT can be reused in many different queries. A good case is when a new table in the DB2 work file as a result of join is scanned a large number of times in a nested loop join: MQTs can be indexed.

The use of MQTs can potentially provide huge savings in a data warehouse environment. It is good for queries, but not for OLTP due to the extra cost for BIND.

For large MQTs, remember to:

- ▶ Use segmented table spaces because of the almost instantaneous mass delete in REFRESH TABLE.
- ▶ Execute RUNSTATS after REFRESH for good access path selection; otherwise, DB2 uses default or out-of-date statistics.

Keep the number of MQTs for a base table down to an acceptable number to avoid the extra cost of BIND time due to the large number of MQTs eligible for query rewrite, as well as the processing necessary for the maintenance of MQTs. For a simple query the measured overhead is about 0.4 ms to examine each qualified MQT.

When creating a user-maintained materialized query table, initially disable query optimization. Otherwise, DB2 might automatically rewrite queries to use the empty materialized query table.

3.3 Star join processing enhancements

In this section we describe how, in DB2 V8, the execution of star join is improved by:

- ▶ Access path enhancements
- ▶ Better estimates of the filtering effect of dimensions
- ▶ Use of sparse indexes in cache for materialized snowflakes
- ▶ Use of in-memory work files

3.3.1 Star schema

The star schema data model is often used in data warehouses, data marts and OLAP. The representation of a star schema is shown in Figure 3-15.

Star Schema

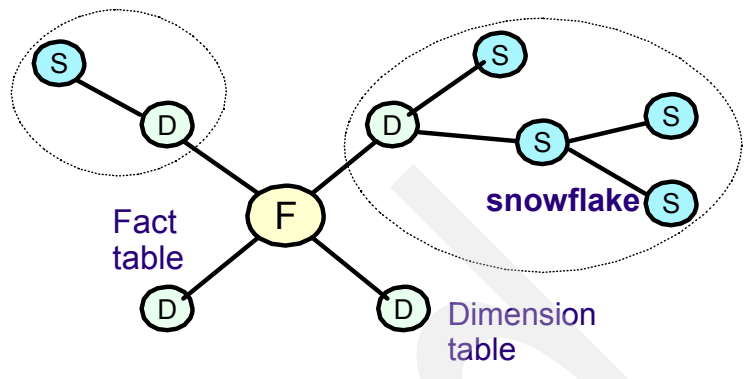


Figure 3-15 Star schema

The attributes of a star schema database are:

- ▶ Large fact table: Like sales tables containing transactions that can be in the order of hundreds of millions, or billions of data rows.
- ▶ Highly normalized design with dimensions (or snowflakes), tables to avoid maintaining redundant descriptive data in the central fact table.
- ▶ Relatively small dimensions: Dimensions can be denormalized to one table (without the tables identified as “S” in Figure 3-15) or normalized in related tables in the form of a snowflake.
- ▶ Sparse “Hyper Cube”: Caused by high correlation among dimensions, leading to a sparse nature of data in the fact table; for example, product sales are dependent on the climate, therefore, the sale of shorts is more likely in a state where the people enjoy hot weather.
- ▶ Fact table is dependent on the dimension tables.

Star schema query

Star schema query is normally a star join with the following characteristics:

- ▶ Equi-join predicates between the fact and dimension tables
- ▶ Local predicates on the dimension tables
- ▶ Large number of tables in the query

For a star join query, DB2 uses a special join type called a *star join* if the following conditions are true:

- ▶ The tables meet the conditions that are specified in the section “*Star join (JOIN_TYPE='S')*” of the *DB2 UDB for z/OS Version 8 Administration Guide*, SC18-7413. Unlike the steps in the other join methods (nested loop join, merge scan join, and hybrid join) in which only two tables are joined in each step, a step in the star join method can involve three or more tables. Dimension tables are joined to the fact table via a multi-column index that is defined on the fact table. Therefore, having a well-defined, multi-column index on the fact table is critical for efficient star join processing.
- ▶ The STARJOIN system parameter is set to ENABLE, and the number of tables in the query block is greater than or equal to the minimum number that is specified in the SJTABLES system parameter.

Note: The default for star join processing is DISABLE.

- ▶ Another system parameter is the maximum pool size (MB) for star join SJMXPOOL.

Star join technical issues

The technical issues surrounding the optimization of decision support and data warehousing queries against a star schema model can be broken down into bind-time and run-time issues.

The first consideration generally is the sheer size of the fact table, and also the large number of tables that can be represented in the star schema.

► Bind-time issues

For non-star join queries, the optimizer uses the pair-wise join method to determine the join permutation order. However, the number of join permutations grows exponentially with the number of tables being joined. This results in an increase in bind time, because of the time required to evaluate the possible join permutations. Star join does not do pair-wise join. It joins several unrelated tables via a (simulated) cartesian join to the fact table.

► Run-time issues

Star join queries are also challenging at run-time. DB2 uses either cartesian joins of dimension tables (not based on join predicates, the result is all possible combinations of values in both tables) or pair-wise joins (based on join predicates).

The result is all possible combinations of values in both tables. The pair-wise joins have worked quite effectively in the OLTP world. However, in the case of star join, since the only table directly related to other tables is the fact table, the fact table is most likely chosen in the pair-wise join, with subsequent dimension tables joined based on cost.

Even though the intersection of all dimensions with the fact table can produce a small result, the predicates supplied to a single dimension table are typically insufficient to reduce the enormous number of fact table rows. For example, a single dimension join to the fact table may find:

- 100+ million sales transactions in the month of December
- 10+ million sales in San Jose stores
- 10+ million sales of Jeans, but
- Only thousands of rows that match all three criteria

Hence there is no one single dimension table which could be paired with the fact table as the first join pair to produce a manageable result set.

With these difficulties in mind, the criteria for star join can be outlined as follows:

► “Selectively” join from the outside-in:

Purely doing a cartesian join of all dimension tables before accessing the fact table may not be efficient if the dimension tables do not have filtering predicates applied, or there is no available index on the fact table to support all dimensions. The optimizer should be able to determine which dimension tables should be accessed before the fact table to provide the greatest level of filtering of fact table rows.

For this reason, outside-in processing is also called the *filtering* phase. DB2 V8 enhances the algorithms to do a better job at selecting which tables to join during outside-in processing.

► Efficient “Cartesian join” from the outside-in:

A physical cartesian join generates a large number of resultant rows based on the cross product of the unrelated dimensions. A more efficient cartesian type process is required as the number and size of the dimension tables increase, to avoid an exponential growth in storage requirements. A key feedback technique is useful for making cartesian joins efficient.

► Efficient “join back”, inside-out:

The join back to dimension tables that are accessed after the fact table must also be efficient. Non-indexed or materialized dimensions present a challenge for excessive sort

merge joins and work file usage. DB2 V8 enhancements introduce support for in-memory work files and sparse indexes for work files to meet this challenge.

► **Efficient access of the fact table:**

Due to the generation of arbitrary (or unrelated) key ranges from the cartesian process, the fact table must minimize unnecessary probes and provide the greatest level of matching index columns based on the pre-joined dimensions.

Of course, the very first step is determine if the query qualifies for a star join. You can refer to the white paper *The Evolution of Star Join Optimization* available from the Web site:

<http://www.ibm.com/software/data/db2/os390/techdocs/starjoin.pdf>

Star join performance characteristics

Star join characteristics are:

- Snowflakes are common for star join queries.
- Snowflakes are generally materialized.
- Snowflake work files could not have indexes defined on them before the implementation of sparse index in V7.
- Sort merge join is usually chosen to join the snowflake to the fact table composite; this can be relatively inefficient.
- Sorting a large intermediate result (composite table) including a join to the fact table (no index) is expensive.

The result of outside-in processing (the star join itself) is a composite table (the composite result of previous operations). It needs to be sorted in join column sequence if sort merge join is used (very likely) during inside out processing.

- Sorting the large intermediate result forces the parallel group to merge (less parallelism).

3.3.2 In memory work file

DB2 V8 supports in memory work files (IMWF) for star join queries. This means the normal work file database is not used. The complete work file is stored in memory instead. Figure 3-16 shows the utilization of IMWF to improve the performance of star join.

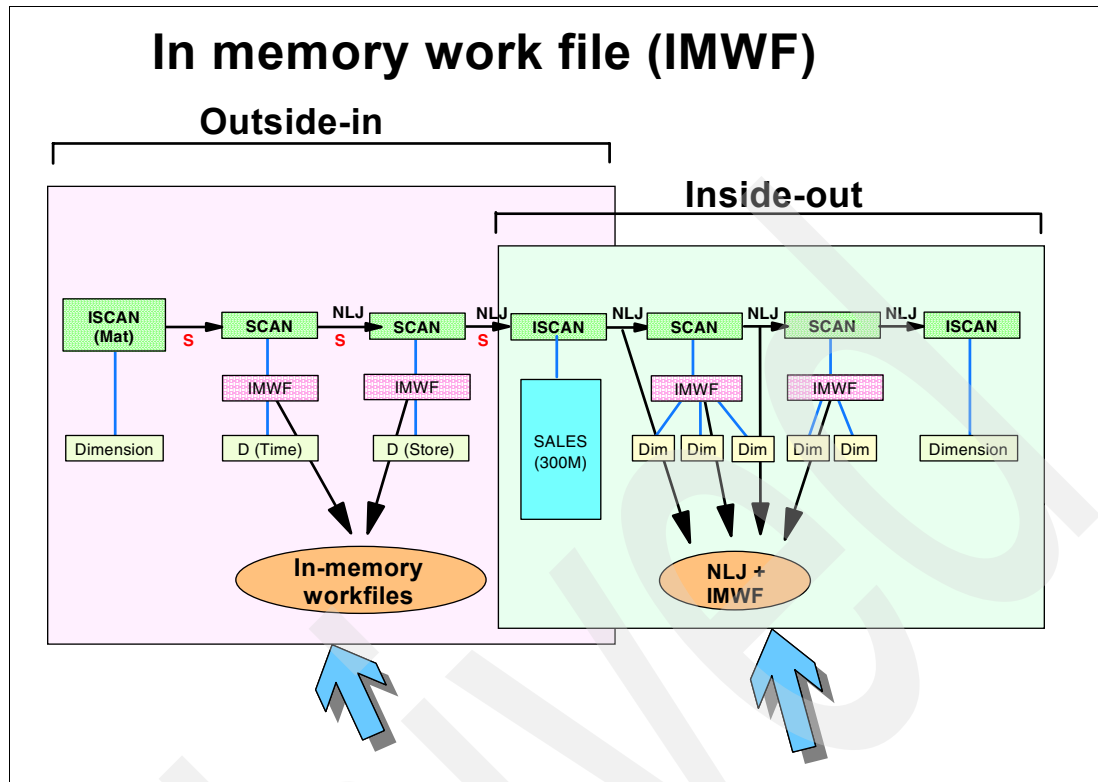


Figure 3-16 In memory work file - Description

The in memory work file is used for caching the data and index keys in a dedicated virtual memory pool. If IMWF fails (for whatever reason), sparse index on the work file is used.

IMWF characteristics:

- ▶ IMWF reduces contention on the work file buffer pool
- ▶ IMWF reduces CPU and work file I/O
- ▶ The memory pool is created only when star join is enabled
- ▶ Can specify the maximum pool size via DSNZPARM SJMXPOOL
- ▶ SJMXPOOL is changeable from 0 to 1024 MB.
 - 20 MB is the default
 - Memory pool is allocated at execution time above the bar
- ▶ Exact amount of required space is allocated
- ▶ The pool contains only the join column and the columns selected
- ▶ Binary search via in memory index keys
- ▶ If the dedicated pool cannot be allocated or the space in it becomes insufficient to store the data, the traditional sparse index is used; that is, a work file is created.

3.3.3 Sparse index

In this section we describe the implementation of a sparse index on the work file during star join execution. The difference between a normal and a sparse index is in the regular index there is one entry for each row in the table and in the sparse index there is one entry for several rows that have the same key. Sparse index was first introduced in DB2 V4 for uncorrelated IN subqueries.

- ▶ In DB2 V7 sparse index can only be used during INSIDE-OUT processing
- ▶ In DB2 V8 in both inside-out and outside-in, but only if IMWF cannot be used

Star join access path without a sparse index

Figure 3-17 shows the access path to execute a star join (before PQ61458, which introduced the sparse index function also in V7).

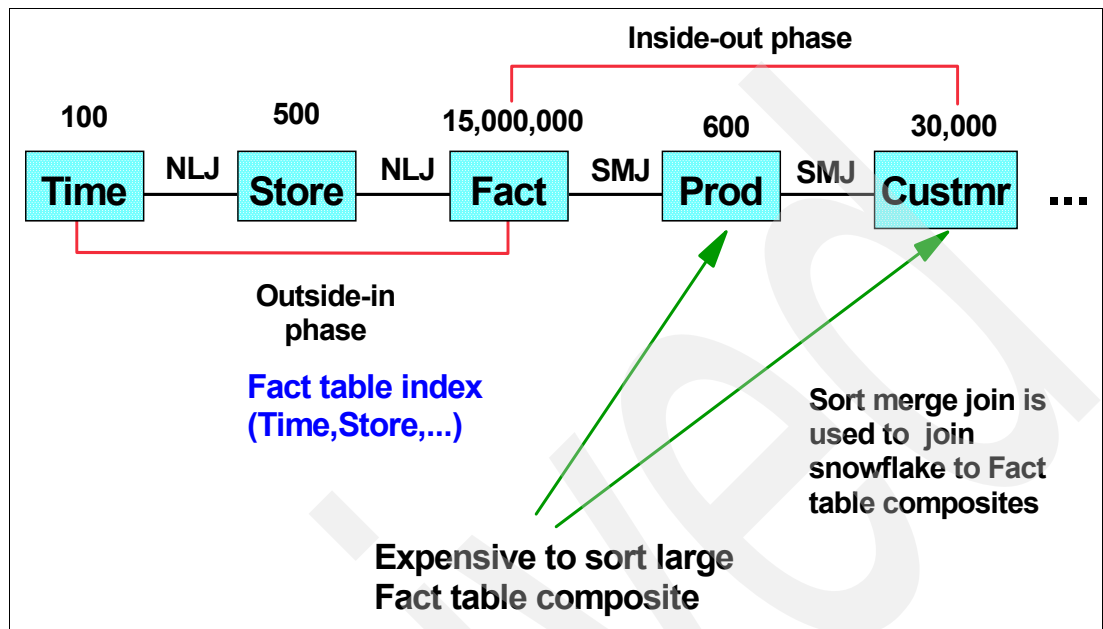


Figure 3-17 Star join access path (before PQ61458)

Characteristics of the star join access path:

- Use of a multi-column fact table index during outside-in phase. Nested-loop join (NLJ) is used.
- No index on the snowflake work file during the inside-out phase. Sort-merge join (SMJ) is used.

This process can be efficient if the number of rows in the fact table composite involves a low number of rows, but can be very expensive for a large fact table composite.

Sparse indexes on work files

Figure 3-18 shows the structure of a sparse index on the work file.

The development of sparse indexes on work files in DB2 V8 improves the performance of the star join execution.

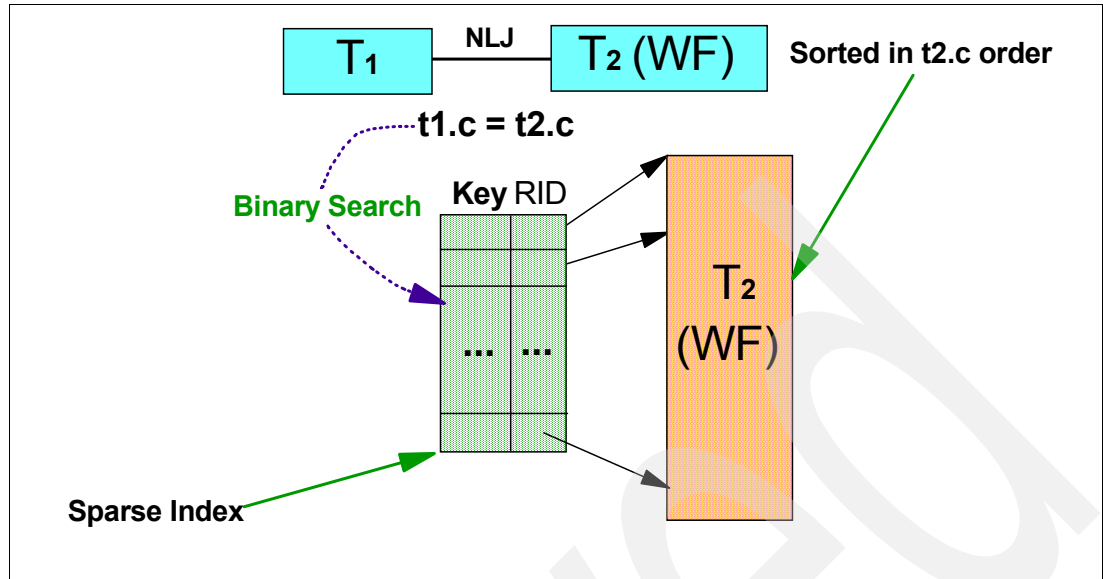


Figure 3-18 Sparse index on work file

With a sparse index there is a binary search to find the entry for the key searched in memory and scans a few rows in the work file.

Sparse index on work file:

- ▶ Made available in DB2 V7 via APAR PQ61458
- ▶ Enhanced in DB2 V8, sort key and a 5 byte RID
- ▶ Sparse indexes are built in cache when they are sorted in join column order
- ▶ Cache up to 240 KB in memory in thread local storage pool
- ▶ Probed through an equal-join predicate
- ▶ New ACCESTYPE='T' in PLAN_TABLE for sparse index access as well as for IMWF
- ▶ Nested-loop join can be used for inside-out phase of star join

Sparse index for star join

Figure 3-19 shows the use of a sparse index in the inside-out phase in a star join.

With the sparse index the access path can be changed to avoid sort-merge-join in the inside-out phase:

- ▶ Use of a multi-column fact table index during outside-in phase
- ▶ Use of a sparse index during inside-out phase

As a result there is no longer the need to sort a large composite file (no sort work file space requirement) and reduction of the overhead (merge and repartition) caused by sort.

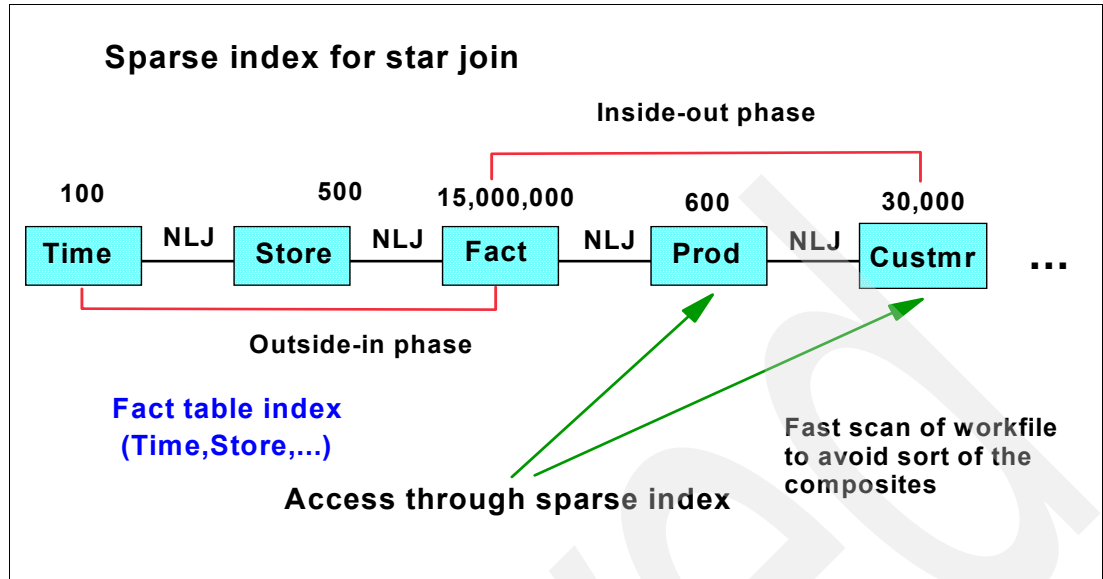


Figure 3-19 Sparse index for star join

The best performance can be obtained if the buffer pool for the work files has enough space to hold the snowflake work files on which the 'T' access in PLAN_TABLE is used.

3.3.4 Performance

Performance measurements show benefits in queries from different workloads as result of the usage of the features described before.

Sparse index performance

Figure 3-20 summarizes the DB2 V7 measurements with (APAR PQ61458) and without sparse indexes (SI) in different queries from two workloads.

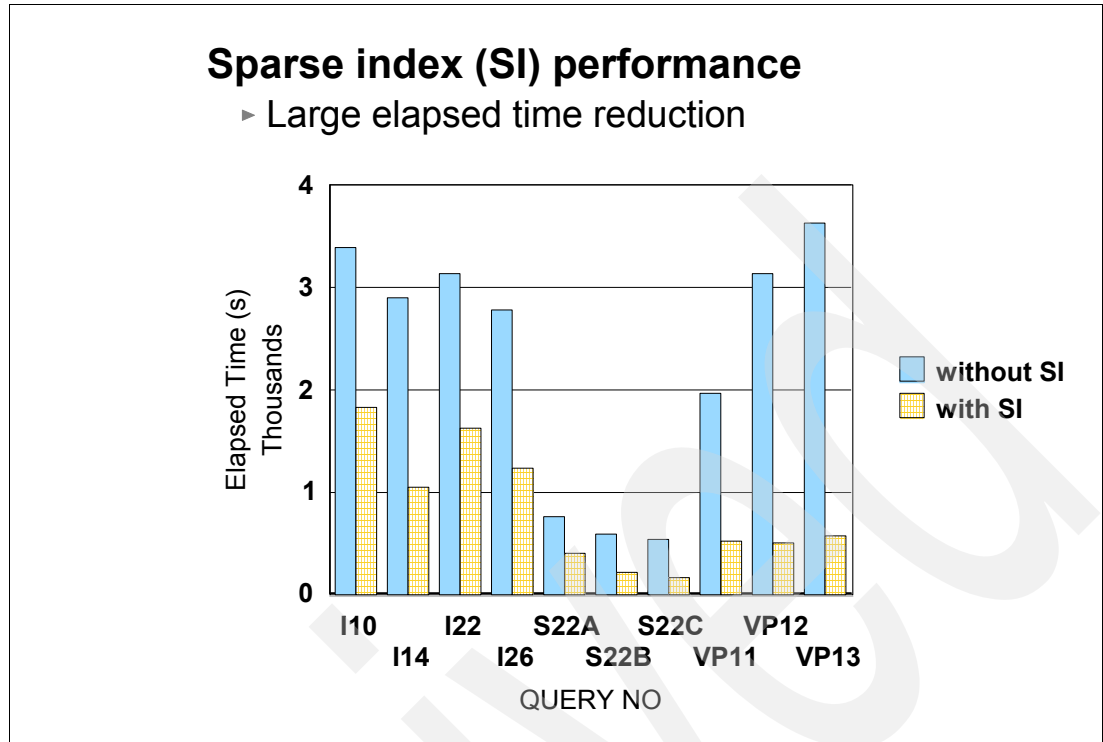


Figure 3-20 Sparse index performance

The performance measurements considering the use of sparse indexes in star join show:

- ▶ 77 queries on two workloads are measured.
- ▶ Among the 41 queries that exploit sparse index (SI), most of queries show 50% - 80% elapsed time reduction. Top queries:
 - Query VP13 elapsed without SI = 3634 sec. and with SI = 561.9 sec. (84% reduction)
 - Query I10 elapsed time without SI = 3399 sec. and with SI = 1838 sec. (45% reduction)
 - Query VP12 elapsed without SI = 3143 sec. and with SI = 498.7 sec. (84% reduction)
 - Query I22 elapsed time without SI = 3142 sec. and with SI = 1625 sec. (48% reduction)
- ▶ With parallelism, they show an additional 30% - 60% elapsed time reduction.
- ▶ Performance improves more when more sorts are involved.

Star join access path

Figure 3-21 shows a comparison on CPU times between DB2 V7 and DB2 V8.

These measurement results were based on all the DB2 V8 supported features, including the in memory work file.

V8 Star Join Access Path

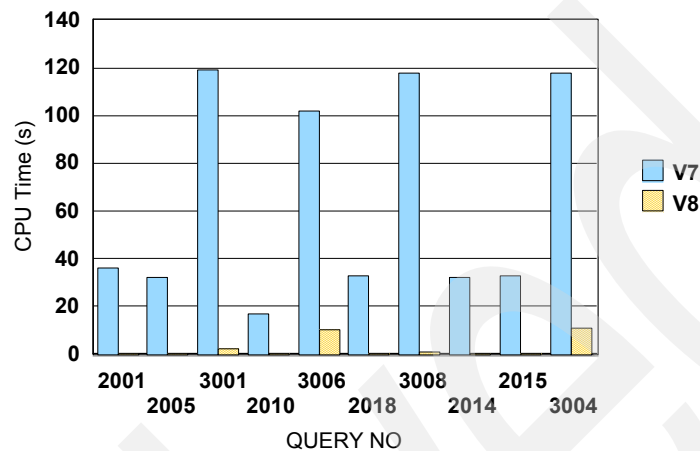


Figure 3-21 Star join access path

Star join access path in DB2 V8 had reduction in CPU time due to:

- ▶ Improved star join cost estimation that better estimates the filtering effect of dimensions
- ▶ Snowflake materialization occurs only if it is cost-effective
- ▶ Updated star join detection logic, optimization and run-time processes
- ▶ Use of in memory work files both in outside-in and inside-out join phases

The measured CPU time for the top queries:

- ▶ Query 3001: CPU time in V7 = 119 sec. and in V8 = 2.3 (reduction of 98%)
- ▶ Query 3008: CPU time in V7 = 118 sec. and in V8 = 1 (reduction of 99%)
- ▶ Query 3004: CPU time in V7 = 118 sec. and in V8 = 11 (reduction of 90%)
- ▶ Query 3006: CPU time in V7 = 102 sec. and in V8 = 10 (reduction of 90%)

In Memory Work File (IMWF) performance

Measurements of the query workload submitted in single thread using various sizes of IMWF show:

- ▶ 20 MB vs. no IMWF - Many queries have performance gain
- ▶ 32 MB vs. 20 MB - Three more queries show improvement
- ▶ 64 MB vs. 32 MB - One more query shows improvement
- ▶ 128 MB vs. 64 MB - No performance gain

Figure 3-22 shows the CPU time reduction with 64 MB IMWF when compared to no IMWF. The result indicates the effect of IMWF (comparison when IMWF is disabled and enabled) on all the other DB2 V8 supported features. When IMWF is disabled, the *traditional* sparse indexes are used, with the same access path.

CPU reduction using in memory work file

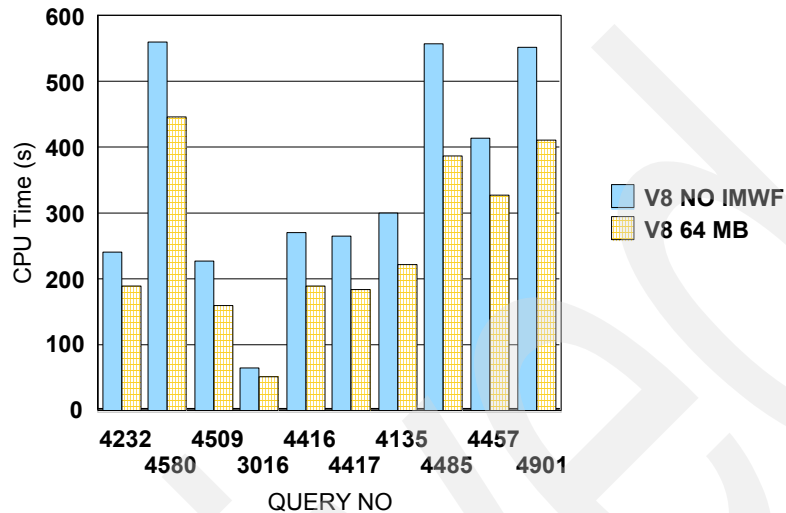


Figure 3-22 In memory work file - CPU reduction

Performance improvements observed on queries from a workload with 64 MB of IMWF compared with no IMWF:

- ▶ Up to 40% elapsed time reduction
- ▶ Up to 31% CPU time reduction. Top queries from the workload:
 - Query 4580: CPU time NO IMWF = 560 sec. and with IMWF = 446 (reduction of 21%)
 - Query 4485: CPU time NO IMWF = 559 sec. and with IMWF = 386 (reduction of 31%)
 - Query 4901: CPU time NO IMWF = 553 sec. and with IMWF = 411 (reduction of 25%)

In practical environments 128 MB to 256 MB can be a good starting point, depending on the workload using the snowflake data model.

Bigger improvements are anticipated in environments where work file data sets are shared concurrently with other threads, such as for sorting.

Performance study on warehouse workload

Total elapsed time and total CPU time were measured for 79 star join queries in a highly normalized data warehouse workload in V7 and V8.

Figure 3-23 shows the improvement in performance with sparse indexes, access path changes and the use of in memory work file.

Star join workload

Total elapsed time and total cpu time for 79 queries on highly normalized warehouse workload

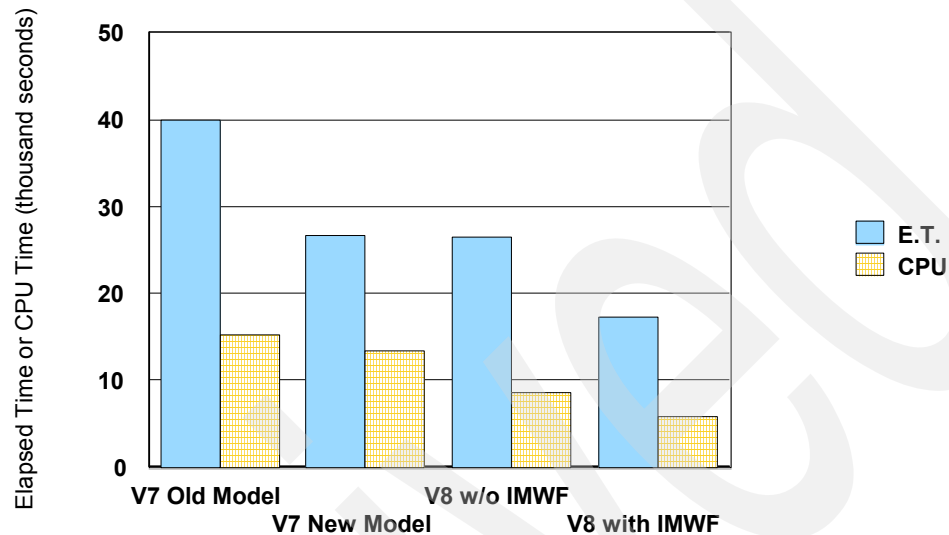


Figure 3-23 Star join workload

Observations on star join workload for 79 queries:

- ▶ V7 old cost model is the default. Sparse index enhancements apply.
- ▶ V7 new cost model - Set SPRMSJNC DSNZPARM to ON (by APAR PQ61458)
 - V7 new cost model provides access path/performance improvement on long running queries
 - Recommend SAP/BW customer use new cost model
- ▶ 33.5% reduction on total elapsed time vs. old cost model
- ▶ V8 without IMWF enhancement
 - Have access path improvements for medium and short running queries
 - 36.4% reduction on total CPU time vs. V7 new cost model
- ▶ V8 IMWF enhancement
- ▶ Reduce the elapsed time of the workload by about 35%

Total elapsed time and total CPU time for 79 queries:

- ▶ V7 old model: 39968 sec. elapsed time and 15219 sec. CPU time
- ▶ V7 new model: 26581 sec. elapsed time and 13462 sec. CPU time
- ▶ V8 w/o IMWF: 26512 sec. elapsed time and 8563 sec. CPU time
- ▶ V8 with IMWF: 17200 sec. elapsed time and 5847 sec. CPU time

The comparison between the V7 old model and V8 with IMWF resulted in a total of a 56% reduction in elapsed time and a 61% reduction in CPU time.

3.3.5 Conclusion

V8 star join queries perform much better than V7 due to:

- ▶ Star join access path enhancements

- In memory work file which caches the data and index keys in a dedicated virtual memory pool

Star join performance is dependent on the circumstances. Your mileage may vary.

3.3.6 Recommendations

The size of the in memory work file can be set on the installation panel DSNTIP8, parameter STAR JOIN MAX POOL.

The in memory data caching for star join is most effective when the schema contains snowflakes. Therefore, for the users whose star schema is highly normalized and has many snowflakes, a larger pool size would be recommended (for example, 256 MB), particularly when the DB2 subsystem is dedicated to star schema workloads. This virtual storage is above the 2 GB bar and the real storage use is limited to only the pages touched, so there should be no significant penalty for those not using star joins and potentially much greater benefit for those using star joins.

Note: An unfiltered snowflake may not be materialized in V8 in contrast to V7 where all snowflakes were materialized into work files.

The sparse index support in inside-out phase and part of the new star join access path selection logic are included in APAR PQ61458 applicable to V7.

After migrating to DB2 V8, specify adequate amounts of memory for maximum pool size in DSNZPARM SJMXPOOL to gain the benefit of in memory work files in a dedicated virtual memory pool for star join.

3.4 Stage 1 and indexable predicates

Figure 3-24 shows one example of the execution of one SQL using index and shows the difference between Stage 1 and Stage 2 predicates, where Stage 1 predicates can be resolved sooner, without involving complex analysis.

The SELECT statement reads all orders from the table TORDER that has order number between 15000 and 17000. DB2 specifies the use of the index on column ORDERNO starting at key value 15000. DB2 uses the index tree to find the leaf page containing the value 15000 and scans forward until it accesses the leaf page containing the value 17000. Then it stops the scan. GETPAGEs are issued for each index page referenced.

In complex SQL statements with multiple predicates, they are applied in the sequence:

1. Matching index predicate
2. Other index key predicates applied
3. All other stage 1 predicates
4. All stage 2 predicates

If the predicate is applied in the earlier stage, the performance is improved. If the predicate is used as an argument for a matching index, DB2 does not need to read data pages not satisfying the predicate. On the other hand, if the predicate is stage 2, DB2 has to read rows that do not satisfy the predicate and send them to further analysis.

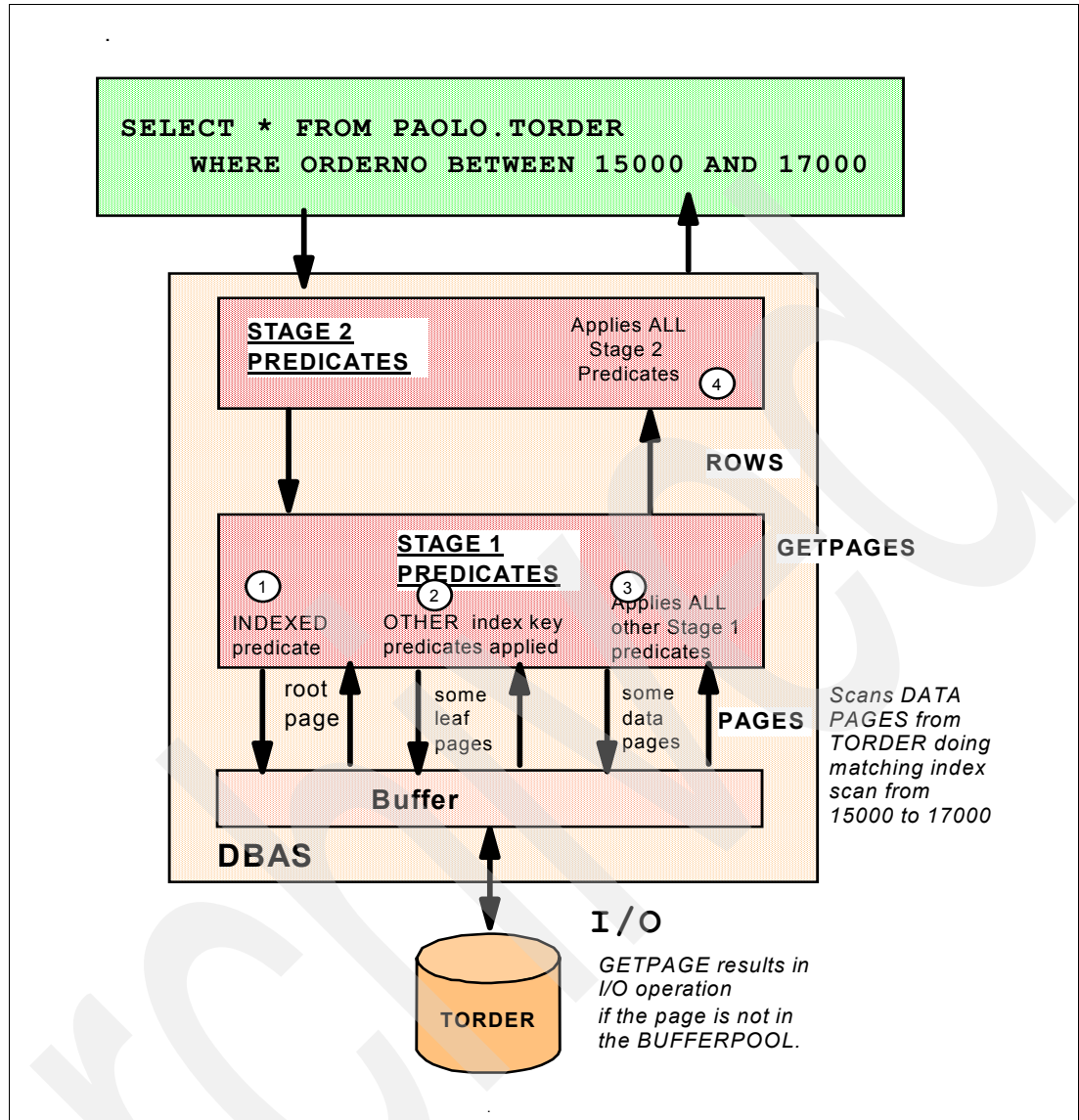


Figure 3-24 Analysis of Stage 1 and Stage 2 predicates

Stage 1 predicates are simple predicates evaluated by DB2. They are evaluated first to reduce processing cost and eliminate the number of rows to be evaluated by the complex predicates. Stage 2 predicates are complex predicates evaluated by DB2's more complex analysis. They are also known as *residual* predicates.

An indexable predicate is a predicate that can use a (matching) index access as an access path. Indexable predicates are always stage 1, but not all stage 1 predicates are indexable.

Stage 1 predicates are better than stage 2 predicates.

However, if your application has been coded in C on a work station, that language does not have a DECIMAL data type, although some of the existing tables might have columns defined as DECIMAL(p,s).

Java does not have a fixed length character string data type; every string is variable length.

3.4.1 New indexable predicates in V7

PTF UQ61237 for APAR PQ54042 resolves a case of incorrect output resulting from SQL statements that contain a BETWEEN predicate. The type of BETWEEN predicate that may have this problem is:

```
col BETWEEN col-expression AND col-expression
```

Where “col” is a simple column and “col-expression” is any expression involving a column.

In addition, BETWEEN predicates of the form:

```
T1.COL BETWEEN T2.COL1 AND T2.COL2
```

can now be indexable when accessing the table on the left hand side of the BETWEEN keyword if the tables on the right hand side of the BETWEEN keyword have already been accessed.

3.4.2 More stage 1 and indexable predicates in V8

In V8, more predicates with mismatched data types and mismatched lengths can be applied by DB2. This allows some stage 2 predicates to become stage 1 predicates, and stage 1 predicates allow more index usage, and it applies to both join and local predicates.

The potential for performance improvement with no application change is very significant.

Note that mismatched string data types in join became stage 1 in V6.

Mismatched numeric types

Figure 3-25 shows an example of a predicate using mismatched numeric types that is stage 2 in V7 and becomes stage 1 and indexable in V8 **NFM**.

In the example, the selection of rows from employee table has to scan the table space in V7 because the predicate indicates salary defined as decimal (12,2) is compared to a host variable defined as floating point that is stage 2.

The same predicate is stage 1 in V8 and it allows matching index scan to find the rows for the execution of the SQL.

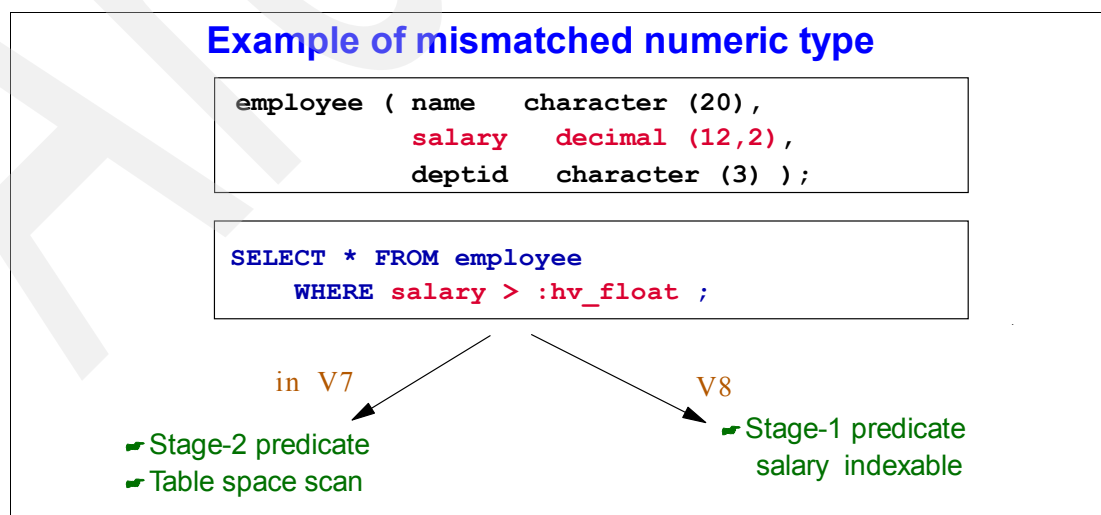


Figure 3-25 Example of mismatched numeric types

But not all mismatched numeric types become stage 1. The exceptions that are still stage 2 predicates are:

- ▶ REAL -> DEC(p,s) where p > 15
where REAL is the right hand side predicate -> DEC(p,s) is the left hand side predicate
- ▶ FLOAT -> DEC(p,s) where p > 15
 - REAL or FLOAT column in the outer table of a join
 - REAL or FLOAT literal or host variable
 - DEC_column > REAL_hostvar

REAL and FLOAT values cannot be converted to decimal (the index column) with precision > 15 without possibly changing the collating sequence.

Mismatched string types

Figure 3-26 shows examples of mismatched string types that were stage 2 in V7 and are stage 1 and indexable in V8 **CM**.

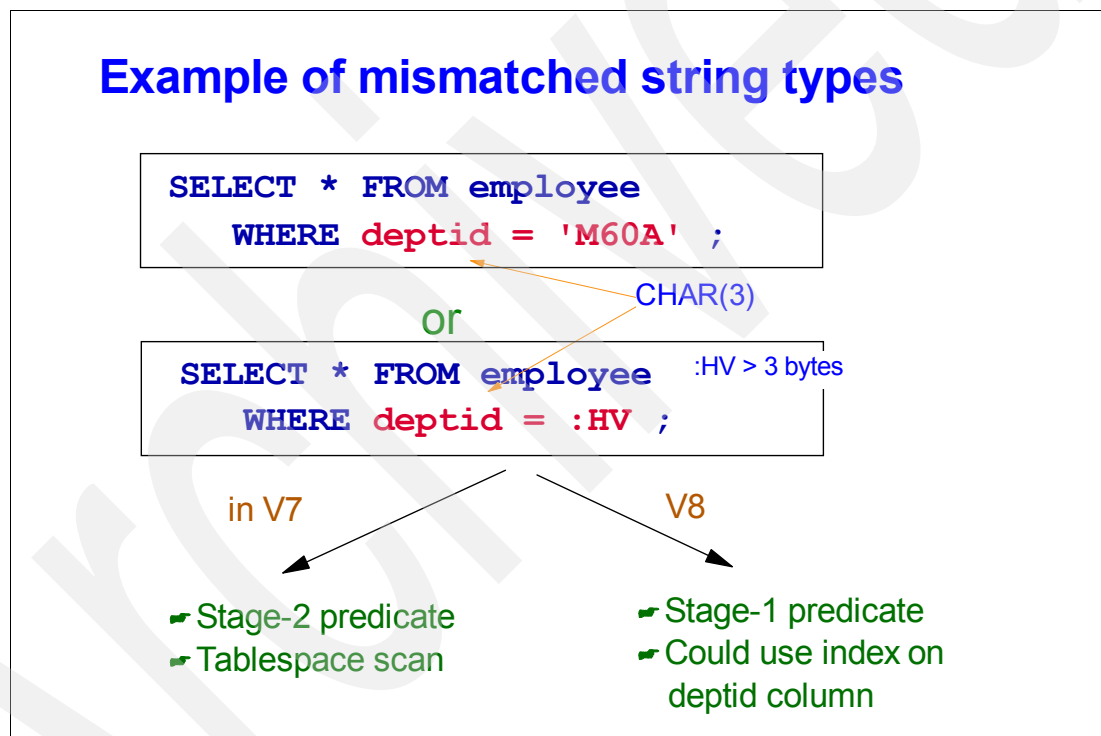


Figure 3-26 Example of mismatched string types

In the example, the selection of rows from the employee table has to scan the table space in V7 because the column DEPTID is defined as CHAR(3) and is compared to a string of four characters. This is a stage 2 predicate in DB2 V7.

The same predicate is stage 1 in V8 and it allows a matching index scan.

Not all mismatched string types become stage 1. The exceptions that are still stage 2 predicates are:

- ▶ graphic/vargraphic -> char/varchar
where, for example, graphic is the right hand side predicate -> char is the left hand side predicate

- ▶ char/varchar(n1)-> char/varchar(n2)
 - where n1>n2 and not equal predicate
- ▶ graphic/vargraphic(n1) -> graphic/vargraphic(n2)
 - where n1>n2 and not equal predicate
- ▶ char/varchar(n1) -> graphic/vargraphic(n2)
 - where n1>n2 and not equal predicate

3.4.3 Performance

Performance measurements were made to show the type of benefits that can be achieved using mismatched numeric data types and mismatched string types in predicates that were stage 2 in V7 and are stage 1 in V8.

Mismatched numeric data types

The measured query is a count of the rows of a result set of a SELECT joining two tables where the columns in the join predicate are INTEGER and DEC(8).

Figure 3-27 shows the SQL text executed in V7 (with DEGREE(1) and DEGREE(ANY)) and V8, the PLAN_TABLE indicating the usage of an index to execute the join, and the measured elapsed and CPU times. The tables used here are the same as in Figure 3-13 on page 50.

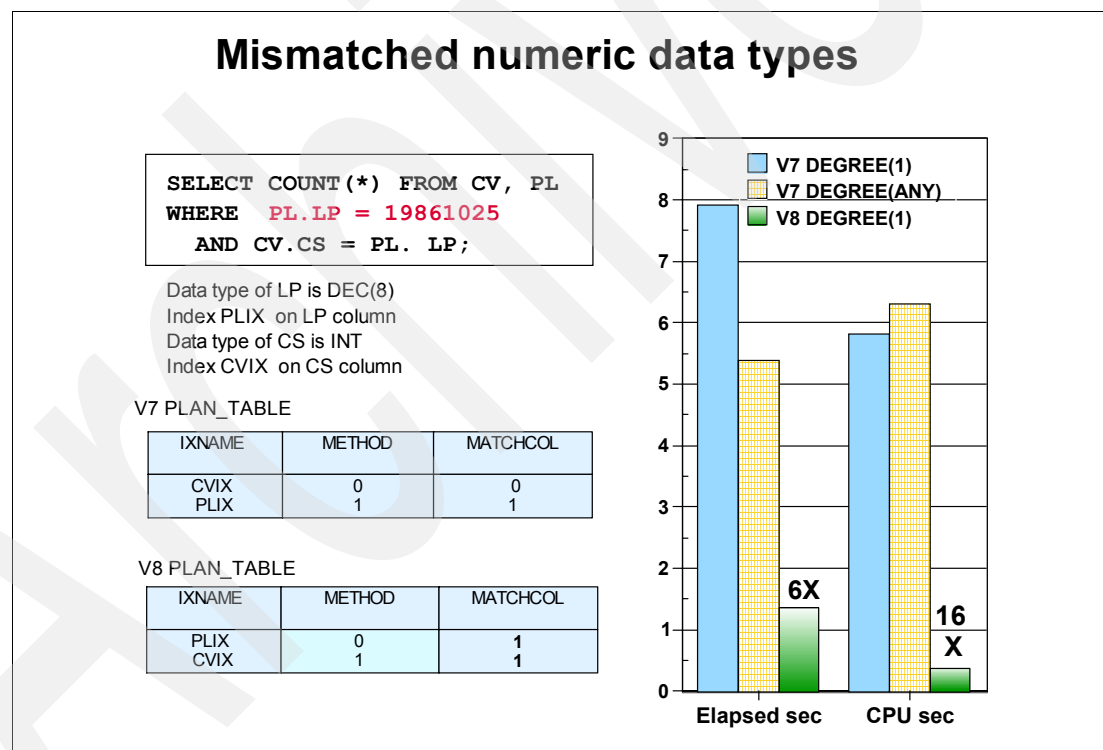


Figure 3-27 Mismatched numeric data types

The columns involved in this equi-join have different numeric data types: Column LP is DEC(8) and column CS is INT.

V7 shows that table CV is joined with table PL using the nested loop join method. PL is accessed via a one column matching index only access. CV is accessed via a non-matching index only access that results in reading all leaf pages in the index.

V8 shows that PL is joined with CV also using nested loop join method. But the sequence is different. Both tables are accessed via a one-column matching index only scan, but the access path in V8 is more efficient.

Other measurement indicators are listed in Table 3-2.

Table 3-2 Measurements for mismatched numeric data types

Measurement	V7	V8	Delta
Elapsed time in sec.	7.9	1.4	-82%
CPU time in sec.	5.8	0.3	-95%
Work file getpages	0	0	0
Data getpages	77	43	-44%
Index getpages	2558	8	-99.7%
Matching index	1	2	100%

Mismatched string data types

This measured query is a join of two tables where the local predicates have different lengths as shown in Figure 3-28.

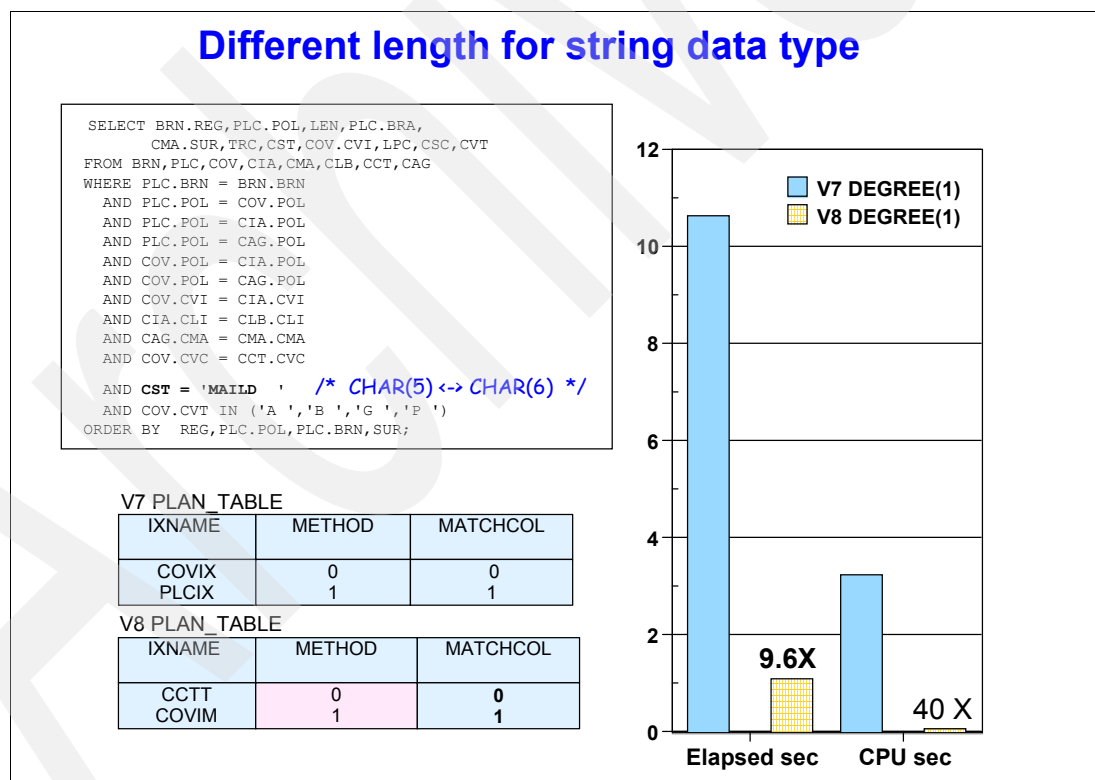


Figure 3-28 Different lengths for string data type

The predicate comparing strings of different length becomes stage 1, and then the DB2 Optimizer changes the sequence of tables in the join to do an earlier filtering, resulting in better performance.

Summary of measurements with a predicate comparing a string data type with different lengths:

- ▶ Elapsed time in V7 = 10.65 sec. and in V8 = 1.11 sec. (reduction of 89%)
- ▶ CPU time in V7 = 3.24 sec. and in V8 = 0.08 sec. (reduction of 97%)

3.4.4 Conclusions

- ▶ Predicates comparing mismatched string data types are stage 1 and indexable with minor exceptions.
- ▶ Predicates comparing mismatched numeric data types are stage 1 and indexable with minor exceptions.

3.4.5 Recommendations

- ▶ Consider rebinding plans or packages to get better access paths.
- ▶ Consider creating an index for the mismatched columns to enable index access.

3.4.6 Considerations on CASTing

In DB2 V7, we recommend you use the CAST function to make both data types the same. However, if you do this, first you add some overhead, and then you generally dictate the join sequence because DB2 favors the index and picks the simple column side as inner table. Let us look at the following SELECT:

```
SELECT *  
FROM CUST AS C, ORD_DEC AS O  
WHERE C.CUST_NO = O.CUST_NO;
```

If you rewrite the predicate as `'CUST_NO = CAST(O.CUST_NO AS INTEGER)'`, DB2 very likely picks the CUST table as inner table. Furthermore, since `ORD_DEC.CUST_NO` is `DEC(15,0)`, this has a bigger range than the `INTEGER` type. Unless you are very sure that all the values in the `ORD_DEC.CUST_NO` are within the range of `INTEGER`, the `CAST` may fail and DB2 may return an error. In DB2 V8, unless you really want to dictate the join sequence, it is better not to add any `CAST` and let DB2 choose the join sequence based on costing. Generally this provides better performance.

3.5 Multi-predicate access path enhancement

Queries which contain two or more predicates referencing a table can get a better access path with statistics for a group of columns (non-indexed or non-leading index columns) collected by `RUNSTATS` in V8.

Long running queries with skewed data distribution or column correlation type environment involving joins with multiple predicates on columns that are not the leading columns of an index or columns that do not participate in indexes are good candidates to benefit from improved access paths.

The new access path in V8 can change the sequence of tables participating in joins involving multiple tables to take advantage of early filtering, and as a result, accessing fewer pages to execute the SQL statement.

Query performance problem

Lack of multi-column statistics can cause a bad access path and poor query performance when a table is referenced by two or more equal, range or BETWEEN predicates.

The possibility of hitting this problem increases when:

- ▶ The number of predicates referencing one table increases
- ▶ The number of joined tables in a query increases

Two, often complementary, DB2 V8 functions help with this issue:

- ▶ Collecting additional statistics
- ▶ Relying on DB2 V8 better filter factor estimation

Additional statistics

Collecting cardinality and multi-column distribution statistics and rebinding is the first step to solve the problem. The Statistics Advisor, of Visual Explain V8, can generate automatically the RUNSTATS statement for the query, after using the Analyze function for a multi-predicate SQL text. See 6.4, “RUNSTATS enhancements” on page 272. Visual Explain is discussed in 3.15, “Visual Explain” on page 110 and 3.15.3, “Statistics Advisor” on page 116.

Example 3-6 shows one SQL statement text used as input to Visual Explain. This is the old query 5 used to verify the DB2 Catalog consistency, discussed in Chapter 9, “Installation and migration” on page 341.

Example 3-6 SQL text for Visual Explain

```
SELECT DBNAME, NAME FROM SYSIBM.SYSTABLESPACE TS
  WHERE NTABLES <>
    (SELECT COUNT(*) FROM SYSIBM.SYSTABLES TB
      WHERE TYPE IN ('T', 'X', 'M')
        AND TB.DBNAME = TS.DBNAME
        AND TB.TSNAME = TS.NAME
        AND TS.NAME <> 'SYSDEFLT');
```

The query has one correlated subquery.

The outer query scans the SYSIBM.SYSTABLESPACE table which has the indicated number of tables NTABLES different from the counts of tables in the inner query based on table SYSIBM.SYSTABLES. For each qualified row in the outer table, the inner query is re-executed. Table type = “X” specifies auxiliary table at the current server for storing LOB data.

The Explain function in Visual Explain for the SQL text results in the graphic for the access path selected by DB2 V8 in Figure 3-29.

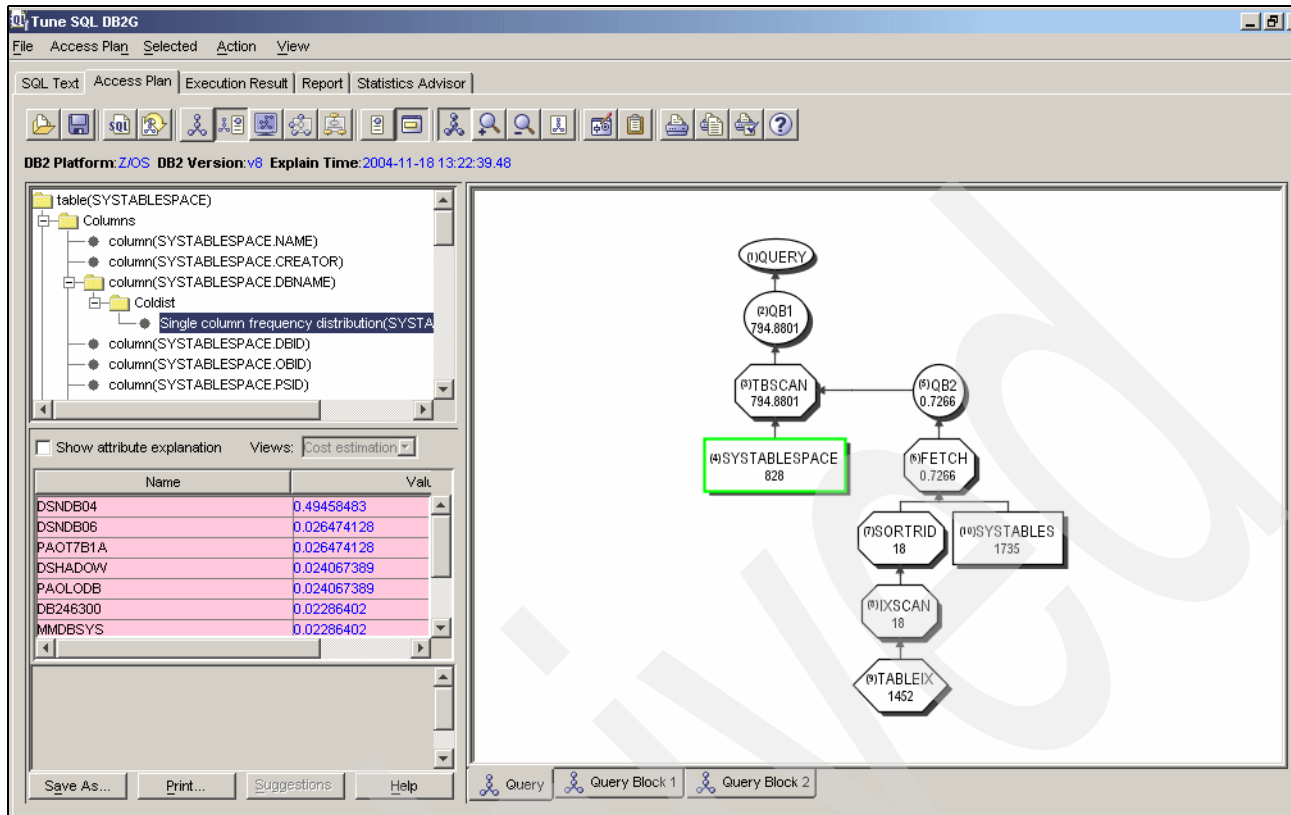


Figure 3-29 Access path graph in Visual Explain

If we click the rectangle indicating (4)SYSTABLESPACE in the graph, it shows that SYSTABLESPACE has 828 rows and it is estimated that approximately 794 rows of the table qualify in the table space scan.

On the left side of the graph there is one tree structure that shows information about the columns, indexes and table space related to the selected table SYSTABLESPACE. If there is a frequency distribution on one column of the table, under the respective column there is a Coldist folder. If you select *i*, you can see the values of the column and the filtering factor for the value.

For each qualified row in the outer table, the subquery is executed using a matching index scan using predicate TB.DBNAME=TS.DBNAME with filtering factor of 0.0098 returning 18 RIDs.

The report function generates information about the cardinalities used. If there are no statistics available, default cardinality is used by the optimizer.

Example 3-7 shows the RUNSTATS statement given by the Statistics Advisor of Visual Explain for the query in Example 3-6 after the execution of the analyze function. Visual Explain enhancements are reported in 3.15, "Visual Explain" on page 110.

Example 3-7 RUNSTATS statement provided by the Statistics Advisor

```
RUNSTATS TABLESPACE DSNDB06.SYSDBASE
TABLE(SYIBM.SYSTABLESPACE)
COLUMN(NAME)
TABLE(SYIBM.SYSTABLES)
COLUMN(TYPE,TSNAME)
COLGROUP(TYPE) FREQVAL COUNT 10
```

```
      SORTDEVT SYSDA
      INDEX(SYSIBM.DSNDX01,
            SYSIBM.SQLDTX01,
            SYSIBM.DSNDTX01,
            SYSIBM.DSNDTX03,
            SYSIBM.DSNDTX02 KEYCARD,
            HAA.TABLEIX)
SHRLEVEL CHANGE REPORT YES
```

Note that the RUNSTATS suggested by the Statistics Advisor specifies the new COLGROUP option introduced in V8 to collect frequency in a column group. For the table SYSIBM.SYSTABLESPACE the statistic is for the column NAME used as a predicate in the correlated subquery to have a better filter factor estimate. The statistic for indexes specifies one index for SYSTABLESPACE and five indexes for SYSTABLES.

You may consider executing RUNSTATS to collect these statistics.

In DB2 V8, RUNSTATS can collect the most and least frequently occurring frequencies on almost any column or column group. RUNSTATS can also collect multi-column cardinality on almost any column group. See also 6.4, “RUNSTATS enhancements” on page 272.

In many queries there are predicates on some matching and some screening columns. You can collect statistics to better reflect the total index filtering. When you have correlations and skews on some indexed and some non-indexed columns, you can collect statistics so the optimizer can more accurately estimate the number of rows which qualify from the table.

The statistics that can be collected are:

- ▶ Frequency distributions for (non-indexed) columns or groups of columns
- ▶ Cardinality values for groups of non-indexed or non-leading index columns
- ▶ LEAST frequently occurring values, along with MOST for both indexed and non-indexed column distributions

The new keywords: COLGROUP, MOST, LEAST, BOTH, SORTNUM, and SORTDEVT

Better filter factor estimation in V8

DB2 V8 can estimate a better filter factor by using statistics inference derivation. This can result in with or without the additional statistics mentioned earlier:

- ▶ Improved query selectivity estimation using statistical inference on available statistics, when applicable
- ▶ Queries which contain two or more predicates referencing a table may get better access paths
- ▶ Significant performance improvement for some complex queries
- ▶ Consider rebinding this type of query

3.5.1 Performance

The performance measurements show the benefits with statistics on multiple columns.

Better filter factor estimation

Statistics on predicate columns allow the DB2 optimizer to select a better access path as reported in the example in Figure 3-30.

Better filter factor estimation

V7 PLAN_TABLE

IXNAME	METHOD	MATCHCOL
TSS on PLO	0	0
CLAIX	1	2
CLBIX	4	1
BLAIX	4	1

V8 PLAN_TABLE

IXNAME	METHOD	MATCHCOL
TSS on PLO	0	0
BLAIX	1	1
CLBIX	1	1
CLAIX	1	2

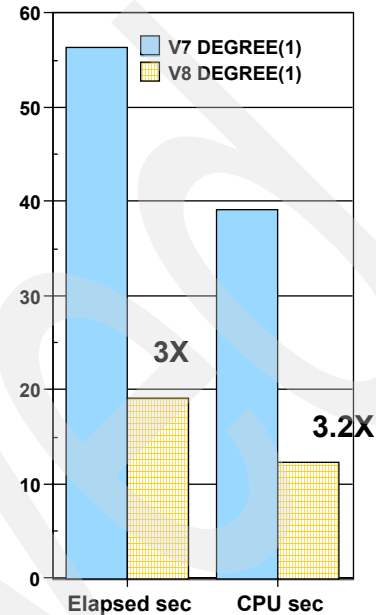


Figure 3-30 Better filter factor estimation

This figure shows a join of four tables and two join predicates between tables PLO and CLA. The plan table for V7 shows a join of table PLO and CLA using NLJ that is followed by a join to table CLB and BLA using hybrid join.

In DB2 V8, the same queries allow the DB2 optimizer to find a better access path applying earlier a better filtering factor. The sequence of join of four tables changes: table BLA, that was processed last, becomes the second table scanned using NLJ, as the tables that follow. In DB2 V8, the hybrid join method is not used.

The measurements of this example show:

- Elapsed time = 56.4 seconds in V7 to 19.1 seconds in V8 (66% reduction, a factor of 3)
- CPU time = 39.1 seconds in V7 to 12.4 seconds in V8 (68% reduction, a factor of 3.2)

The improvement ratio can be even more significant in complex queries involving more tables and more predicates.

Access path improvements - multi-column statistics

Measurements on other queries from the sample workload show improvements in CPU time resulting from a better access path as shown in Figure 3-31.

Access path improvements - multi-column statistics

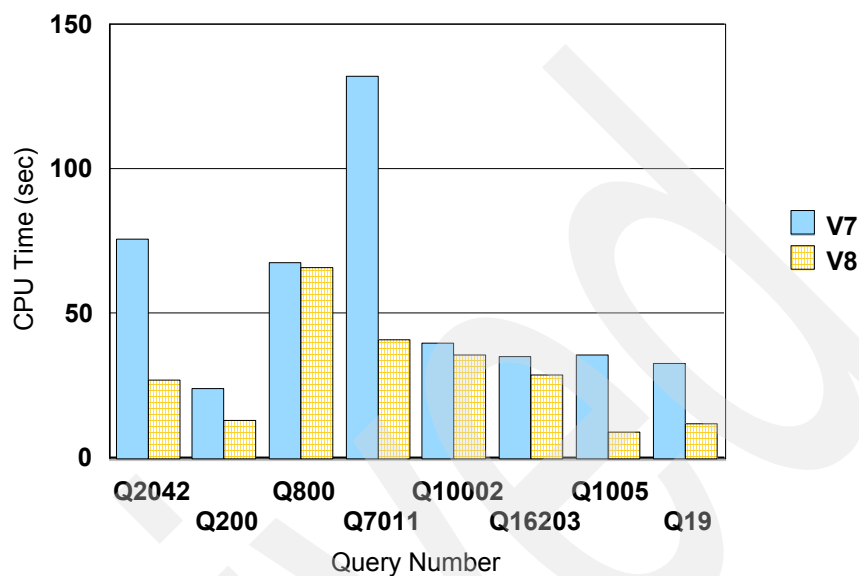


Figure 3-31 Access path improvements - Multi-column statistics

Measurements of CPU time for top consuming queries:

- ▶ Query Q7011
CPU time = 132 sec. in V7 to 27 sec. in V8 (68% reduction, factor of 3.2)
- ▶ Query Q2042
CPU time = 76 sec. in V7 to 27 sec. in V8 (64% reduction, factor of 2.8)
- ▶ Query Q800
CPU time = 68 sec. in V7 to 66 sec. in V8 (2% reduction)

The improvement ratio varies and tends to be more significant in complex queries involving more tables and more predicates.

3.5.2 Conclusions

Multi-column statistics provided by the RUNSTATS utility in V8 allow DB2 to better estimate the filtering factor for a query.

This benefits queries with multiple predicates in one table and joins of multiple tables. As the number of predicates increases and the number of tables joined increases, there is more opportunity for improvements.

3.5.3 Recommendations

In case you are not satisfied with query performance, we recommend collecting the distribution statistics on the non-leading indexed columns and/or non-indexed columns, which are used as predicates in the query (for which there are no statistics in the catalog). Collecting these statistics can ensure DB2 can use them for better access path selection.

Consider this option if the long running queries involve multiple predicates in one table and joins of multiple tables. The Statistics Advisor function of Visual Explain helps significantly by suggesting statistics to be collected, and how to collect them, for the query. This is a better way to resolve bad access path problems than using OPTIMIZATION HINTS.

Rebinding programs that contain an SQL statement with predicates on multiple, not indexed columns and joins, should also be considered.

3.6 Multi-column sort merge join

In V7, when there is more than one predicate for the sort merge join, access path selection makes decisions based only on the first SMJ predicate. If there is a second predicate on SMJ, it is applied at a later stage. The new table in MCSMJ is always sorted, and query parallelism is disabled.

In V8, access path selection supports multi-column predicates in SMJ. The new table is sorted or not, based on cost, and query parallelism is enabled for multi-column sort merge join (MCSMJ). This applies also in case of mismatched string data types. Mismatched numeric types, and DATETIME types are not supported yet. To summarize:

- ▶ For char data types only, mismatched predicates on two and more col are allowed
- ▶ Sort is now cost-based
- ▶ Parallelism is allowed for MCMSJ

3.6.1 Performance

Two examples are presented showing improved performance for multi-column sort merge join.

The first example compares a two table join with mismatched data types executed in V7 and V8 without parallelism. The second example shows the same query without parallelism and with parallelism.

Mismatched data types

Figure 3-32 shows an example of MCSMJ.

The join of two tables P and PX has two mismatched data type predicates, the first comparing CHAR(20) and VARCHAR(20), and the second comparing CHAR(10) and VARCHAR(10).

The difference is in V7 the access path uses only one predicate, and in V8 it applies both predicates to process sort merge join. The elapsed time is reduced from 184 seconds in V7 to 71.2 seconds in V8, without using parallelism in this case.

The sort merge join processing multi-column in V8 reduces the amount of pages used by sort in the work file as a result of better filtering when compared to single column: 1648262 getpages in V7 and 318540 getpages in V8 (reduction of 81%).

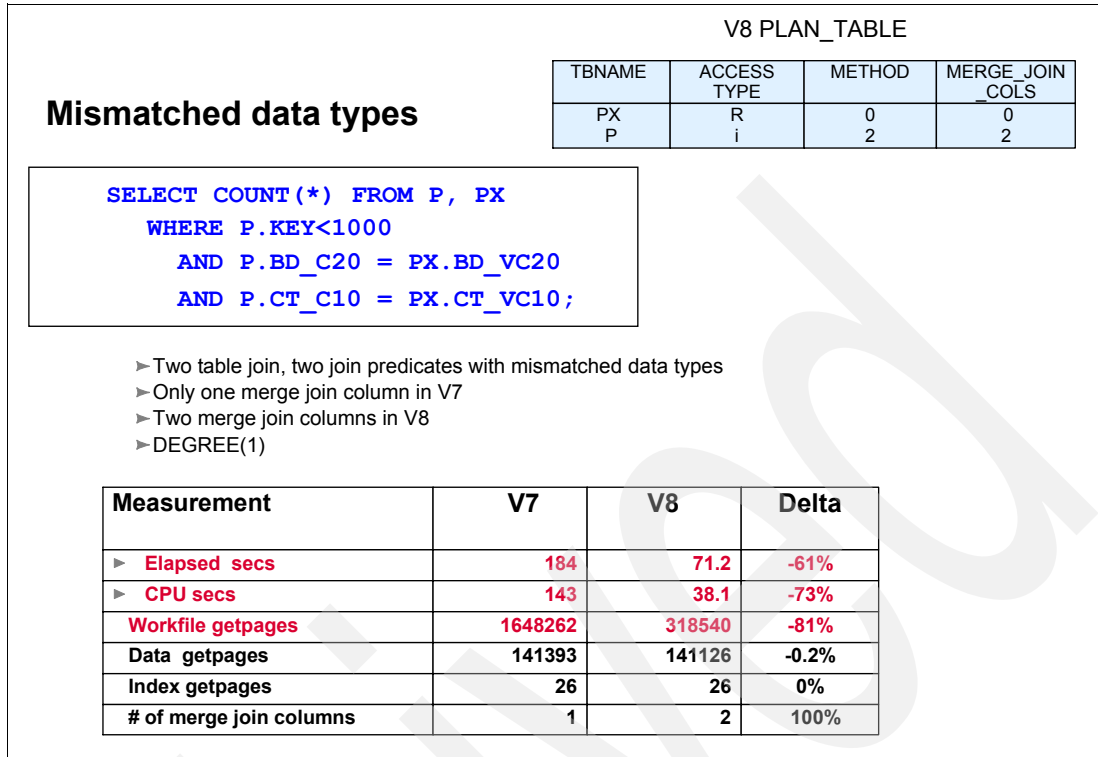


Figure 3-32 Mismatched data types

Query parallelism in V8

For details on how to activate parallelism see *Chapter 34. Parallel operations and query performance in DB2 UDB for z/OS Version 8 Administration Guide*, SC18-7413.

Figure 3-33 shows the further reduction in elapsed time when parallelism is enabled for MCSMJ.

The measurements show the additional improvement to the query used in the previous section, when it is executed in a plan/package with DEGREE(ANY):

- ▶ Elapsed time = 71.2 sec. for DEGREE(1) to 31.2 sec. for DEGREE(ANY), reduction of 56%
- ▶ CPU time = 38.1 sec. for DEGREE(1) to 47.2 sec. for DEGREE(ANY)

The elapsed time is reduced when executed with query parallelism. But more CPU is used due to the coordination process of the parallelism.

Query parallelism in V8

```
SELECT COUNT(*) FROM P,PX
WHERE P.KEY < 1000
      AND P.BD_C20=PX.BD_VC20
      AND P.CT_C10 = PX.CT_VC10;
```

- Data type of P.BD is CHAR(20)
- Data type of P.CT is CHAR(10)

V8 PLAN_TABLE
DEGREE(ANY)

TBNAME	METHOD	MATCHCOL
PX	0	0
P	2	1

V8 PLAN_TABLE
(continue)

ACCESSTYE	ACCESS DEGREE	JOIN DEGREE
R	30	?
I	?	3

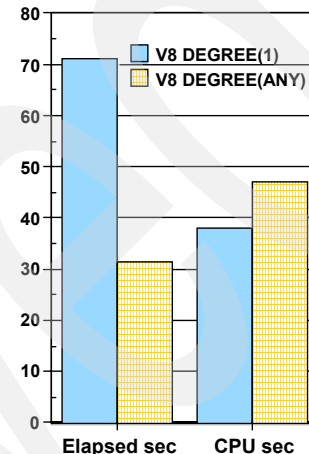


Figure 3-33 Query parallelism in V8

3.6.2 Conclusion

Queries involving joins using MCSMJ (the method is likely to be used when the result set tends to be large) can have benefits because of access path enhancements due to better filtering when multiple columns are used as predicates.

Enabling query parallelism in long running queries can give extra improvement in elapsed time, but generally an increase in CPU time.

3.6.3 Recommendations

Evaluate the usage of sort merge join in long running queries. V8 allows improvement if there are multi-column predicates in sort merge join. If the elapsed time is a concern, the query can be executed in shorter elapsed time if executed with query parallelism.

3.7 Parallel sort enhancement

In V7, parallel sort has the following characteristics:

- Parallel sort enabled for single-table sort composite only
- Not cost-based decision
- No threshold to disable parallel sort for short running queries

Sort composite tables from more than one table can have longer elapsed time because sort is not executed in parallel. If the composite table is from a single table, then parallel sort is invoked in V7.

In V8 the parallel sort has been changed:

- ▶ Parallel sort is enabled for single and multiple-table sort composite
- ▶ Cost-based decision
- ▶ Thresholds to disable parallel sort for short running queries
 - Total sort data size (<2 MB, 500 pages)
 - Sort data size per parallel degree (<100 KB, 25 pages)

Figure 3-34 shows an example of a join of three tables using the multiple table parallel sort merge join method.

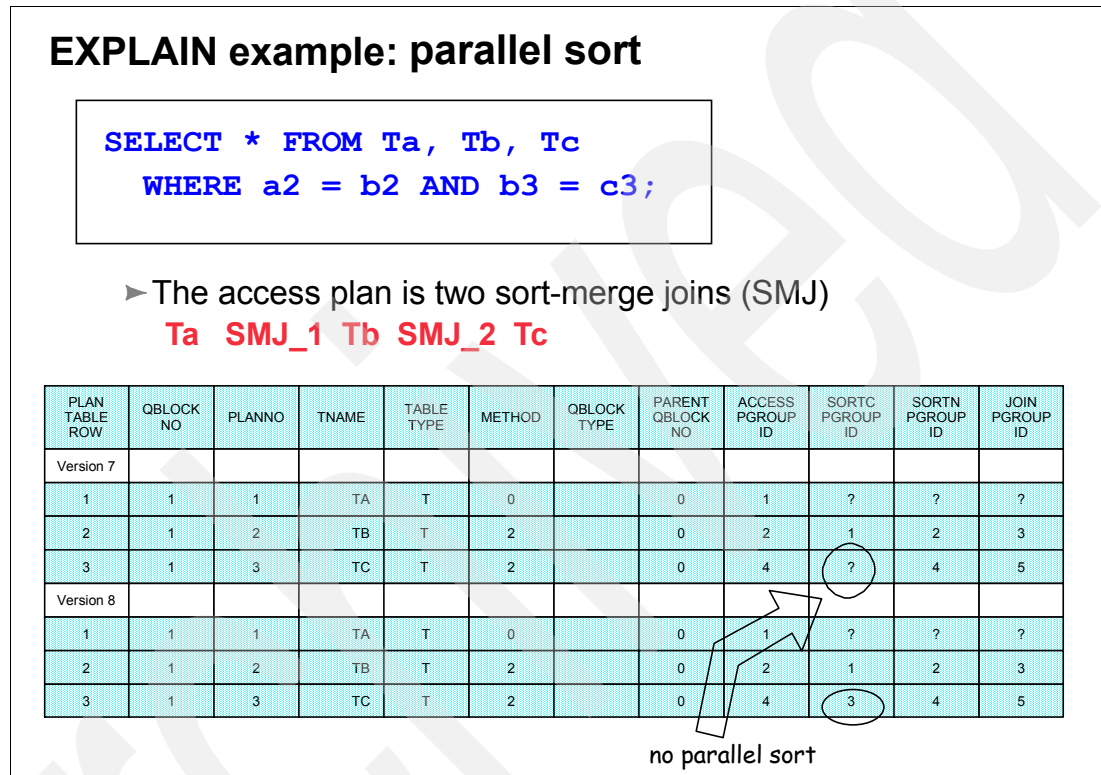


Figure 3-34 EXPLAIN example - Parallel sort

In the example, the plan table shows that, when joining new table Tc with the composite of (Ta SMJ_1 Tb), there is no parallel sort in V7 and parallel sort is used in V8.

For details on how to activate parallelism, see *Chapter 34. Parallel operations and query performance* in *DB2 UDB for z/OS Version 8 Administration Guide*, SC18-7413-00.

3.7.1 Performance

Measurements of the execution of queries were made to compare sort in V7 vs. V8.

Multiple composite table parallel sort

The queries from the workload measured involved multiple tables with large sort.

- ▶ Query elapsed times range from 11 seconds to 35 minutes
- ▶ Up to 6X elapsed time reduction is observed
- ▶ Large improvement when a full merge pass is not required by parallel sort
 - Either rows already in order
 - Or fewer than 32k rows in sort tree (64k rows in V8)

- ▶ Reduced improvement when merge pass is still required
- ▶ Reduced improvement when query has two or more parallel sort composites to be merged

Multiple table GROUP BY sort

Let us look at the query in Example 3-8 where DB2 executes a parallel sort when grouping 4 million rows in 600,000 groups in a join of two tables.

Example 3-8 Multiple table GROUP BY sort query

```

SELECT C_CUSTKEY, C_NAME,
       SUM(O_ORDERKEY * 0.0001) AS REVENUE,
       C_ACCTBAL, C_NAME, C_ADDRESS, C_PHONE, C_COMMENT
FROM CUSTOMER, ORDER
WHERE C_MKTSEGMENT = 'BUILDING'
      AND C_CUSTKEY = O_CUSTKEY
      AND O_ORDERDATE < DATE('1994-12-05')
GROUP BY C_CUSTKEY, C_NAME,
         C_ACCTBAL, C_NAME, C_ADDRESS, C_PHONE, C_COMMENT

```

Figure 3-35 shows the performance improvement.

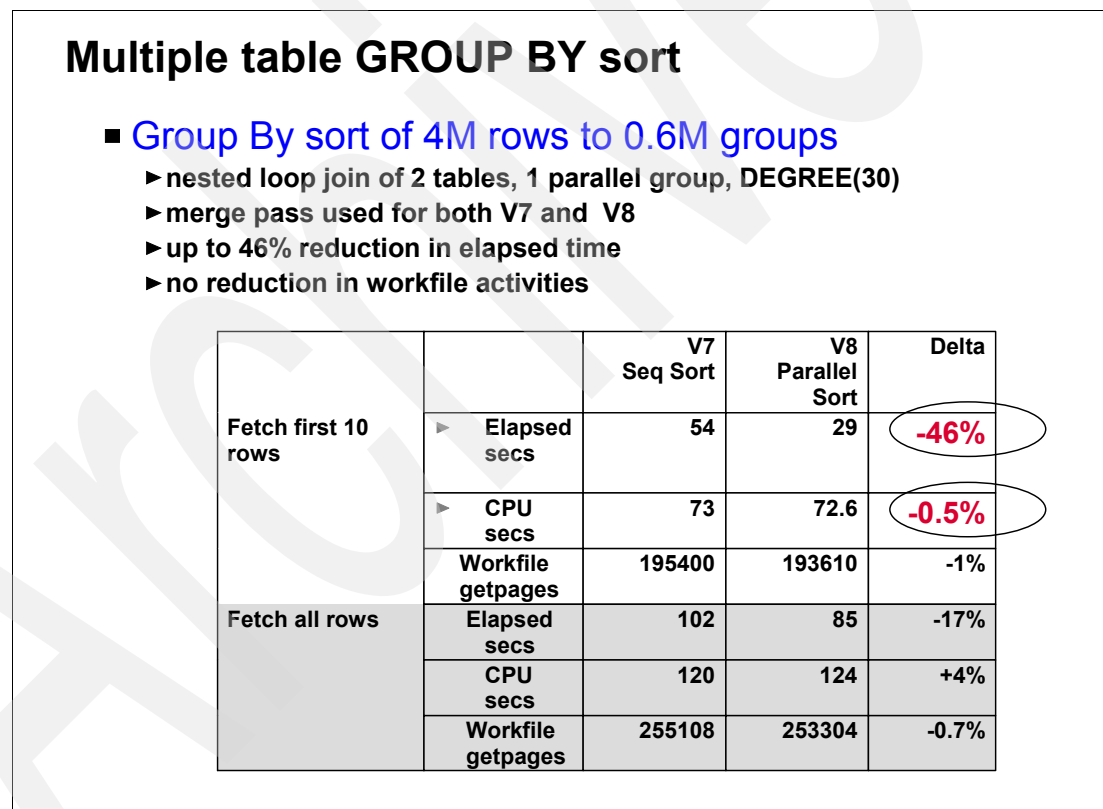


Figure 3-35 Multiple table group by sort

We observe, in the example of GROUP BY, the best situation is when we fetch the first 10 rows (reduction of 46% in elapsed time) when V8 is compared to V7.

Multiple table ORDER BY sort

Let us look at the query in Example 3-9 where the ORDER BY is requested.

Example 3-9 Multiple table ORDER BY sort query

```
SELECT C_CUSTKEY, C_NAME,
       O_ORDERKEY AS REVENUE,
       C_ACCTBAL, C_NAME, C_ADDRESS, C_PHONE, C_COMMENT
FROM CUSTOMER, ORDER
WHERE C_MKTSEGMENT = 'BUILDING'
      AND C_CUSTKEY = O_CUSTKEY
      AND O_ORDERDATE < DATE('1994-12-05')
ORDER BY C_CUSTKEY, C_NAME,
         REVENUE,
         C_ACCTBAL, C_NAME, C_ADDRESS, C_PHONE, C_COMMENT
```

Figure 3-36 shows the performance improvements for ORDER BY due to parallel sort.

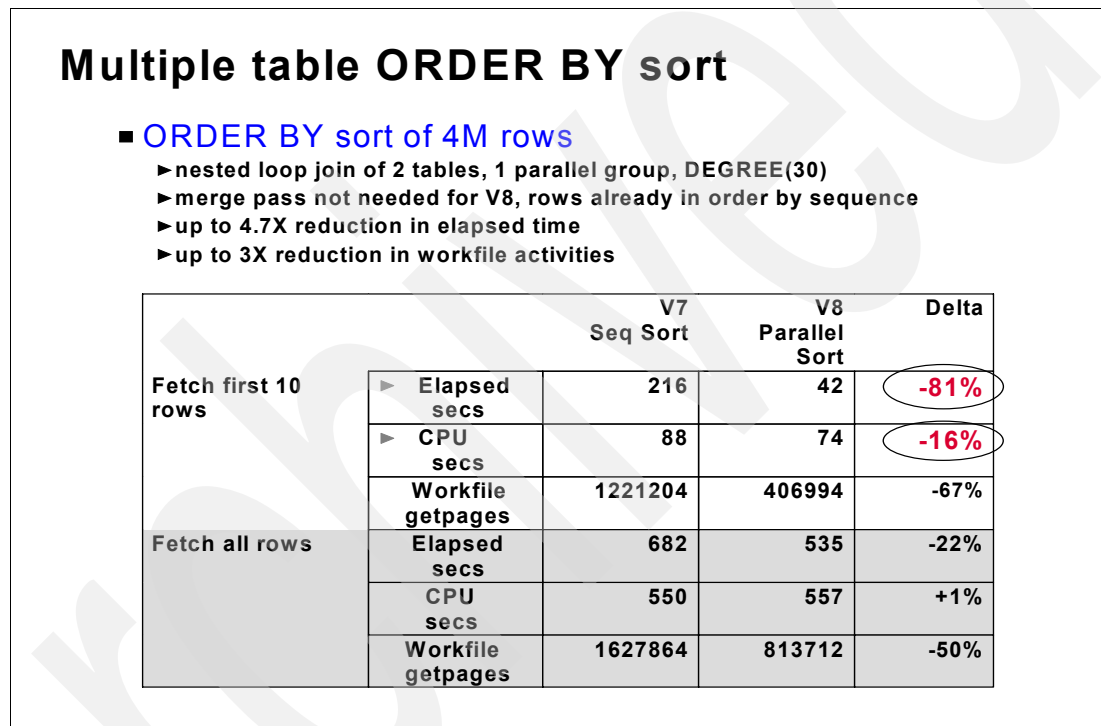


Figure 3-36 Multiple table order by sort

We observe in the example of ORDER BY the best situation is when we fetch the first 10 rows (reduction of 81% in elapsed time) when V8 is compared to V7. In this case there is also a significant reduction in the number of getpages for work files. No merge is needed because the rows are already in the right sequence.

3.7.2 Conclusions

Parallel sort can improve the elapsed time of long running queries when parallelism is used as summarized in the following list:

- ▶ Up to 6x elapsed time improvement observed with no merge pass
- ▶ Typical queries have less improvement due to merge pass
- ▶ May need bigger work file buffer pool size
- ▶ May need to allocate more work files or use parallel access volume (PAV)
- ▶ Work file getpage may increase when query has two or more parallel sort composites

3.7.3 Recommendations

Parallel sort in V8 enables parallelism for multiple-table composite, is cost-based and is disabled for short running queries. When used, it affects the use of work files.

- ▶ Parallel sort uses more work files concurrently
- ▶ May need bigger work file buffer pool size
- ▶ Monitor critical buffer pool shortage and adjust buffer pool size
- ▶ Have sufficient number of work files to reduce contention
- ▶ Use ESS DASD to benefit from PAV

3.8 Table UDF improvement

DB2 allows you to specify the cardinality option passing the expected value when you reference a user-defined table function in an SQL statement. With this option, users give DB2 the information needed for better tuning the performance of queries that contain user-defined table functions.

3.8.1 Description

The cardinality option for user-defined table functions is primarily intended for the integration of DB2 for z/OS with Content Manager to allow improved performance through better optimization.

The table UDF improvement can be achieved by the conjunction of:

- ▶ The cardinality option for a user-defined table function allows users to tune the performance of queries that contains user-defined table functions.
- ▶ A join predicate between a user-defined function and a base table is stage 1, and possibly indexable, if the base table is the inner table of the join and the other stage 1 conditions are met.
- ▶ Rows returned from a user-defined function are prefetched into a work file in its first invocation. This feature is enabled by DB2 based on the access cost estimation.

Queries involving user-defined table functions, in general, could benefit from this feature, when users can estimate the number of rows returned by the function before the queries are run. You should note that this option is a nonstandard SQL extension and specific to DB2 for z/OS implementation.

User-defined table function

The user-defined table function cardinality option indicates the total number of rows returned by a user-defined table function reference. The option is used by DB2 at bind time to evaluate the table function access cost.

A user-defined table function cardinality option can be specified for each table function reference in the following forms:

- ▶ A keyword **CARDINALITY** followed by an integer that represents the expected number of rows returned by the table function.
- ▶ A keyword **CARDINALITY MULTIPLIER** followed by a numeric constant. The expected number of rows returned by the table function is computed by multiplying the given number of the reference cardinality value that is retrieved from the **CARDINALITY** field of **SYSIBM.SYSROUTINES** by the corresponding table function name that was specified when the corresponding table function was defined.

We show three examples.

Example 3-10 shows passing a multiplier value.

Example 3-10 User-defined table function (1/3)

```
SELECT *  
FROM TABLE(TUDF(1) CARDINALITY MULTIPLIER 30) AS X;
```

If the cardinality was specified to be 1 when TUDF was defined, DB2 assumes that the expected number of rows to be returned is 30 (30 x 1) for this invocation of user-defined table function TUDF.

Example 3-11 shows a different multiplier.

Example 3-11 User-defined table function (2/3)

```
SELECT R  
FROM TABLE(TUDF(2) CARDINALITY MULTIPLIER 0.3) AS X;
```

If the cardinality was specified to be 100 when TUDF was defined, DB2 assumes that the expected number of rows to be returned by TUDF is 30 (0.3 x 100).

Example 3-12 shows passing an absolute value.

Example 3-12 User-defined table function (3/3)

```
SELECT R  
FROM TABLE(TUDF(3) CARDINALITY 30) AS X;
```

DB2 assumes that the expected number of rows to be returned by TUDF is 30, regardless of the value in the table function.

For more details on the specification of CARDINALITY and CARDINALITY MULTIPLIER, see the *table-UDF-cardinality-clause* section in *DB2 UDB for z/OS Version 8 SQL Reference*, SC18-7426.

3.8.2 Performance

Table UDFs are driven by Content Manager in all members of the DB2 family:

- ▶ Content Manager uses Text Information Extender/Text Extender to perform text search
- ▶ Results from the table UDF are joined with base or other tables

Measurements show the performance improvements for the table UDF as a result of:

- ▶ Cardinality option:
 - User/Extender can specify the cardinality of the table UDF in SQL levels
 - Better access path selection
- ▶ Table UDF block fetch support
 - Materialize the rows from a table UDF into a work file at UDF open statement
- ▶ Table UDF join predicates become sortable
 - Retrofitted into V7 as PQ54042

Figure 3-37 shows that specifying the cardinality option in SQL reduced both the elapsed and CPU time.

Table UDF – Cardinality Option

```
SELECT .....
FROM ORDER A , TABLE(TUDF(100000 )
                      CARDINALITY 100000) AS B
WHERE A.cust_key = B.int_key
      AND A.order_key BETWEEN 1 AND 1500000 ;
```

Access path is
changed from
NLJ to HBJ
with correct
cardinality
value 100000

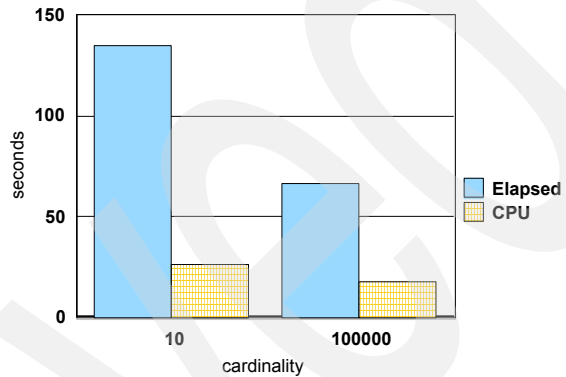
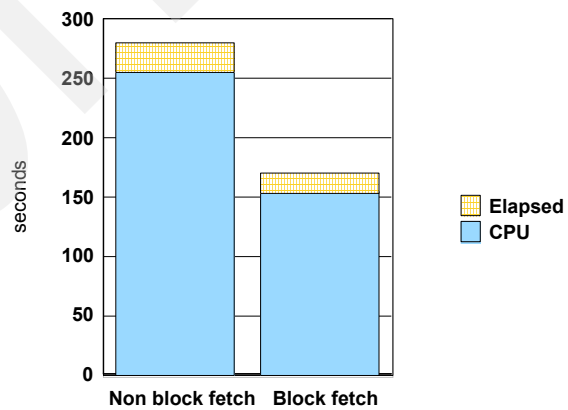


Figure 3-37 Table UDF - Cardinality option

Figure 3-38 shows the measurements with no block fetch and with block fetch.

Table UDF – Block Fetch Support



- Block Fetch support
 - Materialize the rows from a table UDF into a workfile at UDF open call
 - CPU improvement by avoiding task switches.
- Native Measurement using a simple Table UDF
 - Measured in 10 to 1M rows
 - About 40% CPU time was saved using Block Fetch support or a simple Table UDF on Freeway Turbo

Figure 3-38 Table UDF - Block fetch support

3.8.3 Conclusion

The measurements show that SQL statements using user-defined table functions can have performance improvement if using the cardinality option. Improvements apply to applications related to Content Management.

3.8.4 Recommendations

Specify cardinality to help the optimizer

Make sure you have sufficient work file buffers and data sets across I/O devices, preferably with the Parallel Access Volume feature:

- ▶ To avoid work file I/O contention, especially with other concurrently running jobs
- ▶ For high table UDF cardinality

3.9 ORDER BY in SELECT INTO with FETCH FIRST N ROWS

This function allows you to specify ORDER BY in the SELECT INTO statement. It enables SELECT INTO to get the top rows based on a user-specified ordering. This means you can now issue a single SQL statement where in the past you had to issue a number of SQL statements or even write a program.

An example would be:

```
SELECT INTO.... ORDER BY ANNUAL_INCOME FETCH FIRST 1 ROW ONLY
```

In this case up to 30% CPU time reduction can occur when compared to the Open/Fetch/Close sequence which was required in V7 when there were multiple rows to be selected, no sort was required. Generally, the entire result set is retrieved and sorted before the first row is returned.

3.10 Trigger enhancements

Prior to DB2 V8:

- ▶ The AFTER trigger causes the creation of a work file for old and new transition variables at each trigger invocation
- ▶ Work files are always created, even when the WHEN condition evaluates to false

With V8:

- ▶ Avoid work file allocation if the WHEN condition evaluates to false
- ▶ Avoid work file allocation if the transition table fits into 4 KB of working storage.
Use of memory rather than of work files for small transition tables or variable
- ▶ Work files are allocated if the transition table exceeds the working storage

Example 3-13 shows the creation of the row trigger used in the measurements.

Example 3-13 Sample row trigger creation

```
CREATE TRIGGER ADD_AFT  
AFTER INSERT ON USRT001.EMPLOYEE  
REFERENCING NEW AS NROW  
FOR EACH ROW MODE DB2SQL
```

```

WHEN (NROW.EMPDPDPTID='010')
BEGIN ATOMIC
  UPDATE USRT001.DEPTMENT
  SET DPTDPTSZ = DPTDPTSZ + 1
  WHERE DPTDPTID = NROW.EMPDPDPTID;
END;

```

The sample trigger used in the performance measurement has:

- ▶ Name: ADD_AFT
- ▶ Trigger activated: AFTER the operation
- ▶ Operation starting the trigger: INSERT
- ▶ Subject table name: USRT001.EMPLOYEE
- ▶ New transition variable correlation name: NROW
- ▶ Granularity: FOR EACH ROW
- ▶ Trigger condition: WHEN (NROW.EMPDPDPTID='010')
- ▶ Trigger body:

```

BEGIN ATOMIC
  UPDATE USRT001.DEPTMENT
  SET DPTDPTSZ = DPTDPTSZ + 1
  WHERE DPTDPTID = NROW.EMPDPDPTID;
END;

```

For more details on triggers, see *Chapter 12. Using triggers for active data* in the manual *DB2 UDB for z/OS Version 8 Application Programming and SQL Guide*, SC18-7415.

3.10.1 Performance

Performance measurements were made in two cases:

- ▶ 100 inserts activated trigger 100 times
- ▶ 100 inserts activated trigger 100 times but WHEN condition evaluated to true only once

The invocation of the triggers is via dynamic SQL. The triggers are static SQL.

100 inserts activated trigger 100 times

Figure 3-39 shows SQL and work file activity in V7 and V8 for the same SQL execution.

■ 100 Inserts activated trigger 100 times

SQL DML	TOTAL	DB2 V7 - workfile activity			DB2 V8 - workfile activity	
-----	-----	BP1	BPOOL ACTIVITY	TOTAL	BP1	no use reported
SELECT	0					
INSERT	100		BPOOL HIT RATIO (%)	100		
UPDATE	100		GETPAGES	600		
DELETE	0		GETPAGES-FAILED	0		
			BUFFER UPDATES	400		
DESCRIBE	100		SYNCHRONOUS WRITE	0		
DESC.TBL	0		SYNCHRONOUS READ	0		
PREPARE	100		SEQ. PREFETCH REQS	100		
OPEN	0		LIST PREFETCH REQS	0		
FETCH	0		DYN. PREFETCH REQS	0		
CLOSE	0		PAGES READ ASYNCHR.	0		
DML-ALL	401					

Figure 3-39 SQL and BPOOL activity - Trigger 100 times

The execution of 100 inserts activates the trigger 100 times executing update. With DB2 V7, the buffer pool BP1 used by the work files indicates 600 getpages and 400 buffer updates. In DB2 V8 there was no activity related to work files.

Figure 3-40 compares the CPU times when executing triggers in V7 and V8.

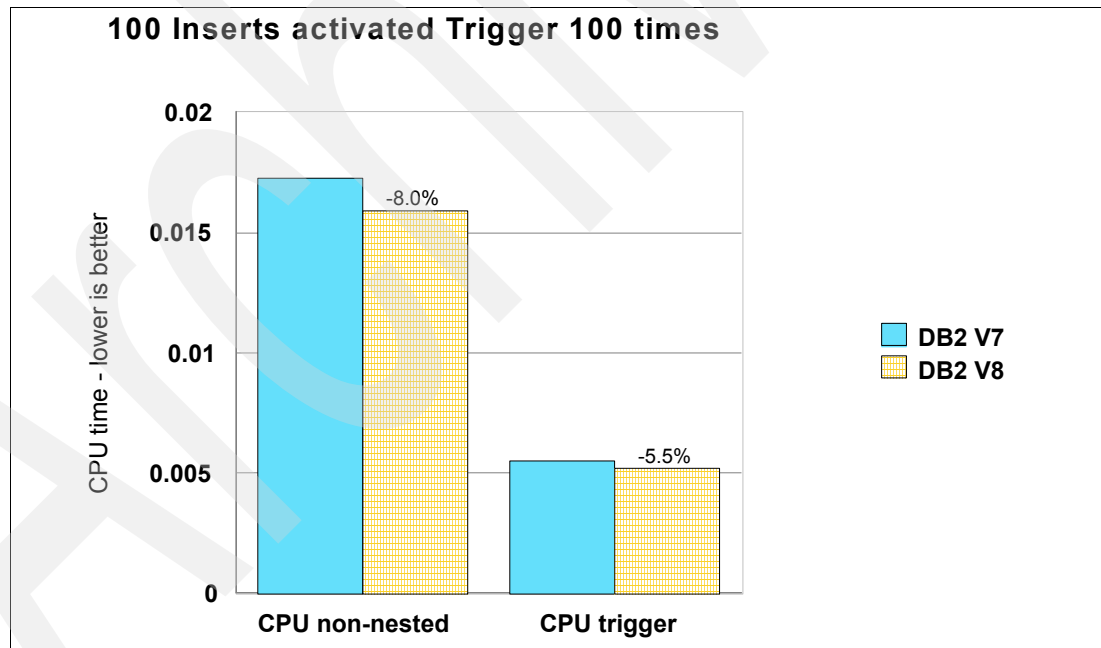


Figure 3-40 CPU time when invoking the trigger 100 times

The figure shows that the CPU used for non-nested (everything that is not in the trigger) was reduced by 8.0% and the CPU to execute the trigger was reduced by 5.5%.

Generally a larger improvement can be anticipated by avoiding work files. In this case, such improvement is partially offset by the increase in CPU usage by going to V8.

The accounting class 1 and class 2 CPU and elapsed times for triggers, stored procedures, and user-defined functions are accumulated in separate fields and exclude the time accumulated in other nested activity.

For more details, see *Chapter 24. Analyzing performance data in the DB2 UDB for z/OS Version 8 Administration Guide*, SC18-7413.

Inserts activated the trigger 100 times but WHEN condition true once

Figure 3-41 shows the measurement when 100 inserts activated the trigger 100 times but 99 times the WHEN condition evaluated false.

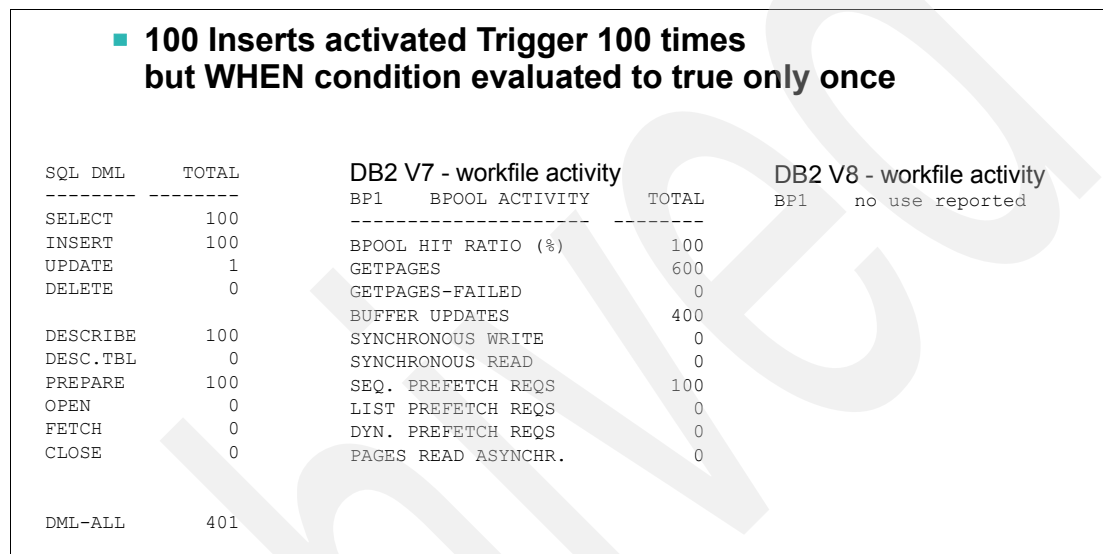


Figure 3-41 SQL and BPOOL activity - Trigger with WHEN true only once

The execution of 100 inserts caused the shown amount of work files activity, to actually execute the update in just one case. The 100 SELECTs shown in the accounting are due to the WHEN condition which causes 100 SELECTs to a DUMMY TABLE.

In V7 the buffer pool BP1 used by the work files indicates 600 getpages and 400 buffer updates. In this case there was no activity related to the work files in V8.

Figure 3-42 compares the CPU times when executing triggers in V7 and V8 and the WHEN condition is evaluated to true only once.

The figure shows that the CPU used for non-nested was reduced 8.3% and the CPU to execute the trigger is reduced by a more visible 24%.

**100 Inserts activated trigger 100 times
but WHEN condition evaluated to true only once**

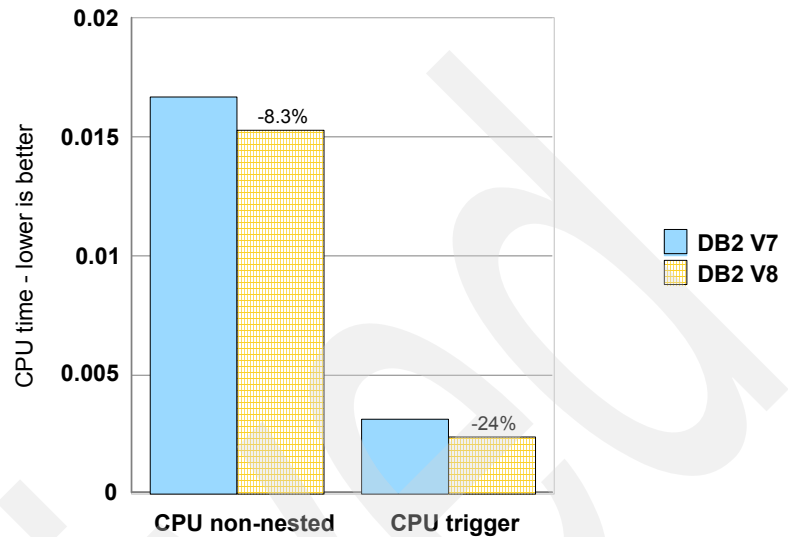


Figure 3-42 CPU time to invoke trigger but WHEN condition true once

3.10.2 Conclusion

The measurements of execution of triggers show:

- ▶ Saved work file usage
- ▶ CPU saving for avoided work file allocation if the rows fit into the work space
- ▶ Performance improvement depends on:
 - Complexity of triggering SQL
 - Complexity of SQL executed in the trigger
 - Number of times the triggers are invoked

3.10.3 Recommendation

You don't need to do anything, not even REBIND, to have the performance benefit if you are using triggers that allocate work files.

3.11 REXX support improvements

The performance of REXX programs accessing DB2 tables is improved in V8 in the case of programs issuing large numbers of SQL statements.

The improvement is due to two changes:

- ▶ DB2 V8 avoids loading and chaining control blocks for each REXX API invocation; they are now built only at initialization time and then pointed to for the successive API calls. This has reduced the CPU time.
- ▶ The REXX exec points to the DSNREXX code. The change now loads the modules and keeps them in local application address space for the duration of the job whenever they have not been preloaded in LPA. This avoids accessing the modules again from disk. This

can provide a large elapsed time improvement in all cases where the DSNREXX module is *not* already loaded in LLA/VLF to reduce the module fetch time.

3.11.1 Performance

Figure 3-43 shows the measurements of a REXX application fetching 100k rows from a table in V7 and V8. There are very significant reductions in elapsed time and some reduction in CPU time.

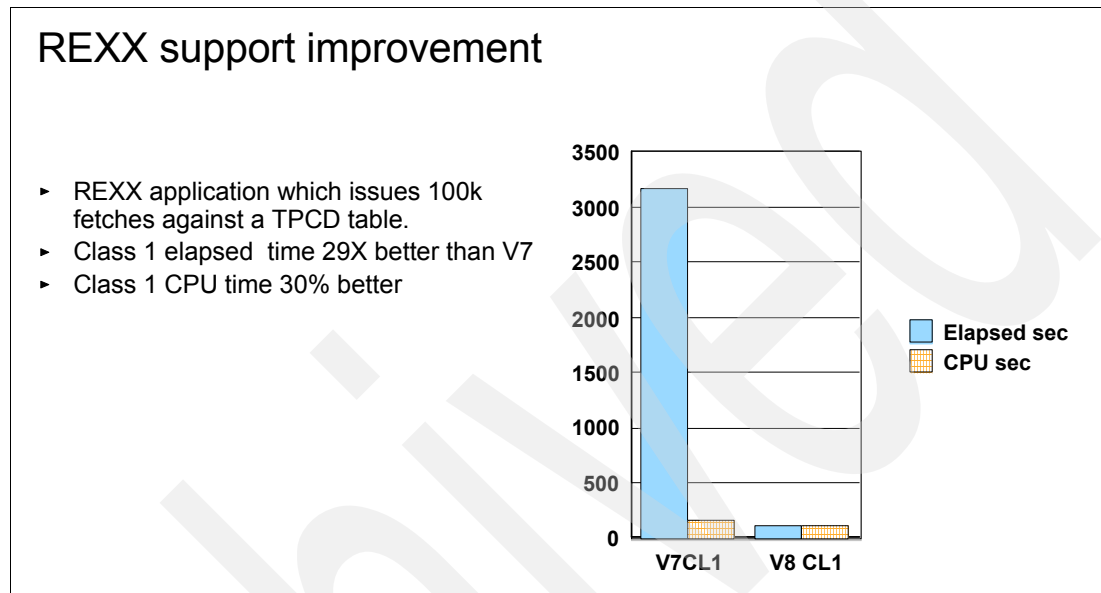


Figure 3-43 REXX support improvement

3.11.2 Conclusion

The measurements show that the elapsed time of programs in REXX language can be significantly reduced (if the DSNREXX module was not already preloaded by other means.) The improvement is larger with larger number of SQL statements.

CPU time is also reduced.

3.11.3 Recommendations

You do not need to do anything to have the performance benefits if you are using REXX programs that issue a large number of calls.

3.12 Dynamic scrollable cursors

Before DB2 V7, you could only scroll cursors in a forward position. So when a cursor was opened, the cursor would be positioned before the first row of the result set. When a fetch was executed, the cursor would move forward one row.

DB2 V7 introduced *static* scrollable cursors as cursors which can be scrolled backwards or forwards, and to an absolute position or to a relative position. They use a DECLARED TEMPORARY TABLE (DTT) introduced first in DB2 V6. When the cursor is opened, DB2

stores the selected columns from each qualifying base row in the result DTT and scrolling is performed on this temporary table. DB2 deletes the result table when the cursor is closed.

Dynamic scrollable cursors are new with DB2 V8. Dynamic scrollable cursors do not materialize the result table. Instead, they scroll directly on the base table and are therefore sensitive to all committed inserts, updates, and deletes. Dynamic scrollable cursors are supported for the index scan and table space scan access paths. DPSIs can also support dynamic scrollable cursors. Dynamic scrollable cursors can also be used with row-level operations or rowset-level operations.

3.12.1 Static scrollable cursor description

Static scrollable cursors introduced in V7 allow scrolling backwards as well as forwards. They also provide the capability to place the cursor to an absolute position or a position relative to the current cursor. In terms of performance, when a static scrollable cursor is opened, the qualifying rows are copied to a DB2-declared temporary table, also commonly known as the result table. This result table has a fixed (static) number of qualifying rows. Performance impact is based upon the number of qualifying rows that must be copied from the base table to the temporary table. Once the qualifying rows are copied to the temporary table, scrolling is then performed on this table in either the forward or backward direction, sequentially or to a specific row. When the cursor is closed, DB2 deletes the result table.

Dynamic scrollable cursors generally do not employ the usage of a temporary table. Therefore, the single most important performance difference between static scrollable cursors and dynamic scrollable cursors is the overhead in building the result table, maintaining a consistent relationship between the base and result tables with an updatable cursor, and deleting the result table at close cursor for static scrollable cursors.

Sensitivity options for static scrollable cursors

A static scrollable cursor can be sensitive or insensitive to update, and can be sensitive or insensitive to changes made by other applications or outside of the cursor. These can be specified in the DECLARE CURSOR statement and FETCH statement. Table 3-3 outlines the available combinations, whether the cursor is read only or updatable, and whether the cursor is sensitive to changes to the base table.

Table 3-3 Static cursor declare and fetch specifications

Specification on DECLARE	Specification on FETCH	Type of Cursor	Comments
Insensitive	Insensitive	Read-only	Not aware of updates, deletes and inserts to base table DB2 uses the predefined TEMP database and segmented table space to create your DTT and move in the results at execution time.
Sensitive	Insensitive	Updatable	Aware of own updates and deletes within the cursor. Changes by another agent are not visible to the cursor. Any inserts are not recognized. DB2 uses the predefined TEMP database and segmented table space to create your DTT and move in the results at execution time.

Specification on DECLARE	Specification on FETCH	Type of Cursor	Comments
Sensitive	Sensitive	Updatable	Aware of own updates and deletes within the cursor. Changes by another agent are visible to the cursor after commit. Any inserts are not recognized. DB2 uses the predefined TEMP database and segmented table space to create your DTT and move in the results at execution time.

The following is the description of the DECLARE CURSOR and FETCH options which were introduced in V7. The DECLARE CURSOR statement specifies that the cursor is scrollable by the SCROLL keyword. If SCROLL is specified, then you need to specify either INSENSITIVE or SENSITIVE STATIC. Any inserts are not visible in static scrollable cursors in every option, because for static scrollable cursors, inserts are not considered cursor operations and the result table is considered fixed after open cursor.

- ▶ If you specify INSENSITIVE in DECLARE CURSOR, the scrollable cursor is strictly read-only. In a FETCH statement you can only specify INSENSITIVE. FETCH INSENSITIVE retrieves the row from the result table. The cursor is not sensitive to updates, inserts, and deletes made to the base table. So the application cannot see the changes made through the cursor, using positioned update and delete, nor the changes made outside the cursor.
- ▶ If you specify SENSITIVE in DECLARE CURSOR, the scrollable cursor is updatable. In a FETCH statement you can specify INSENSITIVE or SENSITIVE to determine whether or not changes to the rows outside the cursor can be seen by the FETCH.
 - INSENSITIVE in a FETCH statement

Applications can see the changes they have made themselves, using positioned updates or deletes through the cursor. In this case DB2 updates both the base table and result table. However, updates and deletes to the base table made outside of the cursor are not visible to the application. The reason is that a FETCH INSENSITIVE cursor only accesses the result table, so the changes to the base table made outside the cursor are not visible.
 - SENSITIVE in a FETCH statement

Applications can see the committed updates and deletes made by other applications outside the cursor, as well as changes they have made themselves using positioned and searched updates and delete through the cursor. A FETCH SENSITIVE first refreshes the rows in the DTT to pick up any committed updates and deletes from the base table, while a FETCH INSENSITIVE just returns the appropriate row from the DTT.

Optimistic locking

The static cursor model introduces the concept of optimistic locking. A specification of Isolation Level Cursor Stability (CS) can be used with Lock Size Row such that locks are held for the minimum duration because locking is optimistic in that no positioned update or delete occurs (lock/unlock the row), and even if it does, a new lock is acquired to compare the values between the base table and temporary table. In short, optimistic locking holds locks for the minimum duration, thereby increasing concurrency for that table. In addition, for applications that compare by value, it frees these applications from coding the comparison by moving this logic to the DBMS for better performance and less application code.

While there is no optimistic locking with the dynamic scrollable cursor model, it is also recommended that CS be used to achieve maximum concurrency. Since dynamic scrollable cursors are most useful for applications that want to see updated rows as well as newly inserted rows, specifying an isolation level of RS or RR restricts updates to the table by other users.

3.12.2 Dynamic scrollable cursor description

Dynamic scrollable cursors allow visibility of inserts as well as updates done by you or other users, while static scrollable cursors do not have a visibility to inserts. The differences of dynamic scrollable cursor compared to static scrollable cursor are:

- ▶ Accesses the base tables directly
- ▶ Inserts done into the base tables are visible
- ▶ New options of DECLARE CURSOR statement:
 - DECLARE CURSOR ASENSITIVE
 - DECLARE CURSOR SENSITIVE DYNAMIC

Dynamic scrollable cursors can be used with row-level or rowset-level operations, so DDF applications should use multi-row FETCH and positioned UPDATE/DELETE for multi-row FETCH.

Sensitivity options for dynamic scrollable cursor

In this section we show the syntax of the DECLARE CURSOR and FETCH statements associated with the dynamic scrollable cursors.

DECLARE CURSOR statement

There are some new options in V8 in the DECLARE CURSOR statement to support dynamic scrollable cursors. Figure 3-44 shows the syntax of the DECLARE CURSOR.

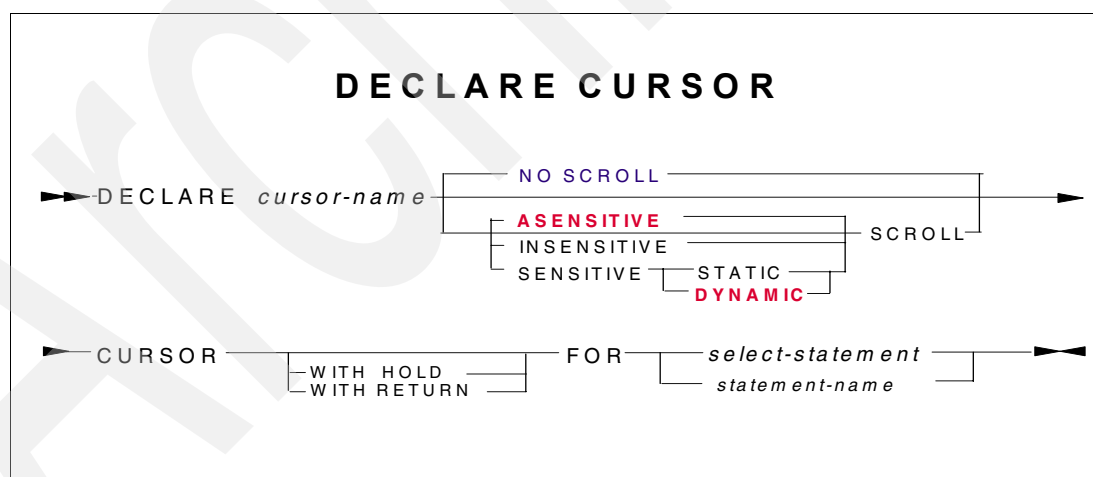


Figure 3-44 DECLARE CURSOR syntax

NO SCROLL is the default and indicates that the cursors are not scrollable. If you specify SCROLL, the cursors become scrollable. ASENSITIVE and SENSITIVE DYNAMIC are new options of scrollable cursors introduced in V8. INSENSITIVE and SENSITIVE STATIC are equivalent to static scrollable cursors in V7.

The new two attributes of dynamic scrollable cursors determine the sensitivity of the cursors.

- ▶ SENSITIVE DYNAMIC SCROLLABLE

If you specify this option, the result table of the cursor is dynamic. It means that the size of the result table is not fixed and it may change depending on the changes made against rows in the underlying table after the cursor is opened. All inserts, updates, and deletes by the same application processes through the cursor are immediately visible. Also all positioned updates and deletes are immediately visible even before commit. And inserts and updates and deletes by other applications are visible once committed. No temporary result table is created for SENSITIVE DYNAMIC, since the FETCH statements are executed against the base table.

► **ASENSITIVE SCROLLABLE**

You can use this option, when you let DB2 determine the sensitivity of the cursor. DB2 determines whether the cursor behaves as SENSITIVE DYNAMIC SCROLLABLE or INSENSITIVE (STATIC) SCROLLABLE depending on the update ability of the associated SELECT statement. ASENSITIVE CURSORS may then require TEMP database since they can resolve to INSENSITIVE cursors, depending on the SELECT statement.

- The cursor behaves as an INSENSITIVE SCROLLABLE cursor, when the SELECT statement is read-only and does not allow the cursor to be sensitive. In these cases, access to the declared temporary table may be necessary.
- The cursor provides maximum sensitivity, which is SENSITIVE DYNAMIC SCROLLABLE, when the cursor is not read-only.

It is suitable for client applications that do not care about whether or not the server supports sensitivity.

Note: There is no DYNAMIC option for INSENSITIVE. The syntax is SENSITIVE DYNAMIC SCROLLABLE cursor (the query cannot be read-only) or INSENSITIVE SCROLLABLE cursor (the query can be anything, but a temp table may be required for materialization).

The current conditions for read-only cursors are listed under Declare Cursor in Chapter 5 of *DB2 UDB for z/OS Version 8 SQL Reference*, SC18-7426-01. The READ ONLY and FETCH ONLY clauses by themselves do not require a temporary data base.

FETCH statement

There is no new option in V8 to use dynamic scrollable cursors. All the positioning keywords for fetch (NEXT, PRIOR, FIRST, LAST, CURRENT, BEFORE, AFTER, ABSOLUTE, and RELATIVE) that were introduced in DB2 V7 to support static scrollable cursors are equally relevant for dynamic scrollable cursors. However, there are some restrictions and considerations about the FETCH statement. Figure 3-45 shows the FETCH syntax.

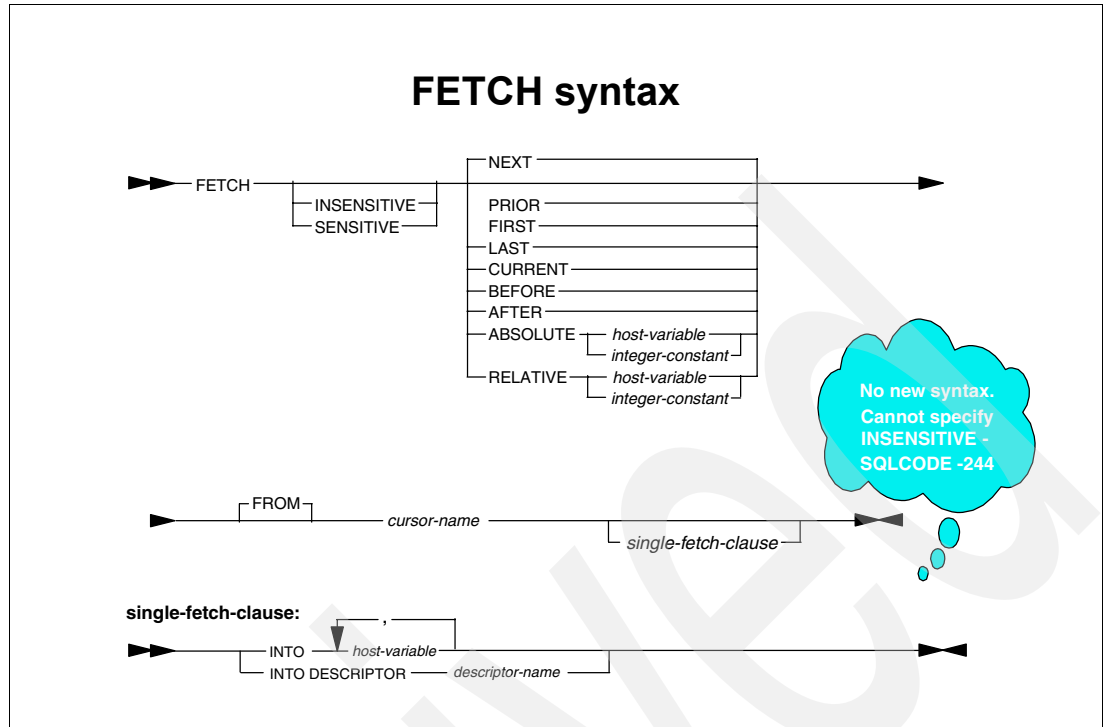


Figure 3-45 *FETCH* syntax

These are implications on the *FETCH* statement using dynamic scrollable cursors:

- ▶ The keyword **INSENSITIVE** is not allowed (SQLCODE -244) in the *FETCH* statement:
 - If the associated **DECLARE CURSOR** statement is specified as **SENSITIVE DYNAMIC SCROLL**, or
 - If the associated **DECLARE CURSOR** statement is specified as **ASENSITIVE SCROLL**, and DB2 selects **SENSITIVE DYNAMIC SCROLL** for the associated **SELECT** statement.
- ▶ SQLCODE +231 is returned in special cases
 - If **FETCH CURRENT** or **FETCH RELATIVE +0** are requested, but the row on which the cursor is positioned has been deleted or updated so that it no longer meets the criteria. It happens only when using **ISOLATION(CS)** and **CURRENT DATA(NO)**, which allows the row retrieved without taking locks.

Data concurrency in dynamic scrollable cursors

The characteristic of using dynamic scrollable cursors is that you can see immediately the newly updated or inserted or deleted data of your application. To get maximum data concurrency and get a high performance, you should use **CURRENTDATA(NO)** as a bind option and **ISOLATION(CS)** as an **ISOLATION** level.

CURRENTDATA is a bind option which helps to determine the currency of data returned by an application cursor. In a local environment, **CURRENTDATA** can only have an impact for an application bound with **ISOLATION(CS)** and executing read-only queries which do not utilize a work file. The primary benefit in using **CURRENTDATA(NO)** is performance. This is especially true in a data sharing environment; using **CURRENTDATA(NO)** should reduce lock contention and CPU utilization.

To describe the difference between CURRENTDATA(NO) and CURRENTDATA(YES), let us give you an example:

1. Program A, bound with ISOLATION(CS) CURRENTDATA(YES), fetches a row from a cursor. The row was read from the base table and not from a work file.
2. Program B tries to update the row just read by Program A. The access is not allowed because the CURRENTDATA(YES) option for Program A caused a lock to be acquired on the page or the row.
3. Program A issues a searched UPDATE of the row just fetched from step 1.

In this example you see that CURRENTDATA protects the integrity of the data read by Program A between the fetch and the searched UPDATE.

If Program A were to be bound with CURRENTDATA(NO) then Program B would be able to access the row just fetched by Program A (assuming correct timing and that lock avoidance worked). When Program A issues its searched UPDATE, it can wipe out the changes just made by Program B. So with the CURRENTDATA(NO) option, you must ensure the program has read intent only, otherwise DB2 advantages of avoiding taking a lock to increase data concurrency can be offset by data integrity issues.

Binding a plan or package with ISOLATION(CS) indicates that the data fetched by a cursor to be committed, but if the application process returns to the same page, another application might have since updated, deleted, or inserted qualifying rows. If the cursor is defined as FOR UPDATE OF, the data returned by the cursor is stable and it may not be updated by another transaction while the updateable cursor is positioned on it. If the cursor is defined as read-only (or is ambiguous), then isolation level CS ensures that the data returned is committed and the stability of the cursor is determined by the CURRENTDATA option.

If the application is bound with ISOLATION(RR) or ISOLATION(RS), the updates of the table by other users are strictly restricted, you cannot acquire the advantage of using dynamic scrollable cursors, since a row or page lock is held for all accessed rows at least until the next commit point.

3.12.3 Performance

Several measurements, using different types of programs, have been implemented. There are four scenarios. The first two cases deal with single-row processing. They compare static scrollable cursors and dynamic scrollable cursors for FETCH and UPDATE programs. The other two cases deal with multi-row processing. They compare single-row and multi-row processing when using static or dynamic scrollable cursors for FETCH and UPDATE programs.

Hardware and software environment

All the test cases use the following environment:

- Hardware: IBM z900 Series 2064-216 (z900 turbo)
 - One LPAR with 2 dedicated processors
 - 7 GB real storage
 - Dedicated ESS model 800 (ESS)
 - 4 dedicated FICON channels
- Software
 - z/OS V1.4
 - DB2 V8
 - Non-data sharing

- ▶ Measurement data collected with DB2 Performance Expert V2
 - Accounting trace class 1,2,3
 - Statistics trace class 1,3,4,5,6,8

Measurement methodology

- ▶ One table defined in one segmented table space is used for all measurements.
- ▶ All the measurements involved only table space scans (1 index is defined).
- ▶ First time effect

Bring up the DB2 subsystem before each measurement to clear the data in the buffer pool, eliminating the inconsistency of different buffer hit ratios from run to run.
- ▶ No concurrency issues

There is no other program accessing the table space during the measurements.
- ▶ Read programs are bound with ISOLATION(CS) and CURRENT DATA(NO)
- ▶ Update programs are bound with ISOLATION(RS)
- ▶ Each program issues one commit after closing the cursor

Measurement cases

- ▶ Read cursor (single row fetch) - Static vs. dynamic
- ▶ Update cursor (single row fetch and update) - Static vs. dynamic
- ▶ Multi-row FETCH using static or dynamic cursors - Single row vs. multi-row
- ▶ Multi-row FETCH and UPDATE or DELETE using static or dynamic cursors - Single row vs. multi-row

3.12.4 Test description and study

The following is the description of our four test cases and study from our measurement data. The programs are all written in PL/I.

Test scenario 1: Read cursor - Static vs. dynamic

This test uses two types of FETCH programs. The intent is to compare insensitive with asensitive cursors for static and dynamic.

The first program, shown in Example 3-14, declares an insensitive scrollable cursor to retrieve data from the table. It means this program is strictly read-only (does not allow positioned update and delete).

Example 3-14 Read program of static cursor

```

DECLARE C1 INSENSITIVE SCROLL CURSOR WITH HOLD FOR SELECT
  COL1, COL2, COL3, COL4, COL5,
  COL6, COL7, COL8, COL9, COL10
  FROM TABLE
  WHERE COL2 < 'ROWNNNNNNNNN';

DO I = 1 TO 50000;
  FETCH INSENSITIVE NEXT C1 INTO
    :COL1,:COL2,:COL3,:COL4,:COL5,
    :COL6,:COL7,:COL8,:COL9,:COL10;
END;
```

The second program, shown in Example 3-15, declares an ASENSITIVE SCROLLABLE cursor, which is the default in DECLARE CURSOR of V8. ASENSITIVE scrollable cursors allow DB2 to choose the maximum sensitivity of a cursor as possible, and in this case the following SELECT statement does not indicate read-only cursor (that is, a UNION or UNION ALL, ORDER BY, FOR READ ONLY, FOR FETCH ONLY) so DB2 determines dynamic as the maximum sensitivity.

Example 3-15 Read program of dynamic cursor

```

DECLARE C1 ASENSITIVE SCROLL CURSOR WITH HOLD FOR SELECT
  COL1, COL2, COL3, COL4, COL5,
  COL6, COL7, COL8, COL9, COL10
  FROM TABLE
  WHERE COL2 < 'ROWNNNNNNNN';

DO I = 1 TO 50000;
  FETCH NEXT C1 INTO
    :COL1,:COL2,:COL3,:COL4,:COL5,
    :COL6,:COL7,:COL8,:COL9,:COL10;
END;

```

The measurements have been done varying the number of FETCHes and the number of qualified rows opened as a cursor. This is done to show how the performance is affected by different result set sizes (the size of the temporary table in the case of the static cursor examples). The summary of our test cases is:

- ▶ Use ISO(CS) and CD(NO) as a bind parameter
For the read program these options provide maximum performance and concurrency.
 - ▶ 50k or 10 FETCHes
 - ▶ FETCH of 10 columns for a total of 50 bytes
 - ▶ Qualified 1 million or 100k rows opened as a cursor
- In the program shown in Example 3-17, we specified 1,000,000 or 100,000 in 'ROWnnnnnnnnnn' row-id.

Table 3-4 shows the test results for static and dynamic cursors from the output of the DB2 PE trace record.

Table 3-4 Read cursor - Static vs. dynamic

Trace data	Static			Dynamic		
	Fetch 50k Open 1M	Fetch 10 Open 1M	Fetch 50k Open 100k	Fetch 50k Open 1M	Fetch 10 Open 1M	Fetch 50k Open 100k
Class 1 ET/CPU	17.33/12.57	16.76/11.91	5.73 / 2.57	0.973/0.879	0.072/0.011	0.971/0.885
Class 2 ET/CPU	17.21/12.44	16.75/11.90	5.61/ 2.43	0.849/0.743	0.069/0.009	0.849/0.748
Class 3 suspend	4.29	4.4	3.02	0.058	0.054	0.059

Since the primary interest in this performance study is the DB2 Class 2 CPU time and elapsed time, the PL/I programs were designed with minimum Class 1 (application) CPU and elapsed time in mind. Therefore, the Class 1 CPU and elapsed times in all measurements is composed mainly of the DB2 Class 2 CPU and elapsed times. The typical business

application has decidedly higher Class 1 CPU and elapsed times as reported in DB2 Performance Expert.

Static read cursors

By manipulating the number of fetches from 50k to 10, but keeping the result set size the same (1 million rows) you can see that the performance does not improve significantly (just over 4% improvement). This is because the overhead for opening and formatting the temporary table (class 3 suspend time) is essentially the same for both cases (4.29 vs. 4.40) and populating it with 1 million rows to build the result set in both cases consumes the majority of CPU cycles.

However, in the third case, you can see a dramatic reduction of class 2 elapsed and CPU time when the qualified rows are changed from 1 million to 100k. In the static cursor case there are data movement I/Os from the base table to the temporary table, so in this case class 2 elapsed and CPU time are dependent on the size of the temporary table to populate. Class 3 time also decreases as the temporary table size decreases.

As a conclusion, performance improvement for static scrollable cursors can be achieved, to a certain extent, by writing SQL statements that limit the result set read into the temporary table to only those rows that are essential.

Dynamic read cursors

See Table 3-4 again for the result trace record of the dynamic cursor case. Response times in all three cases are dramatically improved when compared against the static cursor. The elimination of formatting and building a result set into a temporary table when using the dynamic cursor model accounts for this dramatic performance improvement. Thus, comparing the static vs. dynamic cursor models, the DB2 class 2 elapsed times are:

- ▶ Fetch 50k / Result Set 1 M – 17.21 vs. 0.849 for 95% performance improvement
- ▶ Fetch 10 / Result Set 1 M – 16.75 vs. 0.069 for 99% performance improvement
- ▶ Fetch 50k / Result Set 100k – 5.61 vs. 0.849 for 85% performance improvement

The important thing here is that the amount of FETCHes becomes the deciding factor for the dynamic cursor's performance, whereas the number of qualifying rows was a deciding factor in the static cursor model. When the number of FETCHes is changed from 50k to 10, the CPU time is reduced from 0.849 to 0.069 for a 92% improvement. But the difference is not as large as in the case of static cursor, when the number of qualified rows is changed.

Test scenario 2: Update cursor - Static vs. dynamic

This case tests positioned updates and deletes. The intent is to compare sensitive cursors for static and dynamic.

As the sample program shows in Example 3-16, a *static* scrollable cursor is declared as SENSITIVE STATIC and specified as FOR UPDATE. It means that the cursor works as a cursor for a query to be used in positioned updates of the table. FETCH SENSITIVE is used in the program, so in each FETCH, DB2 re-references the base table to pick up any committed updates or deletes.

Example 3-16 Update program of static cursor

```
DECLARE C1 SENSITIVE STATIC SCROLL CURSOR WITH HOLD
FOR SELECT
  COL1, COL2, COL3, COL4,
  COL5, COL6 FROM TABLE
WHERE COL2 < 'ROWNNNNNNNN'
FOR UPDATE OF COL6;
```

```

DO I = 1 TO 1000;
FETCH SENSITIVE NEXT C1 INTO
:COL1,:COL2,:COL3,:COL4,:COL5,
:COL6;
UPDATE TABLE SET COL6 = :COL6 +10 WHERE CURRENT OF C1;
END;
(Same for Deletes)

```

Example 3-17 shows the program of a *dynamic* scrollable cursor. This program specifies explicitly DYNAMIC cursor in DECLARE CURSOR because the program is for update or delete.

Example 3-17 Update program of dynamic cursor

```

DECLARE C1 SENSITIVE DYNAMIC SCROLL CURSOR WITH HOLD
FOR SELECT
COL1, COL2, COL3, COL4, COL5,
COL6 FROM TABLE
WHERE COL2 < 'ROWNNNNNNNNNN'
FOR UPDATE OF COL6;

DO I = 1 TO 1000;
FETCH SENSITIVE NEXT C1 INTO
:COL1,:COL2,:COL3,:COL4,:COL5,
:COL6;
UPDATE TABLE SET COL6 = :COL6 +10 WHERE CURRENT OF C1;
END;
(Same for Deletes)

```

As in the previous test case, we now examine UPDATES and DELETES with different numbers of qualified rows. The summary description of this test case is:

- ▶ Use ISO(RS) as a bind parameter
This option protects data integrity (not for performance).
- ▶ 100k or 10 FETCHes followed by 100k or 10 UPDATES
- ▶ 100k or 10 FETCHes followed by 100k or 10 DELETES
- ▶ FETCH of 6 columns for a total of 37 bytes
- ▶ Qualified 1 M or 50k rows opened as a cursor

The trace record results for a static and dynamic updatable cursor are listed in Table 3-5.

Table 3-5 Update cursor static vs. dynamic

Trace data	Static			Dynamic		
	UPDATE or DELETE 1k Open 1M	UPDATE or DELETE 10 Open 1M	UPDATE or DELETE 1k Open 50k	UPDATE or DELETE 1k Open 1M	UPDATE or DELETE 10 Open 1M	UPDATE or DELETE 1k Open 50k
Class 1 ET/CPU	16.44/11.51	15.75/11.16	4.95 / 1.33	0.601/0.148	0.218/0.016	0.508/0.144
Class 2 ET/CPU	16.43/11.49	15.75/11.16	4.93 / 1.32	0.589/0.135	0.215/0.014	0.496/0.131
Class 3 suspend	4.5	4.31	3.56	0.454	0.195	0.364

Static updatable cursor

You can see that, for the static cursor model, the performance characteristics are the same as for the read programs. SENSITIVE FETCHes with the static cursor refer to the base table. Buffer pool I/O to the data in the base table occurs in each FETCH. The change of the number of UPDATES or DELETES does not impact performance much when you compare the first two cases, both building a temporary table of 1 million rows. In the second case, where the SQL activity is reduced by a factor of 100, the performance improvement is only 2.9% for class 2 CPU time, and 4% for class 2 elapsed time. Additional CPU time in accounting is consumed by extracting the qualified rows and inserting them into the temporary table.

On the other hand, when the size of the temporary table is drastically reduced, but keeping the SQL activity constant, you can see a significant performance improvement: A DB2 class 2 CPU time improvement of 88.5% and a DB2 class 2 elapsed time improvement of 70%.

Dynamic updatable cursor

The same characteristics as dynamic read cursor apply for the dynamic update cursor model. When we compare Case 2 where we are only updating 10 rows and deleting 10 other rows and compare that with Case 1 where we are updating and deleting 1,000 rows each, the class 2 DB2 CPU time shows a 89% improvement and the class 2 DB2 elapsed time exhibits a 63.7% improvement. On the other hand, when we keep the SQL activity the same (Case 1 vs. Case 3) and vary the size of the result set, the class 2 DB2 CPU time only shows a 3% performance improvement, and the class 2 DB2 elapsed time (0.589 vs. 0.496) shows a 15.7% improvement. Dynamic updatable cursor performance is greatly dependent on the amount of SQL activity.

Comparing static updatable cursors to dynamic updatable cursors using class 2 DB2 elapsed times for the metric, you see:

- ▶ UPDATE or DELETE 1k, result set 1 million – 16.43 vs. 0.589 for 96% performance improvement
- ▶ UPDATE or DELETE 10, result set 1 million – 15.75 vs. 0.215 for 99% performance improvement
- ▶ UPDATE or DELETE 1k, result set 50k – 4.93 vs. 0.496 for 90% performance improvement

Test scenario 3: Multi-row processing with scrollable cursors

DB2 V8 introduces multiple-row processing for both the FETCH and INSERT statements. Fetching multiple rows of data with one SQL statement can be done with both scrollable (static and dynamic) and non-scrollable cursors. This enhancement helps to lower the SQL statement execution cost by reducing the number of trips between the application and database engine, and in a distributed environment reducing the number of send and receive network messages as well.

Since the same performance principles apply regarding the size of the temporary table in the static cursor model and the size of the result set in the dynamic model, regardless of the single fetch or multi-fetch implementation, we only consider a temporary table size of 1 M for the static cursor model and a result set size of 1 M for the dynamic cursor model in the following two test cases.

We have discussed multi-row performance in 3.1, “Multi-row FETCH, INSERT, cursor UPDATE and DELETE” on page 31. So in this chapter we only describe the performance of multi-row FETCH and INSERT used in conjunction with static and dynamic scrollable cursor.

Single row vs. multi-row FETCH

To test multi-row FETCH in static and dynamic scrollable cursors, we used two PL/I programs. BIND options and sensitivity options for the DECLARE CURSOR and FETCH statements are kept the same as the single row read cursor program. Each of the multi-row read programs fetches 100 rows (the row set size) with a single fetch and performs a total of 500 fetches to be compared against the original read program doing 50,000 individual fetches.

Example 3-18 shows a sample of our program for multi-row FETCH using a static scrollable cursor.

Example 3-18 Multi-row FETCH program of a static scrollable cursor

```
DECLARE C1 INSENSITIVE SCROLL CURSOR WITH ROWSET POSITIONING WITH HOLD FOR SELECT
    COL1, COL2, COL3, COL4, COL5,
    COL6, COL7, COL8, COL9, COL10
FROM TABLE
WHERE COL2 < 'ROWNNNNNNNN';
FETCH INSENSITIVE FIRST ROWSET FROM C1 FOR 100 ROWS INTO
    :COL1,:COL2,:COL3,:COL4,:COL5,
    :COL6,:COL7,:COL8,:COL9,:COL10;
DO I = 1 TO 499;
FETCH INSENSITIVE NEXT ROWSET
    C1 INTO
    :COL1,:COL2,:COL3,:COL4,:COL5,
    :COL6,:COL7,:COL8,:COL9,:COL10;
END;
```

Example 3-19 shows a sample of our program for multi-row FETCH using a *dynamic* scrollable cursor.

Example 3-19 Multi-row FETCH program of a dynamic scrollable cursor

```
DECLARE C1 ASENSITIVE SCROLL CURSOR WITH ROWSET POSITIONING WITH HOLD FOR SELECT
    COL1, COL2, COL3, COL4, COL5,
    COL6, COL7, COL8, COL9, COL10
FROM TABLE
WHERE COL2 < 'ROWNNNNNNNN';
FETCH FIRST ROWSET FROM C1 FOR
100 ROWS INTO
:COL1,:COL2,:COL3,:COL4,:COL5,
:COL6,:COL7,:COL8,:COL9,:COL10;
DO I = 1 TO 499;
FETCH NEXT ROWSET
    C1 INTO
    :COL1,:COL2,:COL3,:COL4,:COL5,
    :COL6,:COL7,:COL8,:COL9,:COL10;
END;
```

In this test case we have not measured a different number of FETCHes or a different number of qualified rows, because the purpose of this case is the comparison of multi-row FETCH with single row FETCH. The summary of this test case is:

- ▶ Use ISO(CS) and CD(NO) as a bind parameter
These options provide maximum performance and concurrency for read programs.
- ▶ 100 rows (rowset) of 500 FETCHes
- ▶ Qualified 1 million at open cursor

Multi-row FETCH with a static cursor

We apply multi-row fetching to the read programs and check how much we can improve performance. The 50,000 individual fetch case (listed in Case 1 in Table 3-4 on page 99) is compared with a multi-fetch program that fetches 100 rows at a time 500 times (listed in Table 3-6).

Since the 1 million row temporary table is built row by row, regardless of the fetch implementation, and since we learned that this activity takes up the majority of CPU cycles in the prior static cursor measurements (whether read or updatable programs), using the multi-fetch implementation has only improved the DB2 class 2 CPU time by 3% (12.44 vs. 12.05) and the DB2 class 2 elapsed time by 2% (17.21 vs. 16.85).

Table 3-6 Multi-row FETCH static

	Static single row FETCH 50,000 Open 1 million	Static multi-row FETCH 100 rows 500 times Open 1 million
Class 1 ET / CPU	17.33 / 12.57	16.85 / 12.06
Class 2 ET / CPU	17.21 / 12.44	16.84 / 12.05
Class 3 suspend	4.29	4.34

Multi-row FETCH with a dynamic cursor

The results of applying the same multi-fetch implementation to the dynamic cursor model are listed in Table 3-7. If we compare them to those of the 50,000 individual fetch program in Table 3-4 on page 99, Case 1, we see a large performance improvement. Both the DB2 class 2 elapsed time and CPU time have been cut in half: 0.743 vs. 0.340 and 0.849 vs. 0.426.

Reducing the number of trips between the application and database engine by a factor of 100 (50,000 fetches vs. 500 multi-fetches) accounts for the performance gains observed.

Table 3-7 Multi-row FETCH dynamic

	Static single row FETCH 50,000 Open 1 million	Static multi-row FETCH 100 rows 500 times Open 1 million
Class 1 ET / CPU	0.973 / 0.879	0.440 / 0.343
Class 2 ET / CPU	0.849 / 0.743	0.426 / 0.340
Class 3 suspend	0.058	0.064

Test scenario 4: Single row vs. multi-row FETCH & UPDATE or DELETE

In this case we explored multi-row FETCH followed by UPDATE or DELETE. The multi-row write programs for update and delete use a row set size of 25 rows and perform 40 fetches each. The multi-row updates or deletes are also performed 25 rows at a time. So it is equivalent to 1,000 rows FETCHed and UPDATED or DELETED. We then compare it against single row FETCH and UPDATE or DELETE of 1,000 rows which was done as part of test scenario 2. Example 3-20 shows our multi-row FETCH and UPDATE program using a *static* scrollable cursor. Multi-row FETCH and DELETE programs are prepared similarly.

Example 3-20 Multi-row FETCH program of a static scrollable cursor

```
DECLARE C1 SENSITIVE STATIC SCROLL CURSOR WITH ROWSET POSITIONING WITH HOLD FOR SELECT
  COL1, COL2, COL3, COL4,
  COL5, COL6 FROM TABLE
  WHERE COL2 < 'ROWNNNNNNNNN'
```



```

FOR UPDATE OF COL6;
FETCH SENSITIVE FIRST ROWSET FROM C1 FOR 25 ROWS INTO
:COL1,:COL2,:COL3,:COL4,:COL5,:COL6;
UPDATE TABLE SET COL6 = :COL6+10 WHERE CURRENT OF C1;
DO I = 1 TO 39;
FETCH SENSITIVE NEXT ROWSET FROM C1 INTO
:COL1,:COL2,:COL3,:COL4,:COL5,:COL6;
UPDATE TABLE SET COL6 = :COL6 +10 WHERE CURRENT OF C1;
END;
(Same for Deletes)

```

Example 3-21 shows our multi-row FETCH and UPDATE program using a *dynamic* scrollable cursor.

Example 3-21 Multi-row FETCH program of a dynamic scrollable cursor

```

DECLARE C1 SENSITIVE DYNAMIC SCROLL CURSOR WITH ROWSET POSITIONING WITH HOLD FOR SELECT
COL1, COL2, COL3, COL4, COL5,
COL6 FROM TABLE
WHERE COL2 < 'ROWNNNNNNNNN'
FOR UPDATE OF COL6;
FETCH SENSITIVE FIRST ROWSET FROM C1 FOR 25 ROWS INTO
:COL1,:COL2,:COL3,:COL4,:COL5,:COL6;
UPDATE TABLE SET COL6 = :COL6+10 WHERE CURRENT OF C1;
DO I = 1 TO 39;
FETCH SENSITIVE NEXT ROWSET FROM C1 INTO
:COL1,:COL2,:COL3,:COL4,:COL5,:COL6;
UPDATE TABLE SET COL6 = :COL6 +10 WHERE CURRENT OF C1;
END;
(Same for Deletes)

```

The summary of this test case is:

- Use ISO (RS) as a bind parameter
These options protect data integrity (not for performance).
- 40 FETCHes followed by UPDATE on a rowset of 25 rows
- 40 FETCHes followed by DELETE on a rowset of 25 rows
- Qualified 1 million rows at open cursors

Multi-row FETCH and UPDATE or DELETE with a static cursor

We now move to the static updatable programs and apply multi-row update and delete function. The results are shown in Table 3-8. We can see the same performance apply to this type of program, the bulk of the performance overhead is attributable to the building of the temporary table, resulting in negligible differences in DB2 class 2 elapsed and CPU times: 16.43 vs. 16.39 and 11.49 vs. 11.44.

Table 3-8 Multi-row FETCH and UPDATE or DELETE static

	Static single row UPDATE or DELETE 1k each Open 1M	Static multi-row UPDATE or DELETE 25 rows x 40 times each Open 1M
Class 1 ET / CPU	16.44 / 11.51	16.40 / 11.45
Class 2 ET / CPU	16.43 / 11.49	16.39 / 11.44
Class 3 suspend	4.50	4.51

Multi-row FETCH and UPDATE or DELETE with a dynamic cursor

We now apply the multi-row update and delete function to the updatable programs in the dynamic cursor model. The results are shown in Table 3-9. They show a 17.8% performance improvement in DB2 class 2 CPU time (0.135 vs. 0.111) and a 7% performance improvement in DB2 class 2 elapsed time (0.589 vs. 0.547).

Table 3-9 Multi-row FETCH and UPDATE or DELETE dynamic

	Dynamic single row UPDATE or DELETE 1k each Open 1M	Dynamic multi-row UPDATE or DELETE 25 rows x 40 times each Open 1M
Class 1 ET / CPU	0.601 / 0.148	0.582 / 0.115
Class 2 ET / CPU	0.589 / 0.135	0.547 / 0.111
Class 3 suspend	0.454	0.433

Multi-row update and delete with the dynamic cursor model result in significant performance improvements over the static cursor model, but the multi-row fetch for update programs does not improve as dramatically as the multi-row fetch for read-only programs. Compare them to Table 3-7 on page 104. This is because with the read-only programs for multi-row fetch the bulk of the DB2 CPU time is the reduction of the number of trips between the application and database engine, which has been reduced by a factor of 100 (fetching 100 rows 500 times vs. 50,000 individual fetches). When we apply multi-row fetch to the update program, the DB2 CPU and elapsed times are not only accounting for the reduced number of trips between the application and database engine for the fetches, but also the update (1,000 rows performed 25 rows at a time) and delete (1,000 rows also performed 25 rows at a time) activity. The update and delete activity is performed on a rowset basis rather than individually, but it is a significant consumer of CPU cycles due to its complexity.

3.12.5 Conclusion

For equivalent function, the dynamic cursor model generally outperforms the static cursor model. The performance overhead to build the temporary table with static cursors is the largest contributor of CPU overhead. For the dynamic cursor model, since all accesses are performed on the base table, the amount of SQL activity (from a DB2 class 2 perspective) becomes the deciding factor.

When multi-row processing is applied to static cursors (whether read or updatable), since the temporary table is built row-by-row, regardless of the fetch implementation, this new feature with DB2 V8 does not contribute significantly, at least with this amount of DML activity. But for dynamic read cursors, multi-row fetch shows performance gains of 50% for both DB2 class 2 CPU and elapsed times over the single fetch equivalent program. While the dynamic multi-row fetch and cursor update and delete program show significant improvement over its single fetch, update and delete equivalent program (17.8% improvement in DB2 class 2 CPU time), it is not as high as just the multi-row fetch program due to the intrinsically more complex and costly functions of multi-row update and multi-row delete. In addition, the row set size for the read-only multi-row fetch program was bigger (100 vs. 25) and resulted in less interaction between the application and database engine.

3.12.6 Recommendations

With a static cursor, class 3 time is heavily dependent on the size of the temporary table for both read and updates. So it is important for the performance of a static scrollable cursor to

manage the number of qualifying rows and columns whenever possible. You should write SQL statements that limit the range of rows that are read into the temporary table.

Although from a performance point of view dynamic scrollable cursors always outperform the static cursor model, we do not recommend that the dynamic cursor model should always be used over the static cursor model. As mentioned previously, some customers appreciate the concept of optimistic locking (static cursor model only) since it contributes to concurrency.

The static cursor model should be considered if you have a requirement for an application that must scroll back to a prior screen and retain and display the same values as before. Also, for gathering certain reports, such as average outstanding balance in a certain zip code, where the application programmer is not concerned about the minute-by-minute updates to the outstanding balance or the newly inserted or deleted accounts, the static cursor model with fetch insensitive can build these rows in a temporary table and permit the application to manipulate the rows as necessary, allowing concurrency to the base table.

Besides performance, dynamic scrollable cursors are preferred when it is important for the application to see/access updated as well as newly inserted rows. Maximum concurrency can be achieved with isolation cursor stability.

The summary of the recommendations for static and dynamic cursors is:

- ▶ Static scrollable cursors can be the better solution if:
 - You need to scroll backwards to obtain initial values
 - You do not care about constant changes (such as business intelligence or representative samples).
- ▶ Dynamic scrollable cursors can be preferable in situations where:
 - It is important for the application to access/see updated/inserted rows
 - Performance is important
- ▶ Use multi-row operations whenever possible, particularly for the dynamic cursor model.

3.13 Backward index scan

With the enhancements introduced to support dynamic scrollable cursors, DB2 also provides the capability for backward index scans. This allows DB2 to avoid a sort and it can eliminate the need for an index. With this enhancement, available in CM, it is no longer necessary to create both an ascending and descending index on the same table columns.

To be able to use the backward index scan, you must have an index on the same columns as the ORDER BY and the ordering must be exactly opposite of what is requested in the ORDER BY.

For example, if you have created the index

```
CREATE UNIQUE INDEX <index> ON <table> (A DESC, B ASC)
```

then DB2 V8 can do a forward index scan for a

```
SELECT ... FROM <table> ... ORDER BY A DESC, B ASC
```

and a backward index scan for

```
SELECT ... FROM <table> ... ORDER BY A ASC, B DESC
```

While, for

```
SELECT ... FROM <table> ... ORDER BY A DESC, B DESC
```

or

```
SELECT ... FROM <table> ... ORDER BY A ASC, B ASC
```

DB2 still has to perform a sort.

Table 3-10 shows a set of test cases, in which we compare queries under V7 to V8. We assume that an ascending index on EMPNO has been created for table EMP. Notice that the access type in the PLAN_TABLE is I in both directions.

Table 3-10 Access type and sort with backward index scan

Query	DB2 Version	Access type in PLAN_TABLE	Notes
SELECT EMPNO FROM EMP ORDER BY EMPNO ASC	V7	I	Base case
SELECT EMPNO FROM EMP ORDER BY EMPNO DESC	V7	I	Sort needed
SELECT EMPNO FROM EMP ORDER BY EMPNO ASC	V8	I	Same as V7
SELECT EMPNO FROM EMP ORDER BY EMPNO DESC	V8	I	Sort no longer required

3.14 Multiple distinct

Prior to DB2 V8, DISTINCT cannot be used more than once in a subselect, with the exception of its use with a column function whose expression is a column. That is, it is allowed to specify multiple DISTINCT keywords as an exception on the same column like:

- ▶ SELECT COUNT(DISTINCT(C1)), AVG(DISTINCT(C1))

However, if you use DISTINCT keywords more than once specifying different columns, you get SQLCODE -127 which means "DISTINCT IS SPECIFIED MORE THAN ONCE IN A SUBSELECT" occurs. For instance, in V7 you cannot have the following statement:

- ▶ SELECT COUNT(DISTINCT(C1)), AVG(DISTINCT(C2))

With DB2 V8, you can use multiple DISTINCT keywords in the SELECT statement and HAVING clause. With DB2 V8, the following queries are possible:

- ▶ SELECT DISTINCT COUNT(DISTINCT C1), SUM(DISTINCT C2) FROM T1
- ▶ SELECT DISTINCT COUNT(DISTINCT C1), COUNT(DISTINCT C2) FROM T1 GROUP BY A3
- ▶ SELECT COUNT(DISTINCT C1), AVG(DISTINCT C2) FROM T1 GROUP BY C1
- ▶ SELECT COUNT(DISTINCT(A1)) FROM T1 WHERE A3 > 0 GROUP BY A2 HAVING AVG(DISTINCT (A4)) > 1

In these cases in V7 you have to divide the statements into multiple SELECT statements for each distinct operation of columns. From the performance point of view, you have to execute multiple SELECT statements to retrieve the value in multiple columns.

In V8 you can code SQL statements easily in one query when multiple distinct operations need to be done for multiple columns before any column functions, such as AVG, COUNT, or SUM are applied. This multiple distinct enhancement in V8 benefits usability of SQL by reducing the amount of SQL statements or simplifying some queries. And it supports DB2 family compatibility by making it easier to move your programs to another platform. Performance improvements are experienced when the multiple SQL statements are merged into one.

3.14.1 Performance

The purpose of these measurements is to evaluate the improvement you can get by using a single query statement specifying multiple DISTINCT keywords for multiple columns compared to the case of using several query statements, each with one DISTINCT keyword. In these tests we SELECT columns of all DECIMAL data and GROUP BY an indexed column.

First we measure the following sequence of four SELECT statements performing only one SELECT COUNT(DISTINCT) for each one of the four columns (no buffer pool flush in between):

1. SELECT COUNT(DISTINCT C1), C5 FROM T1 GROUP BY C5;
2. SELECT COUNT(DISTINCT C2), C5 FROM T1 GROUP BY C5;
3. SELECT COUNT(DISTINCT C3), C5 FROM T1 GROUP BY C5;
4. SELECT COUNT(DISTINCT C4), C5 FROM T1 GROUP BY C5;

Then we measure the performance of multiple distinct statements specifying more than one DISTINCT keyword. These are the alternatives in DB2 V8 to the above SELECT statements. We have three cases of using multiple COUNT(DISTINCT) on two, three, and four columns:

5. SELECT COUNT(DISTINCT C1), COUNT(DISTINCT C2), C5 FROM T1 GROUP BY C5;
6. SELECT COUNT(DISTINCT C1), COUNT(DISTINCT C2), COUNT(DISTINCT C3), C5 FROM T1 GROUP BY C5;
7. SELECT COUNT(DISTINCT C1), COUNT(DISTINCT C2), COUNT(DISTINCT C3), COUNT(DISTINCT C4), C5 FROM T1 GROUP BY C5;

Figure 3-46 shows the percentage of improvement with the **multiple distinct** statements compared to the total time of the single distinct statements. We are comparing the total time of test cases 1 and 2 with test case 5, the sum of 1, 2, and 3, to test case 6, and the sum of 1, 2, 3, and 4, to test case 7.

The results show the percentage of the reduction of class 2 elapsed time and CPU time by using multiple distincts. When we merge two SELECT single distinct statements into one multiple distinct statement, there is a 40% elapsed time reduction. When we merge four SELECT single distinct statements into one multiple distinct statement, we observe more than 55% elapsed time reduction.

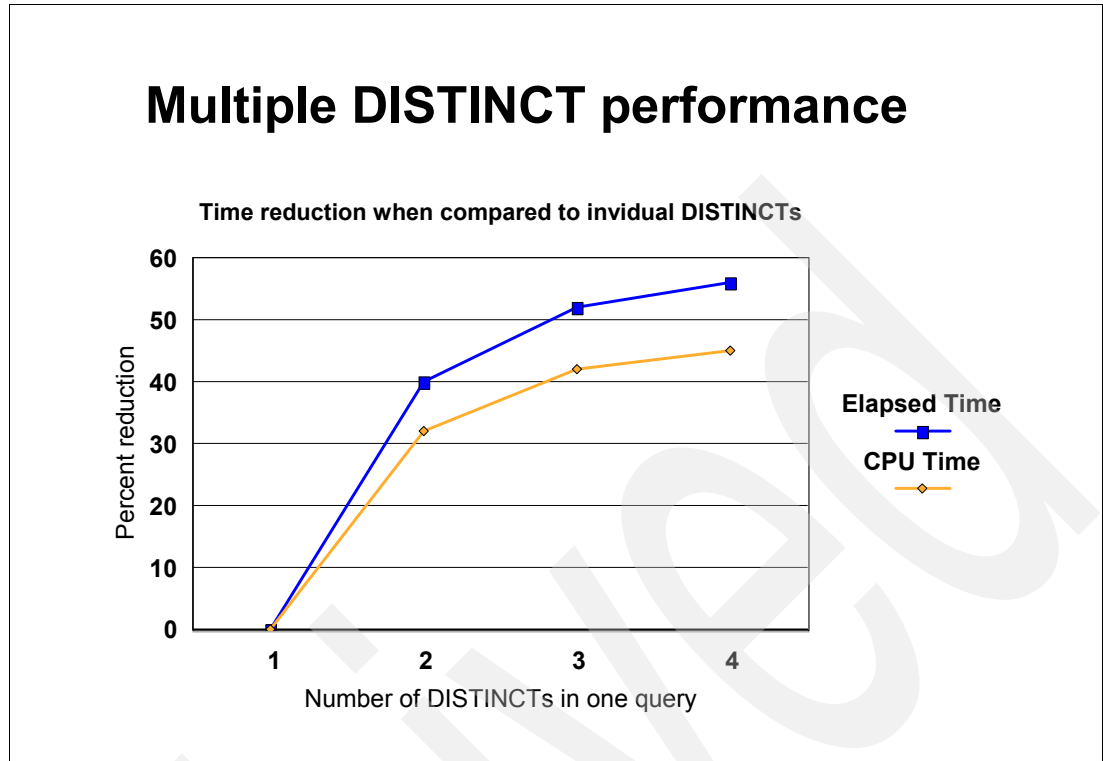


Figure 3-46 Multiple DISTINCT performance

3.14.2 Conclusion

Multiple DISTINCT keywords in a single SELECT statement in V8 can largely enhance application performance, as well as usability of SQL. Specifying four distinct columns in one statement can make approximately 55% elapsed time improvement and 45% CPU improvement compared to the total time of the four individual statements. This improvement is due to reducing the overhead of multiple executions of queries and performing multiple sorts for multiple distinct columns.

3.14.3 Recommendations

Use multiple DISTINCT keywords in a single SELECT statement in your application as often as possible, eliminating unnecessary multiple queries. Performance advantages were found where there more than two DISTINCT clauses in one query.

3.15 Visual Explain

Visual Explain for DB2 V8 provides graphical depictions of the access plans that DB2 chooses for your SQL queries and statements. Such graphs eliminate the need to manually interpret the PLAN_TABLE output filled by DB2 EXPLAIN. See Appendix D, “EXPLAIN and its tables” on page 407 for its contents and the contents of the other EXPLAIN tables. The relationships between database objects, such as tables and indexes, and operations, such as table space scans and sorts, are clearly illustrated in the graphs. You can use this information to tune your queries.

Statistics Advisor is a brand new function of Visual Explain, and it is also available from the Web. Its main function is to help you identify the RUNSTATS executions that help your SQL.

You can also use Visual Explain:

- ▶ To generate customized reports on explicable statements
- ▶ To view subsystem parameters
- ▶ To view data from the plan table, statement table, and function table
- ▶ To list SQL statements stored in the catalog (SYSSTMT and SYSPACKSTMT)

Note: All these functions can be executed with a click in the corresponding icon in the Visual Explain icon bar.

You can link to these suggestions directly from the graph. You can also link from the graph to additional statistics and descriptions for each object or operation that is used in the access plan.

Each graph can display multiple query blocks, so that you can view the entire access plan in one graph. In previous versions of Visual Explain, each graph displayed only one query block. In this version, you still have the option to view only one query block at a time. You can use Visual Explain to catalog and uncatalog databases on your local machine, to list DSNZPARM parameters by installation field name, alphabetically by parameter name, or by installation panel name. You maintain Visual Explain by deleting old records from the EXPLAIN tables.

With DB2 V8 Visual Explain, you can save the access path graph for later use, because Visual Explain uses XML to store the graph information. Every time that you EXPLAIN a statement, the access path graph is displayed on the Access Plan page of the Tune SQL notebook. On this page, select File → Save XML File to save the XML file to your local work station. To retrieve the graph, select File → Open XML File. You can also send the XML file to other people. If they have DB2 Visual Explain installed, they can also see the graph.

Note: Visual Explain has enhanced capabilities related to qualified rows estimates. It provides a wealth of predicate information, more partition scan information, and parallelism details.

Visual Explain uses some sample stored procedures:

- ▶ DSNWZP

This stored procedure is required to support the Browse Subsystems Parameters function. The stored procedure is also used by Service SQL for collection and transmission of subsystem parameter settings to IBM service.

- ▶ DSN8EXP

This stored procedure (see also APARs PQ90022 and PQ93821) supports the EXPLAIN with stored procedure option. The purpose of EXPLAIN with stored procedure is to allow you to EXPLAIN SQL against objects against which you do not have the authority to execute queries. For example, a developer may have responsibilities for SQL performance on a payroll application, but may not have SELECT, INSERT, UPDATE, and DELETE access to the object. The developer can use the stored procedure to execute the Explain against the object. The developer does not need access to the objects, but the authority to execute the stored procedure.

- ▶ DSNUTILS

Statistics Advisor generates RUNSTATS statements. These RUNSTATS statements can be executed dynamically if you have a license to execute RUNSTATS, the DSNUTILS stored procedure is installed, and you have appropriate authority to execute the RUNSTATS

commands. Of course, the RUNSTATS input can also be copied into a batch job and executed outside of Statistics Advisor.

Installing the Java Universal Driver on DB2 V8 provides Visual Explain with the stored procedures used by Visual Explain. See APAR PQ62695. Additionally, the stored procedure SYSIBM.SQLCMESSAGE is installed. This provides error message descriptions for SQL errors in Visual Explain. For more information, see the book *DB2 for z/OS and OS/390: Ready for Java*, SG24-6435.

3.15.1 Installation

Visual Explain is available for download from the Web site:

<http://www.ibm.com/software/data/db2/zos/osc/ve/index.html>

It is also part of the DB2 Management Client Package (no charge feature of DB2) on a CD-ROM. After the install, you need to set up a JDBC connection. You can do so by using the restricted-use copy of DB2 Connect Personal Edition V8 for Windows which is provided with the Management Client Package.

Instead of using the Configuration Assistant tool that comes with DB2 Connect, you can set up your connection via Visual Explain. Make sure the DB2 Connect packages have been previously bound on your target DB2 system. If you are using a TCP/IP connection to the host, you need to have the port number from your subsystem. The configuration is very simple and straightforward and all required information can be entered on a single window.

Visual Explain has a Help function which describes its features. For more details on the results provided by Visual Explain, and for more information on how to use it, refer to section 10.19, *Visual Explain enhancements of DB2 UDB for z/OS Version 8: Everything You Ever Wanted to Know,... and More*, SG24-6079.

Note: When you run Visual Explain for the first time, Visual Explain tells you if you do not have the following tables: DSN_STATEMENT_TABLE, PLAN_TABLE, DSN_DETCOST_TABLE, DSN_PREDICAT_TABLE, DSN_STRUCT_TABLE, DSN_FILTER_TABLE, DSN_SORT_TABLE, DSN_SORTKEY_TABLE, DSN_PGROUP_TABLE, DSN_PTASK_TABLE, DSN_PGRANGE_TABLE. And if you do not have them, Visual Explain creates them for you.

3.15.2 Visual Explain consistency query analysis

As an example, we are going to use Visual Explain to EXPLAIN one of the queries used for catalog consistency checking. These queries are shipped in SDSNSAMP(DSNTESTQ) to help you evaluate the *health* of your DB2 catalog.

But first we describe the access plan graph. This graph can contain three types of nodes:

- Data source nodes

A data source node represents a physical data store in the database, such as a table or index.

- Operator nodes

An operator node indicates an operation, such as a nested loop join, that is performed on the data source, or on the result of a previous operation

- Auxiliary nodes

An auxiliary node includes any nodes that are neither data source nodes nor operator nodes.

Both query nodes and query block nodes are classified as auxiliary nodes. Most of the nodes have corresponding descriptors, which store detailed information about the data source, query, or operation that is represented by the node. The Descriptor window lists all of the attribute values for the data source, query, or operation that is represented by the node. From the Descriptor window, you can also link to any related descriptors. For example, from the descriptor for a table node, you can link to the descriptors for the indexes that are defined on that table.

We use the consistency query 31, listed in Appendix 3-11, “Old consistency query 31” on page 113 to show an example of the EXPLAIN.

Table 3-11 Old consistency query 31

The old Query 31
<pre>SELECT NAME FROM SYSIBM.SYSPLAN PL WHERE PLENTRIES = 0 AND NOT EXISTS (SELECT * FROM SYSIBM.SYSDBRM WHERE PLNAME = PL.NAME);</pre>

This query with correlated subquery selects the names of plans that have PLENTRIES=0 and have no DBRM pointing to the name of the plan. If you want more information about the consistency query, take a look at 9.5, “Catalog consistency query DSNTESQ” on page 359. There you find some measurements and more details about it.

Now let us analyze the Visual Explain from the Query 31. Figure 3-47 shows the EXPLAIN and tells us that the cost of CPU is 2 milliseconds (in service units the cost is 32).

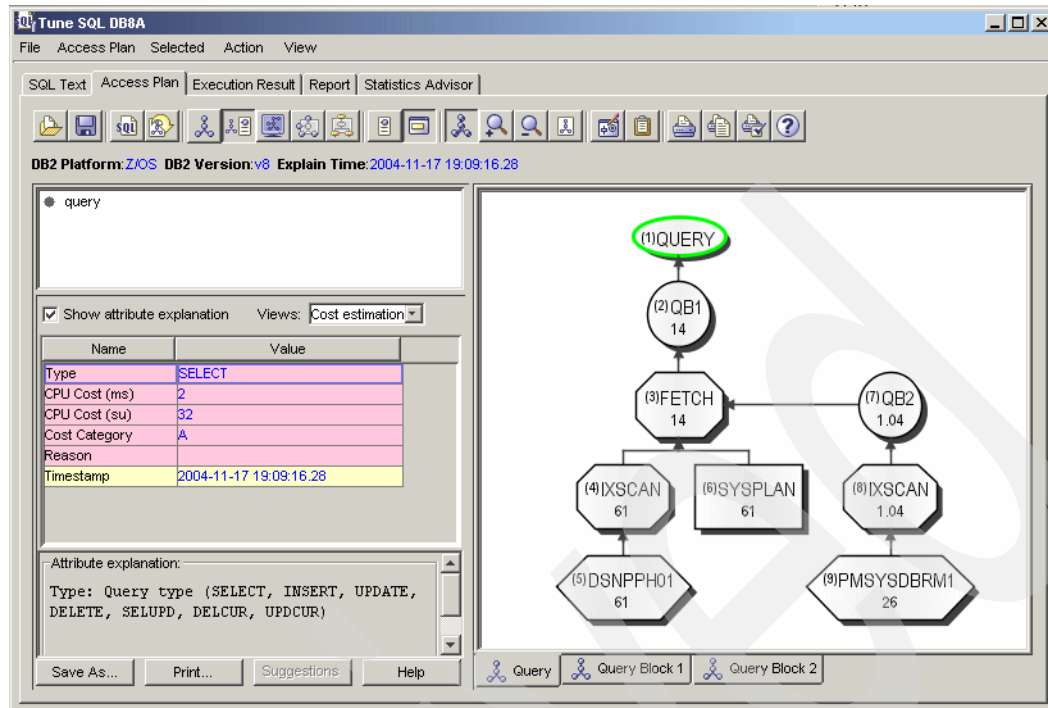


Figure 3-47 Explain of the old consistency query 31

Note: We are using a very small catalog, that is why the numbers are so small. But we want to give just an idea on how useful Visual Explain can be for tuning queries.

If we click on the top node (1) QUERY, the attributes of the node QUERY appear on the left hand side window informing us that:

- ▶ CPU cost (ms): 2
- ▶ CPU cost (SU): 32
- ▶ Cost category: A

This information is taken from the table DSN_STATEMNT_TABLE that has been updated by the EXPLAIN statement execution. The CPU cost in milliseconds is the cost in service units divided by the number of service units that the machine where DB2 is running can do in one millisecond.

The chart shows that this query has two query blocks. The first query block accesses the table SYSPLAN. If we click the node for table (6) SYSPLAN, the attributes of the table pop up informing us of:

- ▶ Table Name: SYSPLAN
- ▶ Creator Name: SYSIBM
- ▶ Correlation Name: PL
- ▶ Cardinality: 61
- ▶ Simple table space

The attributes of the node appear on the left hand side of the graph showing as default the information considered for cost estimation:

- ▶ Information that appeared in the pop-up +
- ▶ Qualifying rows: 14
- ▶ Pages: 61

- ▶ Timestamp: 2004-11-15 13:44:36.771989 (RUNSTATS)
- ▶ Explain time: 2004-11-19 14:56:24:34

If we click the Index node (5) DSNPPH01, the information about the index used to access SYSPLAN pops up: Fullkeycard of 61 and cluster ratio of 0.4918. The attribute side shows the number of leaf pages, number of levels, if the rows are clustered for this index, if the index is padded, and other information.

The index scan node (4) shows the number of input RIDs and output RIDs, scanned leaf pages, total filter factor, and number of matching columns for this scan.

The fetch node shows a cardinality of 14. This is very interesting. The attribute of the fetch node shows:

- ▶ Scanned rows: 61
- ▶ Stage 1 predicate: PL. PLENTRIES=0. Filter factor: 0.459
- ▶ Stage 1 returned rows: 28. If we calculate $61 * 0.459 = 27.999$
- ▶ Stage 2 predicate: NOT EXISTS(SELECT 1 FROM SYSDBRM WHERE SYSDBRM.PLNAME=PL.NAME); Filter factor: 0.5.
- ▶ Stage 2 returned rows: 14

For each entry in the index DSNPPH01 leaf page (no matching index scan) DB2 has to go to the data portion to verify the predicate PL.PLENTRIES=0. PLENTRIES is the number of package list entries for the plan. The size of the row in SYSPLAN is 3613 bytes in a 4 KB page.

Since this query is a correlated subquery, for each row in the outer table the inner query is re-executed. This subquery is an non-matching index-only scan, resulting in many scans to all leaf pages in the index PMSYSDBRM1 because the PLNAME is the second column of the index.

So the largest impact comes from the time to scan all leaf pages for the entries in the index for the table SYSDBRM. If the number of DBRMs is small, this can be a few pages, but if the catalog has a large number of DBRMs, for example, 20,000, and the index on columns PLCREATOR, PLNAME, has a key of 16 bytes, resulting in approximately 200 keys per page, there is a total of 100 leaf pages for each plan that has PLENTRIES=0.

If there are 10,000 plans not using a package list, there are 10,000 scans of 100 pages.

In this example we executed the query under Visual Explain and the output resulted in two rows.

Let us take a look at the new Query 3, listed in Table 3-12.

Table 3-12 Consistency Query 31 New

The new Query
SELECT NAME FROM SYSIBM.SYSPLAN WHERE PLENTRIES = 0 AND NAME NOT IN (SELECT PLNAME FROM SYSIBM.SYSDBRM);

The new query is a non-correlated query. In a non-correlated subquery case the subquery is executed only once and the result of the query is materialized in the work file. The validation of the clause NOT IN is made in the sorted work file.

The optimizer uses the process (scan, sort, work file) to make the performance better as shown in Figure 3-48. This change makes a nice difference when we compare the CPU cost

in ms = 1 and the CPU cost is 16 service units. We save 1 millisecond, or 16 SUs on accessing a very small catalog.

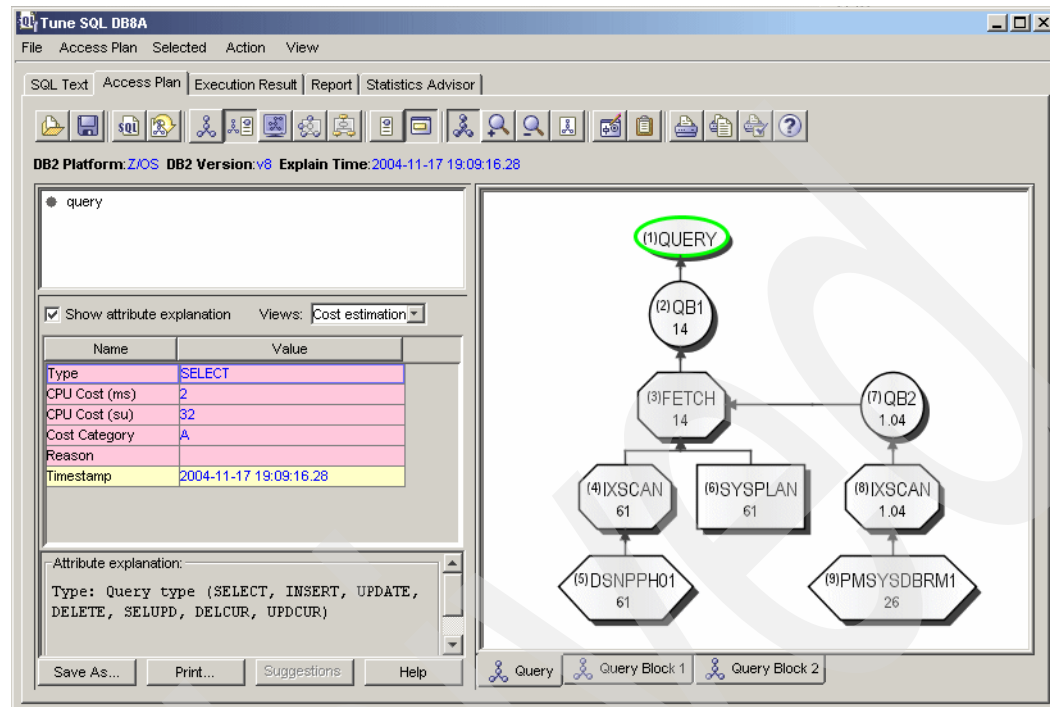


Figure 3-48 Consistency query - 31 New

3.15.3 Statistics Advisor

Statistics Advisor is a new function from the Web, available with Visual Explain. Its main purpose is to help you identify the RUNSTATS executions that help your SQL. Sometimes it is not simple at all to understand which statistics can help your queries. The Statistics Advisor builds the RUNSTATS statements for the correct set of statistics, gives you consistent and accurate information, and you just need to run the RUNSTATS and perform some tests, and the problem is solved. You can continue to run the Statistics Advisor until it no longer provides suggestions.

The Statistics Advisor (SA) analyzes the predicates, column (groups) used as predicates, type of predicates, performs statistical analysis, checks missing statistics (default), conflicting statistics, missing appropriate correlations, and skew statistics. Based on all of that, the Statistics Advisor gives you suggestions on the RUNSTATS to execute. Furthermore, it gives you an explanation and a conflict report.

We use the following simple SQL statement against the EMP table of the DB2 V8 sample database:

```
SELECT * FROM DSN8810.EMP WHERE HIREDATE < '2004-12-31' AND SALARY < 25000 AND SEX = 'M';
```

We start SA from the main panel, shown in Figure 3-49, where we enter the statement to be analyzed, explained, or run.

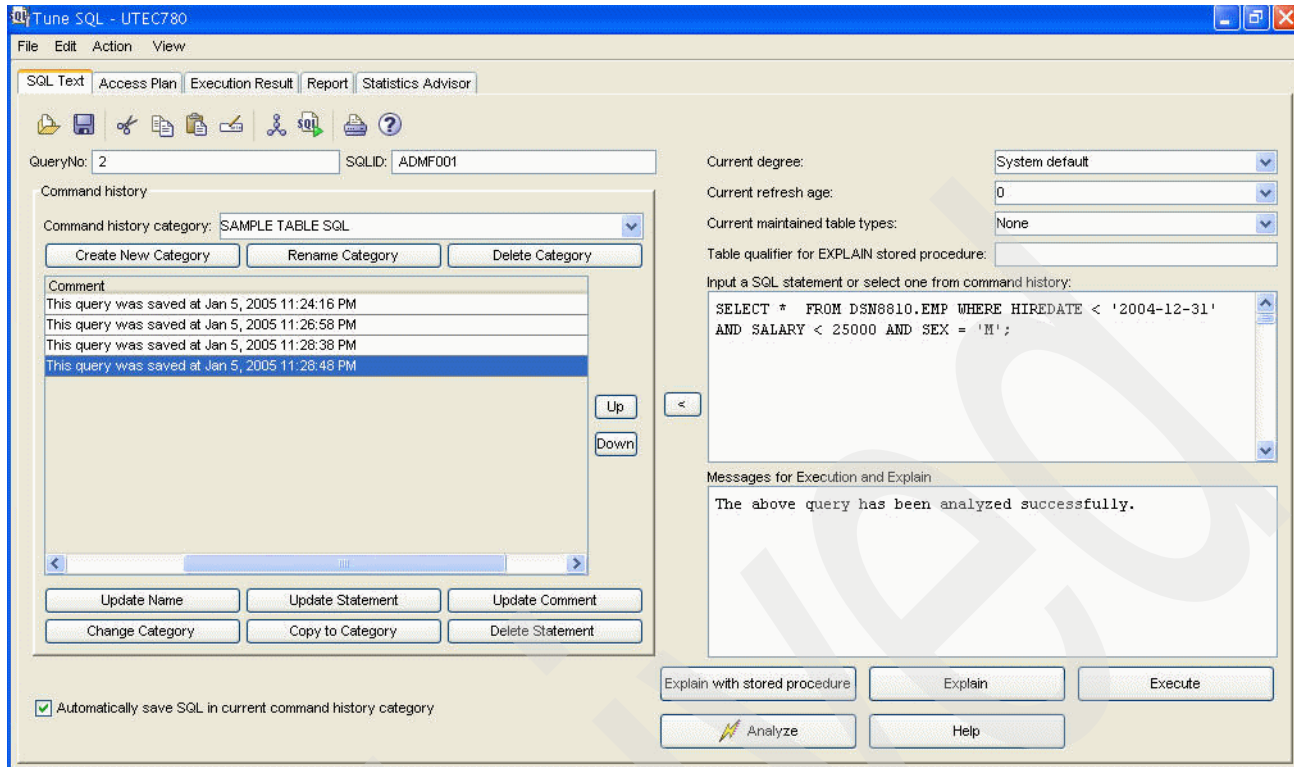


Figure 3-49 Starting to analyze a sample query

You can create a category to save all the related SQL. The related SQL can be automatically saved in a current command history category and you can rerun it anytime.

Note: Before you run an EXPLAIN, you can run the Analyzer. If Statistics Advisor suggests a RUNSTATS, you can execute the RUNSTATS, then execute the Analyzer again, iteratively. When no more RUNSTATS are suggested, run EXPLAIN. And if you want to execute the query, just click **EXECUTE**.

We click the **ANALYZE** button. The screen that pops up, when the analysis is completed, shows the targeted RUNSTATS statements.

Figure 3-50 reports the RUNSTATS suggestion for the SQL statement we analyzed. You can execute that RUNSTATS directly from Visual Explain (VE), after configuring a WLM environment, by invoking DSNUTILS from Statistics Advisor.

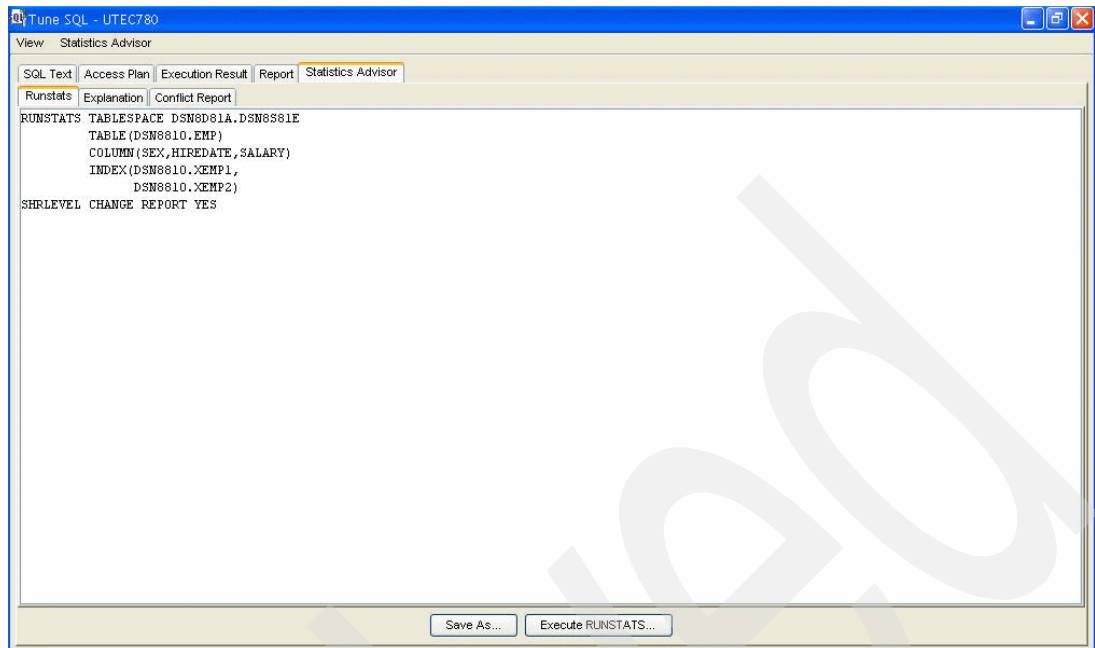


Figure 3-50 Suggested RUNSTATS statements

You can click the **Explanation** tab for an explanation of why the suggestions were made.

Figure 3-51 is the report which shows you the explanation why Statistics Advisor suggests you run the RUNSTATS listed in Figure 3-50. You can save the report in HTML format just for information, or understand how the optimizer works, and Analyze it at any time.

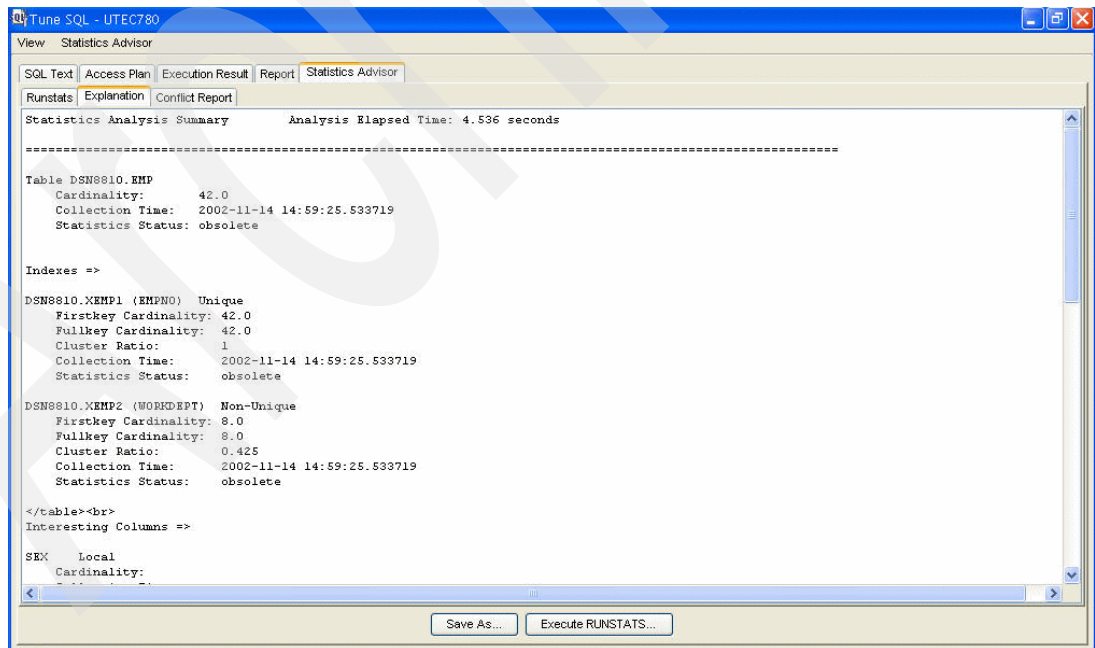


Figure 3-51 Reason for RUNSTATS

Note that the table and index statistics are old, and column statistics have not been collected.

Figure 3-52 shows the history of the saved queries.

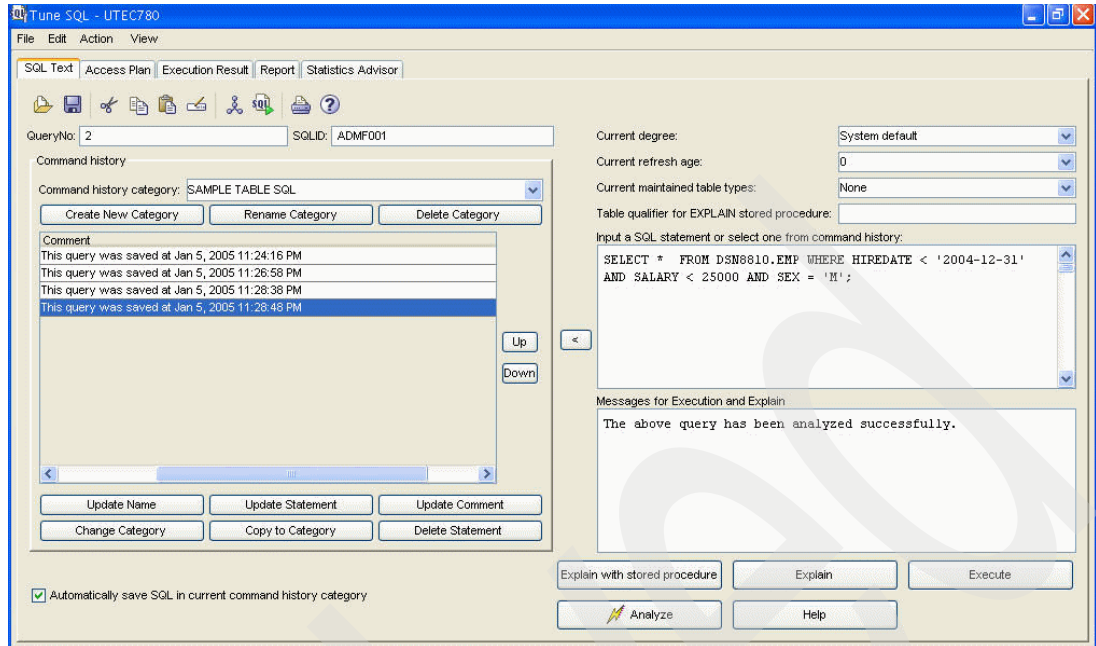


Figure 3-52 Saved queries

We now execute the RUNSTATS (for large tables, running RUNSTATS in batch is best), and then rerun the SQL through SA to see if there are further suggestions.

There are cases when one suggestion is not sufficient and you need to run the Statistics Advisor again to uncover another layer of problems.

SA uses the existing statistics to make determinations about potential correlation and skew. If some columns have default statistics, then SA does not make any suggestions until the defaults are resolved. In this case, SA now suggests you collect frequency statistics on column SEX, as shown in Figure 3-53.

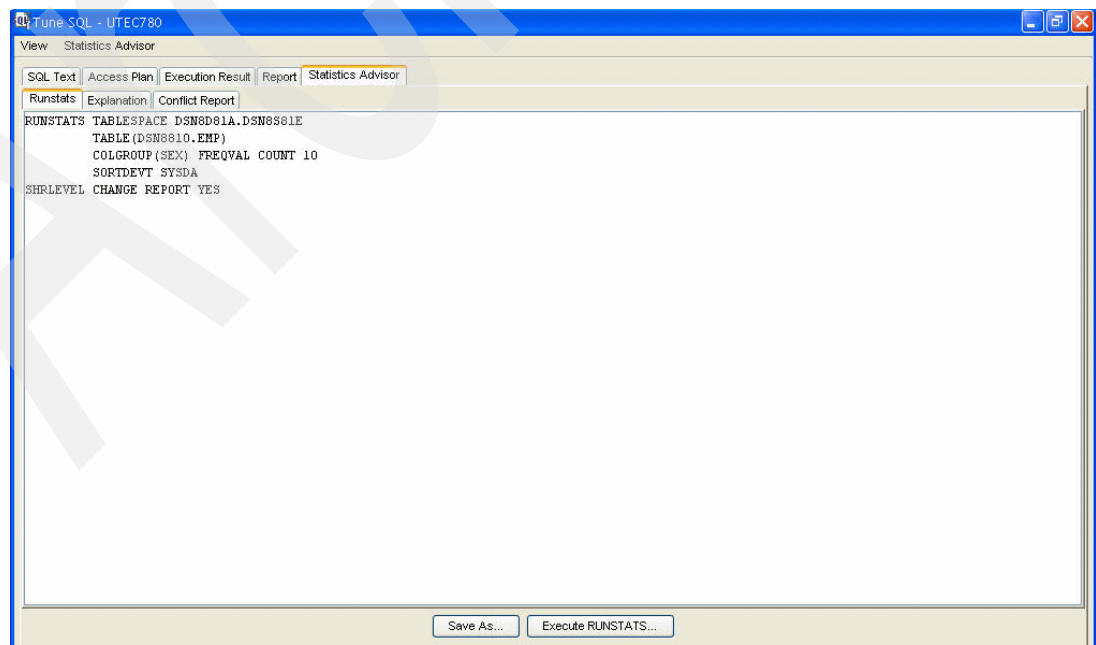


Figure 3-53 New suggested RUNSTATS

Looking at the explanation, we can see that column SEX has COLCARDF of 2. This is a low COLCARDF. Columns with low cardinality (relative to table cardinality) have significant opportunity for skew, for that reason frequencies are desirable. The purpose of this example is to illustrate it may take two runs of SA to get more robust suggestions if defaults are initially provided. We obtain the statistics shown in Figure 3-54.

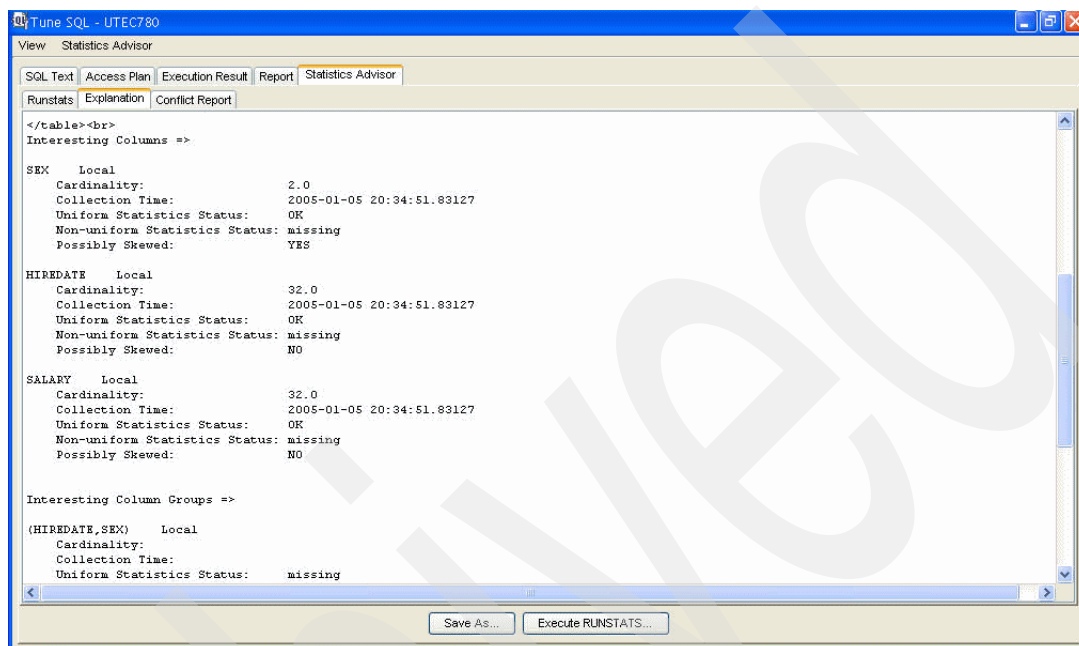


Figure 3-54 Statistical output

To illustrate conflict statistic checking, FULLKEYCARDF has been set to 100,000. The table cardinality is much lower. Rerunning SA returns a complete recollection. When statistics are conflicting, the issue is often due to an inconsistent statistics collection approach.

We have observed some customers collecting statistics very aggressively on the indexes, and not on the tables, or vice versa. When index and table statistics are inconsistent, it can result in inaccurate estimations of filtering for an index, or join size. This can lead to inefficient access paths. SA can solve this problem quickly by identifying and suggesting recollection of inconsistent statistics, as shown in Figure 3-55.

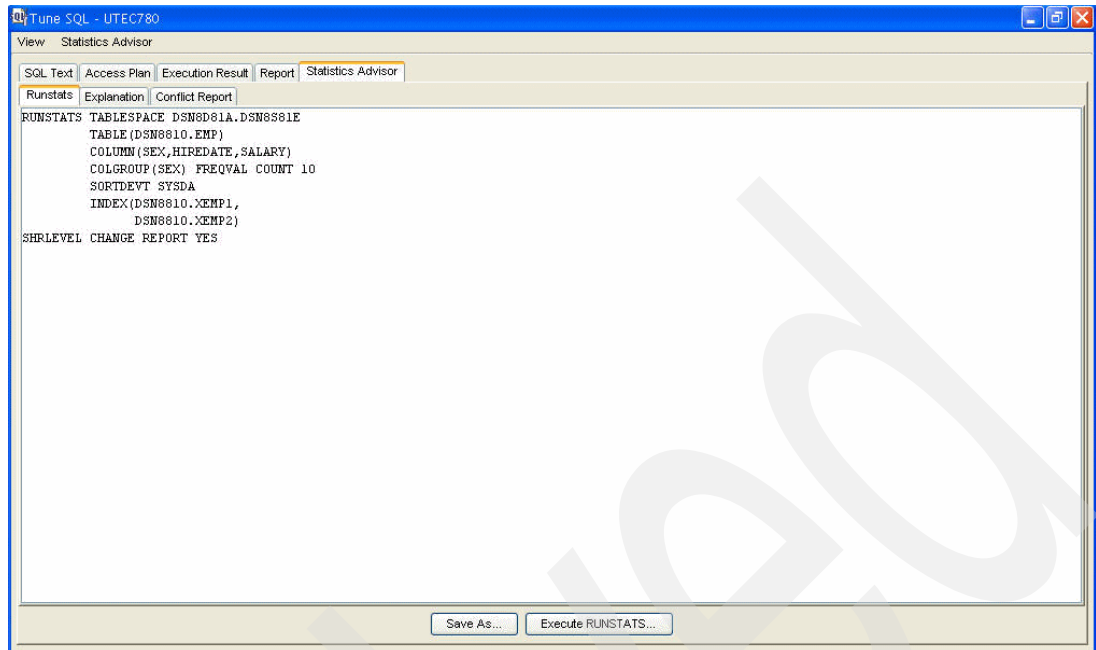


Figure 3-55 Recollection of inconsistent statistics

The conflicts tab shows the statistics which were in conflict. See Figure 3-56.

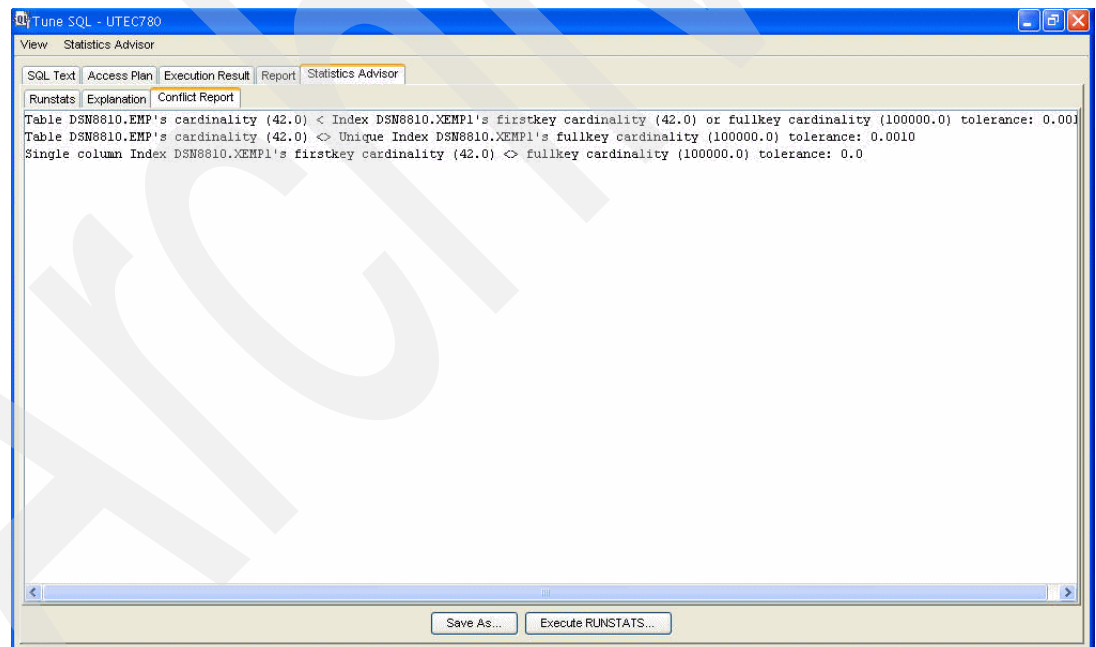


Figure 3-56 Conflicting statistics

The DBA could make a judgement call based on these conflicts. The DBA can determine that only the index is inconsistent with the table cardinality, and limit the recollection.

Since repeated partial statistic collections could result in a cycle of collecting "inconsistent statistics", SA is designed to collect a more complete and consistent set of statistics when inconsistent statistics are present for a table.

The Plug-In Options screen of Statistics Advisor is provided in Figure 3-57:

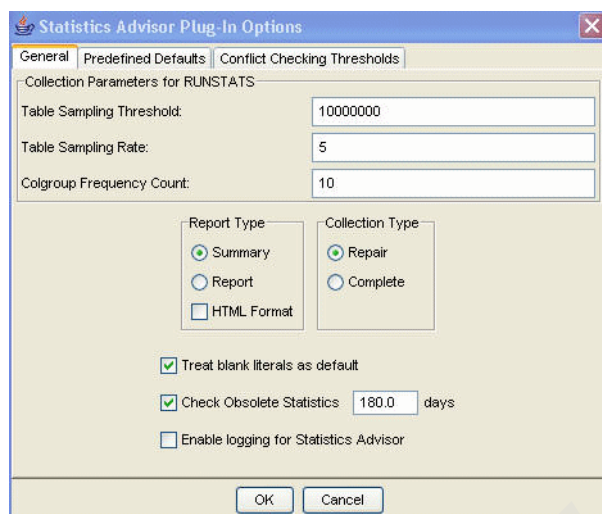


Figure 3-57 Statistics Advisor Plug-integrated with Visual Explain

Predefined defaults (you can add your own business specific defaults):

When Statistics Advisor observes a predicate using a default, SA recognizes that often default values are skewed. So even if a column has high COLCARD, if the SQL searches for a 'typical default' (ACCT_NO = 0), then SA suggests collection of a single frequency on that column. On the screen presented on Figure 3-58, you can define the collection parameters for RUNSTATS, report type, collection type, check obsolete statistics dates, and enable logging for Statistics Advisor. *Logging* means saving files with the name ssa.log, ssa.log.1, ssa.log.2, and so on, and creates a trace.txt file, with the information you need to analyze VE problems.

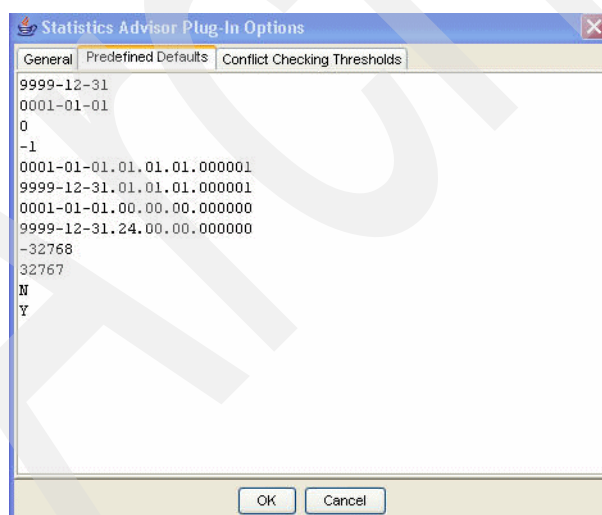


Figure 3-58 Predefined default values

Conflict thresholds tab:

We recognize that you run RUNSTATS with SHRLEVEL CHANGE on online systems. So it is possible for table, index, and column statistics to be slightly inconsistent without causing

problems with optimization. The conflicts threshold tab allows you to control how aggressively SA can enforce inconsistencies. See Figure 3-59.

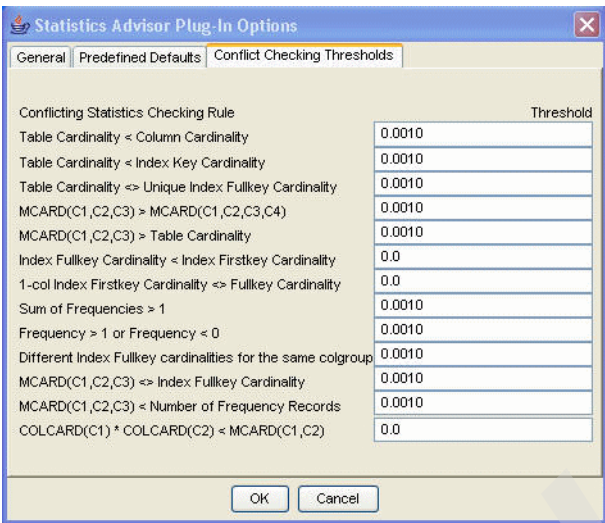


Figure 3-59 Conflict statistics threshold setup

Summarizing what the Statistics Advisor can do:

- ▶ Automate the analysis of the statistics required for an SQL statement. Often a query can have inefficient or unstable performance due to lack of statistics.
- ▶ Generate a report explaining why you need the recommended RUNSTATS
- ▶ Show the statistics conflict that you have in your catalog
- ▶ Submit the RUNSTATS using a stored procedure
- ▶ Automate the solution to many common SQL performance problems and solve SQL performance problems quickly and easily.

3.15.4 Visual Explain and Statistics Advisor on MQTs

To illustrate, we show screens from Visual Explain and Statistics Advisor taken using one query from the member DSNTEJ3M (from SDSNSAMP).

Note: Do not forget to set the Current Refresh Age Parameters = Any (from the Visual Explain - Tune SQL Screen); otherwise, EXPLAIN does not show you the MQTs you are using.

We put the whole query on Table 3-13 in case you want to do the same tests we did.

Table 3-13 Query using MQT table

Query MQT table
<pre>SELECT R1.DAY, R1.MONTH, R1.YEAR, R10.CITYID, R7.DIVISIONID, SUM(R15.QUANTITY) SUMQTY, SUM(R15.COST) SUMCOST, SUM(R15.AMOUNT) SUMAMT, SUM(R15.AMOUNT) - SUM(R15.COST) GPROFIT FROM DSN8810.TIME R1, DSN8810.PRODUCT R5, DSN8810.LOCATION R10, DSN8810.PCATEGORY R6, DSN8810.SALESFACT R15, DSN8810.PFAMILY R7 WHERE R1.TIMEID = R15.TIMEID AND R5.PRODUCTID = R15.PRODUCTID AND R5.CATEGORYID = R6.CATEGORYID AND R6.FAMILYID = R7.FAMILYID AND R10.STOREID = R15.STOREID AND R1.YEAR >= 2000 GROUP BY R1.DAY, R1.MONTH, R1.YEAR, R10.CITYID, R7.DIVISIONID;</pre>

Before the introduction of MQTs, DB2 had to struggle through long data warehouse queries. And the DBA had to pre-elaborate complex SELECTs with lots of UNIONS and JOINS in order to create intermediate tables to be used by application people.

Figure 3-60 shows the graph for the query in Figure 3-13 (no MQT).

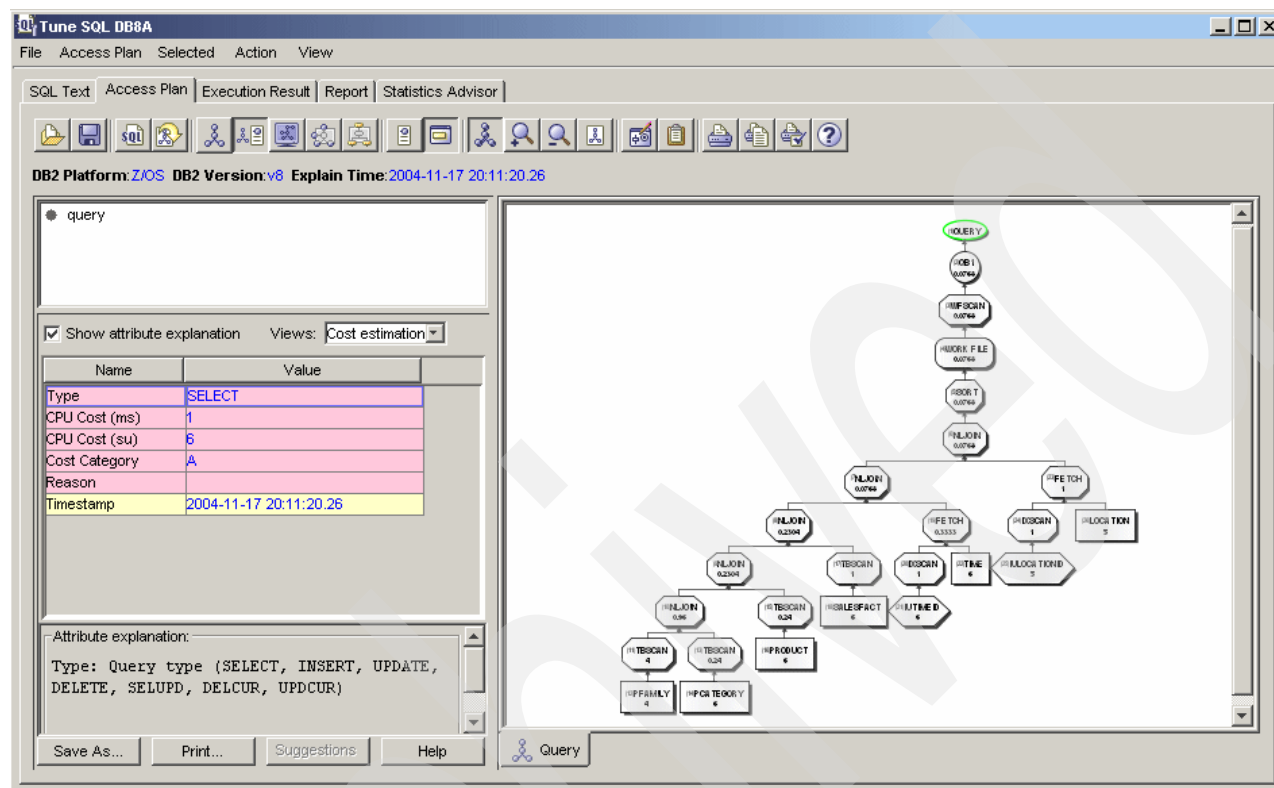


Figure 3-60 Explain without using an MQT table by Visual Explain

DB2 accesses an MQT the same way it accesses a base table with one exception: DB2 cannot use a direct fetch to access a materialized query table.

When underlying data changes, DB2 does not automatically refresh MQTs. To keep the MQT data current, you must issue a REFRESH TABLE statement if the table is system-maintained, or, if the materialized query table is user-maintained, you must issue INSERT, UPDATE, and DELETE statements as needed. To control the use of MQTs, use the CURRENT REFRESH AGE and CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION special registers.

An MQT node is labeled with the name of the MQT and is, by default, displayed as an upside-down trapezoid. The table's correlation name, creator name, or cardinality can also be displayed on the label. If the RUNSTATS utility has not collected statistics for the table, the node is outlined in red. If the optimizer uses the default value for the cardinality, the cardinality is marked as the default.

On Figure 3-61 you can see the EXPLAIN graph generated by the VE of a query using the MQT table with the name MATPROC ID = 14 and Cardinality = 6.

Archived

DB2 subsystem performance

In this chapter, we discuss the following topics and related performance enhancements that affect DB2 subsystem performance:

- ▶ **DB2 CPU experiences:** With every new release of DB2, performance is always a challenge. DB2 V8 is no different. Here we discuss general subsystem performance considerations and what we have seen in V8 in terms of CPU utilization.
- ▶ **Virtual storage constraint relief:** DB2 V8 supports 64-bit storage in the DBM1 address space. This provides significant virtual storage constraint relief (VSCR). We first explain what has changed in V8, then evaluate the impact of these enhancements on virtual storage usage.
- ▶ **Distributed thread storage:** Although 64-bit addressing is exploited by DBM1 and IRLM, the distributed address space does not use 64-bit in DB2 V8. In general we experience an increase in virtual storage requirements of the distributed address space due to the architectural changes in DB2 V8.
- ▶ **Monitoring DBM1 storage:** It is important to proactively monitor virtual storage usage taking advantage of the enhanced instrumentation available to DB2 V7 and V8.
- ▶ **Buffer pool long term page fixing:** DB2 V8 allows you to fix the buffer pool pages once in memory and keep them in real storage. This avoids the processing time that DB2 needs to fix and free pages each time there is an I/O. This applies to CM and NFM.
- ▶ **IRLM V2.2:** DB2 V8 requires IRLM V2.2, which is a 64-bit application. With IRLM V2.2, locks always reside above the 2 GB bar. You no longer have the option to request the IRLM to manage the locks in ECSA by specifying PC=NO.
- ▶ **Unicode:** DB2 V8's increased exploitation of Unicode dictates that DB2 must perform far more conversions to and from Unicode than in the past. Here, we discuss recent enhancements that have been made, both inside DB2 and external to DB2, to improve the performance of these conversions.
- ▶ **Data encryption:** We introduce both DB2 encryption and the IBM Data Encryption Tool and discuss some recent hardware enhancements that improve encryption performance.
- ▶ **Row level security:** DB2 introduces support for row level security for more granular security of DB2 data which simplifies row level security implementation and management significantly.

- ▶ **Larger VSAM CI sizes:** DB2 V8 introduces support for VSAM CI sizes of 8, 16, and 32 KB, which potentially reduces the elapsed time for all I/O bound workloads.
- ▶ **Instrumentation enhancements:** Significant enhancements to DB2 instrumentation have been added in V8.
- ▶ **Miscellaneous items:**
 - *DB2 logging enhancements:* DB2 V8 increases the amount of active log data set pairs from 31 to 93, as well as increases the number of archive log data sets from 1,000 to 10,000. This enhancement increases the amount of log data DB2 has available online for recovery, which can have a dramatic impact on recovery performance and data availability.
 - *Up to 65,041 open data sets:* DB2 V8 increases the number of open data sets from 32k to 65,041 by exploiting z/OS 1.5 which allows open data set control blocks to be allocated above the 16 MB line. This enhancement does not only provide virtual storage relief, it also removes a potential scalability problem with growing systems and growing machine capacity.
 - *SMF Type 89 record collection enhancements:* DB2 V8 provides a new DSNZPARM parameter, SMF89, which lets you specify whether or not DB2 is to do detailed tracking for measured usage pricing. Previous releases of DB2 automatically used detailed tracking of measured usage if SMF type 89 records were activated. This is available in CM.
 - *Lock avoidance enhancements:* DB2 V8 avoids taking locks on overflow records if a row has been relocated to another page. In addition, DB2 performs lock avoidance for singleton SELECT statements with ISOLATION(CS) and CURRENTDATA(YES). This is available in CM.

4.1 DB2 CPU considerations

With every new release of DB2, performance is always a challenge. DB2 V8 is no different. For those applications not taking advantage of any V8 performance enhancements, some CPU time increase is unavoidable to support a dramatic improvement in user productivity, availability, scalability, portability and family consistency of DB2. In particular, enhancements in:

- ▶ DBM1 virtual storage constraint relief with 64-bit addressing. A number of DB2 structures, for example, buffer pools, have moved above the 2 GB bar.
- ▶ Long names and long index keys. Many DB2 names change from CHAR(8) or VARCHAR(18) to VARCHAR(128). Maximum index key size also moves from 255 to 2000 bytes.
- ▶ Longer and more complex SQL statements. The maximum SQL statement size moves from 32 KB to 2 GB.
- ▶ Extended support for Unicode. The DB2 catalog is now Unicode and you can combine multiple CCSIDs in the same SQL statement.

In this section we discuss general subsystem performance considerations and what you may see in V8 in terms of CPU utilization.

As processor power continues to improve, linear scalability, or the ability to exploit increasing processor power without encountering a bottleneck that prevents the full CPU usage, becomes more important. Bigger buffer pools and cache reduce the I/O bottleneck and CPU overhead. Although most of the thread-related storage is still below the 2 GB bar, more concurrent active threads could be supported in V8 as other storage areas are moved above the 2 GB bar. However, whether more threads can be supported in V8 critically really depends on how much thread storage is being used in V7. If the thread storage takes up the majority of virtual storage in V7, there is no relief in V8.

Exploiting bigger and faster hardware without premature constraint is a major driver to the virtual storage constraint relief enhancements in DB2 V8. For example, the z990 (GA in 05/03) is:

- ▶ 1.3 to 1.6 times higher MIPS compared to z900 turbo
- ▶ 1.8 to 2 times higher MIPS compared to z900
- ▶ 256 GB real storage up from 64 GB for z900

From V1R1 1984 to the present, real storage usage has grown in DB2 at about 20 to 30% per release to support enhancements in performance and scalability, more and bigger buffer pools, larger sort pools, and more concurrent threads. DB2 V8 continues this trend, by removing bottlenecks which would have prevented the exploitation of bigger real storage.

64-bit addressing has brought its own performance challenges to DB2. For example, 64-bit instructions are typically 50% bigger than 31-bit instructions. 64-bit addresses are twice as big as 31-bit addresses. So, hardware registers and software control blocks which contain virtual addresses all need to be bigger. All these bigger instructions and bigger registers take longer to load and execute, (more bits and bytes must be staged into registers, destaged and moved around the hardware). Studies have shown some 64-bit instructions can take as much as 15% more CPU to execute. Other studies have shown that the same path length coded in 64-bit mode can take as much as 10% more CPU than coded in 31-bit mode.

The biggest difference is when 31-bit pointers have to be converted to 64-bit before being used in AMODE(64). That overhead does not exist in V7, but is unavoidable since not all components have been converted to 64-bit. Finally, it should be pointed out that the Unicode

support caused almost as much CPU degradation as the 64-bit conversion, and support for variable length indexes was not far behind.

Hopefully you can now appreciate how the new 64-bit architecture, combined with all the enhancements in DB2 V8, have added much more code than V7 and have also presented some significant performance challenges which have had to be overcome in order to minimize the CPU regression of V8 compared with V7.

4.1.1 Performance

We reviewed the performance of DB2 V8, compared with DB2 V7, for a variety of different workloads, to understand the impact V8 may have on CPU for your workload. The performance measurements we compared for each workload were the ITR (Internal Throughput Rate, inversely proportional to CPU seconds used) and CPU.

We begin our discussion by having a look at OLTP workloads. The variety of OLTP workloads we studied include the IRWW (IBM Relational Warehouse Workload, described in *DB2 for MVS/ESA Version 4 Data Sharing Performance Topics*, SG24-4611), both in a data sharing and non-data sharing environment, a number of DRDA client/server transaction workloads, a number of SAP workloads and a CICS/DB2 transaction workload. Each workload studied had a different SQL structure and therefore revealed different performance strengths and weaknesses. For example, some workloads executed reasonably simple SQL while other workloads executed more complex SQL.

A number of query-based workloads have been studied to compare query performance. Over 200 queries gathered from a number of customer sites have been used. Some of these queries achieved up to 20% reduction in CPU, due primarily to improved access path enhancements. For example, a data warehouse application achieved over 30% improvement in CPU, primarily due to star join enhancements.

A number of batch workloads were also studied, including a sampling of commonly used utilities.

Figure 4-1 summarizes some general measurements of V8 CPU consumption compared with V7 for various workloads. These comparisons are made with no application changes to exploit any V8 new function, nor any aggressive configuration changes.

This chart has been compiled from a number of different laboratory tests running various DB2 workloads. A '+' means a CPU increase and a '-' means a CPU decrease in V8 compared with V7. Be reminded that these trends can only be used as a guide. As always, "your mileage may vary", since all DB2 installations and workloads are different.

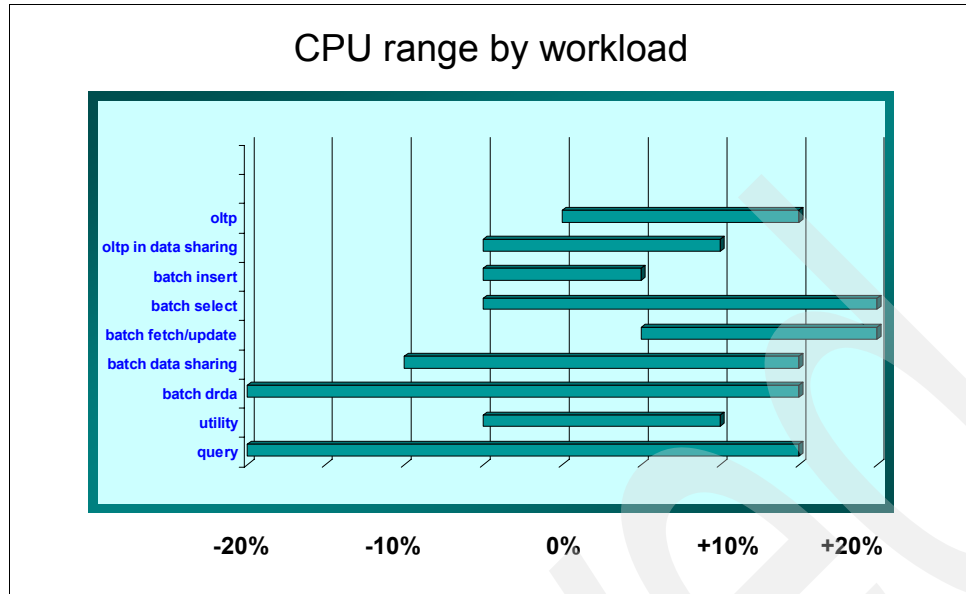


Figure 4-1 CPU workload summary

The typical customer CPU regression of DB2 V8 is anticipated to be 0 to 10% more than V7 on average. This CPU degradation is the result of the new functionality that is provided in DB2 V8, together with the overhead that comes with 64-bit addressing.

Looking at it a little closer, the CPU delta is:

- 0 to +15% regression for online transaction workloads

Remember, we have much more code and function in V8 compared with V7, as well as dealing with the added complication of 64-bit addressing in the DBM1 address space.

- -5 to +10% regression for online transaction workloads in data sharing environments

The reason why we are able to achieve less CPU degradation in data sharing is because there are a number of performance enhancements in V8 that apply specifically to data sharing environments. For example, Lock Protocol Level 2 to reduce global lock contention. See Chapter 8, “Data sharing enhancements” on page 319 for more details.

- -5 to +20% regression in batch workloads

- -5 to +5% with a heavy INSERT workload

This does not represent a significant change and is comparable to DB2 V7.

- -5 to +20% with a heavy SELECT workload

We are able to achieve some CPU reduction for SELECT workloads because the SELECT workloads benefit from the new lock avoidance techniques used in V8. See 4.13.4, “Lock avoidance enhancements” on page 214 for more details. Applications using CURRENTDATA(YES) receive the most benefit as these do not exploit lock avoidance in DB2 V7. We see more CPU degradation (up to a 20%) as the number of columns increases.

- +5% to +20% with a heavy FETCH and UPDATE workload

This increase in CPU varies greatly based on the number of columns being processed. The more columns, the more CPU used. In fact, you may see as much as 20% more CPU consumed in V8 when you are processing over 100 columns in the SQL.

- -10% to +15% regression in batch workloads in a data sharing environment

Regression is about 5% better than non-data sharing. V8 can achieve significant performance improvements over V7 primarily due to some data sharing specific enhancements such as CF Request Batching and protocol 2. See Chapter 8, “Data sharing enhancements” on page 319 for more details.

- -20 to +15% regression in batch DRDA workloads

These huge gains over V7 are primarily due to API reduction for multi-row fetch in a distributed environment. See Chapter 7, “Networking and e-business” on page 281 for more details.

- -5% to +10% regression for utilities

See Chapter 6, “Utilities” on page 261 for details.

- -20% to +15% regression for query workloads

Simple and short queries tend to increase in CPU while long running queries tend to decrease in CPU. However, the gains in performance can be quite substantial. They are primarily due to access path enhancements in V8. See Chapter 3, “SQL performance” on page 29 for more details.

Now, let us have a brief overview of DB2 utilities, where we found a wide variety of results. Figure 4-2 summarizes these results. Once again, a ‘+’ means a CPU increase and a ‘-’ means a CPU decrease.

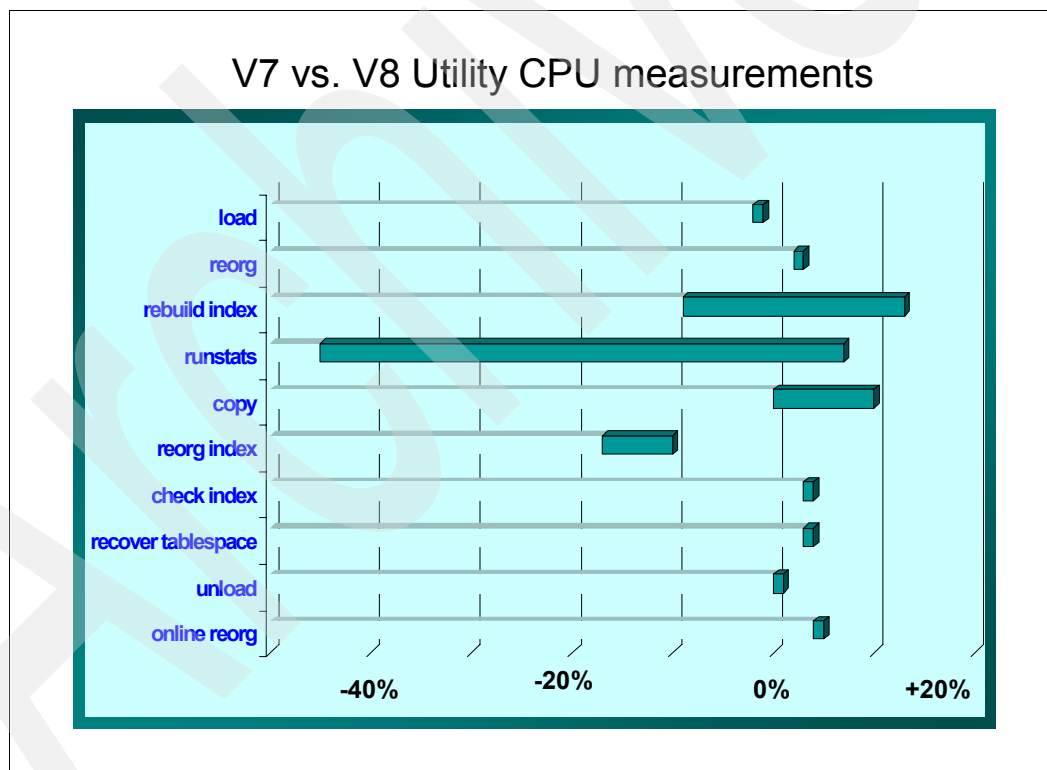


Figure 4-2 V7 vs. V8 Utility CPU measurements

All these utilities were executed with no new V8 function, except where the defaults changed from V7 to V8. We saw that:

- LOAD achieved a 2% reduction in CPU
- REORG achieved a 2% increase in CPU
- REBUILD INDEX varied from a 9% reduction, to a 13% increase in CPU
- RUNSTATS varied from a huge 45% reduction, to a 7% increase in CPU

- COPY varied from a 0% reduction, to a 10% increase in CPU
- REORG INDEX varied from a 10% to 17% reduction in CPU
- CHECK INDEX achieved a 3% increase in CPU
- RECOVER TABLESPACE also achieved a 3% increase in CPU
- UNLOAD achieved no increase CPU
- ONLINE REORG achieved a 4% increase in CPU

Refer to Chapter 6, “Utilities” on page 261 for a discussion of utility performance enhancements in V8.

Overall, the utilities we studied achieved a -5% to +10% regression in CPU used in V8 compared with V7.

The RUNSTATS utility was by far the most variable. RUNSTATS TABLESPACE was generally the best performer with its CPU to the left of the RUNSTATS bar. This was followed by RUNSTATS TABLESPACE and TABLE, then RUNSTATS TABLESPACE and INDEX, with RUNSTATS INDEX generally appearing to the right of the bar.

Important: The numbers we have presented in this section represent what we observed from the laboratory measurements using a variety of different workloads. Your performance studies most certainly vary. DB2 performance is very application and workload dependent. Also remember that DB2 performance may vary every time maintenance is applied to DB2.

Let us have a closer look at the IRWW workload, running in DB2 V7, DB2 V8 in CM and DB2 V8 running in NFM. Table 4-1 summarizes these results.

Table 4-1 CM vs. NFM - Non-data sharing

	DB2 V7	DB2 V8 CM	DB2 V8 NFM	Delta (CM / V7)	Delta (NFM / CM)
Page-fixed Buffer Pools	No	Yes	Yes		
ETR (commits / sec)	386.00	386.50	385.50	+ < 1%	+ < 1%
CPU (%)	63.79	64.41	64.11	+ 1%	- < 1%
ITR (commits / sec)	605.11	600.06	601.31	- 1%	+ < 1%
DB2 class 2 time (msec / commit)					
Elapsed	15.148	15.922	15.462	+ 5%	- < 3%
CPU	1.713	1.842	1.815	+ 7%	- < 2%
DB2 class 2 CPU time (msec / commit)					
MSTR	0.096	0.107	0.107	+ 11%	0%

	DB2 V7	DB2 V8 CM	DB2 V8 NFM	Delta (CM / V7)	Delta (NFM / CM)
DBM1	0.407	0.324	0.321	- 20%	- 1%
IRLM	n/c	n/c	n/c		
Total	0.503	0.431	0.428	- 14%	- < 1%
DB2 class 2 CPU + Total	2.216	2.273	2.246	+ 2%	- < 2%

The results indicate no significant difference in throughput for our workload running in CM compared with NFM. We recorded less than 1% variation which is not significant.

We see 7% more class 2 accounting CPU consumed in V8 compared with V7 and slightly less CPU used in NFM compared to CM, with a less than 2% improvement. We can therefore conclude there is no significant difference in CPU consumed by using DB2 V8 in CM over NFM.

We also see a modest increase of 2% CPU consumed by each transaction in DB2 V8 CM compared with DB2 V7. This would have been a little higher if CPU consumed by the DBM1 address space had not decreased by 20% compared with V7. The DBM1 address space used less CPU because all the buffer pools were long term page-fixed and all the plans and packages were bound in the V8 format.

In addition, we see a 7% increase in class 2 CPU time used by each transaction in DB2 V8 CM compared with DB2 V7. This increase is caused in part by the extra CPU required to process the 64-bit instructions necessary to access the DBM1 memory structures which are now above the 2 GB bar.

Table 4-2 compares the IRWW workload running in DB2 V8 compatibility mode, with DB2 V8 new-function mode, in a data sharing environment.

Table 4-2 CM vs. NFM - Data sharing

	DB2 V7		DB2 V8 CM		DB2 V8 NFM		Delta (CM / V7)	Delta (NFM / CM)
Page-fixed Buffer Pools	No		Yes		Yes			
Locking Protocol Level	1		1		2			
ETR (commits / sec)	287.83	287.50	285.33	283.67	290.00	289.83	- 1%	+ 2%
CPU (%)	67.32	72.60	65.33	70.22	62.29	61.95	- 3%	- 8%
Group ITR (commits / sec)	823.56		840.72		933.41		+ 2%	+ 11%
DB2 class 2 time (msec / Commit)								
Elapsed	34.272	34.859	36.713	39.179	19.415	19.442	+ 9%	- 49%
CPU	2.204	2.380	2.440	2.643	2.525	2.508	+ 8%	- 1%

	DB2 V7		DB2 V8 CM		DB2 V8 NFM		Delta (CM / V7)	Delta (NFM / CM)
DB2 AS class 2 CPU time (msec / commit)								
MSTR	0.309	0.336	0.135	0.150	0.150	0.150	- 56%	+ 0%
DBM1	0.603	0.621	0.451	0.447	0.436	0.434	- 27%	- 3%
IRLM	0.330	0.371	0.331	0.337	0.014	0.014	- 5%	- 96%
Total	1.243	1.328	0.917	0.974	0.600	0.597	- 27%	- 63%
DB2 class 2 CPU + Total	3.447	3.708	3.357	3.617	3.125	3.105	- 3%	- 11%

In these measurements, we see a large 11% decrease in overall CPU used in NFM compared to CM. This is not due to the extra overhead in running in CM over NFM, but as a direct result of Locking Protocol Level 2 being used in NFM, but not available in CM. These savings are realized through a dramatic reduction in the IRLM CPU times. See 8.2, “Locking protocol level 2” on page 325. We can also see the positive impact the Locking Protocol Level 2 has on transaction response times, with quite a significant savings in transaction elapsed time. The drop in transaction elapsed time is caused by a similar drop in application class 3 suspend time, which is not listed in the table.

We see a slight decrease in overall CPU (3%) used in DB2 V8 CM compared with DB2 V7. This is a better result than with non-data sharing. In addition to the performance gains we see through Long Term Page Fixing the buffer pools, data sharing offers more CPU savings through functions such as CF Request Batching which are specific to data sharing. See Chapter 8, “Data sharing enhancements” on page 319 for a more detailed discussion of these enhancements. All these enhancements combine to offer a far greater offset to the CPU increase in DB2 V8.

Compatibility mode versus new-function mode

DB2 V8 brings a new strategy to move an existing DB2 V7 environment to Version 8. You must first ‘migrate’ your DB2 environment from Version 7 to Version 8 compatibility mode (CM). You can fallback to DB2 V7 if you need to, and only partial new Version 8 function is enabled. Once you are comfortable with the function, stability and performance of DB2 V8 in CM, you then move through enabling-new-function mode (ENFM) and into new-function mode (NFM). Once in NFM, you can now exploit all the new function in DB2 V8, but you cannot return to CM or Version 7. See Chapter 9, “Installation and migration” on page 341.

Some users may choose to remain in CM for a period of a few weeks or even a month or two. While other users may choose to migrate to NFM at their earliest opportunity.

Once in CM, DB2 parses all SQL statements in Unicode. In addition, DB2 V8, both in CM and NFM, uses a different format for its DBDs, packages and plans. DB2 V8 requires a new format for DBDs, plans and packages to support the new function such as Unicode and Long Names. So, before DB2 can use a DBD, plan or package from an earlier release of DB2, it must first expand it into the new V8 format. In CM, DB2 must also convert the DBDs, plans and packages to the old format before it can store them in the catalog. This extra overhead is recorded in DB2 class 2 CPU times, and is extra overhead that exists while running in CM and NFM without which you can easily function.

Important:

If you plan to stay in CM for some time and are interested in finding out the performance of V8 on your performance-sensitive workload, you should definitely rebind. Rebinding will give you a sense of the V8 performance.

After you have entered NFM, we recommend that you plan to rebind all of your plans and packages. DB2 then stores these plans and packages in the DB2 catalog in the new format. DB2 will then no longer need to expand the plans and packages each time it needs to use them.

4.1.2 Conclusion

For most workloads, our measurements have shown a less than 10% increase in CPU consumed by moving from V7 to V8, and we did not even change applications to take advantage of any new function or make any aggressive configuration changes. In addition, the difference in CPU consumption between DB2 V8 CM and NFM is only minor and therefore should not be considered significant.

Generally, you should experience less CPU regression if, under DB2 V7:

- ▶ You are already using IRLM PC=YES and LOCKPART YES.
In V8 these are required.
- ▶ You are heavily using hiperpools or buffers in data spaces in V7.
You are taking advantage of simplified buffer pool management above the bar.
- ▶ You already use CURRENTDATA(NO) in your BIND options.
CURRENTDATA(NO) already exploits lock avoidance in V7.
- ▶ You collect SMF type 89 records.
V7 generates these records after every SQL statement while V8 generates these records on each commit. In some environments, this has the potential of saving up to 15% in CPU consumed by DB2 V8 (extreme stress environment).
- ▶ You heavily use DSNTDP2 (now DSNTDP4) and DSNTIAUL which now exploit multi-row operation. This is applicable to NFM only.
- ▶ You heavily use DRDA which now automatically exploits multi-row operation. This is applicable to CM as well as NFM.
Multi-row FETCH/INSERT/UPDATE have the potential to save considerable CPU, especially in DRDA applications.
- ▶ You have the DSNZPARM MINSTOR and CONTSTOR parameters set to YES and are not thread storage constrained.
Having these parameters set to YES in V7 uses extra CPU cycles. Once in V8, if you can set them to NO, this improves the CPU time.
- ▶ You use BIND RELEASE(COMMIT) in data sharing.
In V8 there is potentially less global lock contention as a result of the Locking Protocol Level 2. DB2 V7 must use more CPU cycles to resolve more global lock contention than V8, which potentially has less global lock contention.
- ▶ You have I/O intensive workloads.
I/O intensive workloads tend to benefit from long term page fix option and 180 CI limit removal. Sequential processing with larger CIs can also benefit.

Generally, you should experience more CPU regression if:

- ▶ You use non-padded indexes (the default in V8 install) with small varchar columns.
There is a fixed CPU overhead in processing varchar columns in INDEX KEY. As the size of each varchar column increases, the overhead gets not only diluted but eventually it can result in positive performance gain. See Chapter 5, “Availability and capacity enhancements” on page 217 for more details and information about changing the default.
- ▶ You use DPSIs in some SQL statements.
DPSIs can currently only be non-unique. Some queries, for example, queries with predicates which only reference indexed columns of the index, may experience some performance degradation. These types of queries may need to probe each partition of the DSPI, where only one probe was needed for an NPI. Additional qualifiers might help.
- ▶ You manipulate lots of columns in your SQL.
This code has already been heavily optimized over the years and in V8 it carries the overhead due to 64-bit processing. Reducing the number of columns to the ones really needed is always advisable.

Impact on class 2 CPU time

As you begin to review the CPU utilization for your workload in V8 compared with V7, keep in mind that you may see a slightly higher accounting class 2 CPU time in V8. However the news is not all bad. Table 4-3 compares the increase in accounting class 2 CPU times to the overall CPU consumed.

Table 4-3 Accounting class 2 CPU time increase

	Non-data sharing	Data sharing
Accounting class 2 CPU time	+6%	+14%
MSTR/DBM1/IRLM CPU time	-14%	-51%
Total	+1%	-9%

We can see that for our tests Accounting class 2 CPU times have increased by 6% for non-data sharing and 14% for data sharing in V8 compared to V7.

As we have mentioned earlier, 64-bit addressing has brought its own performance challenges to DB2. 64-bit instructions are typically twice as big as 31-bit instructions and some 64-bit instructions can take as much as 15% more CPU to execute. This impacts class 2 CPU time. This is what contributed to driving the accounting class 2 CPU times higher in DB2 V8.

New DB2 V8 function, like removing the 180 CI limit for list prefetch and castout processing, and long term page fixing of DB2 buffers, works to decrease the overall SRB CPU time, which in turn offsets the increase in accounting class 2 CPU times.

The higher class 2 CPU time in data sharing compared with non-data sharing (14% compared with 6%) is largely attributed to the changes V8 has introduced to the IMMEDIATE BIND option. In V7, DB2 charges the CPU for writing changed pages to the group buffer pool at commit, to the MSTR address space. In V8, this cost is charged to the allied address space. The cost of writing these changed pages has moved from the MSTR address space to the allied agents Accounting class 2 CPU times. See the discussion on IMMEDIATE in Chapter 8, “Data sharing enhancements” on page 319 for more details.

However, the combined impact of other V8 enhancements in data sharing (for example, CF Request Batching for castout processing and Locking Protocol Level 2 which also reduce the overall SRB CPU time), offsets the increase in CPU times seen by the Accounting class 2 reports and produces a lower percentage increase in CPU overall.

Important: Do not just take the accounting class 2 CPU times to compare V7 to V8 performance. You must also factor in any CPU time reduction in the DB2 address spaces from statistics reports.

4.1.3 Recommendations

Even though we have found that the relative overhead of running DB2 V8 in NFM is negligible compared to CM and the extra cost of running a DB2 V7 bound plan/package in V8 is also negligible compared to V7, the following recommendations still apply.

Unless you go to NFM immediately, consider rebinding your plans and packages in V8 CM as soon as you are comfortable you will not fallback to V7. This gives DB2 the opportunity to select a better access path which potentially benefits your SQL performance, especially for more complex queries. The DB2 Optimizer progressively becomes smarter and smarter with each new release of DB2. You are able to experience the *real* V8 performance.

Tuning for CPU usage in V8

If you were virtual storage constrained in V7 and took initiatives to resolve the virtual storage problems, you may choose to reconsider these initiatives in V8. Some of these actions taken to reduce virtual storage usage have a cost of additional CPU. You may no longer need to have these initiatives in place in V8, since you are less likely to be constrained by storage in V8:

- ▶ Increase again the size of buffer pools and other pools you reduced in V7, to once again improve overall performance.
- ▶ Consider rebinding some critical plans and packages with RELEASE(DEALLOCATE) again, only if you were using this bind parameter to improve performance of some key applications in V7. (Remember, however, there is less of a need to have plans and packages bound with RELEASE(DEALLOCATE) in DB2 V8 data sharing than V7 data sharing. See Chapter 8, "Data sharing enhancements" on page 319 for a more detailed discussion of why RELEASE(DEALLOCATE) is less necessary in V8.)
- ▶ Consider turning on thread reuse again, to improve performance of your critical applications.
- ▶ Consider turning off the DSNZPARM EDMBFIT parameter which concerned EDM pool space.
- ▶ Consider turning off the DSNZPARM CONTSTOR parameter which regularly contracts thread local storage pools.
- ▶ Consider turning off the DSNZPARM MINSTOR parameter which changes the way DB2 allocates local storage for thread use.

Other options with significant potential to offset a possible increase in CPU include:

- ▶ Rebind all your plans and packages.
See the above discussion.
- ▶ Long term page fixing your buffer pools

This is a strong recommendation as it has the potential to save as much as 8% in CPU. However, only consider this if you have enough real storage available to back 100% of

your buffer pools. We recommend you implement long term page fixing, buffer pool by buffer pool, for I/O intensive profiles. See 4.5, “Buffer pool long term page fixing” on page 169, for a more detailed discussion on long term page fixing.

- ▶ Multi-row FETCH and INSERT

Although this DB2 V8 enhancement requires you to modify your application, it has a significant potential to save CPU cycles. See the sections on Multi-row FETCH and INSERT in Chapter 3, “SQL performance” on page 29 for more details.

4.2 Virtual storage constraint relief

DB2 V8 exploits 64-bit storage in the DBM1 address space which provides significant virtual storage constraint relief (VSCR).

By way of background, MVS addressing is represented in the power of 2 closest to the corresponding power of 10. Refer to the *z/Architecture Principles of Operation*, SA22-7832 for a more detailed discussion.

2#10 Kilo	(1024 bytes)
2#20 Mega	(1024 * 1024 bytes)
2#30 Giga	(1024 * 1024 * 1024 bytes)
2#40 Tera	(1024 * 1024 * 1024 * 1024 bytes)
2#50 Peta	(a really big number)
2#60 Exa	(an even bigger number)
2#70 Zetta	(too big a number)
2#70 Yotta	(the “Restaurant at the end of the Universe!”)

So 2#64 is 16EB -- and the largest z/Architecture address is 16E-1. DB2 now supports a 16-exabyte DBM1 address space. A 16-exabyte address space is 8 billion times larger than a 2-gigabyte (GB) address space. The new virtual storage area above the old 2 GB limit is referred to as *above the bar*.

Figure 4-3 provides a simplified introductory description of the major storage areas that have moved above the 2 GB bar in the DBM1 address space. The 64-bit storage exploitation in DB2 V8 provides a number of performance improvements to DB2 which we discuss in this section.

The entities which have been moved above the bar in the DB2 V8 DBM1 address space include:

- ▶ 64-bit virtual buffer pool and their control blocks

Larger buffer pools enable I/O reduction for random access, and enable larger page sizes which benefit sequential access. Buffer pool management is also made simpler.

- ▶ DBDs in the EDM pool

More database objects can be defined and I/O to the DB2 directory is reduced because the pool can be larger.

- ▶ EDM statement cache

A larger cache may avoid dynamic SQL PREPAREs.

- ▶ Compression dictionaries

You no longer need to manage the open and close for table spaces to manage the amount of DBM1 storage which is required by compression dictionaries for open compressed table spaces.

- ▶ Sort pools

Frees up space below the bar for other uses, as well as allowing for larger sorts.

- RIDLISTs for the RID pool

Frees up space below the bar for other uses, as well as allowing for larger RID lists to be processed.

- Castout buffers

In case your environment is data sharing.

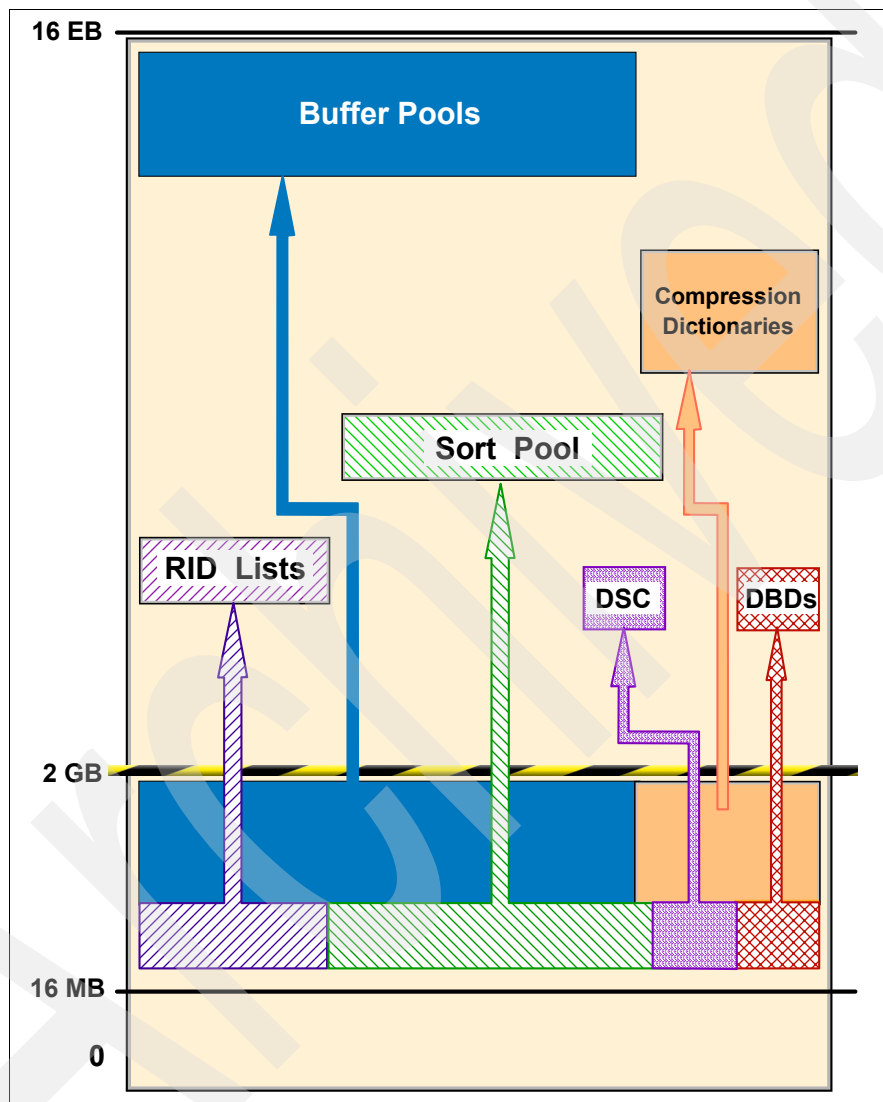


Figure 4-3 DBM1 virtual storage constraint relief (no data spaces nor hiper pools)

By moving storage above the bar, more room is made available below the bar for other entities that must remain there in DB2 V8, such as thread-related storage and storage for plans and packages.

In addition, by moving these structures above the bar, DB2 V8 can support many improvements in scalability, such as larger buffer pools, more compression dictionaries, larger sorts and RID pools.

Likewise, the IRLM address space now exploits 64-bit storage and allows many more locks.

Attention:

LOBs use data spaces with versions prior to V8, and go above the bar in the DBM1 address space with DB2 V8, so they are not influential with respect to VSCR.

If you were using data spaces (max 8 M pages each), and have allocated their buffer pools and global dynamic statements cache (DSC) instead of below the bar in the DBM1 address space, then the VSCR would be less.

If you were using hiper pools (max 8 GB total) to work as cache for your virtual buffer pools, then the VSCR would be less.

4.2.1 The DB2 pools

The GETMAIN and FREEMAIN processes to acquire and release virtual storage in z/OS can be quite expensive in terms of instructions. So, DB2 tends to GETMAIN large blocks of storage from z/OS and then manage the sub-allocation and usage directly.

DB2 maintains two main types of storage pools:

- ▶ **Stack storage (SKB):** Used for very active thread-related storage.
- ▶ **Getmaind block (GMB) storage:** Used for specific purposes such as buffer pools, compression dictionaries, and so on. With DB2 V7 this is generally the largest user of storage below the bar in the DBM1 address space. Storage in the GMB pool includes:
 - Buffer pools and buffer pool control blocks
 - EDM pool
 - Compression dictionaries
 - Data space lookaside buffers (V7)
 - Data space buffer pool control blocks (V7)
 - Hiper pool control blocks (V7)

Pools can also be *fixed* pools, in which the storage is sub-allocated into the same length blocks and optimized for performance or *variable* pools, which are general use pools of storage, sub-allocated into variable length blocks.

Some storage pools are global and shared by multiple threads, for example buffer pools and the EDM pool. In these pools fragmentation is generally smaller, however contention can sometimes be a problem. While other types of storage are thread-related and only used by a single thread, for example, stack, storage and variable pools. In these other types of pools, fragmentation is a bigger problem but contention is much lower. Generally the largest contributors to virtual storage usage are GETMAINED and variable pools; however, the trend in the last few years has been toward growing usage of stack, thread-related, and storage.

We now examine what has changed with DB2 V8 in terms of storage management.

64-bit virtual buffer pool support

The buffer pools are used by DB2 to reduce disk accesses. With more and more data needed by the applications, larger buffer pools have become necessary. The 64-bit virtual addressing is seen as the third major step in the evolution of MVS in the last 30 years. The first step was the transition from MVS/370 (24-bit) to MVS/XA™ (31-bit). The second step was the transition to MVS/ESA™ (including data spaces and hiper pools).

DB2 V7 supported buffer pools in data spaces and hiperpools. They both facilitated larger buffer pools. They each had their advantages and disadvantages. We could not do I/O directly

from the hiperpools, they were not byte-addressable, and their pages could be stolen any time. Data spaces were technically byte-addressable, but, for performance reasons, a lookaside pool was used instead to copy the page. Whenever an application tried to read data from a data space buffer page, the page was copied from the data space into this temporary work area within the 2 GB address space.

Another disadvantage of data spaces and hiperpools is that the control blocks representing these buffers resided in the 2 GB address space. These control blocks created a practical limit to the size of the buffer pools, depending on other demands for virtual storage in the DBM1 region. Given the control block storage requirements, it is unlikely that a single DB2 member could exploit all of the central storage that a z/Series processor is capable of supporting, which is now up to 256 GB, and growing.

DB2 V8 no longer supports data spaces or hiper pools since 64-bit addressing makes them obsolete. DB2 V8 stores all buffers above the bar. DB2 does I/O directly into and out of the buffers and never has to copy the data merely to facilitate byte addressability. DB2 systems management and operations tasks are simplified and the cost of data moves is reduced.

As of DB2 V8, the terms buffer pool and virtual pool become synonymous, with the tendency to use just the term buffer pools.

Buffer pools can now scale to extremely large sizes, constrained only by the physical memory limits of the machine (64-bit allows for 16 exabytes of addressability). However DB2 imposes a 1 TB limit, as a safety valve for real storage available, as follows:

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

The buffer pool control blocks, also known as *page manipulation blocks* (PMBs), are also moved above the 2 GB bar. Castout buffers used for data sharing, are also relocated above the bar.

When you first migrate to V8, DB2 determines the buffer pool size based on the following equation:

$$VPSIZE + HPSIZE = BPSIZE$$

We therefore recommend you review your virtual buffer pool sizes and your hiperpool buffer sizes before you migrate to V8 to make sure you have enough real storage to fully back the new default buffer pool sizes (see 2.7, “New installation default values” on page 24.).

You also need to adjust down the buffer pool thresholds, since they were initially set up for the write activity of just the relatively small virtual pools.

The EDM pool

The Environmental Descriptor Manager (EDM) deals with the DB2 metadata of your databases and applications. It caches and manages the following objects in the EDM pool:

- ▶ Data base descriptors (DBDs), skeleton cursor tables (SKCTs), skeleton plan tables (SKPTs), cache blocks for plans, and skeleton dynamic statements (SKDSs)
- ▶ Private user versions of the plan (CTs and PTs) during execution
- ▶ Authorization cache

DBDs in the EDM pool

In V8, almost all of the storage related to managing DBDs is allocated above the 2 GB bar. This gives the DBDs the needed space to grow and relieves contention with other objects in the EDM pool. DBDs are also larger in DB2 V8 in new-function mode. The size of the DBD

cache is governed by the DSNZPARM EDMDBDC parameter, with a size between 5 MB and 2 GB.

There is little overhead in below the bar storage to support this new DBD pool above the bar. So, make sure you adequately size the EDMDBDC parameter in order to avoid continually loading DBDs from disk.

Storage for plans and packages

Plans and packages (SKCT, CT, SKPT, and PT) and the authorization cache are the only control blocks that are stored in the EDM pool that remains below the bar. The DSNZPARM is EDMPOOL as before. As other DB2 structures have moved above the bar in the DBM1 address space, more space may become available below the bar to accommodate more plans and packages in the EDM pool.

Remember that if you plan to use the current V7 size of the DSNZPARM EDMPOOL parameter in V8, the EDM pool may be over-allocated, since the DBDs (and **potentially** also the dynamic statement cache) no longer reside in that storage area in V8. However, we suggest you monitor your EDM pool performance before making any changes. Plans and packages are bigger in DB2 V8 and consume some of the space in the EDM pool that was vacated by the DBDs. You can monitor the health of the EDM pool by using the pertinent DB2 PE report fields monitoring. If SKPT and SKCT I/O are to be eliminated, oversizing the EDM pool and monitoring SKPT and SKCT requests are best.

The recommendations when monitoring are:

- ▶ Failure due to EDM pool full should be 0
- ▶ Request not found in EDM pool < 1 to 5%
- ▶ CT + PT < 80% of pool

Global dynamic statement cache

In V7, if global dynamic statement caching is active (DSNZPARM CACHEDYN=YES parameter), the statements can be cached in a data space (if the DSNZPARM EDMDSPAC parameter is also >0), or in the normal EDM pool. In V8, cached, dynamic SQL statements are always cached in the EDM statement cache pool above the 2 GB bar. The maximum EDM Statement Cache size, specified by the DSNZPARM EDMSTMTC parameter, is 1 GB.

The default value for EDM STATEMENT CACHE is taken from the EDMPOOL STORAGE SIZE field on panel DSNTIPC, or 5000 K, whichever is larger. The maximum size is 1048576 KB.

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

Although the EDM Statement Cache pool is located above the bar, its size has a real impact on storage below the bar. It impacts the Statement Cache Blocks pool, which contains various control blocks for dynamic SQL statements that are cached in the EDM Statement Cache pool above the bar. It impacts the dynamic SQL pools (local cache) below the bar that are used for statement execution and locally cached statements resulting from KEEP DYNAMIC(YES). The storage consumed below the bar is not only dependent on the size of the EDM Statement Cache above the bar, but is also dependent on the number of SQL statements in the cache.

Note that the PTF for APAR PQ96772 further relieves virtual storage constraint for systems that are very active with dynamic statement caching by moving statement control blocks above the bar.

So, we advise you to not overallocate the EDM Statement Cache, but choose a size based on your needs. Otherwise use the default values. Monitor the EDM Statement Cache and tune it so that it only contains statements which are frequently in use. Having statements in the cache which are rarely used can lead to an impact on performance.

Finally, it is worth noting that SQL statements cached in the EDM Statement Pool Cache may be much larger in V8 than SQL statements cached in V7. This is primarily due to DB2 V8 support for larger names and Unicode.

Compression dictionaries

The compression dictionary for a compressed table space is loaded into virtual storage for each compressed table space or partition as it is opened. Each compression dictionary occupies 64 KB bytes of storage (sixteen 4 KB pages).

Moving the dictionary above the 2 GB bar provides significant storage relief for customers making extensive use of compression. Remember that DB2 V8 can further increase the compression dictionary storage requirement as V8 also implements support for 4096 partitions for a single table; if they are all compressed and open, you would have 4096 compression dictionaries in memory. Notice that most of the open data set DB2 control blocks are also above the bar.

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

SORT pools

Sorting requires a large amount of virtual storage, because there can be several copies of the data being sorted at a given time.

The DB2 Sort pool is allocated on a per thread basis up to a maximum of 128 MB depending on the DSNZPARM SRTPOOL parameter. For example, if SRTPOOL=10, then each concurrent thread doing an RDS sort could allocate up to 10 MB.

Two kinds of storage pools are used for DB2 V8 sort (also known as RDS sort) to store various control structures and data records. One is a thread-related local storage pool below the bar, and the other is a thread-related storage pool sort pool above the bar. We estimate that over 90% of the sort pool is moved above the 2 GB bar.

To take advantage of the 64-bit addressability, some high level sort control structures remain in thread-related storage below the 2 GB bar. These structures contain 64-bit pointers to areas in the thread-related storage pool above the 2 GB bar. Sort tree nodes and data buffers, which comprise 90% of the Sort pool, are located above the 2 GB bar. The maximum Sort pool size, specified by the DSNZPARM SRTPOOL parameter is 128 MB. In V7 it was 64 MB.

In V7 the RDS OP pool is allocated as a single global storage pool. In V8 this has changed. The thread-related local storage pool below the bar is now allocated per thread rather than global. Storage fragmentation can be higher, but contention is reduced since there is no RDS OP pool anymore.

RID lists for the RID pool

The RID pool is split into two parts. The RID pool part below the 2 GB bar stores the RID maps which are usually small in number, and the RID pool part above the 2 GB bar contains the RID lists which normally comprise the bulk of the RID pool storage. Storage size

relationship between RID map and RID list varies from 1 to 1 to 1 to many. For a typical DB2 subsystem with heavy RID pool usage, the great majority of RID pool usage is for RID lists.

The RID map, which is 10% of the RID pool, is located below the 2 GB bar. The RID list, which is 90% of the RID pool, is located above the 2 GB bar. The maximum RID pool size, specified by the DSNZPARM MAXRBLK parameter, is now 10 GB, in DB2 V7, it was 1 GB.

Because of these changes, there are some slight modifications in estimating the size for the RID pool. The same size RIDMAPs would have held half as many RIDLISTs, (because they need to contain pointers to 64-bit memory addresses). The RIDMAP size is doubled to accommodate the same number of 8 byte RIDLISTs, and each RIDLIST now holds twice as many RIDs. Each RIDBLOCK is now 32 KB in size.

If you have a large RID pool, much less storage is used below 2 GB.

LOB data

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

- ▶ LOBVALA (the size per user)
- ▶ LOBVALS (the size per system)

Other virtual storage topics

There are also several other virtual storage-related enhancements such as the star join memory pool, DSNZPARM CONSTOR parameter, DSNZPARM MINSTOR parameter, DSNZPARM MAXKEEPD parameter, and the short prepare performance enhancement.

- ▶ Star join memory pool

The star join memory pool is a new pool of storage in DB2 V8, which only exists when star join is enabled. This new pool is allocated above the 2 GB bar. The star join memory pool extends DB2 V8 support for sparse indexes for star join work files to allow caching data in virtual memory as well as the keys for star join queries. When the star join memory pool is allocated, the in-memory index is no longer sparse because data caching is most effective when the entire work file data is cached. See 3.3, “Star join processing enhancements” on page 53.

A new DSNZPARM parameter, SJMXPOOL, allows you to specify the maximum extent of the new memory pool for star join and can range from 0 to 1024 MB. DB2 may use a smaller portion of the space depending on the number of work files kept in the pool at a certain point of the time. For example, if a value 20 is specified, (which is the default), the total amount of up to 20 MB of data is stored in the pool at a time. When a value of 0 (zero) is specified, the star join memory pool is not allocated. In this case, workfiles are not cached in memory for any star join-qualified queries.

If a non-zero positive value is specified for the SJMXPOOL, the initial allocation size for the pool is the smaller of the specified value and 4 MB. If the specified value is larger than 4 MB, the actual maximum is rounded up to a multiple of 4. For example, if 21 is specified, the pool expands up to 24 MB. One block is allocated for a workfile, if enough space exists in the pool. When the execution of a star join-qualified query finishes, the allocated blocks in the pool to process the statement are freed from the pool. If the memory pool is not created, or the allocation of an individual work file space fails due to the limitation of the pool size, the star join process continues without using the data caching feature. A sparse index (the existing feature for inside-out processing) may be used, instead.

The star join memory pool is externalized in IFCID 225 records.

See 4.3, “Monitoring DBM1 storage” on page 157 for the contents of the 225 record.

► The DSNZPARM CONTSTOR parameter

The CONTSTOR parameter determines whether or not thread storage contraction is to be performed. Storage contraction tries to free up allocated storage that is no longer in use by attempting to contract thread local storage pools at commit. If CONTSTOR is set to YES, two other thresholds are used to govern when storage contraction is performed for a thread.

- SPRMSTH provides a threshold for the amount of storage allocated to a thread before contraction occurs.
- SPRMCTH provides a threshold for the number of commits performed by a thread before contraction occurs.

► The DSNZPARM MINSTOR parameter

The MINSTOR parameter is used to direct DB2 to use storage management algorithms that minimize the amount of working storage consumed by individual threads, when DB2 allocates thread-related storage from local storage pools for threads. Essentially, a best fit storage search algorithm is used instead of a first fit algorithm. The trade-off is less storage fragmentation however slightly more CPU may be consumed.

MINSTOR should normally be set to its default, NO. However, for subsystems that have many long-running threads and are constrained on storage in the DBM1 address space, specifying YES may help reduce the total storage used in the DBM1 address space.

► The DSNZPARM MAXKEEPD parameter

The MAXKEEPD parameter indirectly controls the size of the **Local** Dynamic Statement Cache, which is allocated at the thread level. MAXKEEPD specifies the total number of prepared, dynamic SQL statements to be saved past commit. This is a system wide limit, which is enforced at the thread level when the next commit point is reached for the thread.

The Local Dynamic Statement Cache provides for prepare avoidance and the Global Dynamic Statement Cache (now above the bar in V8) provides for short prepares. Prepare avoidance is cheaper than a short prepare, but the cost of a short prepare is much cheaper (100x) than a typical long prepare.

A suggested tuning approach based on trading CPU for virtual storage constraint relief is to incrementally reduce the size of the Local Dynamic Statement Cache by reducing MAXKEEPD, even setting it to zero, and then monitor performance and storage utilization. You may have to increase the Global Dynamic Statement Cache to compensate, if the Global Dynamic Statement Cache Hit Ratio starts to deteriorate.

In V7, MAXKEEPD has a default of 5000 statements. As the MAXKEEPD threshold is only evaluated at commit, it is common to see the number of SQL statements cached exceed this limit. This is not a problem. For a description of MAXKEEPD at V7 level, see the article *DB2 UDB for OS/390 Storage Management* by John Campbell and Mary Petras, published in The IDUG Solutions Journal Spring 2000 - Volume 7, Number 1.

► Short prepare performance enhancement

The short prepare copies a referenced statement from the EDM statement cache pool into the Local Dynamic Statement Cache and sets it up for execution. The overhead varies depending on the statement execution time but may be 1-2% for long running statements.

There is one Local Dynamic Statement Cache pool allocated for each thread in DB2 V7. This has changed in V8. Now the Local Dynamic Statement Cache is allocated in an array of 30 global pools. This change should reduce the storage fragmentation below the bar, however some contention may occur as a result.

The new approach aims to reduce the cost of the OS level GETMAINs and FREEMAINs by going to a centralized storage approach. With this approach, DB2 uses a number of

storage pools that are owned by the system, not a particular thread. The new implementation uses a fixed number of pools.

The size of the local cache for dynamic statement associated to a long running thread is governed by the MAXKEEPD DSNZPARM.

When a new piece of storage is needed, a hash function is used, to randomly assign a pool. The hash uses the statement identifier as input. Each of the pools is a best fit pool and it extends its size as needed, in 1 MB chunks. With this approach, DB2 relies on the best fit logic to keep the pools at a minimum size. With the thread-based storage model, further reductions in contractions would have led to some storage increase.

The EDM statement cache contains the skeletons of dynamic SQL if your installation has YES for the CACHE DYNAMIC SQL field of installation panel DSNTIP4. The EDM storage statistics have information that can help you determine how successful your applications are at finding statements in the cache. KEEP DYNAMIC (YES) specifies that DB2 keeps dynamic SQL statements after commit points.

Performance measurements show:

- Moving prepared statements from agent pools to multiple shared pools. At MAXKEEPD=12K, the ITR improvement was marginal at 1.4% with negligible delta in DBM1 VSTOR usage.
- The advantage of the new code becomes more apparent at MAXKEEPD=0. There is an ITR improvement of 3.9% with a net DBM1 VSTOR savings of 108 MB.
- Most of the ITR gain comes from the reduction of GETMAINs and FREEMAINs issued.

In DB2 installations using long running threads with a big number of SQL short prepares for dynamic SQL statements, the performance can be improved using MAXKEEPD=0 in V8.

A short prepare is always more expensive than simply using the prepared statement in the centralized shared pools. Short prepare costs 1% or more over prepare avoidance.

MAXKEEPD=0 means do not keep any statements in local dynamic statement cache after commit. In this particular experiment, we are interested in the effectiveness of the new code to support more efficient short prepare. With MAXKEEPD=0, all prepares are short prepare rather than avoiding PREPARE, thereby maximizing any impact from the new code change. This is not the best performing option, you can see on “KEEP DYNAMIC YES” on page 304 that using KEEP DYNAMIC instead can bring 20% ITR improvement.

However MAXKEEPD=0 is good if the environment is virtual storage constrained.

4.2.2 Performance

We examine DB2 real and virtual storage usage by comparing storage usage between V7 and V8 with the same workload at equivalent thread levels. We used various distributed workloads, as well as a local workload and a data sharing workload.

The performance tests used here are for a common workload running with 500 concurrent threads for both DB2 V7 and V8, in all of our comparisons. The IRWW workload with different transaction characteristics was used in order to evaluate the storage considerations when moving different workloads from DB2 V7 to V8.

Note that in all the measurement data we present in this section, we removed the buffer pool storage to get a more accurate picture of the effect on virtual storage.

The DB2 V7 and V8 system storage configuration we used is documented in Table 4-4.

Table 4-4 System storage configuration DB2 V7 vs. DB2 V8

	DB2 V7	DB2 V8
BP type	Primary (MB)	Above 2 GB bar (MB)
4 KB BPs	438	438
8 KB BPs		16
16 KB BPs		16
EDM pool	146	146

DB2 V7 uses a data space for the dynamic statement cache. In addition, there is a negligible use of the Sort pool, RID pool, and global dynamic statement cache in the EDM pool (less than 1 MB). The parameter KEEP_DYNAMIC was set to NO so there is no local dynamic statement cache.

Here are the different types of workloads we examined to understand the impact of static vs. dynamic SQL:

- ▶ Dynamic SQL using CLI and ODBC on AIX, connecting to DB2 Connect and using DRDA to communicate with DB2.
- ▶ Dynamic SQL using the DB2 Universal Driver Type 4 on AIX, connecting to DB2 Connect and using DRDA to communicate with DB2.
- ▶ Static SQL embedded in a local application running on AIX, connecting to DB2 Connect and using DRDA to communicate with DB2.
- ▶ Static SQL using the DB2 Universal Driver Type 4 on AIX, using DRDA to directly connect to DB2.
- ▶ A local IRWW OLTP workload directly connecting to DB2 using IMS attach.
- ▶ A local IRWW OLTP workload directly connecting to a two member data sharing group running DB2 using IMS attach.

During the performance tests, RMF storage statistics were gathered as well as DB2 storage statistics using IFCID 225. In order to validate the storage statistics reported by DB2 PE, and storage information collected from the RMF monitor reports, system dumps were taken of all DB2 address spaces. In addition, SDSF DA (Display Active) real storage usage was captured. As a result, we were able to compare the difference between DB2 V7 and V8 DBM1 and DIST virtual and real storage per DBAT and active connection. The results are presented here.

Remember that your results may be very different as most customer workloads do not resemble this configuration; usually customers have a mix of different applications all running in a single DB2 subsystem. Take this into consideration as you examine the potential impact of your migration to DB2 V8.

Virtual and real storage comparison for DBM1

We first look at virtual storage used by the DBM1 address space.

Figure 4-4 compares the total virtual storage usage for the DBM1 address space for all of these different workloads with DB2 V7 vs. V8. However, to be fair, the virtual buffer pools were removed from the equation; you can see that GETMAINed storage is represented by GM - BP in the graph where the storage for these buffer pools were removed.

A significant decrease in storage below the 2 GB bar is possible. With V8, several DBM1 address space structures that consume large amounts of storage below the bar, have moved

to above the bar, for example, buffer pools, compression dictionaries, DBDs in the EDM pool and some Sort pool and RID list storage. These all come from GETMAINED storage.

The GETMAINED storage after the virtual buffer pools are subtracted in DB2 V7 does not change significantly. The predominant storage areas that increase as you migrate from V7 to V8 in all cases are stack and variable storage. This is primarily thread-related storage which must be larger to support new V8 functionality such as long names, Unicode support and some 64-bit addressing requirements. In addition, in V8 the RDS OP Pool used during bind has been replaced by thread variable storage.

Getmained area is shown as constant for all distributed workloads, but in reality V8 is smaller. The problem is that virtual pool size is artificially subtracted without accounting for an overhead associated with data space buffer pool, such as lookaside buffer and data space buffer control blocks.

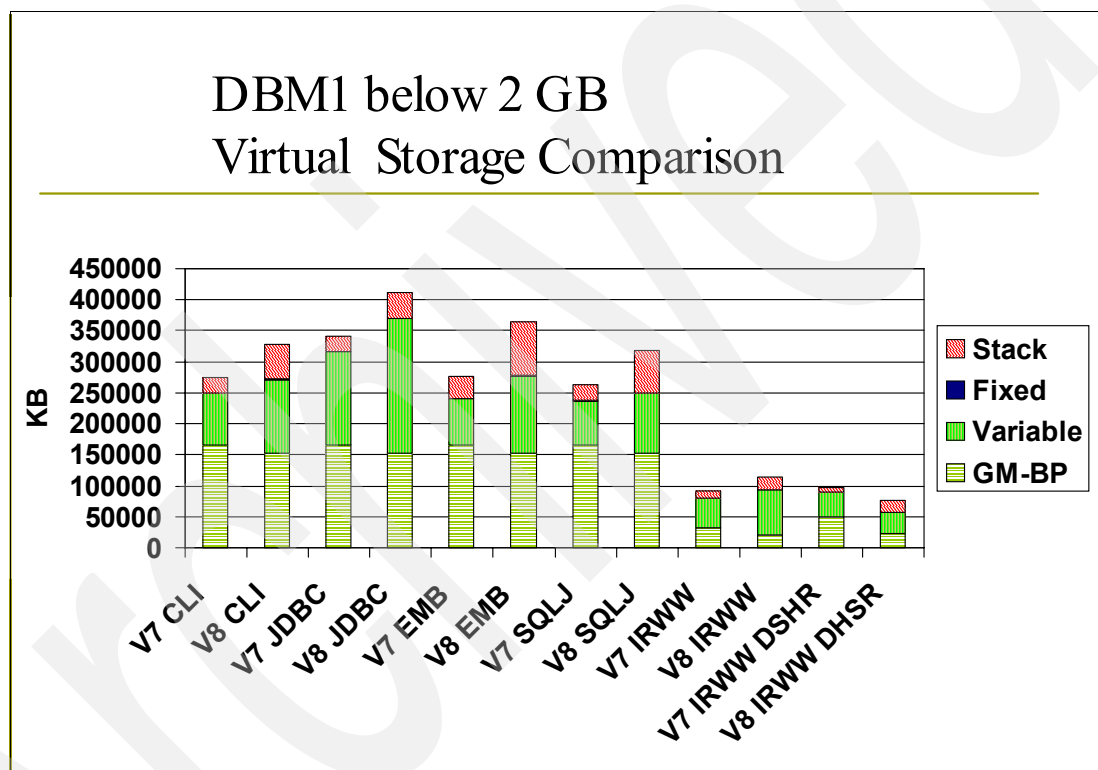


Figure 4-4 DBM1 below 2 GB virtual storage comparison for different workloads

Non-distributed performance

Here we report on the IRWW workload for a non-distributed environment comparing the DBM1 virtual storage consumption for the different main virtual storage areas in V7 and V8 for a non-data sharing environment and a data sharing environment.

Table 4-5 Non-distributed DBM1 virtual storage comparison

	V7 IRWW	V8 IRWW	% change	V7 IRWW	V8 IRWW	% change
	Non-data sharing			Data sharing		
GM - BP	33075	21146	-36%	50248	22830	-55%
Variable	47565	71967	51%	38738	33669	-13%
Fixed	102	328	322%	87	195	124%

	V7 IRWW	V8 IRWW	% change	V7 IRWW	V8 IRWW	% change
	Non-data sharing			Data sharing		
Stack	11356	21033	85%	8648	19799	129%
Total	92098	114473	24%	97721	76493	-22%

Here we observe that the data sharing virtual storage overall decreases by 22% whereas the virtual storage increase in a non-data sharing environment is 24% more. This is also seen visually in Figure 4-4 on page 149.

Part of the increase in total virtual storage in non-data sharing is because DB2 V8 increases the number of deferred write engines from 300 to 600, (however storage for deferred write engines is only allocated as the high water mark for in use deferred write engines increase). This is in addition to larger plan/package/SQL-related structures in DB2 V8 which are required to support long names, Unicode and some 64-bit addressing.

Important: PTF UK14283 for APAR PK21237 has modified the buffer manager engines to use a single common above-the-bar storage pool, rather than having a separate pool for each engine. Additionally, the default maximum for the number of engines has been reduced.

Data sharing shows an overall decrease because the growth in storage described above is offset by the castout buffers also moving above the bar in DB2 V8. This can be shown by the significant decrease in GMB storage in data sharing.

Distributed performance

Table 4-6 summarizes the storage usage results for distributed workloads using CLI, JDBC, embedded SQL, and SQLJ so they can be easily compared.

Table 4-6 Virtual storage comparison for CLI, JDBC, embedded SQL, and SQLJ workloads

	CLI		JDBC		Embedded SQL		SQLJ	
	V8	V7 to V8% increase	V8	V7 to V8% increase	V8	V7 to V8% increase	V8	V7 to V8% increase
DBM1 VS in MB below 2 GB	319	-55% to 26%	402	-48% to 26%	355	-50% to 38%	311	-55% to 28%
DBM1 VS in KB per user thread	338	59%	509	49%	318	52%	323	72%
DBM1 VS in KB agent sys storage per sys agent	223	76%	202	24%	239	32%	244	39%
DIST VS in KB per active connection	358	162%	403	168%	254	97%	383	201%

Attention: The ranges of percentages reported in the row **DBM1 VS (Virtual Storage) in MB below 2 GB** actually reflect the percent increase moving from V7 to V8 with and without the virtual buffer pools. Naturally, a decrease in the virtual storage is possible since the virtual buffer pools move above the 2 GB bar. In the case of large buffer pools, they are likely to be using hiperpools and data space buffer pools, so this would not apply. You also see a larger decrease if you compress some of your data, since compression dictionaries also move above the bar.

We can make the following conclusions about the performance data in Table 4-6:

A general increase in DB2 V8 DBM1 virtual storage below the 2 GB bar in the distributed environment is observed:

- ▶ The virtual storage usage for the DBM1 address space below the 2 GB bar in V8 increases in the range of 24 to 29%. This increase does not take into account the storage for virtual buffer pools. If it did, then the virtual storage usage for the DBM1 address space below the 2 GB bar in V8 decreases in the range of 55 to 48%.

This is primarily due to larger thread-related storage structures required by DB2 V8 to support new functionality such as long names, longer SQL statements, Unicode and 64-bit addressing.

The DBM1 virtual storage increase per distributed user thread is in the range of 49 to 72%. The predominant contributors are non-system thread storage and user stack storage.

For the SQLJ workload the virtual storage per active connection increased by 201%.

- ▶ Total DBM1 VS without BP - up to 38% increase for the embedded SQL workload
- ▶ Per User Thread - up to 73% increase for the SQLJ workload
- ▶ Per System Agent - up to 76% increase for the CLI workload

As virtual storage increases, real storage usage will generally increase in order to back the virtual storage requirement.

User thread storage comparison

The increase in storage is primarily thread-related, as the GETMAINed areas and fixed storage pools did not change with the different workloads in either V7 or V8.

DB2 V8 uses more storage for each thread in the DBM1 address space due to:

- ▶ Thread structures increased in size to support larger names in DB2.
- ▶ The RDS OP pool storage that was used at bind time in V7 is now acquired from the thread's variable storage pool.
- ▶ Some DB2 structures must increase in size to support 64-bit addresses.

This section compares the user thread storage in DBM1 below the 2 GB bar and references Figure 4-5.

DBM1 below 2 GB VS User Thread Storage Comparison

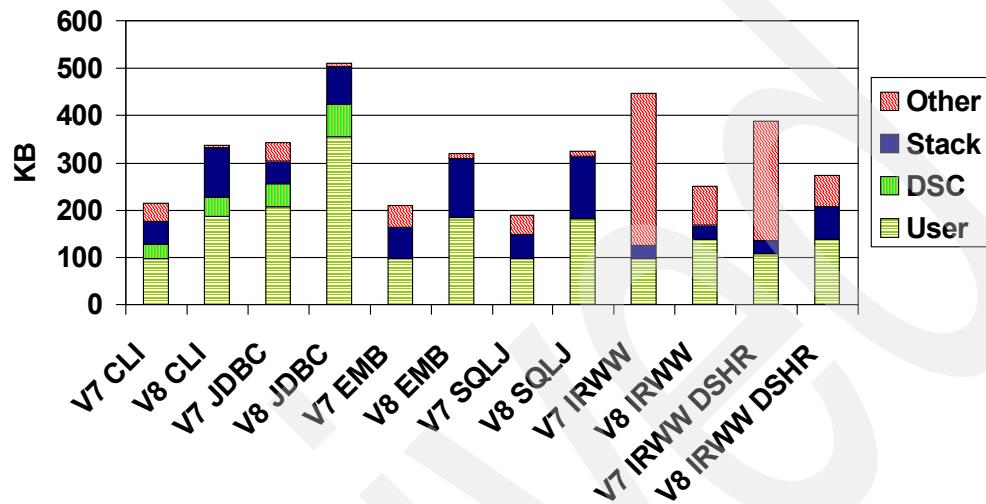


Figure 4-5 DBM1 user thread storage comparison DB2 V7 vs. V8

OTHER includes RDS OP pool (in V7 only), RID pool, and trace tables.

This graph shows that, in general, the DBM1 storage required for each thread increases as we move from DB2 V7 to V8. Comparing this graph with Figure 4-4, we can conclude the increase in thread storage is the most significant driver for the overall storage increase in the DBM1 address space, as we move from DB2 V7 to V8 with distributed applications.

For dynamic SQL, here we can see that more storage is required by DB2 V8 to prepare the dynamic SQL. Once again, this increase in storage is needed to support DB2 functionality such as processing SQL in Unicode and support for long names. We also observe that the dynamic statement cache (DSC) memory usage has increased.

Tip: PTF UK00991for APAR PQ96772 provides significant virtual storage relief for large, dynamic SQL cache environments by moving the DSC control blocks above the bar.

DIST address space

Now, let us turn our attention briefly to the DIST address space, since the total DB2 demands on storage extend beyond the DBM1 address space.

Table 4-6 on page 150 compares the virtual storage consumed by active distributed connections in the DIST address space for DB2 V7 and V8 for all distributed workloads we studied. The table shows quite a significant increase in virtual storage used by active distributed connections in the DIST address space, from 45% to 169% for SQLJ and JDBC workloads respectively. This increase may not be an issue with distributed inactive thread processing (CMTSTAT=INACTIVE).

The majority of this growth is stack storage and the variable subpools. These are very dynamic pools with the storage owned and consumed by thread-related activities.

Thread-related storage is much larger in V8 to support all the new function consumed by threads. The main driver is support for much larger identifiers within DB2.

IRLM address space

IRLM V2.2 which is shipped with DB2 V8, is now a 64-bit application in which the locks now always reside above the 2 GB bar. These changes also have a significant impact on virtual storage used by DB2, if you currently specify PC=NO.

Refer to 4.6, “IRLM V2.2” on page 171, for a more detailed discussion of IRLM and virtual storage.

MSTR address space

The MSTR address space has not caused any problems related to being virtual storage-constrained. In addition, the MSTR address space is still all coded as 31-bit. We therefore have not done any specific studies on virtual storage usage of the MSTR address space in V8 compared with V7.

Real storage

An implication of the z/OS implementation of the z/Architecture in 64-bit mode is there is no more Expanded Storage. If a page or ‘frame’ needs to be paged out of real memory by the system, it is paged out directly to disk. This can have an extraordinary impact on DB2 performance, especially if the buffer pools, or part of the buffer pools, are paged out to auxiliary storage.

DB2 V8 requires more virtual storage to accommodate:

- ▶ Bigger modules, control blocks, and working storage
- ▶ Higher default and maximum buffer pool size, RID pool size, sort pool size, EDM pool size and authorization cache
- ▶ More and larger user threads
- ▶ More deferred write engines and castout engines (300 -> 600 maximum)
- ▶ More parallelism enabled
- ▶ Locks moved above the 2 GB bar

This increased demand for virtual storage therefore increases the demand on the amount of real (or main) storage that is available on the system (maximum 512 GB on a z9 EC model S38 or S54) or LPAR (maximum 128 GB).

There are messages regarding what storage is currently available but the storage information contained in these messages can be overly optimistic. The real storage is by z/OS image or LPAR, and you could have several other DB2 subsystems, each contending for the same real storage. The messages are documented in Table 4-7.

Table 4-7 DB2 storage messages

Message	Message information
DSNB536I	Indicates that the total buffer pool virtual storage requirement exceeds the size of real storage capacity of the z/OS image.
DSNB610I	Indicates that a request to define or increase the size of the buffer pool exceeds the smaller of twice the amount of real storage of the z/OS image or 1 TB.
DSNB508I	Issued when the total size used by all buffer pools exceeds 1 TB.

For the same buffer pool size and other configuration settings, in general, a 1 to 20% increase in demand for real storage for medium to large DB2 V8 subsystems is experienced. For smaller DB2 subsystems, we experience a higher increase. Measurements have seen as little as 4 to 5% increase demand for real storage, in the workloads used.

Your mileage will definitely vary, depending on the type of DB2 workload (for example, simple OLTP SQL compared to complex, query-based SQL) and the current demands DB2 is placing on your real storage. It is important to ask if DB2 is a major user of real storage on your system or a minor user of real storage.

We first compare real storage measurements for various distributed workloads. These scenarios tend to be a “worst case” for studying real storage requirements in DB2 V8, because the DIST address space storage has also increased in V8, which also drives the demand for real storage in the LPAR. Normally you would see less demand on real storage when you are only running a local workload since the DIST address space is not used.

Distributed performance

We can make the following conclusions about the performance data in Table 4-8.

The real storage used for CLI ODBC and embedded SQL workloads is significantly less than those for JDBC and SQLJ. The real storage increase on an LPAR basis is as high as 20% for the JDBC workload, compared with 11% for the CLI workload.

Table 4-8 Real storage comparison for CLI, JDBC, embedded SQL, and SQLJ workloads

	CLI		JDBC		Embedded SQL		SQLJ	
	V8	V7 to V8% increase	V8	V7 to V8% increase	V8	V7 to V8% increase	V8	V7 to V8% increase
Real storage in MB for LPAR	1196	11%	1410	20%	1175	11%	1238	18%
DBM1	547	24%	688	29%	566	29%	523	25%
DIST	70	64%	137	169%	74	73%	63	45%
IRLM	5	139%	5	139%				
MSTR	19	19%	18	19%				
OTHER	555	-3%	563	-2%	535	-8%	652	12%

Note: The row name OTHER records the real storage consumed by the whole LPAR minus those address spaces separately listed in the table.

The real storage values for IRLM and MSTR were not separately captured for the static cases; as a result, these times are reflected in the table under OTHER. The real storage consumed by the IRLM and MSTR address spaces does not have significant changes from the values already noted for the CLI and JDBC tests.

Figure 4-6 illustrates the difference in real storage between DB2 V7 and V8 for the same series of performance tests already discussed in this section. We measure the real storage consumed for the different workloads comparing DB2 V7 usage to that of V8 for the DBM1 and DIST address spaces. As you can see, the real storage growth in DB2 V8 is significant and something you need to prepare for, as you migrate to V8.

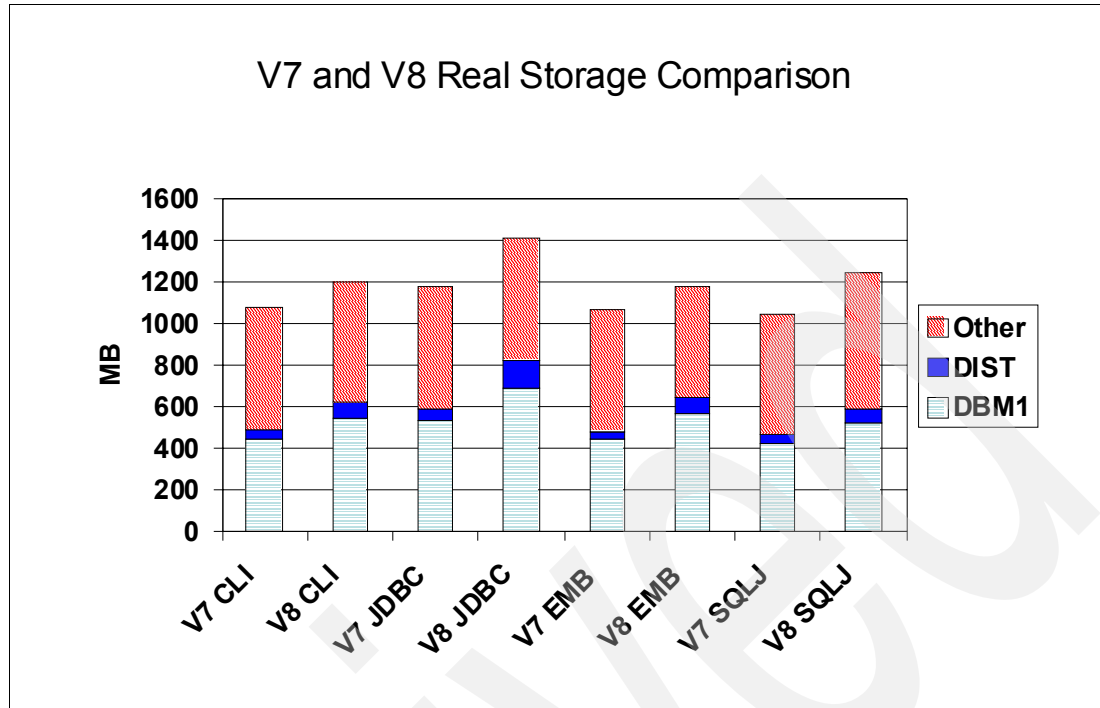


Figure 4-6 Real storage comparison DB2 V7 vs. V8

The “OTHER” bar in the graph represents the sum of the real storage consumed by MSTR, IRLM and other system address spaces. As you can see, this is fairly stable for all the tests. We see an increase in both the DBM1 address space and the DIST address space real storage usage, as we execute the same workload levels (500 DBAT threads) for each distributed workload.

There is a real storage increase of 20% in the JDBC application within the LPAR. For JDBC, the DBM1 real storage increases 29%, the DIST real storage increases 169%. Overall, the DIST address space uses far less storage than the DBM1 address space, even in V8. So, the huge increase in DIST address space storage should not be considered a big problem, unless you have a large distributed workload and are already real memory-constrained.

Although this is not shown on the graph, the MSTR address space also sees an increase of 19% over V7. However, as the MSTR address space only has a small storage footprint, we do not consider this a problem.

Similarly, the IRLM real storage increased 139%. This is also not shown on the graph. For the test workloads we used, the IRLM is not a huge consumer of storage, even in V8, since the workloads we use do not generate a large amount of locks. Your workloads probably generate many more locks and drive IRLM real storage requirements a little more, since many customers generally run a large and very diverse mix of workloads concurrently, which in turn tends to generate more locking activity. The IRLM requires more real storage in V8, primarily due to the increase in size of IRLM control blocks. We therefore recommend you also consider the IRLM storage requirements in your real storage estimates.

Real storage values for MSTR and IRLM do not change much and are included under OTHER.

Nondistributed performance

We examined the performance characteristics for a local workload in both data sharing and non-data sharing environments. The measurement test used the IRWW OLTP workload using 78 concurrent threads with 480 MB of virtual buffer pools in DB2 V7 vs. V8.

There was an 8% increase in real storage usage in DB2 V8 at the LPAR overall level as compared to DB2 V7.

A batch workload with a small number of threads each doing a single-thread table space scan with a 1200 MB buffer pool configuration showed only a 3% increase in real storage usage in DB2 V8 at the overall LPAR level as compared to DB2 V7.

4.2.3 Conclusion

DB2 V8 has significant improvements in virtual storage constraint relief as many of the large DB2 structures (for example, buffer pools, EDM DBDs, and compression dictionaries) in the DBM1 address space have moved above the 2 GB bar. The most important factor here is the percentage of storage used for threads and local DSC compared to other users of DBM1 below 2 GB in V7. This can vary widely among installations.

For the same DB2 configuration (for example, buffer pool sizes), we experienced a 1% to 20% increase in real storage requirements for medium and large DB2 subsystems. Larger increases are possible for smaller DB2 subsystems. This is a result of higher defaults, minimum and maximum buffer pool sizes, RID pool size, Sort pool size, EDM pool size and bigger threads. Bigger modules, control blocks and working storage also contribute to the increase in demand for real storage.

How many additional threads, if any, can be supported depends on the *Thread Footprint*® which varies by workload depending on the duration of the thread, SQL workload, RELEASE parameter setting on plan and package bind, effectiveness of thread storage contraction (CONSTOR=YES), and storage for pools, control blocks, and work areas moved above the 2 GB bar.

4.2.4 Recommendations

DB2 V8 64-bit support does not totally eliminate virtual storage constraint below the 2 GB bar in the DBM1 address space. However, V8 provides valuable additional relief, but the relief varies by installation.

DB2 V8 is able to exploit all available processors' storage on the latest processor models (currently 256 GB, current DB2 limit of 1 TB), through:

- ▶ Supporting extremely large buffer pools to eliminate I/O
- ▶ Exploiting increased ESA Compression
- ▶ Using larger thread Sort pool
- ▶ Allowing more compressed table spaces and partitions

You must have sufficient real storage to fully back increased storage use usage.

You must still continue to plan for, monitor and tune virtual storage usage below the 2 GB bar. The DB2 Lab's recommendation is that users should implement a methodology which incorporates the development and measurement of a production workload to achieve accurate projections for virtual storage recommendations. The resulting measurements need to be evaluated using formulas for calculating the thread storage footprint (virtual storage per thread) and the estimation of the maximum possible threads.

If you have not measured the amount of storage per thread, here are some rough estimates:

- ▶ Low-end: 200 to 400 KB - Static
- ▶ Mid-range: 500 KB to 2 M - Static/Dynamic
- ▶ High-end: 3 MB to 10 MB or higher - Dynamic, heavy sort, parallelism applied, long running persistent thread (WFI, CICS Protected Entry Threads with many packages with RELEASE(DEALLOCATE) or (RELEASE(COMMIT) with CONTSTOR=NO).

These ranges are based on historical observations of typical SQL statements utilized in customer production environments, the low-end of the range representing static SQL vs. the high-end representing dynamic SQL-based workloads. The estimation should be verified and modified appropriately through the actual measurement using the above formulas.

4.3 Monitoring DBM1 storage

Despite the fact that DB2 V8 provides many enhancements to relieve virtual storage constraints in DB2, it is still important you proactively monitor virtual storage usage within DB2.

It is also important for you to monitor storage usage above the bar. Within the z/OS implementation of the z/Architecture, there is no expanded storage. Paging goes directly to disk, and then you will obviously suffer the performance impact of writing to and reading from disk. DB2 should rarely page to disk. So, it is more important than ever now to monitor the number of real frames in use and ensure auxiliary storage is never used.

DB2 provides two new messages to help you not overallocate your buffer pools relative to the amount of real storage available. This is important as buffer pools can now scale to extremely large sizes, constrained only by the physical memory limits of the machine (64-bit allows for 16 exabytes of addressability).

- ▶ **DSNB536I:** This warning message indicates the total buffer pool virtual storage requirement of this DB2 subsystem exceeds the size of real storage of the z/OS image. To relieve this situation you can either allocate more real storage to the z/OS image or, if there is not enough real storage to hold the buffers, then the number of buffers needs to be reduced.
- ▶ **DSNB610I:** This warning message indicates that a request to increase the size of the buffer pool will exceed twice the real storage available to the z/OS image, or the normal allocation of a buffer pool not previously used will cause an aggregate size which exceeds the real storage. Either request is then limited to 8 MB (2000 pages for 4 KB, 1000 pages for 8 KB, 500 pages for 16 KB, and 250 pages for 32 KB).

DB2 still monitors Short on Storage (SOS) conditions below the bar in the DBM1 address space by monitoring a storage cushion within the DBM1 address space. When the head room is less than this storage cushion value, DB2 attempts to contract and reclaim free memory segments which have been fragmented. This work can impact DB2 performance if DB2 is continually in this state.

DB2 V8 introduces a number of new counters to IFCID 225 to support DB2 V8. In addition the IFCID 225 is now available via IFI READS to V8 and V7 (APAR PQ85764), this makes its contents immediately available to online monitors instead of the last interval values.

The statistics gathered represent a point-in-time view of storage used in the DBM1 address space at the time the record is cut. They do not record information about storage utilization in any other address space. IFCID 225 records are cut at statistics intervals whenever the DB2 Statistics class 6 trace is active.

Important: With PTFs UK04394 (DB2 V8) and UK04393 (DB2 V7) for APAR PQ99658 the availability of virtual storage diagnostic record IFCID 225 is added to Statistics Class 1, the system parameter STATIME default time is reduced from 30 minutes to 5, and an internal cache of IFCID 225 records is being kept to track storage changes over time.

Example 4-1 shows an extract from a DB2 PE V8.1 Statistics Long Trace, showing the DBM1 STORAGE STATISTICS sections for a DB2 V7 subsystem. The report was produced with the fix for APAR PQ91101 applied to DB2 V7 and the fix for APAR PQ94872 applied to DB2 PE. See also 10.2, “DB2 Performance Expert” on page 364, and Appendix B, “The DBM1 storage map” on page 393.

Example 4-1 DBM1 Storage Statistics for a DB2 V7 subsystem

DBM1 AND MVS STORAGE BELOW 2 GB		QUANTITY	DBM1 AND MVS STORAGE BELOW 2 GB		CONTINUED	QUANTITY
TOTAL DBM1 STORAGE BELOW 2 GB	(MB)	779.63	24 BIT LOW PRIVATE	(MB)		0.21
TOTAL GETMAINED STORAGE	(MB)	599.18	24 BIT HIGH PRIVATE	(MB)		0.43
VIRTUAL BUFFER POOLS	(MB)	437.50	31 BIT EXTENDED LOW PRIVATE	(MB)		24.64
VIRTUAL POOL CONTROL BLOCKS	(MB)	13.67	31 BIT EXTENDED HIGH PRIVATE	(MB)		794.60
EDM POOL	(MB)	146.48	EXTENDED REGION SIZE (MAX)	(MB)		1628.00
COMPRESSION DICTIONARY	(MB)	0.00	EXTENDED CSA SIZE	(MB)		256.83
CASTOUT BUFFERS	(MB)	0.00				
DATA SPACE LOOKASIDE BUFFER	(MB)	0.00	AVERAGE THREAD FOOTPRINT	(MB)		0.30
HIPERPOOL CONTROL BLOCKS	(MB)	0.00	MAX NUMBER OF POSSIBLE THREADS			2593.01
DATA SPACE BP CONTROL BLOCKS	(MB)	0.00				
TOTAL VARIABLE STORAGE	(MB)	153.39				
TOTAL AGENT LOCAL STORAGE	(MB)	109.16				
TOTAL AGENT SYSTEM STORAGE	(MB)	3.54				
NUMBER OF PREFETCH ENGINES		5				
NUMBER OF DEFERRED WRITE ENGINES		26				
NUMBER OF CASTOUT ENGINES		0				
NUMBER OF GBP WRITE ENGINES		0				
NUMBER OF P-LOCK/NOTIFY EXIT ENGINES		0				
TOTAL AGENT NON-SYSTEM STORAGE	(MB)	105.63				
TOTAL NUMBER OF ACTIVE USER THREADS		500				
RDS OP POOL	(MB)	1.15				
RID POOL	(MB)	0.57				
PIPE MANAGER SUB POOL	(MB)	0.00				
LOCAL DYNAMIC STMT CACHE CNTL BLKS	(MB)	0.99				
THREAD COPIES OF CACHED SQL STMTS	(MB)	22.77				
IN USE STORAGE	(MB)	N/A				
STATEMENTS COUNT		N/A				
HWM FOR ALLOCATED STATEMENTS	(MB)	N/A				
STATEMENT COUNT AT HWM		N/A				
DATE AT HWM		N/A				
TIME AT HWM		N/A				
BUFFER & DATA MANAGER TRACE TBL	(MB)	16.95				
TOTAL FIXED STORAGE	(MB)	0.30				
TOTAL GETMAINED STACK STORAGE	(MB)	26.77				
STORAGE CUSHION	(MB)	132.58				
DBM1 STORAGE ABOVE 2 GB		QUANTITY	REAL AND AUXILIARY STORAGE			QUANTITY
FIXED STORAGE	(MB)	N/A	REAL STORAGE IN USE	(MB)		516.13
GETMAINED STORAGE	(MB)	N/A	AUXILIARY STORAGE IN USE	(MB)		0.00
COMPRESSION DICTIONARY	(MB)	N/A				
CACHED DYNAMIC SQL STATEMENTS (MAX)	(MB)	N/A				
DBD CACHE (MAX)	(MB)	N/A				
VARIABLE STORAGE	(MB)	N/A				
VIRTUAL BUFFER POOLS	(MB)	N/A				
VIRTUAL POOL CONTROL BLOCKS	(MB)	N/A				
CASTOUT BUFFERS	(MB)	N/A				
STAR JOIN MEMORY POOL	(MB)	N/A				

The format of the DBM1 Storage Statistics sections of the DB2 PE Statistics Detail Report is enhanced over previous versions. See *DB2 for z/OS and OS/390 Version 7 Selected Performance Topics*, SG24-6894, for an example of the previous report. The new report is hierarchical, more complete and logically organized.

Be aware that IFCID 225 represents a “snapshot” of DBM1 storage usage at a particular point in time. On an active subsystem, it is not uncommon for DB2 to be in the process of allocating storage while freeing up other storage. So, the numbers presented in IFCID 225 may keep changing and do not always add up. However, they become consistent after a while and can be used as a good guide to what is happening in the DBM1 address space. In addition, keep in mind that these numbers show the DB2 side. MVS has a different picture which tends to be more pessimistic. DB2 generally acquires storage from MVS in *chunks* but may not immediately use the storage for any particular purpose within DB2. MVS sees the whole chunk as being allocated and used.

We have seen that DB2 maintains four main types of storage:

- ▶ Stack Storage (SKB): Used for very active thread-related storage.
- ▶ Getmaind Block (GMB) Storage: Used for specific purposes such as buffer pools and compression dictionaries.
- ▶ Fixed pools : Storage is sub-allocated into the same length blocks and optimized for performance.
- ▶ Variable pools: General use pools of storage, sub-allocated into variable length blocks.

In this example, the TOTAL DBM1 STORAGE BELOW 2 GB is 779.63 MB.

This total can be split up into the following components:

- ▶ TOTAL GETMAINED STORAGE = 599.18 MB
- ▶ TOTAL VARIABLE STORAGE = 153.39 MB
- ▶ TOTAL FIXED STORAGE = 0.30 MB
- ▶ TOTAL GETMAINED STACK STORAGE = 26.77 MB

GETMAINED Storage

The TOTAL GETMAINED STORAGE is storage acquired by DB2 as GETMAINED Blocks (GMB). In V7, this is generally the largest consumer of storage in the DBM1 address space. It includes space for virtual buffer pools, virtual buffer pool control blocks, the EDM pool, compression dictionaries, castout buffers, the data space lookaside buffers, hiper pool control blocks and data space buffer pool control blocks. We can see these indented in the report. Note however that these numbers may not add up to TOTAL GETMAINED STORAGE because DB2 does not collect statistics on all GETMAINED storage.

Buffer pools are usually the major consumers of DBM1 storage.

- ▶ The storage required for the virtual buffer pools is shown by the following indicator:
 - VIRTUAL BUFFER POOLS = 437.50 MB
- ▶ Independent of the page size, a control block of 128 bytes (V7) or 144 bytes (V8) is allocated for each page in the virtual buffer pool. The storage required is shown by the following indicator:
 - VIRTUAL BUFFER POOL CONTROL BLOCKS = 13.67 MB
- ▶ Hiperpools are not allocated in the DBM1 address space but storage is allocated inside the DBM1 address space for the 56-byte hiperpool control block associated with each hiperpool buffer. The storage required is shown by the following indicator:
 - HIPERPOOL CONTROL BLOCKS = 0 MB (hiperpools are not used here)
- ▶ If you allocate the buffers to data spaces (V7), additional storage is required in the DBM1 address space. A control block of 128 bytes is allocated for each data space buffer defined. The lookaside buffers also utilize a sizable amount of DBM1 virtual storage. The storage required is shown by the following indicators:

- DATASPACE BP CONTROL BLOCKS = 0 MB (data spaces are **not** used here)
- DATASPACE LOOKASIDE BUFFER = 0 MB

In our example, buffer pools and control structures use a total of 451.17 MB, which represents almost 58% of the total DBM1 storage.

Besides the virtual storage required for the buffer pools, other components can be large consumers of virtual storage in the DBM1 address space depending on the workload and system parameter settings:

- ▶ The EDM pool containing active and skeleton plans and packages, dynamic statements, and Database Descriptors (DBDs). The maximum size is specified by the system parameter EDMPOOL in DSNZPARM. The storage used is shown by the following indicator:
 - EDM POOL = 146.48 MB
- ▶ Compression dictionaries can also consume significant amounts of storage in the DBM1 address space, especially when a compression dictionary is required for every partition of a partitioned table space. The compression dictionary for each table space/partition that is opened. The size of a compression dictionary is 64 KB. The storage required is shown by the following indicator:
 - COMPRESSION DICTIONARY = 0 MB (compression is not used here)
- ▶ DB2 allocates Castout buffers only in a data sharing environment. Castout buffers are used by DB2 to read changed pages from the GBP into the DBM1 address space, as a part of writing them out to disk. Since 128 KB are allocated per castout engine, and you can have up to 600 castout engines, the total can be up to 77 MB of storage.
 - CASTOUT BUFFERS = 0.00 MB (data sharing is not used)

Variable pools

The TOTAL VARIABLE STORAGE contains storage used by threads or agents as working storage, RID pool, RDS OP pool, Pipe manager subpool (working storage pools used by parallel tasks), local dynamic statement cache control blocks, thread copies of cached SQL statements (each thread's Local Dynamic Statement Cache) and storage for various trace tables. Agent storage is further itemized into storage for "system agents" (such as prefetch engines, deferred write engines, castout engines, GBP write engines, P Lock exit engines) and "non-system agents", which includes the number of all active allied threads and active DBAT threads.

- ▶ TOTAL AGENT LOCAL STORAGE = 109.16 MB combines both TOTAL AGENT SYSTEM STORAGE and TOTAL AGENT NON-SYSTEM STORAGE.
- ▶ TOTAL AGENT SYSTEM STORAGE = 3.54 MB. This is the total storage consumed in the variable pools by the various system agents (for example, prefetch engines, deferred write engines, and castout engines).
- ▶ TOTAL AGENT NON-SYSTEM STORAGE = 105.63 MB. This is the total storage consumed in the variable pools by all the active threads. The normal range for DB2 V8 is 200 KB to 10 MB per thread. Simple threads like CICS transactions can be as low as 200 KB while SAP threads can be as much as 10 MB per thread.
- ▶ RID POOL = 0.57 MB. This is the total storage consumed in the variable pools for the RID pool. The RID pool is used for list prefetch, multiple index access processing, and hybrid joins. The maximum size is specified by the DSNZPARM parameter MAXRBLK.
- ▶ The Local Dynamic Statement Cache size is indirectly determined by the DSNZPARM parameter MAXKEEPD when KEEP DYNAMIC(YES) is used. The storage used in the virtual pools is shown by the following indicators:

- LOCAL DYNAMIC STMT CACHE CTL BLKS = 0.99 MB
- THREAD COPIES OF CACHED SQL STMTS = 22.77 MB

DB2 V7 does not further break down the storage used for THREAD COPIES OF CACHED SQL STMTS. So, the values are recorded as N/A.

- RDS OP POOL = 1.15 MB. For DB2 V7 there is a single RDS OP pool for all threads which is used for sort, prepares and other similar functions.
- BUFFER & DATA MANAGER TRACE = 16.95 MB

Stack Storage and Fixed Pools

The TOTAL FIXED STORAGE is typically used for DB2 code and does not tend to vary a great deal, while the TOTAL GETMAINED STACK STORAGE is typically used for program stack usage, for example, save areas. It does vary as your workload varies.

DBM1 and MVS Storage below 2 GB

Figure 4-7 presents an overview of the MVS storage map. You may like to refer to this illustration as we continue to explain more items in the DBM1 Storage Statistics extract in Example 4-1.

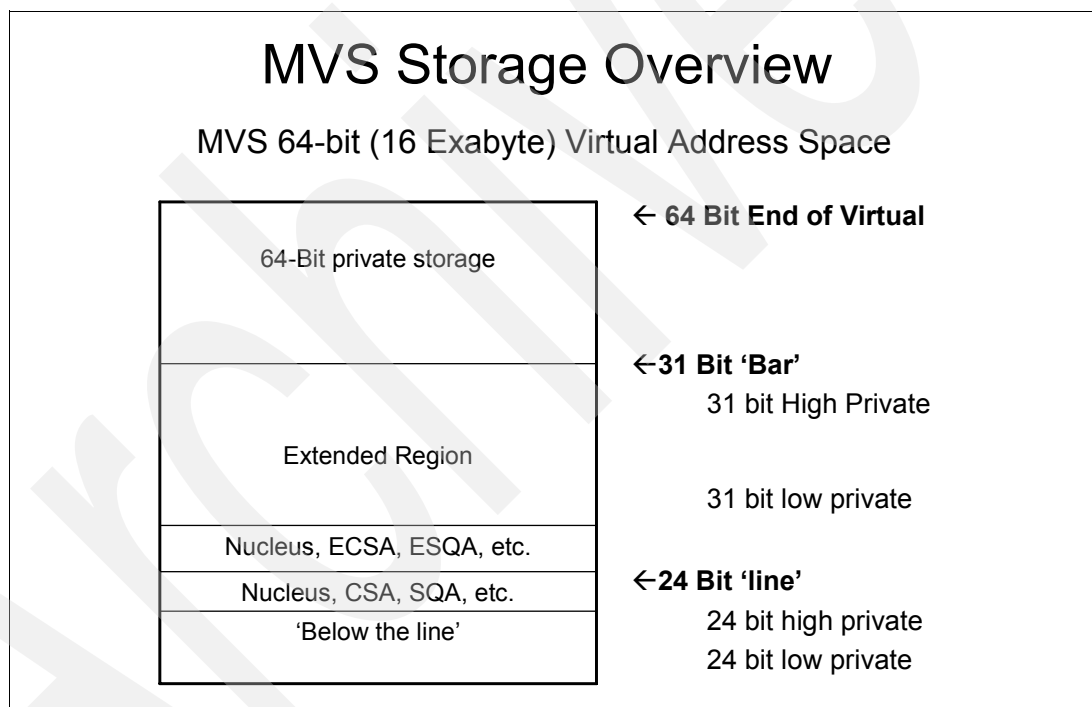


Figure 4-7 MVS storage overview

The 31-bit EXTENDED LOW PRIVATE is the amount of used storage just above the 16 MB line. It is generally used by unauthorized programs. DB2 code is loaded here, as well as data set control blocks. This storage grows from the bottom up in the extended region. The 31-bit EXTENDED HIGH PRIVATE is the amount of storage above the 16 MB line generally used by authorized programs. This storage grows top down in the extended region. All GETMAINED DB2 storage is obtained in this way. MVS has a rule that high private cannot grow into low private. The REGION parameter in the JCL controls how high the low private storage can grow (0 MB indicate indefinitely). For example, the REGION parameter can be used to save a few MB of extended private storage by forcing some low private growth into 24-bit low private. Extended low private is often controlled with the system-wide MVS IEFUSI exit which can treat differently different address spaces.

The EXTENDED REGION SIZE (MAX) is the size of the region available to DB2 after MVS has allocated all common ECSA, ELPA, and ENUC type storage. It is commonly thought that DB2 has 2 GB available, but this is generally more in the range of 1000 MB - 1600 MB. Most of the storage DB2 allocates below the bar is here, in MVS subpool 229 or 230 key 7.

The general recommendation for monitoring and tuning DB2 storage is to have approximately 200 MB free for fluctuations in workload. This recommendation is also true for DB2 V8. 200 MB is a good conservative number, but you may think of reducing this if your DB2 subsystem is small.

The EXTENDED CSA SIZE is the common storage area across all address spaces for a given LPAR. A large ECSA allocation would be .5 to .8 GB, while a typical allocation would be .3 to .5 GB.

MVS is very likely to report a different number in EXTENDED REGION SIZE (MAX) by as much as 100 MB higher (the difference is in MVS overhead that DB2 cannot track.)

When DB2 thinks it is running low on extended virtual storage, DB2 begins a process called "system contraction", which attempts to freemain any available segments of storage in fixed and variable pools back to MVS. (This process can be CPU-intensive and cause latch contention). If the storage DB2 has reserved based on the number of threads that are allowed in the system, plus the storage DB2 has reserved for open and close of data sets based on DSMAX, plus the storage DB2 has reserved for must complete operations (such as ABORT and COMMIT), is less than DB2's opinion of what storage is available, then contraction occurs.

Therefore, it is important to correctly estimate CTHREAD, MAXDBAT and DSMAX, and not be overly specific in estimating them.

Storage estimates for threads

The following formulas apply to storage between the 16 MB line and the 2 GB bar.

The current formula for the **Thread Footprint (TF)** is as follows:

$$\frac{(\text{TOTAL VARIABLE STORAGE} - \text{TOTAL AGENT SYSTEM STORAGE})}{\text{TOTAL NUMBER OF ACTIVE USER THREADS}}$$

$$(153.39 - 3.54) / 500 = 0.2997$$

This value is also reported in the report as AVERAGE THREAD FOOTPRINT as 0.30 MB.

The fixed amounts of storage within the DBM1 address space do not vary a great deal as workload increases. They can be considered as a fixed overhead, for example, buffer pools, EDM pool, buffer pool control blocks and compression dictionaries. The total fixed storage overhead (F) can be estimated as:

$$\text{TOTAL GETMAINED STORAGE} + \text{TOTAL GETMAINED STACK STORAGE} + \text{TOTAL FIXED STORAGE} + 31 \text{ BIT EXTENDED LOW PRIVATE}$$

or

$$599.18 + 26.77 + 0.30 + 24.64 = 650.89 \text{ MB}$$

The **value V for upper limit variable areas** is defined as:

$$V = R - C - F$$

where:

- R is the theoretical maximum region size

= MVS EXTENDED REGION SIZE (MAX) - MVS 31 BIT EXTENDED LOW PRIVATE

or

1628.00 - 24.64 = 1603.36 MB

► C is the basic cushion constant of 200 MB.

► F is the value for fixed areas

= TOTAL GETMAINED STORAGE + TOTAL GETMAINED STACK STORAGE + TOTAL FIXED STORAGE

or

599.18 + 26.77 + 0.30 = 626.25 MB

The **maximum theoretical number of threads** can therefore be estimated as:

V/TF

For our example, the theoretical maximum number of threads can be estimated as:

(1603.36 - 200 - 626.25) / 0.3 = 777.11 / 0.30 = 2590.37.

This is also reported in the report as MAX NUMBER OF POSSIBLE THREADS as 2593.01. The difference between 2590 and 2593 can be explained because the above calculation is based on the rounded MB values of the report, while Performance Expert uses the exact byte values of IFCID 225.

Be warned that this is a maximum theoretical limit and should only be used as a guide or rough estimate. It does not provide for things such as growth in buffer pools and more compression dictionaries that may be required to support the extra workload.

The stack storage size is directly proportional to the number of threads, not FIXED. Even Getmained ones are really not fixed. Threads would typically be much bigger in an environment which has many more threads. If a measurement run has 100 threads and the maximum is calculated much higher for the projection, such as 1000, the result in the maximum number of threads allowed tends to be much higher than really possible.

So, what has changed in DB2 V8?

Example 4-2 shows the same extract from a DB2 PE V8.1 Statistics Long Trace, but this time it shows the DBM1 STORAGE STATISTICS section for a DB2 V8 subsystem. Once again, the report was produced with the fix for APAR PQ91101 applied to DB2 and the fix for APAR PQ94872 applied to DB2 PE.

Example 4-2 DBM1 Storage Statistics for a DB2 Version 8 subsystem

DBM1 AND MVS STORAGE BELOW 2 GB		QUANTITY	DBM1 AND MVS STORAGE BELOW 2 GB		CONTINUED	QUANTITY
-----		-----	-----		-----	-----
TOTAL DBM1 STORAGE BELOW 2 GB	(MB)	539.30	24 BIT LOW PRIVATE	(MB)		0.21
TOTAL GETMAINED STORAGE	(MB)	236.09	24 BIT HIGH PRIVATE	(MB)		0.43
VIRTUAL BUFFER POOLS	(MB)	N/A	31 BIT EXTENDED LOW PRIVATE	(MB)		30.45
VIRTUAL POOL CONTROL BLOCKS	(MB)	N/A	31 BIT EXTENDED HIGH PRIVATE	(MB)		553.99
EDM POOL	(MB)	234.38	EXTENDED REGION SIZE (MAX)	(MB)		1628.00
COMPRESSION DICTIONARY	(MB)	N/A	EXTENDED CSA SIZE	(MB)		256.83
CASTOUT BUFFERS	(MB)	N/A				
DATA SPACE LOOKASIDE BUFFER	(MB)	N/A	AVERAGE THREAD FOOTPRINT	(MB)		0.46
HIPERPOOL CONTROL BLOCKS	(MB)	N/A	MAX NUMBER OF POSSIBLE THREADS			2377.81
DATA SPACE BP CONTROL BLOCKS	(MB)	N/A				
TOTAL VARIABLE STORAGE	(MB)	231.47				
TOTAL AGENT LOCAL STORAGE	(MB)	195.72				
TOTAL AGENT SYSTEM STORAGE	(MB)	2.32				
NUMBER OF PREFETCH ENGINES		6				
NUMBER OF DEFERRED WRITE ENGINES		10				
NUMBER OF CASTOUT ENGINES		0				
NUMBER OF GBP WRITE ENGINES		0				
NUMBER OF P-LOCK/NOTIFY EXIT ENGINES		0				
TOTAL AGENT NON-SYSTEM STORAGE	(MB)	193.40				
TOTAL NUMBER OF ACTIVE USER THREADS		500				

RDS OP POOL	(MB)	N/A
RID POOL	(MB)	0.35
PIPE MANAGER SUB POOL	(MB)	0.00
LOCAL DYNAMIC STMT CACHE CNTL BLKS	(MB)	3.93
THREAD COPIES OF CACHED SQL STMTS	(MB)	29.10
IN USE STORAGE	(MB)	16.13
STATEMENTS COUNT		2096
HWM FOR ALLOCATED STATEMENTS	(MB)	16.13
STATEMENT COUNT AT HWM		2096
DATE AT HWM		10/23/04
TIME AT HWM		00:59:04.55
BUFFER & DATA MANAGER TRACE TBL	(MB)	N/A
TOTAL FIXED STORAGE	(MB)	0.46
TOTAL GETMAINED STACK STORAGE	(MB)	71.28
STORAGE CUSHION	(MB)	123.79

DBM1 STORAGE ABOVE 2 GB	QUANTITY	REAL AND AUXILIARY STORAGE	QUANTITY
FIXED STORAGE (MB)	0.02	REAL STORAGE IN USE (MB)	694.48
GETMAINED STORAGE (MB)	2221.78	AUXILIARY STORAGE IN USE (MB)	0.00
COMPRESSION DICTIONARY (MB)	0.00		
CACHED DYNAMIC SQL STATEMENTS (MAX) (MB)	1024.00		
DBD CACHE (MAX) (MB)	1024.00		
VARIABLE STORAGE (MB)	173.78		
VIRTUAL BUFFER POOLS (MB)	468.75		
VIRTUAL POOL CONTROL BLOCKS (MB)	0.14		
CASTOUT BUFFERS (MB)	0.00		
STAR JOIN MEMORY POOL (MB)	0.00		

The TOTAL GETMAINED STORAGE has decreased from 599.18 MB in V7 to 236.09 MB in V8. This is a direct result of buffer pools, buffer pool control blocks, data space lookaside buffers, castout buffers and compression dictionaries all moving above the bar. We can see them now recorded in a new area of the report, titled DBM1 STORAGE ABOVE 2 GB. You will also notice these fields are now marked as N/A under the TOTAL GETMAINED STORAGE section of the report.

For V8, RID Pool is only referred to RID map, since RID lists have been moved above the bar.

Note that the RDS OP pool is recorded as N/A for V8. In DB2 V7, there was a single RDS OP pool reported, which all threads use. However in DB2 V8, the RDS OP pool storage for each thread is allocated from the thread's variable length storage pool. We can see the TOTAL GETMAINED STACK STORAGE jump from 26 MB to 71 MB. This increase is primarily due to larger thread-related structures in V8, since they must be larger to accommodate long names.

So, in V8 we see a considerable decrease in TOTAL GETMAINED STORAGE and a considerable increase in TOTAL GETMAINED STACK STORAGE.

DB2 V8 also breaks down the THREAD COPIES OF CACHED SQL STMTS storage into more meaningful statistics. Cached SQL statements for dynamic SQL can cause quite a lot of virtual storage to be consumed.

- THREAD COPIES OF CACHED SQL STMTS reports the virtual storage consumed by copies of currently executable SQL statements and the executable statements that are kept beyond commit when KEEP DYNAMIC YES is used.
- IN USE STORAGE reports the amount of virtual storage that is actually allocated to locally cached dynamic SQL statements, discounting storage not used in the storage allocation segment.
- STATEMENTS COUNT is the number of locally cached dynamic SQL statements that are actually using the storage.

The following formula can give you some insight into the degree of fragmentation of storage which is used to locally cache dynamic SQL statements.

$$(\text{THREAD COPIES OF CACHED SQL STMTS} - \text{IN USE STORAGE}) / \text{STATEMENTS COUNT}$$

In addition, DB2 shows a high water mark for storage consumed by locally cached dynamic SQL statements. The high water marks are measured since the last time the IFCID 225 record was externalized.

Now, with DB2 V8, if you look only at virtual storage, you get a partial view. DB2 V8 now can allocate very large amounts of virtual storage above the bar which may not be backed by real memory. This can be very bad for performance. The DBM1 STORAGE ABOVE 2 GB counters do not provide the whole picture.

So the IFCID 225 record and DBM1 Storage Statistics section of the DB2 PE Statistics reports also report on DBM1 Storage usage above the 2 GB bar, as well as provide an indication of real storage frames used by the DBM1 address space. These statistics are displayed for both DB2 V7 and DB2 V8.

REAL STORAGE IN USE reports on the amount of real storage in megabytes, used by the DBM1 address space. Ideally, this should be significantly less than the amount of virtual storage consumed. (TOTAL DBM1 STORAGE BELOW 2 GB + all the counters in TOTAL DBM1 STORAGE ABOVE 2 GB.)

AUXILIARY STORAGE IN USE is the amount of storage in megabytes the DBM1 address space has allocated in auxiliary storage (disks). This should be zero or very close to it.

The following points are indicators DB2 may be suffering from storage-related problems, such as a DB2 code error (storage leak) or a really stressed system due to a lack of real storage:

- ▶ 100% real frames consumed. Note that although a physical machine may have 30 GB of real storage, a given LPAR may only have a fraction of this real storage dedicated.
- ▶ An excessive number of auxiliary paging slots in use.
- ▶ A general performance degradation.

Dumps are also another useful tool to monitor storage usage within the DBM1 address space. However, they only capture a point-in-time picture of storage usage and do not capture peaks. An IPCS formatted dump using the parameter

```
VERBEXIT DSNWDMP 'ALL SM=3'
```

produces the useful summary of DBM1 storage usage shown in Example 4-3.

Example 4-3 DBM1 storage usage summary

===Stack(SKB) total:	55196K	53M
===Fixed subpool total:	384K	0M
GMB total:	151755K	148M
ADMF AGL system total:	3076K	3M
ADMF AGL non system total:	93360K	91M
ADMF AGL 31 total:	72064K	70M
ADMF AGL VL total:	24372K	23M
ADMF local total:	96436K	94M
SQL cache total:	16080K	15M
Variable subpool total:	119736K	116M
Var+Fix+GMB+Stack total:	327071K	319M
===Above the bar		
Fixed subpool total:	2604K	2M
ADMF AGL 64 total:	76072K	74M
ADMF AGL 64 system total:	0K	0M
Variable subpool total:	154616K	150M
VSAS blocks:	2201M	

Note: When you see ADMF in the DB2 literature (and the example above), it does not refer to the Asynchronous Data Mover Facility, the MVS function previously used to move pages to hiper pools. ADMF is merely another name for the Database Services Address Space, more often called the DBM1 address space.

As you can now appreciate, it is important you monitor your system paging rates. Even a small rate of paging to DASD is usually a warning sign of real storage issues. You can use the online RMF Monitor III Option 3.7 Storage Frames (STORF) report to gauge paging rated by address space. This online report monitors the number of REAL+AUX frames used by the DB2 DBM1 address space in real time. Figure 4-8 shows a sample RMF Monitor III STORF report, showing the relative paging rates to the right of the report. Performance problems arise from overcommitment of real storage and auxiliary paging slots. The general recommendation is close to zero (or <1 page per second) paging to DASD because of the LRU scheme used for buffer replacement and the use of long term page fixing the bufferpools.

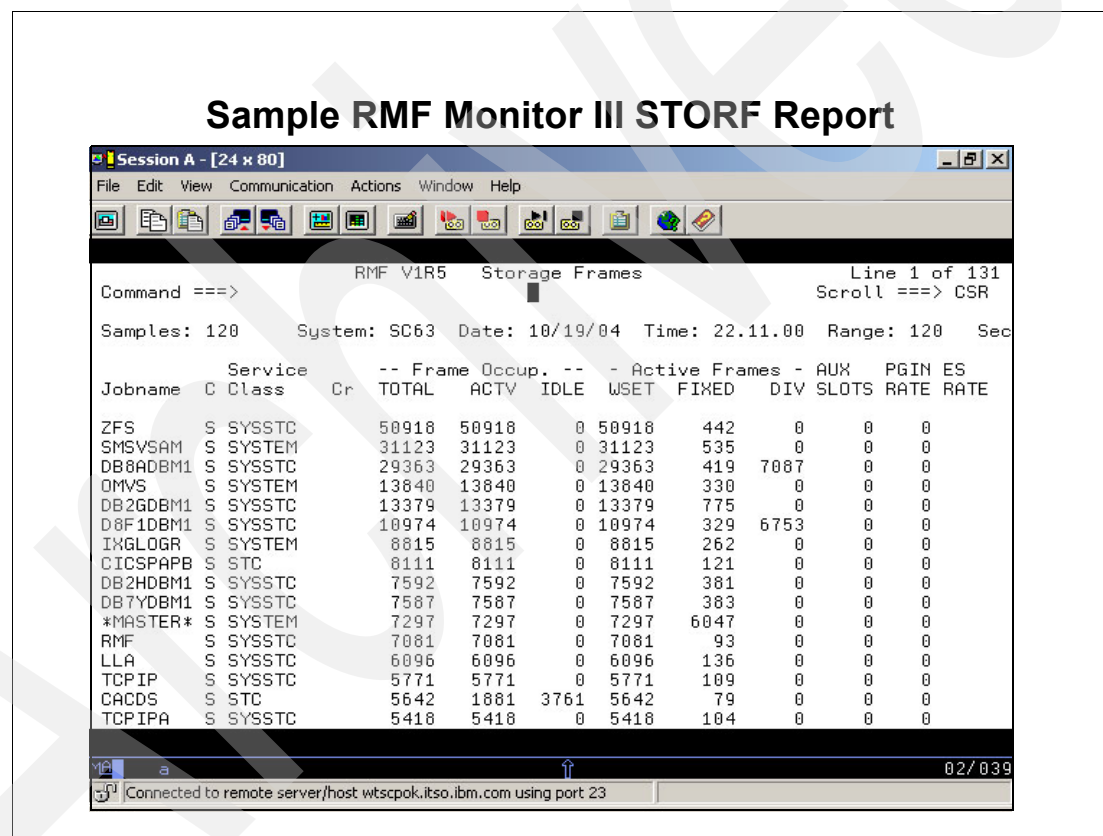


Figure 4-8 Sample RMF Monitor III STORF Report

4.4 Profiles do vary

In this section we show it is important to look at storage utilization across time to make sure you are evaluating a meaningful profile. We do that by showing measurements taken with PMDB2 for a DB2 V7 customer using a 4-way data sharing. Only one subsystem is shown.

PMDB2 is the DB2 performance consulting offering from IBM Global Services, taking DB2 instrumentation (such as Accounting Trace and the DB2 Catalog) and supplementing it with performance data from z/OS sources, such as RMF.

Figure 4-9 shows how the IFCID 225 picture varies by time of day. The chart is based on DB2 Statistics Trace Class 6 (IFCID 225) data. This customer has undertaken most of the sensible tuning actions to reduce virtual storage, such as moving all the buffer pools and the Prepared Statement Cache to data space. What remains is mostly thread-related, with some areas relevant to data sharing and Dataspace Lookaside buffers.

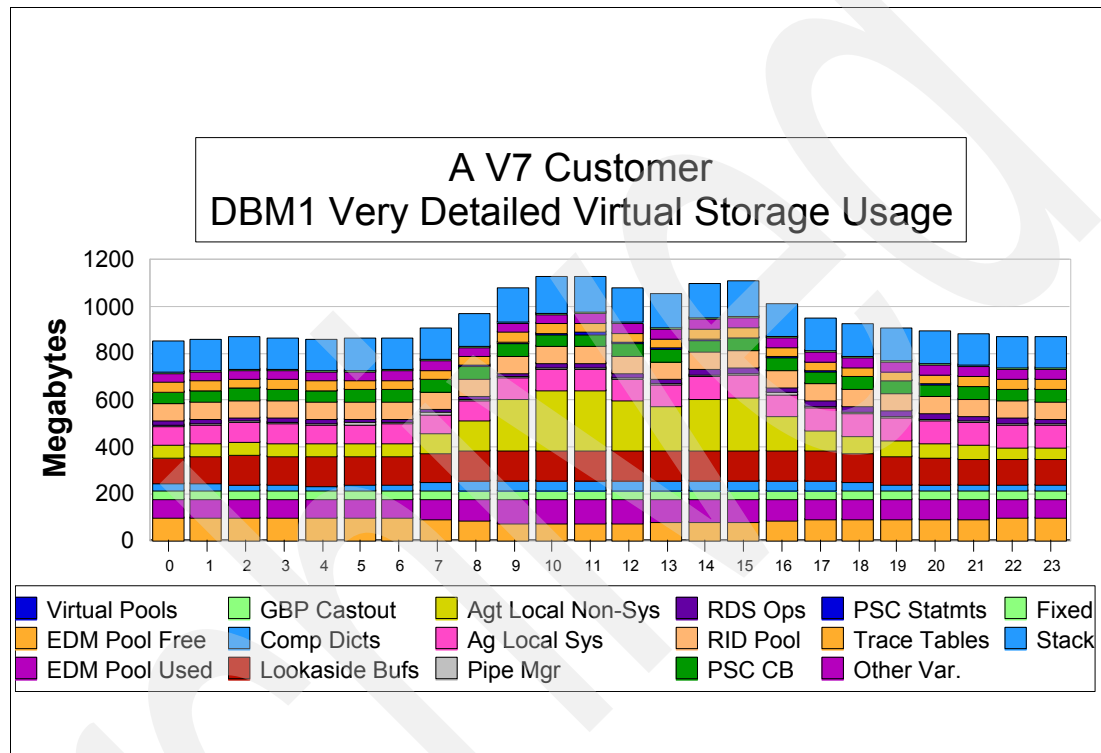


Figure 4-9 DBM1 storage by time

Figure 4-10 shows how the thread-related storage in DBM1 varies by time of day.

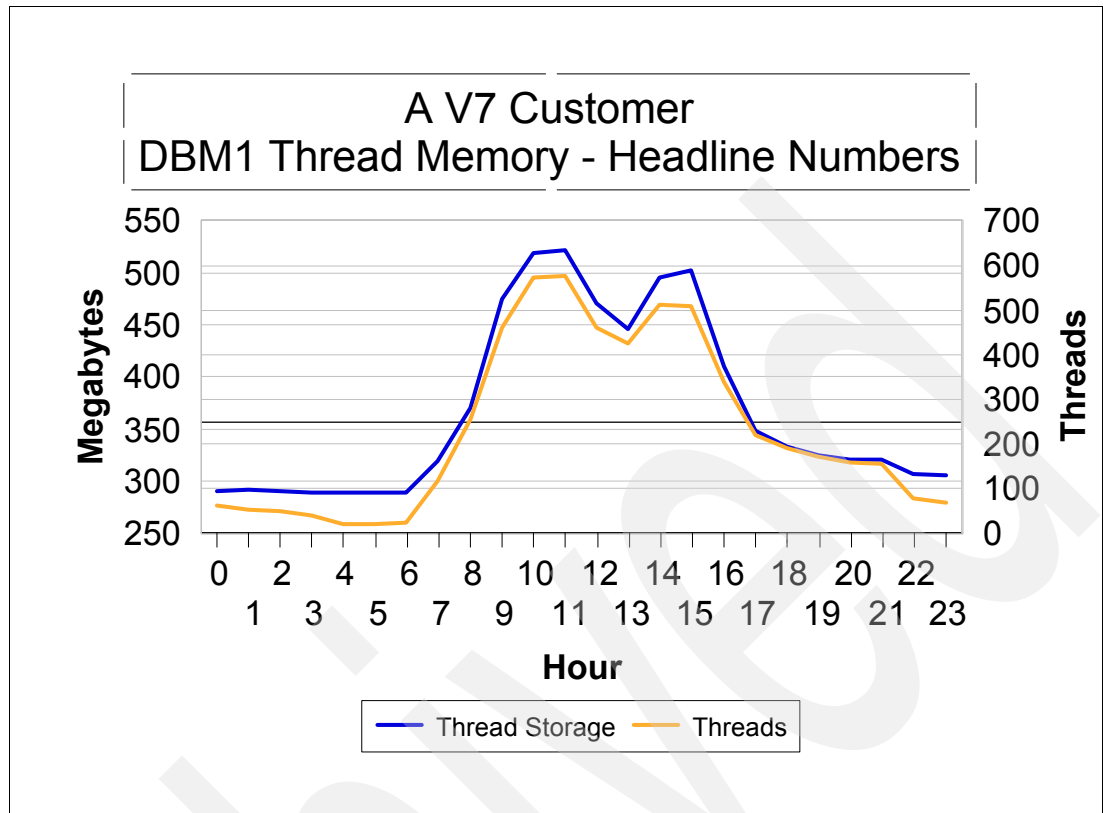


Figure 4-10 Thread-related storage by time

This chart shows that the number of threads varies largely between day and night, with more and lighter threads during the day. Storage per thread stays more or less constant at 1 MB each during normal working hours.

Figure 4-11 shows in detail how the memory per thread varies by time of day. Overnight there are fewer threads but they each have a larger footprint. In general each daytime thread (when the demand for virtual storage is at its highest) has a footprint of about 1 MB.

It is useful to calculate the storage per thread so that thread limits such as CTHREAD can be properly set. This picture can be further refined using IFCID 217 thread-level data.

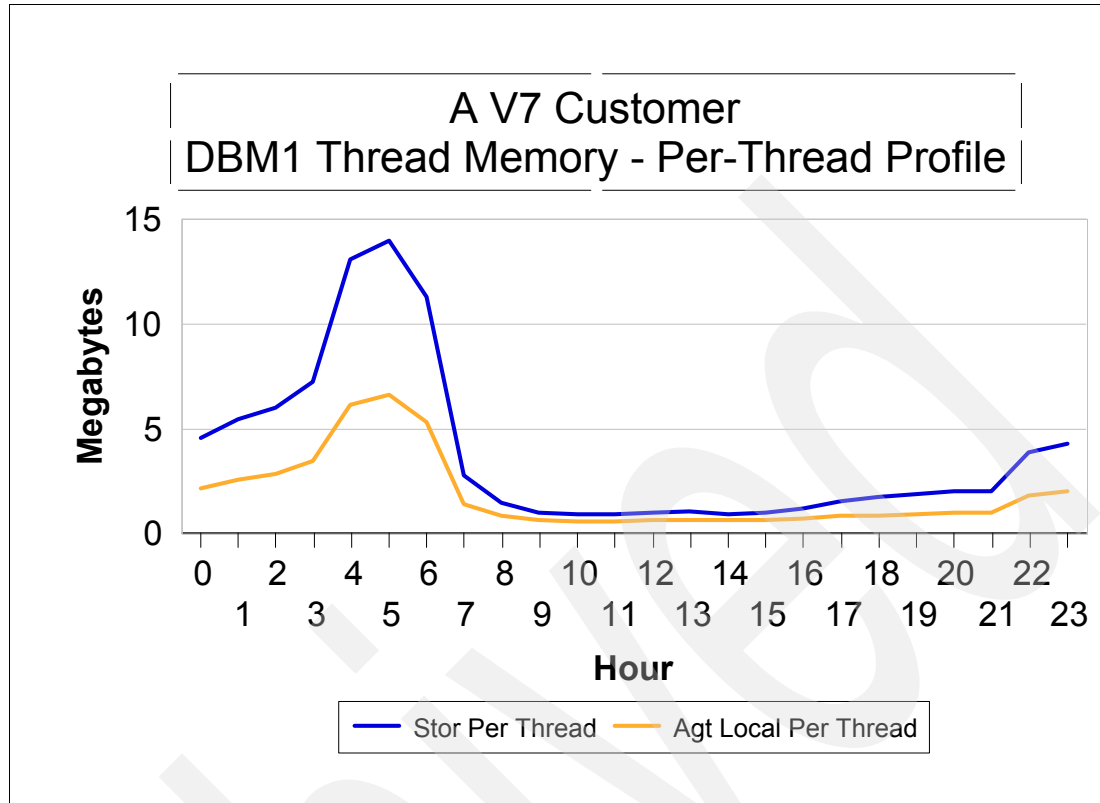


Figure 4-11 Storage per thread by time

4.5 Buffer pool long term page fixing

DB2 V8 introduces a new ALTER BUFFERPOOL parameter called PGFIX. This parameter allows you to fix the buffer pages once in memory and keep them fixed in real storage. This avoids the processing time that DB2 needs to fix and free pages each time there is an I/O.

When virtual storage is used for an I/O operation, the channels require that the storage be “page-fixed” during the time span of the I/O, because the channels must use real storage addresses for I/O and therefore the page cannot be allowed to move between the time when the channel program is built and when the I/O is complete.

The operating system normally “fixes” the storage temporarily during the life of the I/O operation. This extra CPU cost can be as high as 10%.

DB2 V8 utilizes a z/OS service requested by the long-term page fixing option to eliminate this expense. The PGFIX(YES/NO) option may be specified for individual buffer pools, and it can be altered dynamically. PGFIX(NO) is the default.

To use this option, you can use the PGFIX(YES) option on the ALTER BUFFERPOOL command. The ALTER takes effect the next time the buffer pool is allocated. DB2 message DSNB543I indicates the buffer pool is now in a pending state, as a result of changing the PGFIX parameter to YES. DB2 message DSNB406I shows the current PGFIX setting for the buffer pool. Be reminded however, that the page fixing is at the buffer pool level.

The most buffer pools DB2 allows you to page fix are up to 80% of the real storage of the z/OS LPAR. This limitation is in place to protect other workloads on the LPAR from being “squeezed” by DB2.

4.5.1 Performance

We tested the impact of long term page fixing all of our buffer pools, both in a data sharing and non-data sharing environment. We used the same IRWW workload for both tests. We also used an I/O intensive workload, by limiting the storage used by the buffer pools to 400 MB.

Let us first look at non-data sharing. Table 4-9 compares PGFIX(NO) to PGFIX(YES) in a non-data sharing test.

Table 4-9 Long term page fix - non-data sharing

PGFIX	NO	YES	Delta (YES/NO)
DBM1 (msec / commit)	0.455	0.322	-29%
Total DB2 CPU time (msec / commit)	2.436	2.238	-8%
ITR (msec / commit)	587.22	601.42	+6%

Page fixing DB2 buffer pools has a significant positive impact on performance. We can see almost a 30% reduction in CPU consumed by the DBM1 address space per transaction. This is because prefetch requests and deferred write I/O requests stand to benefit the most from page fixing. DB2 no longer has to page fix a large number pages before initiating each prefetch and write I/O request. DB2 also does not have to release the pages after the I/O has completed. DB2 only incurs the cost of page fixing when the buffer pool page is first referenced in an I/O request. Subsequent I/Os do not incur this cost again.

We can also see a 8% reduction in total CPU time used by DB2 and a gain of 6% in transaction throughput.

Now let us have a look at data sharing. Table 4-10 compares PGFIX(NO) to PGFIX(YES) in a data sharing environment with two members. Notice that a long term page fix applies to GBP read and write. It does not apply to castout work area, since cast work area is not a part of buffer pool and a long term page fix is a buffer pool option.

Table 4-10 Long term page fix - data sharing

PGFIX	NO		YES		Delta (YES / NO)
DBM1 (msec / commit)	0.560	0.563	0.456	0.457	-19%
Total DB2 CPU time (msec / commit)	3.318	3.306	3.109	3.357	-2%
Group ITR (commit / sec)	890.70		897.00		+1%

The performance improvements in a data sharing environment are not as pronounced, with only a 19% improvement in DBM1 CPU used, compared with 29% in non-data sharing. The main reason is a higher fixed cost with data sharing.

4.5.2 Conclusion

Remember that our measurements were taken using an I/O intensive workload. You might not see as dramatic an improvement in CPU savings as we did. However our test cases show a CPU time reduction of up to 8% for I/O intensive transaction workload, and as much as 10% for more I/O intensive batch types of workloads.

Although these are still significant CPU savings to be obtained in a data sharing environment, the CPU savings are generally not as great as in a non-data sharing environment. This is primarily due to a higher fixed cost overhead with data sharing.

4.5.3 Recommendations

Long term page fixing DB2 buffer pools is effectively a trade-off between real storage and performance. It is available in V8 CM and beyond.

Long term page fixing of DB2 buffer pools typically results in a CPU savings in the range of 0% to as great as 10%. The performance benefits are inversely proportional to the buffer pool hit ratio. The higher the hit ratio, the lower the potential performance benefit. We therefore recommend you first page fix your smaller buffer pools with a low hit ratio and frequent I/O, (for example sequential workloads). You then should consider page fixing your other buffer pools, buffer pool by buffer pool, to minimize potential impact on other concurrently running workloads. However, you need to be a little more cautious when considering larger buffer pools. Page fixing buffer pools may drive system paging rates higher, thereby impacting overall system performance.

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

4.6 IRLM V2.2

DB2 V8 is shipped with IRLM V2.2, which is now a 64-bit application. With IRLM V2.2, DB2 locks always reside above the 2 GB bar.

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

IRLM V2.2 is also shipped with IMS V9. However, note that at the time of writing this book, IRLM V2.2 does not support earlier versions of DB2 or IMS.

IRLM V2.2 supports a far greater number of locks than V2.1. IRLM V2.1 which is shipped with DB2 V7, is a 31-bit application and is therefore restricted to the 2 GB address space limit. The maximum number of concurrently held locks in IRLM V2.1 is approximately 8 million locks, (2 GB/approximately 260 bytes per lock). IRLM V2.2 which is shipped with DB2 V8, is a 64-bit application and therefore not restricted to the 2 GB address space limit. As the average lock size in DB2 V8 is increased to approximately 540 bytes, only 3.7 million locks can be

maintained in a 2 GB IRLM address space. However V2.2 is not restricted to this 2 GB limit. The new maximum number of locks is theoretically 100 million, around 16 times more than V2.1, assuming that ECSA does not become a bottleneck sooner.

Locks no longer reside in ECSA. Therefore, the PC=NO parameter in the IRLM procedure is no longer honored. PC=YES is always used. This also means that the MAXCSA parameter no longer applies, as it was related to ECSA storage usage. You can still specify both PC=NO and MAXCSA= for compatibility reasons, but they are ignored by IRLM. Note that the IRLM parameters are positional, so you cannot remove them altogether.

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

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

- **MLMT - Max storage for locks:**

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

APAR PQ87611 allows you to dynamically change the amount of storage used for locks above the bar by using the z/OS command:

```
MODIFY irImproc,SET,PVT=nnnn
```

where nnnn can be specified in nnnnM (MB) or nnnnG (GB). A SET,PVT= command causes the MEMLIMIT to be updated; never lower than the default 2 G, and never lower than the amount of storage in use, plus the 10% of reserved space.

- **PGPROT - Page protect:**

Acceptable values are NO and YES (default). The page protect IRLM startup procedure parameter specifies whether or not IRLM loads its modules that reside in common storage into page-protected storage. YES indicates that modules located in common storage are to be loaded into page-protected storage to prevent programs from overlaying the instructions. We recommend YES because it requires no additional overhead after the modules are loaded, and the protection can prevent code-overlay failures. NO indicates that common storage modules are to be loaded into CSA or ECSA without first page-protecting that memory.

Some large DB2 environments, for example, large SAP R/3® environments, should consider a MEMLIMIT of 4 GB for their IRLM if the real storage is available.

4.6.1 Performance

Table 4-11 shows the CPU time consumed by IRLM while running the IRWW workload in a two member data sharing environment. Each value in the IRLM CPU row of the table represents a DB2 member. All tests were run using PC=YES, including the V7 test. Note that IRLM request CPU time is not included in IRLM CPU. Instead it is included in accounting class 2.

Table 4-11 IRLM CPU - V2.1 vs. V2.2

DB2	V7		V8		V8		Delta for P1 % (V8-V7) / V7	Delta for P2 % (V8-V7) / V7
IRLM (PC=YES)	2.1		2.2		2.2			
Lock Protocol	n/a		1		2			
Group ITR (commits/sec)	823.56		804.69		933.41		-2.29%	+13.34%
IRLM CPU msec/commit	0.330	0.371	0.335	0.379	0.014	0.014	+1.85%	-96.01%

We can make some judgements here to assess the impact on CPU the IRLM V2.2 has over the IRLM V2.1. Keep in mind that IRLM V2.2 is now a 64-bit application and the locks are managed above the 2 GB bar. These figures show only a modest 1.8% increase in CPU with this workload. The IRLM is generally not a significant user of CPU compared with other DB2 address spaces such as the DBM1 address space. So, this slight increase is not really considered significant.

Significant savings in the IRLM CPU can be seen in the third test, when Locking Protocol Level 2 was used, compared with the first test under DB2 V8. This CPU savings can directly be attributed to the Locking Protocol Level 2, which significantly reduces XES contention resolution. Refer to Chapter 8, “Data sharing enhancements” on page 319 for a more detailed discussion of the new locking protocol implemented in DB2 V8.

PC=NO vs. PC=YES

In the past, the IRLM PC parameter, an option on the IRLM startup procedure JCL, determined where the IRLM maintains its locks. PC=NO instructs the IRLM to maintain its lock structures in ECSA, which is governed by the MAXCSA parameter. PC=YES instructs the IRLM to maintain its lock structures in the IRLM private region. PC=YES causes cross-memory linkage PC constructs to be used between DB2 and the IRLM, rather than the cheaper branch constructs.

In the past, PC=NO was the general recommendation for performance, and PC=NO is the default in DB2 V7. However, Cross Memory calls have become less expensive and recent enhancements in lock avoidance techniques have greatly reduced the advantage of using PC=NO.

Prior to DB2 V8, PC=YES has been the recommendation for high availability. PC=YES helps you avoid “out of storage” problems related to exhausting ECSA. It also allows you to decrease the ECSA size, and thus increase the private region size. It also allows you to raise lock escalation limits and max locks per user limit in DSNZPARM, reducing the frequency of lock escalations.

For some workloads, going from PC=NO to PC=YES may increase the CPU cost of each IRLM request by as much as 10%. This is because the number of instructions required to process a lock increase by about 5%, and cross memory instructions are generally more expensive to process than normal instructions. However if 2% of the overall CPU time is consumed by the IRLM, the overall impact of having PC=YES is 0.2%, (10% * 2%), which is probably not noticeable. So, even in DB2 V7, the availability benefits of moving from PC=NO to PC=YES outweigh the very slight performance benefits of PC=NO in most cases.

In any event, the PC and MAXCSA parameters are no longer used in the IRLM V2.2. However, you must still specify them in the IRLM JCL for compatibility reasons.

4.6.2 Conclusion

We can conclude that there is some increase in IRLM CPU caused by moving from V2.1 to V2.2, generally compensated by the Locking Protocol Level 2 enhancements in a data sharing environment. In a non-data sharing environment, the increase in CPU usage by the IRLM address space is included in the accounting.

By maintaining its lock structures in private storage above the 2 GB bar, the IRLM is able to manage many more locks than before. This enables DB2 to manage locks at a smaller granularity, resulting in less lock contention.

4.6.3 Recommendations

Since IRLM V2.2 can manage many more locks than IRLM V2.1, you may consider increasing NUMLKTS (maximum number of locks per table space before lock escalation occurs) or NUMLKUS (number of lock per user before a resource unavailable occurs).

Because the lock structures no longer reside in ECSA, considerable ECSA storage is now freed up in V8 and can be used by other subsystems (such as WebSphere) and other applications. This is good news for MVS Systems Programmers.

Since the space required to maintain each individual lock has increased, some large DB2 installations, such as large SAP R/3 environments, should consider a MEMLIMIT of 4 GB for their IRLM. However, the same rules apply for the storage areas that move above the 2 GB bar in DBM1. Make sure there is enough real storage to back this up.

4.7 Unicode

V8 vastly improves DB2's support for Unicode and lifts many of the limitations that existed in V7. For example, when DB2 has been migrated to V8 new-function mode, the DB2 catalog is converted to Unicode. In addition all the SQL statements must be parsed as Unicode, even in CM, while DB2 V8 NFM allows you to combine different encoding schemes, CCSIDs, in the same SQL statement.

As a result, DB2 now must perform much more code page conversion than ever before, which potentially impacts performance. In this section we discuss the major improvements that have occurred in converting to and from Unicode. These improvements come from three major sources; improvements in the zSeries processor hardware, improvements in the z/OS Conversion Services, and optimization inside DB2 V8.

For details, see the new manual *DB2 UDB Internationalization Guide* available from:

<http://www.ibm.com/software/data/db2/zos/v8books.html>

Another source of information is the article titled "Unicode Performance in DB2 for z/OS", by Jeffrey Berger and Kyoko Amano, published in The IDUG Solutions Journal Volume 11 Number 2.

DB2 V8 supports three types of encoding schemes: EBCDIC, which is used by mainframe environments, ASCII, which is used by Unix and Windows environments, and more recently, Unicode.

We identify the character set for a language by a numeric CCSID. A CCSID encoding scheme consists of a single byte character set (SBCS), and optionally a double byte character set (DBCS) along with a mixed character set. Over time each geography developed their own EBCDIC or ASCII encoding schemes. The variety of CCSIDs made it very difficult for

multinational companies to combine data from different sources. Unicode was developed to solve the problem. By storing data in a single Unicode CCSID, text data from different languages in different countries can be easily stored and managed.

DB2 V5 introduced the ability to create ASCII databases, table spaces and tables via the CCSID ASCII clause. This gave you the choice to store your data in either EBCDIC or ASCII, and DB2 would convert data between these encoding schemes where appropriate. DB2 V7 built upon that support, introducing support for Unicode objects. However, where EBCDIC and ASCII conversions are done within DB2, DB2 V7 always relied on OS/390 Conversion Services to handle Unicode conversion. The performance of Unicode conversion in DB2 V7, with older levels of the operating system and zSeries hardware, was less than optimal. DB2 V8 enhances support for Unicode, and at the same time takes advantage of the faster conversion provided by the current versions of z/OS services and zSeries processors.

When storing data into and retrieving data, DB2 converts the data from one CCSID to another CCSID where necessary. Obviously, it is preferable that no conversion be carried out, because conversion impacts performance. However, IBM has done considerable work to improve conversion performance on zSeries. In this section, we discuss the recent enhancements IBM have made to conversion and how they impact performance.

When does CCSID conversion occur?

CCSID conversion occurs whenever there is a mismatch between the CCSID of a source and target string, such as between a host variable and its associated column. This conversion support in DB2 has existed since DB2 began to support client/server connections in V2.3 between different EBCDIC CCSIDs. DB2 V5 began to support ASCII tables and ASCII host variables, so CCSID conversion was expanded to include conversion between EBCDIC and ASCII.

To perform these conversions (not involving Unicode), DB2 uses a translate table which is stored in SYSIBM.SYSSTRINGS. We refer to this type of CCSID conversion as SYSSTRINGS conversions.

The specific EBCDIC and ASCII CCSIDs used to store data by a DB2 system must be specified in the DSNHDECP data only module, and they should not be changed. Starting in V8, they cannot be changed. Within a single DB2 system, all EBCDIC tables have the same EBCDIC CCSIDs and all ASCII tables have the same ASCII CCSIDs. In addition, numeric columns and date/time columns are stored as binary fields, but they may involve CCSID conversion when loading or unloading such fields using the EXTERNAL attribute. However, the utilities always apply the EXTERNAL attribute to date/time fields.

DB2 V7 introduced the ability for user tables to be defined as Unicode by specifying “CCSID UNICODE” on the database, table space or table definition. Within a Unicode table, CHAR and VARCHAR columns are stored in UTF-8 format. GRAPHIC and VARGRAHIC columns are stored in a Unicode table as UTF-16. Any CCSID conversion involving Unicode is called Unicode conversion and is necessary whenever there is a mismatch between the CCSID of a source and target string. DB2 V7 uses the z/OS Conversion Services exclusively for all Unicode conversions, which can affect the CPU time substantially.

In V7 the DB2 Catalog remains in EBCDIC and all SQL statement parsing is still done in EBCDIC. Therefore, for applications such as the Universal Java Driver that provide an SQL statement in Unicode, DB2 must convert the statement to EBCDIC. Similarly, any incoming metadata, such as package names and authorization ids, may also require conversion by DB2.

One of the limitations of V7 is that you cannot join two tables that use different CCSIDs. Another limitation is performance. V7 completely relies on the z/OS Unicode conversion

service for Unicode conversions, and prior to z/OS 1.4 the conversion service was very slow, especially on pre-z900 processors.

DB2 V8 brings many more enhancements in DB2's support for Unicode as well as removing many of the limitations that were in V7. When you first start V8, you are running in CM in which the DB2 catalog is still in EBCDIC and you are prevented from exploiting new function. However, all SQL parsing in V8 is done in Unicode, (that is, UTF-8), even in compatibility mode. SQL statements that are input in EBCDIC, ASCII or UTF-16 must be converted to UTF-8. In compatibility mode, all the DB2 object names the SQL statements reference must again be converted to EBCDIC in order to compare them to DB2 catalog data. In addition, all Utility control statements must be converted to Unicode before they are parsed.

When you migrate the catalog to new-function mode (NFM), the DB2 catalog is converted to Unicode, and fallback to V7 is no longer possible. New DB2 V8 subsystems always run in NFM. SQL statement conversion is unaffected by the switch to NFM, but all the DB2 object names the SQL statements reference no longer need to be converted since the DB2 object names are already in the CCSID of the DB2 catalog. DB2 V8 also introduces the capability to join two tables of different CCSIDs. Needless to say, Unicode conversion is necessary to do this and the extra CPU time is likely to be substantial.

Now, let us turn to understanding how or when CCSID conversion impacts performance. Recall that 'class 1 CPU time' and 'class 2 CPU time' are CPU metrics reported in the DB2 accounting statistics. Class 1 CPU time represents the CPU time consumed between the creation of a DB2 thread and the termination of the thread. Class 2 CPU time is a subset of the class 1 CPU time and it represents the CPU time from when the application passed the SQL statements to the local DB2 system until return. Sometimes, class 2 time is referred to simply as 'in DB2 time'. In general, conversions performed by remote clients affect neither class 1 nor class 2 CPU time, but conversions performed by local Java clients only affect class 1 CPU time. The DBM1 address space handles all CCSID conversions on behalf of old mainframe legacy applications, including DB2 utilities. The DIST address space handles all metadata conversions for remote applications, if any conversion is needed by DB2.

Let us have a look in more detail. Figure 4-12 shows the flow of data for a typical legacy application accessing a Unicode table, where the application is bound using the default application encoding scheme, which is EBCDIC.

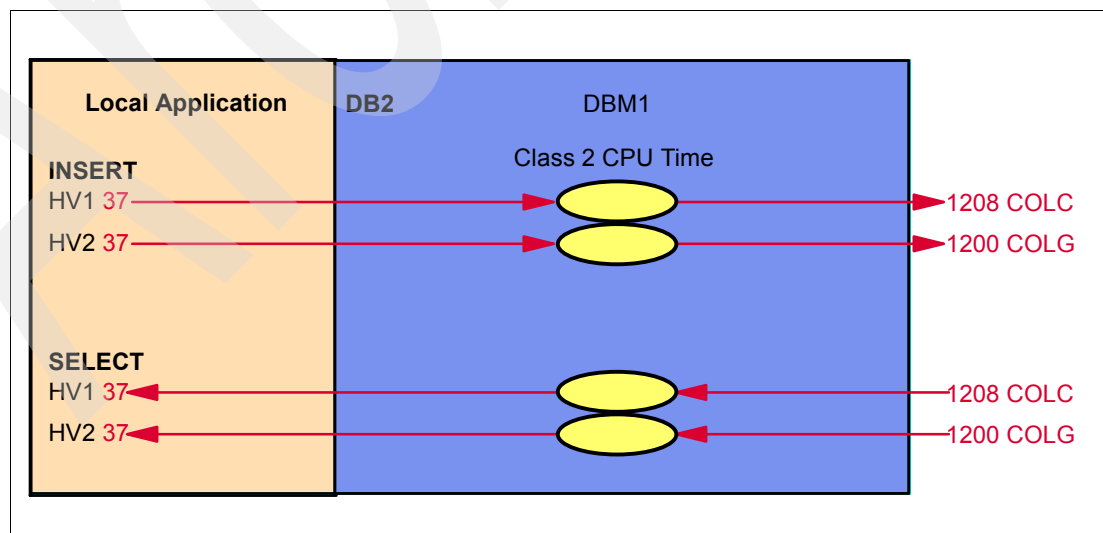


Figure 4-12 CCSID conversion - Local application

Since the database is defined as Unicode, any data being sent or received by the application must be converted by the DBM1 address space, such as in this example in which the EBCDIC code page is 37. Had the database remained as EBCDIC 37, no CCSID conversion would be necessary. The table contains one CHAR column called COLC (CCSID 1208) and one GRAPHIC column called COLG (CCSID 1200). When the application inserts host variables HV1 and HV2 into the table, the strings are converted by the DBM1 address space to Unicode. The CPU time for these conversions is added to the class 2 CPU time.

Figure 4-13 shows a DB2 Connect example using the same Unicode table.

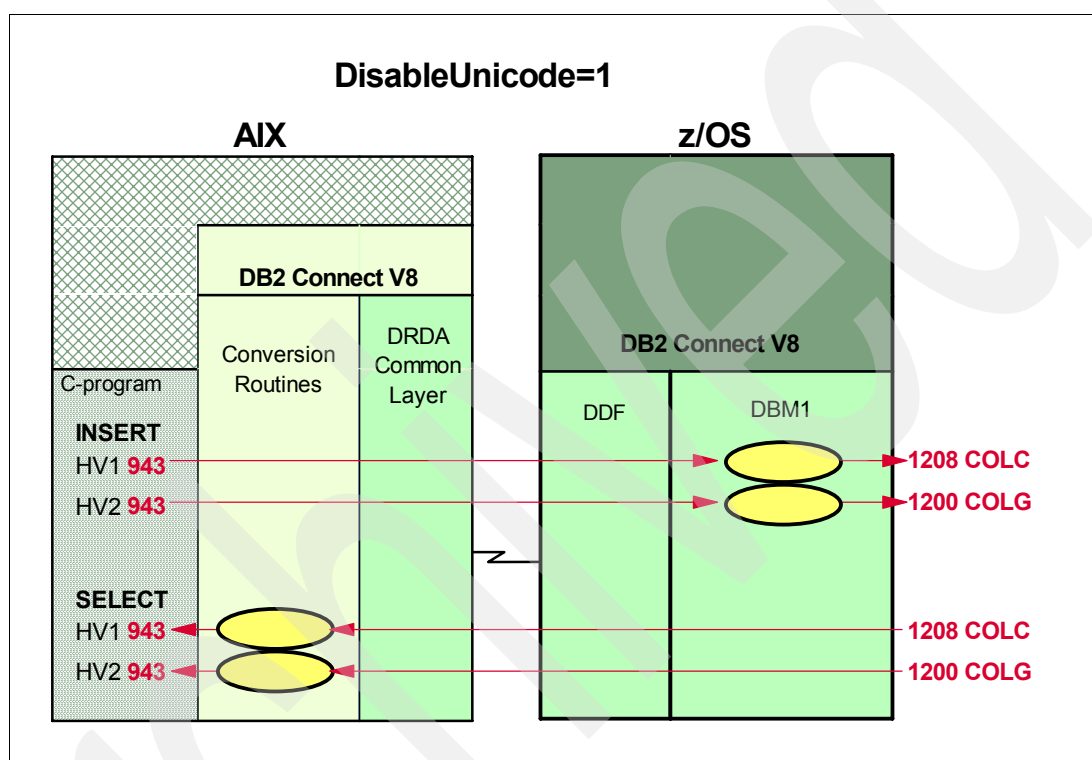


Figure 4-13 CCSID conversion - Remote application - 1

The application is running remotely on an AIX client and uses the Japanese ASCII CCSID 943. The application uses DB2 Connect V8 to communicate to the z/OS server.

The DRDA protocol and DB2 maintain a convention strategy in which the "receiver makes right". Hence, DRDA lets DB2 for z/OS do the conversions when sending data to the server, and DB2 for z/OS lets the client do the conversions for data being retrieved by the client.

Note however that the conversion is actually performed by the DBM1 address space, not the DIST address space. The DBM1 address space actually does the inbound CCSID conversion for character strings, while the DIST address space performs the conversion for numeric data.

IBM first introduced a JDBC Driver in V5 and V6, which is now known as the Legacy JDBC Driver. On workstations, the Legacy JDBC Driver interfaced with DB2 Connect. When sending data to a DB2 for z/OS V6 server, the Legacy JDBC Driver converted the data from UTF-16 to EBCDIC, because V6 was unable to handle incoming data in Unicode. (All IBM implementations of Java use CCSID 1200 corresponding to UTF-16.)

However, DB2 V7 supports Unicode databases. If DB2 Connect were to convert UTF-16 data to EBCDIC and then the server converted it back to Unicode, there would be a potential for data loss since not all Unicode characters exist in the intermediate EBCDIC character set. To

overcome this, when DB2 Connect is interfacing with DB2 for z/OS V7, DB2 Connect sends the data in Unicode (by default). If you still require the data to be sent in ASCII instead of Unicode, then you may set the `DISABLEUNICODE=1` parameter. `DISABLEUNICODE` is specified in the `db2cli.ini` file. Older z/OS systems were slow to convert Unicode so it was desirable to specify `DISABLEUNICODE=1`. Otherwise, installations migrating from DB2 for z/OS V6 to V7 would experience an increase in class 2 CPU time. Only for Java Applications was it preferable not to specify `DISABLEUNICODE=1`.

This performance problem goes away with z/OS 1.4, because now Unicode conversion is faster than ASCII conversion. Furthermore, if the database is actually stored in Unicode, setting `DISABLEUNICODE` adds Unicode conversions to the class 2 CPU time.

In the above example, the DB2 Connect properties file indicates `DISABLEUNICODE=1`, which prevents DB2 Connect from sending data as Unicode. The data sent on `INSERTS` is in ASCII. The DBM1 address space converts the ASCII data to Unicode.

For the `SELECT` statement, DB2 Connect on the client converts the data from Unicode back to ASCII. Only those conversions done by the `DIST` address space are counted as class 2 CPU time.

Class 2 CPU time is unaffected by the conversions done by the client, but if the client is running locally on z/OS, the CPU time for conversions done by the client may be part of class 1 CPU time. Conversions done by a non-z/OS client do not use zSeries mainframe resources.

Figure 4-14 shows the effect of not disabling Unicode in DB2 Connect, which is the default.

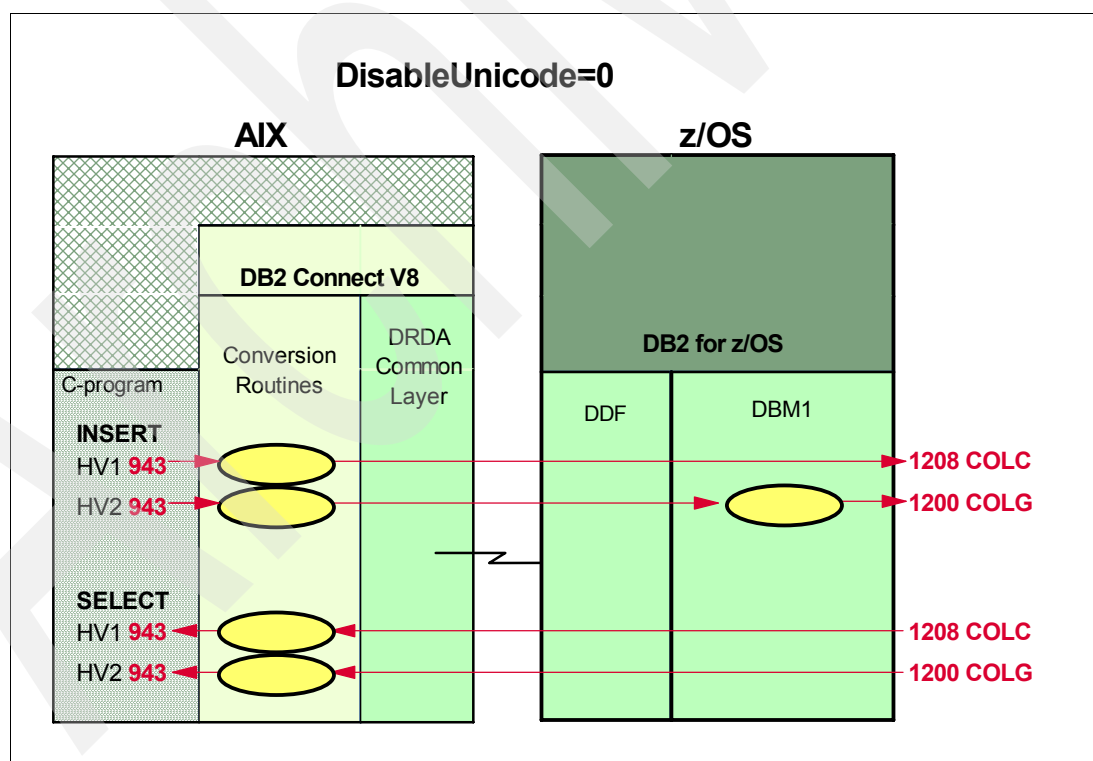


Figure 4-14 CCSID conversion - Remote application - 2

In this case, DB2 Connect sends all data as 1208, no matter what the type of column is. The DB2 for z/OS conversions for `CHAR` columns disappear. However, there is a double conversion for `GRAPHIC` columns, but only the second conversion is done by DB2 for z/OS.

Still, this second conversion is a conversion from 1208 to 1200 which is somewhat less expensive than an ASCII conversion (provided that z/OS 1.4 is installed).

In summary, for CHAR columns, not setting DISABLEUNICODE shifts the Unicode conversion during the Insert from the z/OS server to the workstation client, helping to reduce class 2 CPU time. For GRAPHIC columns, DISABLEUNICODE has a more subtle effect, because in one case the class 2 CPU time contains a conversion from UTF-8 to UTF-16, and in the other case the class 2 CPU time contains a conversion from ASCII to UTF-16 (a Java application would send all data as UTF-8.)

The same DISABLEUNICODE parameter affects both host variables and SQL statements. Since DB2 V8 does SQL parsing in Unicode, it becomes more important to avoid setting DISABLEUNICODE. This is especially true if any of the code points used in the SQL statement is greater than 127. If all code points are less than 128, CCSID 1208 is equivalent to ASCII, and no conversion of the SQL statement actually occurs. See “DB2 V8 major and minor conversion” on page 185 for more details.

Now let us have a look at Unicode conversion for Java clients.

Along with DB2 V8, IBM introduced the Universal Driver for JDBC and SQLJ to replace the old Legacy JDBC Driver. The Universal Driver does not use DB2 Connect. The Universal Driver supports both a Type 2 and Type 4 Driver. While the Type 2 Driver is optimally used by local Java applications running on z/OS, the Type 4 Driver can be used by workstation clients as well as z/OS clients. The Type 4 Driver uses DRDA protocols.

Since the Universal Driver does not use DB2 Connect, the DISABLEUNICODE parameter is not applicable. Instead the Universal Type 4 Driver supports a new “UseServerEncoding” data source property. Unlike the Legacy Driver, the Universal Driver always sends the SQL statement in UTF-8 no matter how the property is set. The Universal Java Driver is shipped with DB2 V8 and V7 with PQ8841. Although it can connect to DB2 V7, it is optimized to work with DB2 V8. Since the Universal Driver always sends the SQL statement in UTF-8, a V7 server has to convert the SQL statement to EBCDIC, and the result is higher class 2 CPU time.

Because there is no disadvantage to using UseServerEncoding=1, our discussion assumes that UseServerEncoding=1 is always used.

Figure 4-15 shows the Universal Java Type 2 Driver locally connecting to DB2 V8, first using an EBCDIC database and then using a Unicode database.

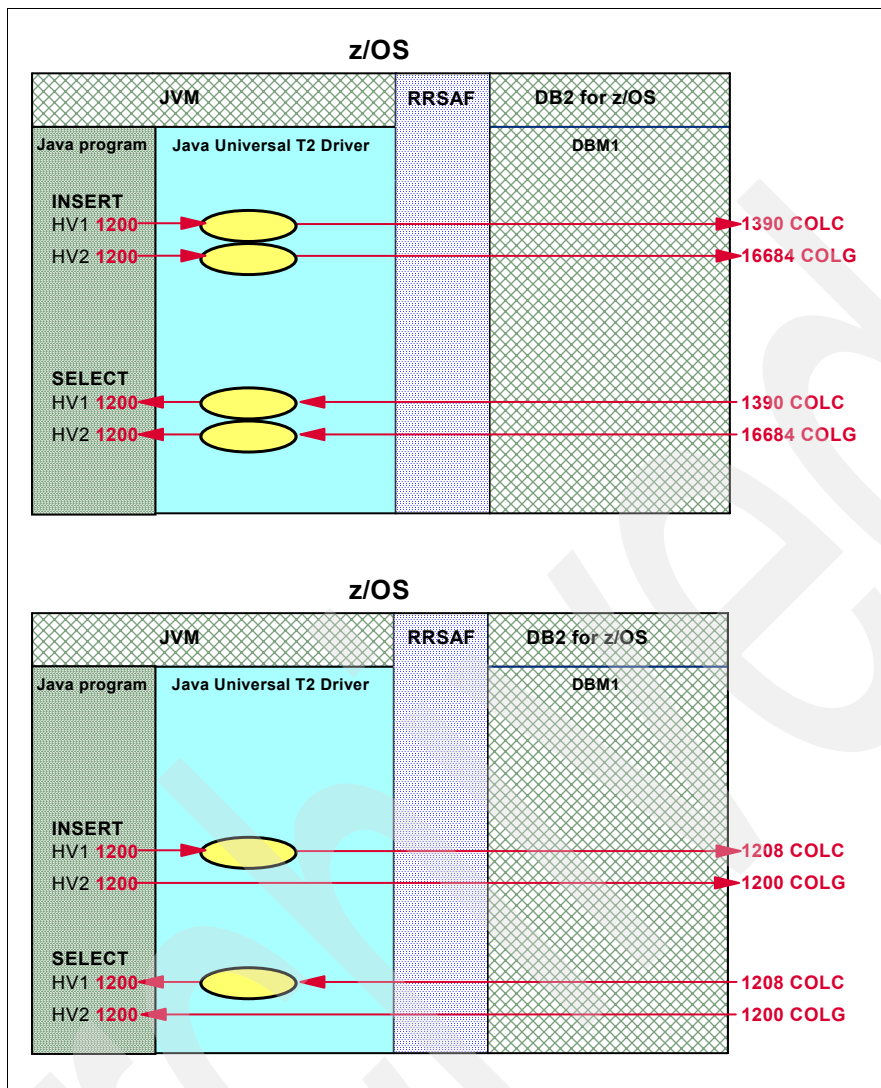


Figure 4-15 CCSID conversion - Local z/OS Universal Java Type 2 Driver

Here we see that DB2 for z/OS never has to do any CCSID conversion because the Universal Java Driver always converts the data to the correct CCSID. Hence, Unicode conversion does not affect class 2 CPU time, but since the JVM™ (Java Virtual Machine) is running on z/OS, conversions performed under the JVM affect the class 1 CPU time. Note also that when a string is being inserted into or fetched from a Graphic Unicode column, there are no conversions under the JVM.

As an aside, the Unicode conversions performed by the JVM in z/OS do not use the z/OS Conversion Services because it is difficult for the JVM to use z/OS system services. Prior to JDK™ Release 1.4.1 the conversions within Java did not use any of the special zSeries translation instructions, and the conversions were inefficient for long strings. Starting with JDK 1.4.1, some of these conversions are beginning to use the zSeries translate instructions, namely the TROO, TROT and TRTO instructions, which are useful for converting among byte codes and UTF-16 and SBCS EBCDIC characters. So, we can see improved performance for the Universal Type 2 Driver, with particularly significant improvements with large SBCS strings on the z990 processor. Such an improvement does not occur when a string contains DBCS characters.

Figure 4-16 shows the Universal Java Type 4 Driver remotely connecting to DB2 V8 from AIX. It also assumes that Deferred Prepare is used. By default, the Type 4 Driver defers the Prepare in order to minimize network overhead.

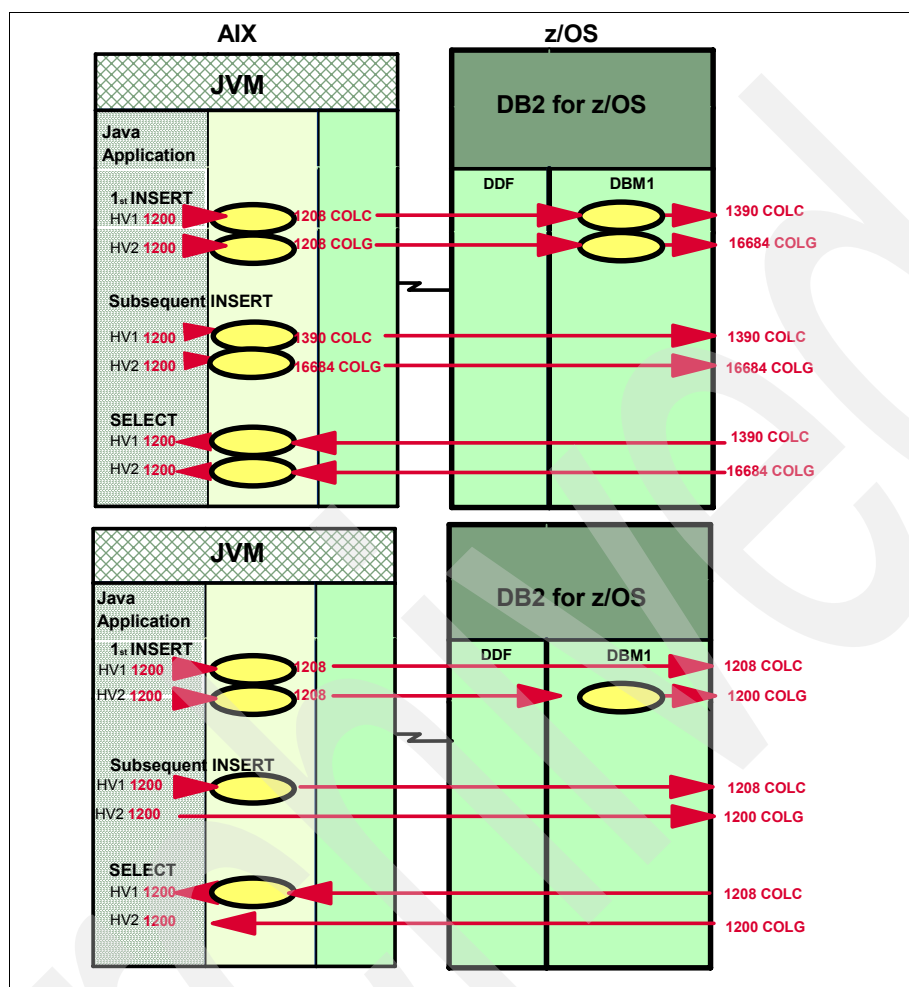


Figure 4-16 CCSID conversion - Remote Prepare using Java Universal Type 4 Driver

The Type 4 Driver offers a new interface called ParameterMetaData to retrieve the table characteristics, but of course it requires network communication. After ParameterMetaData is invoked, the Universal Driver uses CCSIDs in the same way the Type 2 Driver does. If ParameterMetaData is not used, and Deferred Prepare is used, the first Insert sends the data in UTF-8; if the table is stored in EBCDIC, DB2 for z/OS needs to convert the data to EBCDIC. Subsequent inserts do not require any conversions by DB2 for z/OS. Again, as with the Type 2 Driver, subsequent INSERTs and SELECTs do not require any conversions for Graphic columns.

Note that for static SQL processing where the SQL statements are prebound by SQLJ, Deferred Prepares are not applicable because the CCSIDs are already known by SQLJ at execution time.

SQL statements and DB2 object names such as Authorization Ids and package names may also require CCSID conversion. Figure 4-17 shows an AIX application using DB2 Connect and DRDA to remotely prepare an SQL statement remotely, connecting to DB2 V7 and DB2 V8.

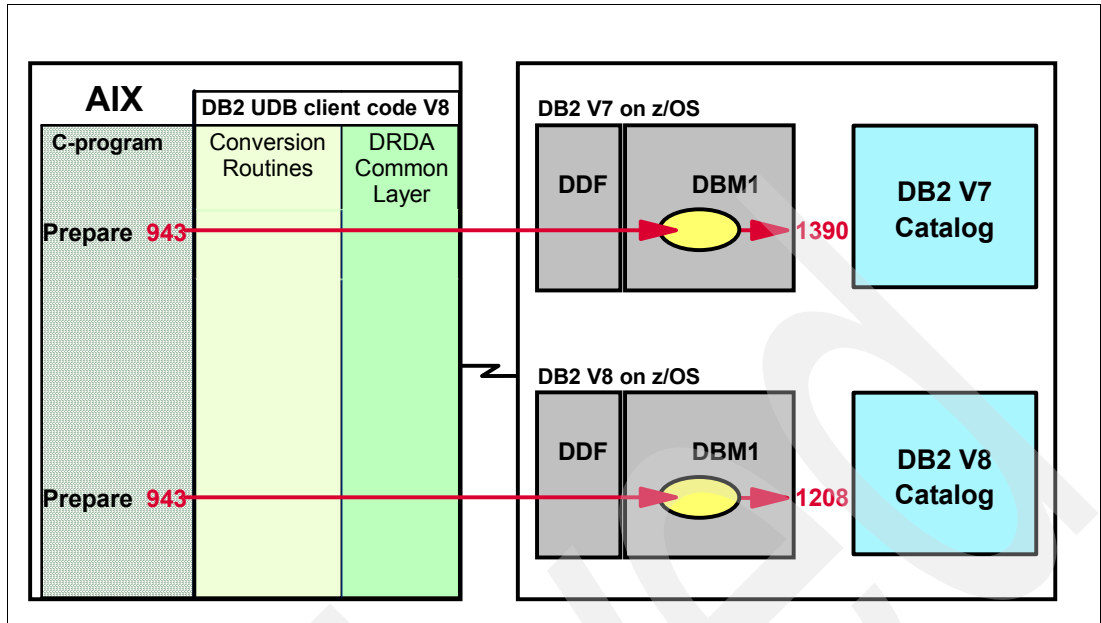


Figure 4-17 CCSID conversion - Remote Prepare of SQL statement

The application on AIX typically inputs the SQL statement in ASCII, whether the target DB2 server is V7 or V8. So, DB2 V7 converts the statement from ASCII to EBCDIC, and DB2 V8 converts the SQL statement from ASCII to UTF-8.

Finally, Figure 4-18 shows a local prepare of an SQL statement using the Universal Java Type 2 Driver to connect to DB2 V7 and DB2 V8.

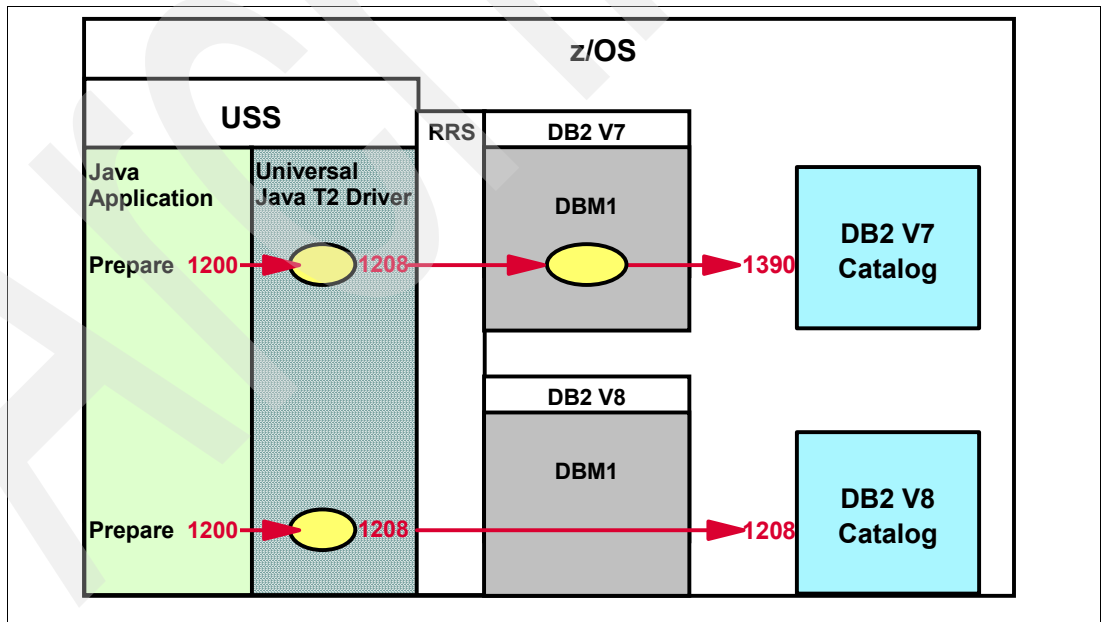


Figure 4-18 CCSID conversion - Local Prepare using Universal Java Type 2 Driver

The Legacy JDBC Driver converts the SQL statement from UTF-16 to UTF-8, which requires no conversion on DB2 V8.

Choosing between UTF-8 and UTF-16

When choosing to store data in a Unicode database, you have a choice of using either UTF-8 (CHAR) or UTF-16 (GRAPHIC). This choice includes both performance and non-performance considerations. For example, since the collating sequences of UTF-8 and UTF-16 are different, you can choose column types based on the desired collating sequence rather than performance.

However, the major performance consideration when comparing UTF-8 to UTF-16 is the amount of space used. The reason space is important is not just the cost of the space itself, but also the impact that the space has on I/O bound utilities and queries, including the time to recover, which affects data availability. Buffer pool hit ratios are also affected by space.

A UTF-8 character is variable length and can be 1 to 4 bytes in length, of which the first 128 code points match 7-bit ASCII. A UTF-8 character is used by DB2 to store CHAR or VARCHAR columns. This is opposed to a character in UTF-16, which is 2 or 4 bytes. DB2 uses UTF-16 to store GRAPHIC or VARGRAPHIC columns. The first 128 code points match UTF-8 and ASCII, while the rest have a different collating sequence.

Consider the type of data you will be storing in DB2 and choose UTF-8 (CHAR) or UTF-16 (GRAPHIC), depending on the characteristics of the data. UTF-8 is efficient for text that is predominantly English alphanumeric while UTF-16 is more efficient for Asian characters.

To reduce the space required by Unicode tables, you can use DB2 compression. With no DB2 compression and for alphanumeric characters, UTF-8 has the same space characteristics as EBCDIC, however UTF-16 doubles the storage requirements. Compressed alphanumeric UTF-16 strings use more storage than compressed alphanumeric UTF-8 strings, but the ratio is generally much less than 2 to 1. Tests have shown this ratio can be less than 1.5 to 1. Whether compression is used or not, the overall effect of Unicode on space usage depends on the amount of text data versus non-text data.

CPU time can also be another consideration. Compression itself costs a lot of CPU time, although its I/O benefits can outweigh the CPU costs. In some applications Unicode conversion can be avoided, but if conversion must be done, it is best done by a remote client in order to distribute the CPU overhead. If conversion must be done on the zSeries machine, then you should consider what the dominant application CCSID will be. Generally, the choice which tends to minimize space usage also tends to minimize CPU usage.

Choosing between fixed and variable length strings may also be another consideration. Remember, the length of a UTF-8 string is unpredictable, unless the application limits the characters to the common set and fixed length strings.

Consider, for example, a column that was defined as CHAR(1) in EBCDIC. If you try to insert a multibyte UTF-8 character into CHAR(1), you get a -302 SQL error, because the data does not fit into one byte. If you change the column definition to GRAPHIC(1), you need two bytes. If you change the column definition to VARCHAR, you need 3 bytes in order to account for the two-byte length field. Knowing that a field is restricted to alphanumeric characteristics allows you to continue using CHAR(1).

It is especially good to avoid using variable length fields in indexes, especially nonpadded indexes, since key comparisons of variable length fields are more expensive. On the other hand, you should try to avoid padding costs with fixed length fields if Unicode conversion is being done.

Remember, there is nothing stopping you from storing both UTF-8 and UTF-16 encoded data in the same table.

4.7.1 Performance

The improvements in Unicode conversion come from three sources. Each of these sources affects different facets of conversion performance, such as the type of characters, and the length of the character strings. The three sources are:

- ▶ **DB2 V8 major and minor conversion:** Improves the performance of all types of conversions, but the improvements are most dramatic with English alphanumeric characters because DB2 does not need to invoke the Unicode conversion service.
- ▶ **z/OS Conversion Services:** z/OS Release 1.4 implemented an enhancement to z/OS Conversion Services with PTF UA05789: This enhancement, referred to as HC3, helps the performance of both DB2 V7 and V8.
- ▶ **zSeries processor hardware:** The z900 model introduced some new Unicode hardware instructions that are simulated by the z/OS Conversion Services on older processors. In addition, the speed of a z990 engine is approximately 50% faster than the z900 Turbo, but when it comes to the Unicode conversion of large strings, the z990 hardware is twice as fast (in this book, we refer to the z900 Turbo model).

Figure 4-19 summarizes the step by step improvements in conversion performance each of these sources have made. For these measurements, a single table scan of 1 million rows was used, using a single cursor with no predicates. The table was defined using 20 VARCHAR columns per row with 10 characters per column.

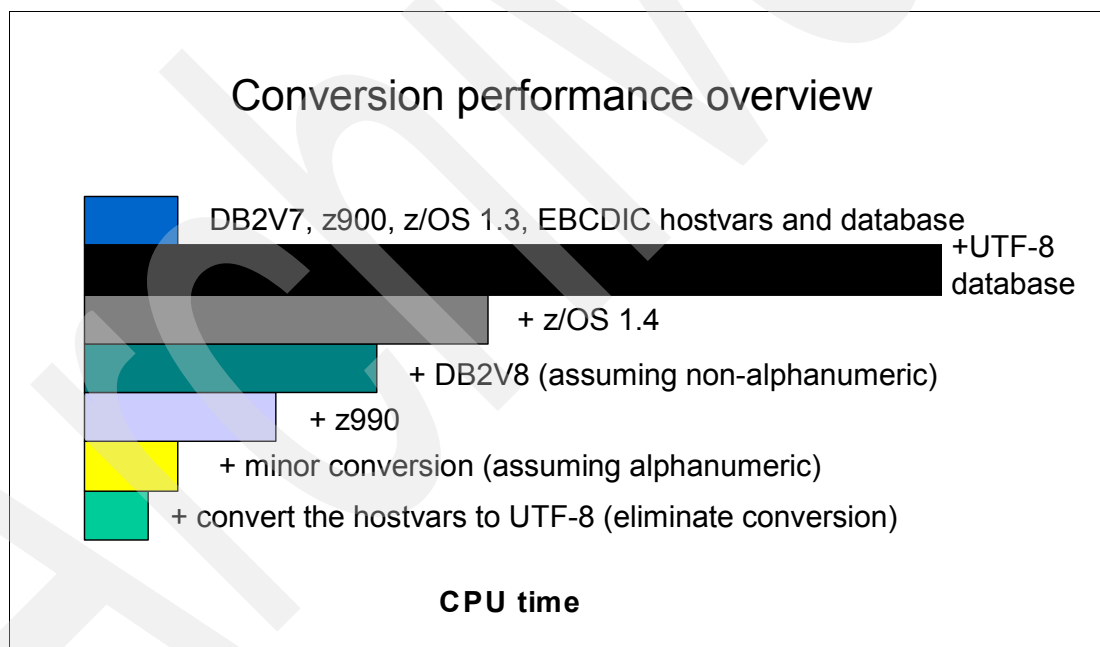


Figure 4-19 Conversion performance overview

Note that the table is not drawn exactly to scale, but it does show the progressive gains in performance that have been achieved.

The first bar represents DB2 V7 running on z/OS 1.3 with EBCDIC host variables and database. CCSID conversion is done here by DB2 using SYSIBM.SYSSTRINGS. Next, we see a huge jump in CPU time as we ask DB2 V7 to use Unicode. DB2 V7 now must invoke z/OS Conversion Services for any conversion to and from Unicode. By simply moving to z/OS 1.4, we see quite a improvement in CPU. This is primarily due to advancements in the z/OS Conversion Services that dramatically improved the performance of (but not exclusively) conversions between UTF-8 and EBCDIC (or ASCII). The next step in performance is

achieved by simply moving to DB2 V8. DB2 V8 is able to exploit the 64-bit interface into the z/OS Conversion Services, which is much more efficient than the 31-bit interface used by DB2 V7. Next, we see a greater improvement by moving to the z990 hardware, which brings a number of specific hardware instructions to support Unicode conversion. Next, we introduce optimization in DB2 V8 called “minor conversion”. Finally, we introduce further optimization in DB2 V8 which in certain cases allows the system to avoid conversion.

Now, we explore each of these enhancements in more detail.

DB2 V8 major and minor conversion

Conversion of SQL statements and DB2 object names generally involves English alphanumeric characters. For many years the DB2 Catalog has been EBCDIC. To improve the performance of DB2, V8 developed a faster way to convert such characters without calling the z/OS Conversion Services. This faster way is called *minor conversion*. Therefore, the term *major conversion* refers to those conversions performed by the z/OS Conversion Services.

DB2 maintains a translate table associated with the EBCDIC SBCS/mixed CCSIDs that are specified during DB2 installation. The same translate table is also used for access to the DB2 catalog and user data. Minor conversion is supported for all European EBCDIC character sets, and some Asian EBCDIC character sets. Minor conversion does not apply to UTF-16 (GRAPHIC) or ASCII; it may only apply to conversions between EBCDIC and UTF-8.

How does minor conversion work?

The zSeries instruction set has always included a TR instruction that translates one byte for one byte, based on a 256-byte translation table. A TRT instruction has also existed which could be used to test a string for certain one byte characters. As long as a single byte English alphanumeric character can be represented in UTF-8 by a single byte, a TR instruction can be used to convert the byte to UTF-8. DB2 V8 has built-in translate tables for most common single byte CCSIDs and if the TRT instruction is “successful”, DB2 can translate the string using a TR instruction. The presence of shift-in/shift-out characters in a mixed string always causes the TRT instruction to fail. If the TRT fails, then DB2 must invoke the z/OS Conversion Services.

The performance of minor conversion is much better than the performance of major conversion because minor conversion avoids calling the z/OS Conversion Services. Hence, minor conversion is one of the significant advantages of DB2 V8 compared to DB2 V7, and represents an advantage of UTF-8 over UTF-16. Even when DB2 V8 needs to use major conversion, V8 is still much faster than V7, which we will see later.

Note that DB2 V8 still uses the catalog table SYSIBM.SYSSTRINGS for conversions that do not involve Unicode.

DB2 V8 introduces a further optimization for ASCII conversion to UTF-8. DB2 uses the TRT instruction to ensure all characters are less than 128, and because the first 128 code points are the same between ASCII and UTF-8, no conversion is necessary. DB2 can skip conversion altogether.

Studies using strings containing only English characters show that minor conversion in V8 can be more than four times faster than major conversion in V8. Compared to conversion in V7, minor conversion can be more than six times faster. Keep in mind that minor conversion is only successful for single byte characters.

z/OS Conversion Services

The z/OS Conversion Services contain both a 31-bit and a 64-bit service. DB2 V7 uses the 31-bit service, while DB2 V8 most often uses the 64-bit service.

In October 2003, a new enhancement was shipped to z/OS in PTF UA05789, known as HC3, that included very significant performance enhancements, especially (but not exclusively) for conversions between UTF-8 and EBCDIC (or ASCII). The UTF-8 enhancement applies to both the 31-bit service and the 64-bit service. In order to take advantage of this performance enhancement, you must rebuild your conversion tables after this PTF has been applied. With this enhancement, Conversion Services under z/OS 1.4 can be as much as 3.75 times faster than under z/OS 1.3.

In addition to the UTF-8 enhancement, the 64-bit service implemented some improved module linkage that gives DB2 V8 an advantage over DB2 V7. Studies have shown that DB2 V8 improves the performance of major conversion over DB2 V7 and that this improvement is most evident with small strings. With a 10-byte string, V8 is 1.5 times faster. With a 200-byte string, V8 is only about 15% faster.

zSeries processor hardware

The z/OS Conversion Services take advantage of a number of new hardware instructions that are faster with the z990 and z890 processors. These instructions are:

- ▶ CUTFU - Convert from UTF-8 to UTF-16
- ▶ CUUTF - Convert from UTF-16 to UTF-8
- ▶ TROO - Translate from One byte to another One byte (used to convert single byte codes)
- ▶ TRTO - Translate from Two bytes to One byte (used to convert from UTF-16 to EBCDIC)
- ▶ TROT - Translate from One byte to Two bytes (used to convert from EBCDIC to UTF-16)
- ▶ TRTT - Translate Two bytes to Two bytes (used to convert between UTF-16 and double byte EBCDIC)

z/OS understands the term Unicode to be UTF-16, while DB2 for z/OS uses Unicode UTF-8 for the catalog and all CHAR columns within Unicode tables. So, all z/OS Conversion Services translations between SBCS EBCDIC and UTF-8 require two steps. Step 1 uses the TROT instruction to convert from EBCDIC to UTF-16, and step 2 uses the CUUTF instruction to convert from UTF-16 to UTF-8. Conversion in the opposite direction first uses CUTFU and then TRTO. In other words, UTF-16 is always used to transit between EBCDIC and UTF-8. Combining these two steps into one step was the key to HC3's improved performance, which was discussed earlier, but UTF-16 is still used as a transit point.

Conversions involving DBCS make use of the TRTT instruction, and mixed strings involve some combination of all of these instructions. If the caller need only convert between UTF-8 and UTF-16, only the CUUTF or CUTFU instruction is used, so the performance is faster than conversions between UTF-8 and EBCDIC or ASCII.

Like many other instructions, the CPU time of these instructions increases in proportion to the string length. In a workload involving Unicode conversion, the CPU time effects of these instructions can be high, especially with large strings. The z990 and z890 processors have provided significant enhancements to the TROO, TROT, TRTO and TRTT instructions, making them as much as five times faster than the z900 and z800 processors. The larger the string is, the larger the performance benefit of the z990 and z890 processors.

Studies have shown that the z990 reduces the conversion CPU time using V7 for 10 byte strings by 1.5 times. This is a typical z990 improvement that we have measured for most applications. This increases to a two times improvement for 200 byte strings, which is even better. Performance of other string lengths may be inter- or extrapolated from 10-byte and 200-byte strings. We see similar improvements in DB2 V8.

4.7.2 Conclusion

The enhancements we have seen in DB2 V8, z/OS Conversion Services and the z990 hardware dramatically improve Unicode conversion performance.

Prior to the z/OS Conversion Services improvement (HC3), Unicode conversion to EBCDIC was always slower than ASCII conversion to EBCDIC, but this is not true with HC3. In V7 Unicode conversion is about 20% faster than ASCII conversion. In addition, V8 major Unicode conversion is about 12% faster than ASCII conversion, but minor conversion is much faster still.

The cost of conversion can affect the total CPU time of your application. As an example, consider a table space scan of 1 million rows with no predicates or indexes. On the z900 with V8, the table space scan used 18.6 CPU seconds when there was no conversion, and 63.3 CPU seconds for major conversion. In other words, 67% of the time was spent doing conversion. For this study, each major conversion used 2.3 microseconds and was 25 times faster.

4.7.3 Recommendations

Although DB2 V8 can run on z/OS 1.3, we recommend you migrate to at least z/OS 1.4, to take advantage of the enhancements that have been made to the z/OS Conversion Services. Notice that z/OS 1.3 was out of service as of March 31, 2005. In addition, migrating to current hardware brings performance improvements to the z/OS Conversion Services.

When choosing to store data in a Unicode database, you have a choice of using either UTF-8 (CHAR) or UTF-16 (GRAPHIC). This choice includes both performance and nonperformance considerations. However, the major performance consideration when comparing UTF-8 to UTF-16 is the amount of space used. CPU time can also be another consideration. Obviously compression itself costs a lot of CPU time, although its I/O benefits may outweigh the CPU costs. Choosing between fixed and variable length strings can also be a consideration. The decision between UTF-8 and UTF-16 therefore has the potential to have a real impact on application performance.

We recommend you closely consider the type of data you will be storing in DB2 and choose UTF-8 (CHAR) or UTF-16 (GRAPHIC) depending on the characteristics of the data. UTF-8 is efficient for text that is predominantly English alphanumeric while UTF-16 is more efficient for Asian characters.

In addition, there are big advantages to limiting the data to single byte characters, so that minor conversion can succeed. However, we do not think you run the risk of compromising your application design in order to take advantage of DB2 minor conversion techniques. In any event, using multi-byte characters does not impact performance if Unicode conversion is unnecessary.

Some customers who use DB2 Connect to connect to DB2 for z/OS have been using `DISABLEUNICODE=1` to tell DB2 Connect to send the data to DB2 for z/OS in ASCII rather than Unicode. In the past, converting from ASCII to EBCDIC was much less expensive than converting from Unicode to EBCDIC. However now in z/OS 1.4, Unicode conversion is faster than ASCII conversion. Furthermore if the database is stored in Unicode, setting `DISABLEUNICODE` adds Unicode conversion times to the class 2 CPU times. We therefore recommend you not to set the `DISABLEUNICODE` parameter when you are using DB2 Connect to connect to DB2 V8.

Since the Universal Driver does not need DB2 Connect, the `DISABLEUNICODE` parameter is not generally applicable. Instead, the Universal Type 4 Driver supports a new

UseServerEncoding data source property. Since there is no disadvantage to using UseServerEncoding=1, we recommend you should always have this property set, especially if your DB2 system is using MIXED=NO.

4.8 Data encryption

DB2 V8 ships a number of built-in functions which allow you to encrypt and decrypt data. IBM also offer an encryption tool called the *IBM Data Encryption Tool for IMS and DB2 Databases*, program number 5799-GWD, also mentioned in 10.8, “Data Encryption for IMS and DB2 Databases” on page 369. The performance implications for encryption are roughly similar to data compression when only considering CPU overhead. In this section, we introduce both DB2 encryption and the IBM Data Encryption Tool and discuss recent hardware enhancements that improve encryption performance.

Today, many organizations are paying much more attention to the security of their data, to comply with various security regulations or as a result of emerging new technologies, for example, the emergence of the internet, Storage Area Networks (SAN) and more intelligent storage controllers. (Do you trust your storage vendor not to look at your data?)

However, data encryption has a number of challenges; including making changes to your application to encrypt and decrypt the data, encryption key management and the performance overhead of encryption.

Machines prior to the z990 have a Cryptographic Coprocessor Feature to improve the performance of encryption and decryption. However, only CPU 0 and 1 could perform encryption. To encrypt and decrypt data, tasks running on other processors need to be redispatched to run on CPU 0 or 1. Performance is therefore a problem if there is contention among tasks (for example, parallel query). In addition, dedicated LPARs might not be able to use the encryption hardware feature.

The z990 introduced a new hardware instruction, CP Assist for Cryptographic Function (CPACF), which can run on all CPUs and is a feature available only on the z990 hardware and later, not the older z900. The z990 also introduces a PCIXCC card which is needed for the IBM Data Encryption Tool, but not for the DB2 encryption function. Now, we briefly introduce these two encryption functions.

DB2 column level encryption

DB2 V8 ships a number of built-in functions which allow you to encrypt data at the *cell* level. These functions are ENCRYPT_TDES (or ENCRYPT), DECRYPT_BIN, DECRYPT_CHAR, and GETHINT. The SET ENCRYPTION PASSWORD statement allows you to specify a password as a key to encryption. Because you can specify a different password for every row that you insert, you can really encrypt data at the cell level in your tables. However, you are responsible for managing all these keys. So, make sure you have a mechanism in place to manage the passwords that are used to encrypt the data. Without the password, there is absolutely no way to decrypt the data. These encryption functions use the Triple Data Encryption Standard (Triple DES) to perform the encryption.

The DB2 built-in encryption functions require:

- ▶ DB2 V8
- ▶ Integrated Cryptographic Service Facility (ICSF)
- ▶ On z990, CPACF is required (PCIXCC card is not, unless DRDA encryption is necessary)
- ▶ Pre-z990, cryptographic coprocessor is required.

Each CP on the z990 has an assist processor on the chip in support of cryptography. This feature provides for hardware encryption and decryption support. Peripheral Component Interconnect Extended Cryptographic Coprocessor (PCIXCC) provides a cryptographic environment with added function. The PCIXCC consolidates the functions previously offered on the z900 by the Cryptographic Coprocessor feature (CCF) and the PCI Cryptographic Coprocessor (PCICC) feature. For a more detailed discussion of CPACF and PCIXCC, refer to the book *IBM eServer zSeries 990 (z990) Cryptography Implementation*, SG24-7070.

Applications implementing DB2 encryption must apply the DB2 encrypt and decrypt built-in functions for each column to be encrypted or decrypted. All encrypted columns must be declared “for bit data”. So, unchanged read-applications see data in encrypted form. Applications may apply a different key for each column, but may also supply the key in a special register. It is strongly recommended for performance to specify the key in the special register.

LOAD and UNLOAD utilities do not support DB2 encryption, but SQL-based programs such as DSNTIAUL do support encryption. Encryption of numeric fields is not supported. The length of encrypted columns must allow for an additional 24 bytes, rounded up to a double word boundary. So, space usage may be a concern if you plan to use DB2 to encrypt small columns.

Indexes are also encrypted. Predicates that depend on the collating sequence of encrypted columns, (for example range predicates), may produce wrong results (unless modified to use built-in functions correctly). For example:

```
SELECT COUNT(*) WHERE COL = :HV;
```

produces wrong results.

```
SELECT COUNT(*) WHERE COL = ENCRYPT_TDES(:HV);
```

produces correct results and almost no impact to performance.

```
SELECT COUNT(*) WHERE COL < ENCRYPT_TDES(:HV);
```

produces wrong results.

```
SELECT COUNT(*) WHERE DECRYPT_CHAR(COL) < :HV;
```

produces correct results with large impact on performance.

IBM Data Encryption Tool for IMS and DB2 Databases

IBM also offers an encryption tool called IBM Data Encryption Tool for IMS and DB2 Databases. This tool performs row level encryption using EDITPROCs. Unlike the DB2 encryption functions shipped with DB2, the Data Encryption Tool uses different keys to encrypt different tables. The encryption keys can be either *clear*, like the DB2 encryption functions, or *secure* and they are managed through ICSF (Integrated Cryptographic Service Facility). Clear keys are generally better performing. The tool also supports single, double, or triple DES. Once again, refer to the book, *IBM eServer zSeries 990 (z990) Cryptography Implementation*, SG24-7070, for a more detailed explanation of clear keys and secure keys.

You can find more information about the IBM Data Encryption Tool for IMS and DB2 Databases by visiting the Web site at:

<http://www.ibm.com/software/data/db2imstools/db2tools/ibmencrypt.html>

The IBM Data Encryption Tool for IMS and DB2 Databases supports all versions of DB2 but only encrypts the whole row. No application changes are required, however the DDL must be modified to include the EDITPROC. The applications do not need to be aware of encryption keys.

The following software is required to support clear keys

- ▶ APAR PQ93943 and PQ94822 for the tool
- ▶ ICSF SPE APAR OA08172 (availability TBD)
- ▶ FMID HCR770A or HCR770B
- ▶ OS/390 V2R10 or later

The following hardware is also required to support “clear” keys

- ▶ On z990, both CPACF and a PCIXCC card
- ▶ Pre z990, cryptographic coprocessor

While, secure keys require the following hardware:

- ▶ On z990, a PCIXCC card
- ▶ Pre z990, the cryptographic coprocessor

The PCIXCC card is required to install ICSF, which in turn, manages the encryption keys.

4.8.1 Performance

Studies on z990 have shown DB2 encryption costs approximately 2.3 microseconds per column, plus 0.0086 microseconds per byte. While the Encryption Tool (clear key) costs roughly 1 microsecond per row, plus 0.006 microseconds per byte.

So, to fetch 100k rows, encrypting two 32 byte columns using DB2 encryption:

$$\text{DB2 cost of encryption} = 2 \times (2.3 + 32 \times 0.0086) \times 100,000 = 520,000 \text{ microseconds}$$

To fetch 100k rows with row size of 204 bytes using the Encryption Tool, the additional CPU time is:

$$\text{Encryption Tool cost} = (1 + 204 \times 0.006) \times 100,000 = 220,000 \text{ microseconds}$$

For an equivalent number of bytes, we found it is faster to use DB2 encryption to encrypt and decrypt the data, in situations where the number of columns to be encrypted is three or less. Otherwise, may be a better choice. The breakeven point varies depending on how many other columns are in the table and how big each column is, but generally the IBM Data Encryption Tool is a better choice than DB2 column level encryption from the performance level point of view.

The cost of encryption is reduced on the z990 hardware, compared with the older z900 hardware. CPU time for the Encryption Tool is 10 times faster on a z990 versus the z900, and the elapsed time for multithreaded applications using encryption is much faster on z990 versus z900 depending on the number of concurrent tasks. Similar observations are applicable to the z890 processor, which has the same level of relative performance as the z990.

4.8.2 Conclusion

The CPU performance implications for encryption are similar to data compression. However the impact on encryption on CPU is likely to be slightly higher than compression.

OLTP performance is less affected than query performance. The best query performance is achieved by designing good indexes to avoid “touching” many rows.

4.8.3 Recommendations

For both DB2 encryption and the IBM Data Encryption Tool, we recommend you move to the current z990 or z890 hardware, where the hardware-assisted encryption instructions are available on all processors.

The IBM Encryption Tool (row level) performs generally better than the DB2 encryption and decryption (column level). The break

even point varies depending on how many other columns are in the table and their size. However, performance is not the only reason to choose one encryption strategy over the other, as they also vary in function.

Encryption is done before compression, so compression and encryption cannot be effectively combined. Encrypted data does not tend to compress very well because repeating unencrypted characters are no longer repeating after they are encrypted.

Important: Be aware that if you plan to use DRDA encryption, you need the PCIXCC function on your z990, because the hardware requires it.

4.9 Row level security

DB2 recognized the need for more granular security of DB2 data and introduced support for Row Level Security in V8. Multilevel security support in z/OS 1.5 and RACF Security Server 1.5, in conjunction with DB2 V8, simplify row level security implementation and management significantly.

In this section we first introduce Row Level Security, then describe the performance impact of Row Level security on DB2 workloads.

Security is becoming increasingly important in the past few years. Row level access control is increasingly critical. Many customers need to extend the granularity from table level to row level, so that an individual user is restricted to a specific set of rows. Good examples are Web hosting companies that need to store data from multiple customers into a single subsystem, database or table.

In the past, views have been used to *hide* data. They can subselect data to provide only certain columns or fields. By creating a view and granting privileges on it, you can give someone access to only a specific combination of data. However, the application tends to become much more complex when views are used in this way. Separating the data into different databases or base tables has also been used to provide more granular security.

DB2 recognizes this need and introduces support for Row Level Security in V8. Multilevel security (MLS) support in z/OS 1.5 and RACF Security Server 1.5, in conjunction with DB2 V8, simplify row level security implementation and management significantly.

Multilevel security is a security policy that allows the classification of data and users based on a system of hierarchical security levels, combined with a system of non-hierarchical security categories. A multilevel-secure security policy prevents unauthorized individuals from accessing information at a higher classification than their authorization (read-up), and prevents individuals from declassifying information (write-down).

You can use MLS for multiple purposes in DB2:

Multilevel security with row-level granularity

In this combination, DB2 grants are used for authorization at the DB2 object level (database, table, and so forth). Multilevel security is implemented only at the row level within DB2. External security is used only by multilevel security itself issuing the RACROUTE. This is the configuration on which we focus for the remainder of this section.

Multilevel security at the object level with external access control

In this combination, external access control, such as the RACF access control module, is used for authorization at the DB2 object level. In addition, you can now define a proper hierarchy of security labels for DB2 objects. For example, a database can be defined with a higher security label than its table spaces. The RACF access control module has been enhanced to use security labels to perform access checking on DB2 objects as part of multilevel security.

Multilevel security with row level granularity with external access control

This option combines both options mentioned above. It uses multilevel security to control the access to the DB2 objects, as well as multilevel security (SECLABELs) to control access at the row level within DB2.

In the following sections we describe some of the concepts of multilevel security. Multilevel security is complex and describing the details of it is beyond the scope of this publication. For more information, refer to the z/OS Security Server publications. An introduction can be found in *z/OS Planning for Multilevel Security and Common Criteria*, SA22-7509.

Security label: SECLABEL

A security label (SECLABEL) is defined for each object which defines the sensitivity of that object. This security label indicates the hierarchical level or classification of the information (such as top secret, confidential or general-use), as well as indicates to which non-hierarchical category the information belongs within that level (such as group ABC or group XYZ).

In RACF you must define two profiles in the RACF SECDATA resource class; one to define the security levels (SECLEVEL profiles) and the other to define the security categories (CATEGORY profile) for the system.

Security level: SECLEVEL

The hierarchical security level (SECLEVEL) defines the degree of sensitivity of the data. In the following example, security level, "LO" is defined to be a security level 10. The security administrator can define up to 254 security levels.

```
RDEFINE SECDATA SECLEVEL UACC(READ)
RALTER SECDATA SECLEVEL ADDMEM(LO/10 L1/30 L2/50 L3/70 L4/90)
```

Security category: CATEGORY

The non-hierarchical CATEGORY profile further qualifies the access capability. The security administrator can define zero or more categories that correspond to some grouping arrangement in the installation. The CATEGORY profile contains a member for each non-hierarchical category in the system. For example, C1 through C5 are security categories.

```
RDEFINE SECDATA CATEGORY UACC(READ)
RALTER SECDATA CATEGORY ADDMEM(C1 C2 C3 C4 C5)
```

Defining Security labels

After defining the SECLEVEL and CATEGORY profiles, the security administrator defines a profile in the SECLABEL resource class for each security label. The SECLABEL is a name of

up to eight uppercase alphanumeric or national characters and each SECLABEL profile specifies the particular combination of:

- ▶ A SECLEVEL member
- ▶ Zero or more members of the CATEGORY profile that apply to the security label

For example:

```
RDEFINE SECLABEL L1C12 SECLEVEL(L1) ADDCATEGORY(C1 C2) UACC(NONE)
```

In addition, RACF has a number of built-in security labels.

- ▶ SYSHIGH: Is equivalent to the highest security level defined and covers all categories defined.
- ▶ SYSLOW: Is equivalent to the lowest security level defined and has no categories signed. It is dominated by all other security labels.
- ▶ SYSNONE: Is treated as equivalent to any security label to which it is compared. SYSNONE, like SYSLOW, should be used for data that has no classified data content.
- ▶ SYSMULTI: Is treated as equivalent to any defined security label. It is intended to be used by users for access to data that has multilevel data classified.

Assigning security labels

A subject (or user ID) can have more than one security label, but can only use one security label at a given time. To authorize a subject to use a security label, the security administrator permits that subject's user ID to the profile in the RACF SECLABEL resource class for the security label. Do not forget to RACLIST REFRESH the SECLABEL class.

```
PERMIT L1C12 CLASS(SECLABEL) ACCESS(READ) ID(USER01)
```

The security label that a subject uses at a given time can be assigned in a number of ways. For example, a TSO/E user can specify a security label on the logon panel, and a batch user can specify a security label on the JOB statement. The default SECLABEL is defined in the user's RACF profile.

A resource can have only one security label. For most types of resources, the security administrator assigns a security label to each resource in the system that is to be protected by ensuring that the resource is protected by a profile, and by specifying the security label in the profile.

For DB2 row level security, DB2 maintains the security label in a new column attribute within the table itself. To implement multilevel security, the DBA adds a column with the security label data type to an existing table. This is done by defining a column with the AS SECURITY LABEL attribute. Each row value has a specific SECLABEL. This new column is populated with security label information when data is inserted into the table. Every row can have a different security label although this is not likely in practice. More likely, the number of distinct security labels is much smaller than the number of rows in a table.

How Row Level Security works

Briefly, access to tables is controlled by security labels which are assigned to users as well as to the data. If the security labels of a user and the data are an equivalent level, that user can access the data.

Dominance

One security label dominates another security label when both of the following conditions are true:

- ▶ The security level that defines the first security label is greater than or equal to the security level that defines the second security label.
- ▶ The set of security categories that defines the first security label includes the set of security categories that defines the second security label.

You can also look at dominance in a simplistic way as one security label being “greater than” another.

Reverse dominance

With reverse dominance access checking, the access rules are the reverse of the access rules for dominance access checking. This type of checking is not used by DB2.

In loose terms, it can be looked as “less than or equal to” checking.

Equivalence

Equivalence of security labels means that either the security labels have the same name, or they have different names but are defined with the same security level and identical security categories.

You can look at this type of checking as “equal to” checking. (One way to check is if both dominance and reverse dominance are true.)

Disjoint

Two security labels are considered disjoint when they have at least one category that the other does not have. Neither of the security labels dominates the other.

Read-up

Multilevel security controls prevent unauthorized individuals from accessing information at a higher classification than their authorization. It does not allow users to “read-up” or read above their authorization level. Read-up is enforced through dominance checking.

Write-down

Multilevel security also prevents individuals from declassifying information. This is also known as write-down, that is, writing information back at a lower level (down-level) than its current classification. Write-down is prevented by doing equivalence checking.

However, there may be cases where you want to allow write-down by selected individuals. The security administrator controls whether write-down is allowed at the system level by activating and deactivating the RACF MLS(FAILURES) option (using the SETROPTS command), or for controlled situations of write-down in which z/OS allows the security administrator to assign a “write-down by user” privilege to individual users that allows those users to select the ability to write down. To do so, a user has to have at least read authority on the IRR.WRITEDOWN.BYUSER profile in the RACF FACILITY class.

Accessing data

When a query fetches data from a table, DB2 checks the security label of the user submitting the query to the security label of the row. If there is a match (equivalent or dominating), the row is returned to the user. To perform this check, DB2 invokes RACROUTE for the determination. DB2 also caches the decision made by RACF so that subsequent rows with the same data security label could be determined locally.

For more detail on Multilevel Security, see *z/OS Planning for Multilevel Security and Common Criteria*, SA22-7509, and the recent book, *z/OS Multilevel Security and DB2 Row-level Security Revealed*, SG24-6480.

4.9.1 Performance

We can implement RACF external security with MLS at the DB2 object level to control access to databases, table spaces, tables, views, indexes, storage groups, buffer pools, plans, packages, stored procedures, and more. This is done by implementing the DB2 and RACF security exit DSNX@XAC. See the *DB2 UDB for z/OS Version 8 Administration Guide*, SC18-7413-01, and the *DB2 UDB for z/OS Version 8 RACF Access Control Module Guide*, SC18-7433, for discussions regarding how to implement RACF External Security for DB2.

At the time of writing this book, there have been no performance studies conducted comparing RACF External Security for DB2 objects, (either with or without using MLS), with DB2 native security. However, we consider the impact of DB2 External Security over DB2 native security (GRANT and REVOKE) insignificant.

We can also implement RACF MLS security with row level granularity, as is described here. These performance studies explore the performance impact of DB2 Row Level Security where there was no RACF external security at all, only DB2 native security. No application changes were made and there was no “application” row level security implemented (for example via views). Obviously, if you wish to change your application to remove any application-implemented row level security techniques and implement DB2 Row Level Security, your application probably becomes less complex and therefore less CPU intensive. Any CPU overhead caused by DB2 Row Level Security should be offset by these savings.

A user or thread has only one SECLABEL “active” at any one time. When rows are accessed, DB2 checks the thread’s, or user’s SECLABEL value with each new SECLABEL value from the rows being accessed. To avoid going to RACF for each and every row accessed, DB2 caches all the SECLABEL values that have been checked, both successfully or unsuccessfully. The cache is maintained per thread and is flushed on commit and rollback. So, if immediate changes are required in the security policy, then long-running applications which have not committed or rolled back may need to be canceled.

In a general sense, you can think of DB2 evaluating SECLABEL values in the same way as DB2 evaluating Stage 1 predicates, only in that DB2 attempts to perform SECLABEL checking as a Stage 1 activity. During access path selection, the DB2 optimizer does not consider the task of evaluating the SECLABELs the same way as it would any Stage 1 predicates. This is similar to resolving referential constraints. However, the need to evaluate SECLABEL values can impact SQL access paths. For example, Index Only access may now be regressed to an Index plus data access (to retrieve each row’s SECLABEL value), if the index being selected does not also contain the SECLABEL column. Row Level Security can therefore have a significant impact on your application performance by changing access paths.

The overhead of Row Level Security in DB2 is recorded as Accounting class 2 CPU time and varies depending on the complexity of your SQL, how many rows your query needs to “touch” during Stage 1 processing, and how many different SECLABEL values you have in your table. The more complex the query, the less impact Row Level Security has on CPU. This is because the overhead of Row Level Security is diluted with more CPU intensive queries. Therefore, your mileage will vary!

SELECT

The user’s SECLABEL is compared to the data SECLABEL of each row to be selected, also for the rows that do not qualify. If the SECLABEL of the user dominates the SECLABEL of the row, DB2 returns the row. If the SECLABEL of the user does not dominate the SECLABEL of the row, DB2 does not return the data from that row, and DB2 does not generate an error.

DB2 must call RACF for this dominance checking if the row's SECLABEL has not already been encountered and therefore cached.

If RACF is called, studies have shown up to a 10% impact on CPU with SELECT statements using Row Level Security. If the SECLABEL is found in the cache, this impact reduces to 2%. For a FETCH rather than a SELECT, this can be as high as 5%. This is because the CPU time which can be attributed to the overhead of Row Level Security consumes a larger percentage of the total CPU time.

INSERT

If the user has write-down privilege or write-down control is not enabled, the user can set the SECLABEL for the row to a valid security label that is not disjoint in relation to the user's security. If the user does not specify a value for the SECLABEL, the SECLABEL of the row becomes the same as the SECLABEL of the user. Alternatively, If the user does not have write-down privilege and write-down control is enabled, the SECLABEL of the row becomes the same as the SECLABEL of the user.

The cost of each INSERT statement can increase by up to 2%. This is also dependent on the complexity of the INSERT statement, the number of columns to be INSERTed and the number of indexes that must be updated. The extra overhead results in DB2 extracting the user's SECLABEL value and building the new row with this extra column.

UPDATE

If the SECLABEL of the user and the SECLABEL of the row are equivalent, the row is updated and the value of the SECLABEL is determined by whether the user has write-down privilege. If the user has write-down privilege or write-down control is not enabled, the user can set the SECLABEL of the row to a valid security label that is not disjoint in relation to the user's security. If the user does not have write-down privilege and write-down control is enabled, the SECLABEL of the row is set to the value of the SECLABEL of the user.

Alternatively, if the SECLABEL of the user dominates the SECLABEL of the row, the result of the UPDATE statement is determined by whether the user has write-down privilege. If the user has write-down privilege or write-down control is not enabled, the row is updated and the user can set the SECLABEL of the row to a valid security label that is not disjoint in relation to the user's security. If the user does not have write-down privilege and write-down control is enabled, the row is not updated.

Finally, if the SECLABEL of the row dominates the SECLABEL of the user, the row is not updated.

As is the case with SELECT, RACF is not consulted unless the user's SECLABEL is not already cached. Therefore we see the same degree of overhead in using Row Level Security as in the case of SELECT. We see up to a 10% performance impact if the user's SECLABEL is not cached and up to 2% if it is cached.

DELETE

If the SECLABEL of the user and the SECLABEL of the row are equivalent, the row is deleted.

If the SECLABEL of the user dominates the SECLABEL of the row, the user's write-down privilege determines the result of the DELETE statement: If the user has write-down privilege or write-down control is not enabled, the row is deleted. Alternatively, if the user does not have write-down privilege and write-down control is enabled, the row is not deleted.

Finally, if the SECLABEL of the row dominates the SECLABEL of the user, the row is not deleted.

The performance impact of Row Level Security on DELETE is therefore very similar to the performance impact on UPDATE.

Utilities

You need a valid SECLABEL and additional authorizations to run certain LOAD, UNLOAD, and REORG TABLESPACE jobs on tables that have multilevel security enabled. All other utilities check only for authorization to operate on the table space; they do not check for row-level authorization and are therefore not impacted. Offline utilities such as DSN1COPY and DSN1PRNT do not impose Row Level Security and are also not impacted.

At the time of writing this book, no actual performance studies have been undertaken to quantify the impact of Row Level Security on utilities. However, we can make some comment as to what performance impact may occur. As always the performance overhead on utilities varies greatly, depending on how many columns are in the table and how complex the definitions of the columns are. A utility processing more columns and performing more complex data conversion sees less impact using Row Level Security because any impact of SECLABEL processing is diluted. However, the CPU overhead of using Row Level Security may be significant when you are processing a large number of rows with few columns in each row. As always, your mileage will vary.

Executing a LOAD RESUME utility against a table space containing tables with Row Level Security requires that the user is identified to RACF and has a valid SECLABEL. LOAD RESUME adheres to the same rules as INSERT. Without write-down authorization, the SECLABEL in the table is set to the SECLABEL associated with the user ID executing the LOAD RESUME utility. With write-down, any valid security label that is not disjoint in relation to the user's security can be specified. So, the performance impact of Row Level Security on LOAD RESUME is similar, but slightly higher than that of INSERTs.

LOAD REPLACE deletes all rows in a table space. Therefore, write-down authority is required. If the user ID associated with the job that is running the LOAD REPLACE utility does not have write-down privilege, and write-down is in effect, an error message is issued.

Executing the UNLOAD or REORG UNLOAD EXTERNAL utility against tables with Row Level Security requires that the user associated with the job is identified to RACF, and have a valid SECLABEL. For each row unloaded from those tables, if the user SECLABEL dominates the data SECLABEL, then the row is unloaded. If the user's SECLABEL does not dominate the data SECLABEL, the row is not unloaded and no error is returned. The performance impact is therefore similar to that of SELECT.

The UNLOAD utility has a sampling option where you can specify a percentage of rows to be unloaded. The sampling filter operates only on the rows you are allowed to access, so it is not possible for an unauthorized user to calculate the total size of the table using the UNLOAD utility. However, the RUNSTATS utility, which also has a sampling function, does not invoke Row Level Security. RUNSTATS also updates the DB2 catalog with column distribution statistics and column values. We therefore recommend that if you are planning to implement MLS Row Level Security to protect your data, you should also review who has access to your DB2 catalog tables.

Executing REORG... DISCARD on tables with Row Level Security requires that the user is identified to RACF and has a valid SECLABEL. REORG with the DISCARD option adheres to the same rules as the SQL DELETE statement. For each row unloaded from those tables, if the row qualifies to be discarded, the user SECLABEL is compared to the data SECLABEL. We would therefore evaluate the performance impact to be slightly higher than that of DELETEs.

4.9.2 Conclusion

The CPU increase caused by Row Level Security occurs because DB2 must now access the SECLABEL column in the table, evaluate the user's SECLABEL against the SECLABEL of the data, manage a cache of SECLABEL values, and periodically consult RACF. At a very rough level, the performance impact of row level security can be considered in the same ballpark as DB2 data compression with less than a 5% impact for a typical online transaction but higher for a CPU-bound sequential scan.

However, these CPU costs are very much dependent on the complexity of the SQL, the number and size of columns manipulated by the SQL and the number of indexes that need to be updated. Anticipate seeing more overhead with simple queries which must touch a large number of unique seclabels and users, so that the cache is ineffective, rather than complex queries. A simple table space scan returning a small number of rows can potentially experience the highest impact, even with caching.

Users who have already implemented a form of row level security in their application and want to convert to DB2's implementation of Row Level Security should not see a high CPU increase. The applications probably are less complex and the SQL less complex and expensive to run. So, the extra cost in CPU caused by Row Level Security should be compensated by the savings in CPU resulting from a simpler application.

4.9.3 Recommendations

Row Level Security using MLS only has a small overhead compared to no row level security at all. In addition, the MLS Row Level Security probably out performs any application implementation which provides similar function. MLS Row Level Security also offers a more complete or secure solution than many application implementations of row level security.

When you add MLS Row Level Security into your existing applications, or design new ones with it, you should consider the performance impact to the application and possibly review the physical design.

The performance of accessing tables that contain a security label can be impacted if the SECLABEL column is not included in the indexes. The SECLABEL column is used whenever a table with Row Level Security enabled is accessed. Access paths may change as DB2 must now access the SECLABEL column. An index only scan may now turn into an index plus data scan. Therefore, it is a good idea to include the SECLABEL column in your existing indexes if your index design allows, especially when your queries are using index-only access today.

DB2 caches security labels to avoid extra calls to RACF. This caching would work best if there are a relatively small number of security labels to be checked compared with the number of rows accessed.

Some application processes may also need to change to accommodate write-down processing and preserve existing SECLABEL values in the table. For example, you may need to run multiple SQL or batch jobs against a table where you would normally run a single execution. In this way, you preserve the SECLABEL values already in the table.

For example, if you have an SQL process to increase everyone's salary by 10%:

```
UPDATE PAYROLL  
SET SALARY = SALARY * 1.1
```

You need write-down authority to update the whole table. Unfortunately DB2 sets ALL SECLABEL values to the one value.

To preserve the SECLABEL values that are in the table, you need to execute the SQL in separate tasks with the various unique seclabels in the table, and include the predicate:

```
WHERE SECLABEL_COLUMN = GETVARIABLE(SYSTEM.SECLABEL)
```

An alternative is to include another SET clause in the SQL UPDATE statement in order to preserve the SECLABEL values already in the table.

```
UPDATE PAYROLL  
SET SALARY = SALARY * 1.1,  
    SECLABEL_COLUMN = SECLABEL_COLUMN
```

This technique would require write-down authority for the user running the job. If there are any rows in the table that dominate (not equivalent) the user's seclabel, then they are not updated.

In summary, although MLS Row Level Security can simplify the separation in an application, it is not transparent. You need to review how your application INSERTs and UPDATEs the data, as these processes probably need to change in order to manage and maintain the SECLABEL values in the data. There are similar considerations for batch processes and some utilities. Utilities may also take a little longer to execute. Processes for managing and controlling which processes have the write-down authority must also be developed.

4.10 New CI size

With V7, DB2 only uses the VSAM Control Interval (CI) size of 4 KB. If the page size is 8, 16 or 32 KB, DB2 treats the page as a chain of 2, 4, or 8 CIs and requires VSAM to only allocate CIs in 4 KB blocks.

DB2 V8 introduces support for VSAM CI sizes of 8, 16, and 32 KB, activated by the new DSNZPARM parameter, DSVCI, in panel DSNTIP7. This is valid for both user-defined and DB2 STOGROUP-defined table spaces. Index spaces only use 4 KB pages.

Once you have activated the new CI sizes (in NFM), all new table spaces are allocated by DB2 with a CI corresponding to the page size. The table spaces already existing at the time of migration to DB2 V8 are later converted by the first execution of LOAD REPLACE or REORG. When using DB2 user-defined objects (USING VCAT), it is your responsibility to define the underlying VSAM cluster with the correct CISIZE.

DB2 uses VSAM linear data sets, (LDS), which only uses a CI size which is a multiple of 4 KB, and no CIDE. CIs are written by VSAM in physical blocks which try to use the best fit within the CI, and keep track occupancy decent.

Now, consider Table 4-12, which describes the existing relationship between DB2 V7 page sizes and VSAM CI sizes assuming a 3390 disk.

Table 4-12 DB2 V7 - Sizes in KB (usable track size 48 KB for 3390)

DB2 page size	VSAM CI size	VSAM physical block size	Blocks per track	DB2 pages per track
4	4	4	12	12
8	4	4	12	6
16	4	4	12	3
32	4	4	12	1.5

VSAM does not know the DB2 page size, nor the fact that DB2 chains the VSAM CIs to make up its “logical” page size. VSAM I/Os cannot span cylinder boundaries or data set extent boundaries. So, we can see by Table 4-12 that if we have a DB2 page size of 32 KB, VSAM can write 22.5 32 KB DB2 pages per cylinder, (15 X 1.5). A 32 KB DB2 page can therefore span a cylinder boundary and therefore span DASD volumes. This is not a problem since DB2 chains the physical pages and guarantees the data integrity by ensuring all 8 VSAM pages are written correctly.

Since a 32 KB DB2 “logical” page can span DASD volumes, restrictions exist in V7 that Concurrent Copy cannot support 32 KB pages and striping of objects with a page size > 4 KB is not supported for DB2 data sets. In V7, striping is only allowed to 4 KB page sets and Concurrent Copy is not allowed for 32 KB table spaces. Concurrent Copy can copy the two volumes at different points in time; however, a single DB2 32 KB “logical” page can span these two DASD volumes. A DB2 page should not be allowed to span a CA boundary and in the case of striping, a CI cannot span a cylinder.

Now, we see what has changed in V8. Consider Table 4-13, which describes the relationship between DB2 V8 page sizes and VSAM CI sizes.

Table 4-13 DB2 V8 - Sizes in KB (usable track size 48 KB for 3390)

DB2 page size	VSAM CI size	VSAM physical block size	Blocks per track	DB2 pages per track
4	4	4	12	12
8	8	8	6	6
16	16	16	3	3
32	32	16	3	1.5

VSAM now deals with the 4 different CI sizes, and, knowing the track size, now allows CI size equal to page size. VSAM writes the CI in a physical block of the same size as the page, but splits the 32 KB CI over two blocks of 16 KB in order not to waste space due track size (48 KB). It fits the 32 KB CIs 1.5 per track. VSAM also takes care of spanning track boundaries and operates I/O at the CI level. A 16 KB block is wasted every 15 tracks (CA size) because the CI cannot span a CA boundary. This is a small price to pay, now that VSAM is aware of the DB2 page size.

Now VSAM can guarantee a DB2 page does not span CAs and therefore DASD volumes. The new CI sizes reduce integrity exposures, and relieve restrictions on concurrent copy (of 32 KB objects) and the use of striping (of objects with a page size > 4 KB). It also resolves small integrity exposure with the Split Mirror solution using FlashCopy on 32 KB pages (SAP) if at the extent border. A DB2 page now is always written in a single I/O and is totally contained in the same extent and the same device (even if striping is used).

4.10.1 Performance

What happens to I/O performance if we increase the VSAM CI size from 4 KB, to 8 KB, to 16 KB? Preliminary measurements show that large CIs are beneficial for FICON channels and storage controllers for table space scans. Table 4-14 shows I/O service times on a ESS 800 for 4 KB, 8 KB, 16 KB and 32 KB page sizes, respectively.

Table 4-14 I/O service time on ESS 800

I/O service time on ESS 800 (ms)	Page sizes			
	4 KB	8 KB	16 KB	32 KB
Cache hit	0.4	0.43	0.5	0.7
Cache miss	7.0	7.03	7.1	7.3

We see fewer “blocks” on a FICON link which results in the improved performance. We can also see that changing the VSAM CI size has little impact on I/O performance.

However, there is an impact on DB2 buffer pool hit ratios. For example, if you increase your DB2 page size from 4 KB to 8 KB and do not increase your buffer pool size, then you are only able to cache half as many pages in the buffer pool. This certainly impacts the buffer pool hit ratio. If you do plan to use larger DB2 page sizes, we recommend you also review your DB2 buffer pool sizes. However always ensure you have enough real storage available to back any increase in buffer pools, to avoid any paging to disk.

Now we turn our attention to DB2. Figure 4-20 compares a DB2 table space scan with 4 KB, 8 KB and 16 KB page sizes respectively.

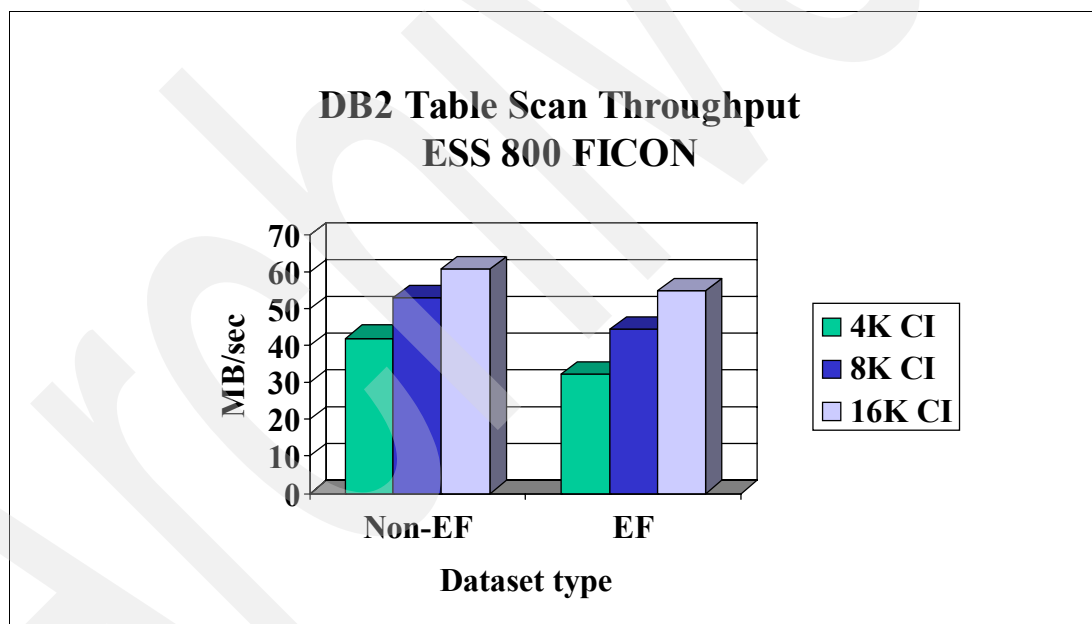


Figure 4-20 Larger CI size impact on DB2

A DB2 page size of 32 KB is not shown as VSAM implements this as 2 16 KB CIs. So, the results are the same as a DB2 16 KB page size. EF data sets are VSAM Extended Format data sets. EF data sets are required to allocate table spaces larger than 4 GB and for DFSMS™ striping.

The performance improvements are most pronounced when using Extended Format (EF) VSAM data sets. We see 45% improvement when we increase the VSAM CI size from 4 KB to 16 KB, and we see huge 70% improvements if these data sets are also EF-Enabled.

4.10.2 Conclusion

All I/O bound executions benefit with using larger VSAM CI sizes, including COPY and REORG. This enhancement can also potentially reduce elapsed time for table space scans.

There is little to be gained in performance by going above 16 KB CI size.

4.10.3 Striping

Now, for a few brief words on VSAM striping. In DB2 V7, VSAM striping was only recommended for DB2 log data sets and DB2 work file data sets. Although you could also use VSAM striping for application data with 4 KB pages, it was not generally recommended or widely used, because it was not extensively researched or tested. However, some customers have used VSAM striping rather successfully in V7 on DB2 indexes, to dramatically increase the performance of REBUILD INDEX. Once you have saturated your channels, VSAM striping does not really help.

I/O striping is beneficial for those cases where parallel partition I/O is not possible, for example, segmented tablespaces, work files, non-partitioning indexes, and so on.

4.10.4 Recommendations

If you already have large pages, migrate them to large CIs. The default CI size is equal to the page size. If you have large table space scans, larger CIs can also help.

4.11 IBM System z9 and MIDAWs

MIDAW stands for Modified Indirect Data Address Word facility, and as the name says, it is a modification to the standard IDAW channel programming technique that has existed since the S/360™ days. MIDAWs adds a new method of gathering data into and scattering data from discontinuous storage locations during I/O operations. This improves the performance of FICON channels.

MIDAWs require the IBM System z9™ server and z/OS Release 1.7 or APAR OA10984 with z/OS Release 1.6, see Figure 4-21.

z9-109 I/O Overview

■ I/O Enhancements

- Up to 28 FICON Express/FICON Express2 features per I/O cage
 - 4 channels/feature FICON/FCP
 - 1, 2 Gbps auto-negotiated
- Modified Indirect Data Address Word (MIDAW) facility
- Multiple (2) Subchannel sets (MSS)
 - Increase to 63.75K Subchannels for Set-0
- Up to 64* x 2.7GB STI's (21 STIs max for 3 I/O cages)

■ Storage Area Networks (SANs) enhancements

- N_Port ID Virtualization
- Program Directed re-IPL
- FICON Link Incident Reporting

■ Networking enhancements

- HiperSockets IPv6
- OSA-Express2 1000BASE-T Ethernet
- OSA-Express2 OSN (OSA for NCP support)
- GARP VLAN management (GRVP)



* z9-109 exploits a subset of its designed I/O capability

Figure 4-21 IBM System z9

The use of MIDAWs reduces the number of frames and sequences flowing across the link, making the channel more efficient.

Figure 4-22 summarizes the comparisons with and without MIDAW for a partitioned table space scan. Page fixing is active and the table space is in EF.

The I/O response time is reduced by about 50%, and gets even better increasing the number of partitions. Channel utilization is largely reduced, and the throughput increases up to 186 MB/sec, close to the FICON link nominal capability of 200 MB/sec in one direction.

Parallel DB2 Table Scan, EF 4K (single channel)

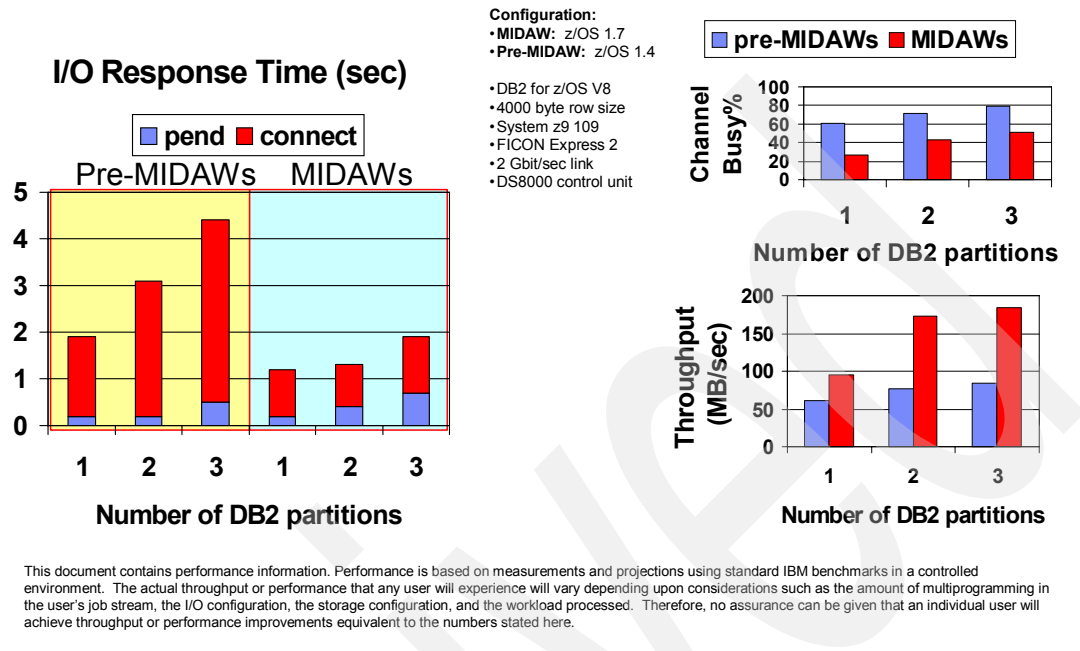


Figure 4-22 Table space scan with MIDAWs

The measurements have also shown that EF and non EF data sets have similar throughput using MIDAW for sequential prefetch of 4 KB pages.

Notice that large pages do not require MIDAWs to achieve excellent I/O performance, they already benefitted from the performance improvements of FICON Express2 and the DS8000™. MIDAWs add benefits targeted at the 4 KB pages.

4.12 Instrumentation enhancements

There are several instrumentation changes with DB2 V8. See Appendix F of the *DB2 UDB for z/OS Version 8 Release Planning Guide*, SC18-7425-01, for an overview of the record changes. Details are in the DSNWMSGs member of the SDSNIVPD library and recent related maintenance is listed in APAR PQ86477. This is a list of the major changes:

- ▶ Package level accounting
- ▶ Accounting rollup for DDF & RRSAF
- ▶ New and improved traces, larger monitor area
- ▶ Long-running reader
- ▶ Lock escalation
- ▶ Full SQL statement
- ▶ PREPARE attributes
- ▶ High water marks
- ▶ Secondary authorization ids
- ▶ Storage 64-bit, READS
- ▶ Dynamic statement cache
- ▶ Temporary space usage
- ▶ Auditing for multilevel security
- ▶ Option for EBCDIC or Unicode data

In this section we describe the significant enhancements to DB2 instrumentation in V8, paying special attention to the new IFCIDs that provide detailed information to help you monitor and tune your DB2 environment.

4.12.1 Unicode IFCIDs

DB2 V8 introduces a new DSNZPARM parameter, UIFCIDS, which governs whether output from IFC records should include Unicode information.

DB2 V8 supports a Unicode catalog and long names, also in Unicode. This means that DB2 object names, for example table names, can be longer than before and use “special” Unicode characters. Normally the various DB2 identifiers that are externalized in IFCID records are converted to EBCDIC. This is done to make it easier for applications that read IFCID records, like online monitors, to provide “toleration support” for DB2 V8.

Only a subset of the character fields is encoded in Unicode. The remaining fields maintain the same encoding of previous releases, EBCDIC. The fields that can be written as Unicode are identified in the IFCID record in the DSNQWxx mapping macros which are shipped in the SDSNMACS dataset. Each field is identified by a %U in the comment area to the right of the field declaration.

All of the information that is presented in the IFCID records in Unicode are stored in Unicode in memory. If you choose to have your IFCID records continue to contain only EBCDIC data, each of those data elements must be converted to EBCDIC from Unicode. See 4.7, “Unicode” on page 174 for more information.

4.12.2 Statistics enhancements

In this section we discuss the IFC enhancements related to storage trace records that can be obtained through the statistics trace.

Enhanced DBM1 and z/OS storage usage reports

Additional fields were added to storage-related IFCIDs 225 and 217 for 64-bit addressing. Sample DB2 Performance Expert (PE) statistics trace reports, showing the IFCID 225 record, are discussed in more detail in 4.2, “Virtual storage constraint relief” on page 139. IFCID 217 has more detailed information, but is normally only used under the guidance of the IBM Support team, so it is not discussed in more detail in this publication.

In addition, IFCID 225 and 217 have been added to monitor class 1 and are available via IFI READS requests.

Additional high water mark counters in statistics records

DB2 V8 adds the following high water mark counters to the statistics record, which should assist you in more accurately setting the appropriate DSNZPARM parameters:

- ▶ Q3STHWIB - High water mark for IDBACK, the number of background connections
- ▶ Q3STHWIF - High water mark for IDFORE, the number of foreground connections
- ▶ Q3STHWCT - High water mark for CTHREAD, the number of active threads

These fields are also reported in DB2 Performance Expert (DB2 PE) statistics reports, in the Subsystem Services section, as shown in Example 4-4.

Example 4-4 DB2 PE enhanced subsystem services reporting

SUBSYSTEM SERVICES	QUANTITY	/SECOND	/THREAD	/COMMIT
--------------------	----------	---------	---------	---------

IDENTIFY	0	0.00	0.00	0.00
CREATE THREAD	2	261.75	1.00	1.00
SIGNON	0	0.00	0.00	0.00
TERMINATE	2	261.75	1.00	1.00
ROLLBACK	0	0.00	0.00	0.00
COMMIT PHASE 1	0	0.00	0.00	0.00
COMMIT PHASE 2	0	0.00	0.00	0.00
READ ONLY COMMIT	0	0.00	0.00	0.00
UNITS OF RECOVERY INDOUBT	0	0.00	0.00	0.00
UNITS OF REC.INDBT RESOLVED	0	0.00	0.00	0.00
SYNCHS(SINGLE PHASE COMMIT)	2	261.75	1.00	1.00
QUEUED AT CREATE THREAD	0	0.00	0.00	0.00
SUBSYSTEM ALLIED MEMORY EOT	0	0.00	0.00	0.00
SUBSYSTEM ALLIED MEMORY EOM	0	0.00	0.00	0.00
SYSTEM EVENT CHECKPOINT	0	0.00	0.00	0.00
HIGH WATER MARK IDBACK	1	130.87	0.50	0.50
HIGH WATER MARK IDFORE	1	130.87	0.50	0.50
HIGH WATER MARK CTHREAD	1	130.87	0.50	0.50

In addition, the high water mark for MAXDBAT and CONDBAT are listed in the enhanced DB2 PE Statistics reports, under the DRDA Remote Locations section.

Package information added to deadlock trace records

When a deadlock (IFCID 172) occurs, DB2 now adds package and DBRM related information about the holder and blocker that allows you to identify the culprit and victims more easily. The following information is added for both the blockers and holders of the locks involved in the deadlock:

- ▶ Location name
- ▶ Collection name
- ▶ Program or package name
- ▶ Consistency token

This information is also available from DB2 PE, as shown in Example 4-5.

Example 4-5 Enhances deadlock trace record

LOCATION: PMODB2A			DB2 PERFORMANCE EXPERT (V2)				PAGE: 1-3		
GROUP: N/A			RECORD TRACE - LONG				REQUESTED FROM: NOT SPECIFIED		
MEMBER: N/A							TO: NOT SPECIFIED		
SUBSYSTEM: SDA1							ACTUAL FROM: 04/16/04 09:53:59.42		
DB2 VERSION: V8							PAGE DATE: 04/16/04		
PRIMAUTH	CONNECT	INSTANCE	END_USER	WS_NAME		TRANSACT			
ORIGAUTH	CORRNAME	CONNTYPE	RECORD TIME	DESTNO ACE	IFC	DESCRIPTION	DATA		
PLANNAME	CORRNMBR		TCB CPU TIME		ID				

N/P	N/P	00BB46094741	N/P	N/P		N/P			
N/P	N/P	'BLANK'	09:53:59.43270268	103075	3 172	DEADLOCK DATA	NETWORKID: G998C442	LUNAME: HF05	LUWSEQ: 1
N/P	N/P		N/P						

INTERVAL COUNT:		152949	WAITERS INVOLVED:		2	TIME DETECTED: 04/16/04 09:53:59.424504			

LOCK RES TYPE: PAGESET LOCK				UNIT OF WORK					
LOCK HASH VALUE: X'00000906'				R E S O U R C E					
				DBID: DSNDB06		OBID: SYSDBASE			
B L O C K E R									

```

PRIMAUTH : AND          PLAN NAME : DISTSERV  CORR ID : javaw.exe      CONN ID : SERVER
NETWORKID : G998C442    LUNAME : HF05      OWNING WORK UNIT: 223   UNIQUENESS VALUE: X'00BB46094741'
MEMBER : SDA1          DURATION : COMMIT    STATE : INTENT EXCLUSIVE ACE : 3
TRANSACTION : javaw.exe WS_NAME : B55Z5WHO    END_USER: and
PROGRAM NAME: SYSSH200  LOCATION : 'BLANK'    PKG/COLL ID: NULLID
CONS TOKEN : X'5359534C564C3031'
STATUS : HOLD
QW0172HF: X'12'

PRIMAUTH : CHM          PLAN NAME : DSNESPRR  CORR ID : CHM           CONN ID : TSO
NETWORKID : DEIBMIPS    LUNAME : IPSASDA1     OWNING WORK UNIT: 132   UNIQUENESS VALUE: X'BB13C6E7AD25'
MEMBER : SDA1          DURATION : COMMIT    STATE : SHARED INTENT EXCLUSIVACE : 2
TRANSACTION : 'BLANK'   WS_NAME : 'BLANK'     END_USER: 'BLANK'
PROGRAM NAME: DSNESM68  LOCATION : 'BLANK'    PKG/COLL ID: DSNESPRR
CONS TOKEN : X'149EEA901A79FE48'
DB2S ASIC : 5008      REQ WORK UNIT: 132    EB PTR : X'206F8540'    REQ FUNCTION: CHANGE
WORTH : X'12'         QW0172WG: X'32'

LOCK RES TYPE: DATA BASE LOCK
LOCK HASH VALUE: X'00000080'

RESO U R C E
DBID: DSNDB04      OBID: 0

B L O C K E R
PRIMAUTH : CHM          PLAN NAME : DSNESPRR  CORR ID : CHM           CONN ID : TSO
NETWORKID : DEIBMIPS    LUNAME : IPSASDA1     OWNING WORK UNIT: 132   UNIQUENESS VALUE: X'BB13C6E7AD25'
MEMBER : SDA1          DURATION : COMMIT    STATE : EXCLUSIVE      ACE : 2
TRANSACTION : 'BLANK'   WS_NAME : 'BLANK'     END_USER: 'BLANK'
PROGRAM NAME: DSNESM68  LOCATION : 'BLANK'    PKG/COLL ID: DSNESPRR
CONS TOKEN : X'149EEA901A79FE48'
STATUS : HOLD
QW0172HF: X'10'

PRIMAUTH : AND          PLAN NAME : DISTSERV  CORR ID : javaw.exe      CONN ID : SERVER
NETWORKID : G998C442    LUNAME : HF05      OWNING WORK UNIT: 223   UNIQUENESS VALUE: X'00BB46094741'
MEMBER : SDA1          DURATION : COMMIT    STATE : EXCLUSIVE      ACE : 3
TRANSACTION : javaw.exe WS_NAME : B55Z5WHO    END_USER: and
PROGRAM NAME: SYSSH200  LOCATION : 'BLANK'    PKG/COLL ID: NULLID
CONS TOKEN : X'5359534C564C3031'
DB2S ASIC : 5008      REQ WORK UNIT: 223    EB PTR : X'2027DFB8'    REQ FUNCTION: LOCK
WORTH : X'11'         QW0172WG: X'30'
-----

```

Detecting long-running readers

DB2 V8 introduces a new DSNZPARM, LRDRTHLD, as a threshold value in number of minutes an application can hold a read claim against an object before notification. You can set this value in order to be able to detect long-running readers. If statistics class 3 is active, an additional field type in IFCID 313 is recorded.

4.12.3 Lock escalation IFCID 337

Many installations go to great lengths to avoid or minimize lock escalation. However, DB2 V7 does not produce an IFCID record when lock escalation occurs, thus making it difficult for performance monitors to report on lock escalation. You can trap console message DSNIO31I today. It is produced whenever lock escalation occurs. However, performance monitors prefer a single interface for all performance-related information through the DB2 Instrumentation Facility Component.

DB2 V8 introduces a new IFCID 337 to report on lock escalations. IFCID 337 is added to the statistics trace class 3 and performance trace class 6. If activated, IFCID 337 is written whenever lock escalation occurs. The record contains a superset of the information that is reported in message DSNIO31I. Refer to Appendix C of *DB2 UDB for z/OS Version 8: Everything You Ever Wanted to Know, ... and More*, SG24-6079, for a more detailed discussion of IFCID 337, or member DSNWMGSG in DB2 V8 SDSNSAMP for a description of its contents.

In addition, message DSNIO31I is enhanced to include the collection ID in order to be consistent with IFCID 337.

This message is completed with the COLLECTION-ID if available, see Example 4-6.

Example 4-6 DSNIO31I - Lock escalation

```
DSNIO31I  -D8F1 DSNILKES - LOCK ESCALATION HAS
OCCURRED FOR
  RESOURCE NAME = DSND804.SSECQTYO
  LOCK STATE = X
  PLAN NAME : PACKAGE NAME = DSNESPCS : DSNESM68
COLLECTION-ID = DSNESPCS
  STATEMENT NUMBER = 000063
  CORRELATION-ID = PAOLR6
  CONNECTION-ID = TSO
  LUW-ID = USIBMSC.SCPD8F1.BC1C562B4800
  THREAD-INFO = PAOLR6 : * : *
```

4.12.4 Accounting enhancements

In addition to reducing the overhead of collecting class 2 and class 3 accounting information, a number of other enhancements have also been made to the accounting trace records.

Accounting roll-up for DDF and RRSF threads

To avoid flooding the system with accounting records, DB2 V8 allows applications coming into DB2 through DDF or RRS to roll up the accounting information of individual “transactions” into a single record that is written out at a user-defined interval.

Package level accounting

DB2 V8 adds SQL, buffer pool, and locking counters to package level accounting. As before, the gathering of package level accounting level information is triggered by activating accounting trace classes 7 and 8. This is especially important for applications that come in through the network via DRDA, since they only execute packages, not plans, and they produce a large amount of accounting-related information.

The new SQL counters collected at the package level are:

- ▶ The number of SELECT statements
- ▶ The number of INSERT statements
- ▶ The number of UPDATE statements
- ▶ The number of DELETE statements
- ▶ The number of DESCRIBE statements
- ▶ The number of PREPARE statements
- ▶ The number of OPEN statements
- ▶ The number of CLOSE statements
- ▶ The number of FETCH statements
- ▶ The number of LOCK TABLE statements
- ▶ The number of SQL CALL statements

The buffer pool counters at the package level are identical to those you have in today’s plan level accounting record. However, there is only one section for all buffer pools used by the package. For example, if the package did 5 getpages for objects in BP1 and 9 getpages for objects in BP2, in the package level buffer pool section, you find 14 getpages.

Note, however, that adding up the numbers in the buffer pool sections of all packages touched by a plan, may not add up to the same number that is shown at the plan level. This can be the case because DB2 does some work while the plan is running but before the package is

invoked; for example, the loading of the package into the EDM pool, or work that is done via DBRMs in the plan that do not have a package.

The locking information at the package level is identical to the information you have in today's plan level accounting record.

Note: In V8, package-related accounting information is always stored in IFCID 239 and no longer stored as part of IFCID 3 (plan level accounting).

Writing accounting records with KEEP_DYNAMIC(YES)

If the DB2 DSNZPARM parameter CMTSTAT is set to INACTIVE (new default for DB2 V8), DB2 writes an accounting record when a transaction commits and the connection qualifies to become inactive. However, when using the KEEP_DYNAMIC(YES) bind option, DB2 cannot disconnect the connection from the thread (which happens when the connection becomes inactive), because the thread contains information about locally cached statements.

Therefore DDF threads always have to remain active when using KEEP_DYNAMIC(YES). As a consequence accounting records are not cut at transaction boundaries. Likewise, DDF does not reestablish WLM enclaves at transaction boundaries.

Although KEEP_DYNAMIC(YES) still prevents DDF threads from becoming inactive in DB2 V8, using KEEP_DYNAMIC(YES) still allows DB2 to cut accounting records at transaction boundaries. When a new request arrives from the client system over the connection (that is still active), a new enclave is created and a new accounting interval is started.

At commit time, when DB2 evaluates whether a DDF connection is eligible for inactivation, if the only reason why it cannot become inactive is the presence of cached dynamic SQL statements due to KEEP_DYNAMIC(YES), DDF still cuts an accounting record, and also completes the WLM enclave (as if KEEP_DYNAMIC(YES) were not specified). As in V7, the presence of held cursors or declared temporary tables keeps the thread active and does not allow accounting intervals or WLM enclaves to complete.

With this new behavior, you may now consider period-based WLM goals for threads that commit frequently, but cannot become inactive only because they use KEEP_DYNAMIC(YES). Threads that cannot become inactive for other reasons do not reset the WLM enclave, and period-based goals are probably not right for them (which was the case in the past).

4.12.5 SQLCODE -905 new IFCID 173

The current method of monitoring dynamic SQL statements that exceeded the resource limit facility (RLF) ASUTIME time limit is inadequate. It does not provide the DBA with information at the system level.

Description

Currently DB2 issues an SQLCODE -905 for the dynamic SQL statement which exceeds the ASUTIME time limit set by the DBA. This is inadequate for the DBA to identify and act on it if this is a recurring problem.

A new IFCID 173 trace record is written whenever a SQLCODE -905 is issued. For a distributed system, IFCID 173 is issued at the site that the -905 error is issued except for a statement that references a remote table with a three part name, then the IFCID 173 can be issued at both the server and the requester.

This new IFCID 173 trace record is contained in trace type STAT CLASS(4) and PERFM CLASS(3).

This enhancement is introduced by PTFs UQ90755 (DB2 V7) and UQ90756 (DB2 V8) for APAR PQ87848.

IFCID 173 DSECT

Example 4-7 shows the DSECT for the new IFCID 173.

Example 4-7 IFCID 173 DSECT

QW0173	DSECT		
QW0173ID	DS	CL8	%U AUTH ID
QW0173PC	DS	CL18	%U PACKAGE COLLECTION ID
QW0173PK	DS	CL8	%U PACKAGE NAME
QW0173PL	DS	CL8	PLAN NAME
QW0173CN	DS	CL18	%U CURSOR NAME if it exists
QW0173UT	DS	F	(S) Time used so far
QW0173AT	DS	F	(S) User ASUTIME
QW0173SN	DS	H	SECTION NUMBER
	DS	CL2	Reserved
QW0173ST	DS	F	STATEMENT NUMBER
QW0173CS	DS	XL4	cached statement ID, or zero
QW0173DI	DS	CL128	(S) RLF Diagnostic information area
*			
QW0173ID_Off	DS	H	Offset from QW0173 to
*			AUTH ID
*			If QW0173ID truncated
QW0173PC_Off	DS	H	Offset from QW0173 to
*			PACKAGE COLLECTION ID
*			If QW0173PC truncated
QW0173PK_Off	DS	H	Offset from QW0173 to
*			PACKAGE NAME
*			If QW0173PK truncated
QW0173CN_Off	DS	H	Offset from QW0173 to
*			CURSOR NAME if it exist
*			If QW0173CN truncated
*			
QW0173ID_D	Dsect		Use if QW0173ID_Off^=0
QW0173ID_Len	DS	H	Length of the following field
QW0173ID_Var	DS	OCL128	%U AUTH ID
*			
QW0173PC_D	Dsect		Use if QW0173PC_Off^=0
QW0173PC_Len	DS	H	Length of the following field
QW0173PC_Var	DS	OCL128	%U PACKAGE COLLECTION ID
*			
QW0173PK_D	Dsect		Use if QW0173PK_Off^=0
QW0173PK_Len	DS	H	Length of the following field
QW0173PK_Var	DS	OCL128	%U PACKAGE NAME
*			
QW0173CN_D	Dsect		Use if QW0173CN_Off^=0
QW0173CN_Len	DS	H	Length of the following field
QW0173CN_Var	DS	OCL128	%U CURSOR NAME if it exist

4.12.6 Performance trace enhancements

In this section we discuss enhancements to IFCIDs that are related to DB2 performance trace classes.

Full SQL statement text trace record

DB2 V8 provides a new IFCID 350, which is both similar and in addition to IFCID 63, which traces the entire SQL statement. Previous versions of DB2 limited the SQL statement text to a maximum of 5,000 bytes (the maximum length of a trace field).

Since there is a 5,000 byte limit on a single data item being traced, a “repeating group” of 5,000 byte items are placed in each IFCID 350 record up to a maximum of 6. If that is not enough, multiple IFCID 350 records are produced. IFCID 350 is written under the same circumstances that trigger the writing of IFCID 63. PREPARE attributes are also traced.

Enhanced trace recording for PREPARE attribute

Since DB2 V7, you can change the attributes of a cursor or SELECT statement on the PREPARE statement. To do so, you use the ATTRIBUTES clause on the PREPARE. For example, it allows you to change a cursor’s isolation level by providing a new attribute string. However, in V7 the attribute string is not always present in IFCID records, which makes problem determination and performance analysis more complex.

DB2 V8 adds the attribute string to the following trace records:

- ▶ IFCID 63: (which contains the first 5,000 bytes of the SQL statement text) now also contains the attribute string on a short prepare.
- ▶ IFCID 124: (which is used to obtain information about the currently executing SQL statement through a READS request) now also contains the attributes string.
- ▶ IFCID 317: (which is used to retrieve the SQL statement string of a statement in the dynamic statement cache) now also contains the attributes string.

Add cached statement ID to IFCID 124

IFCID 124 provides information about the currently executing SQL statement through the READS interface. DB2 V8 enhances IFCID 124 to also include the dynamic cached SQL statement ID. The 4 byte field QW0124ST was added to contain the cached dynamic SQL statement identifier.

This enhancement works together with another enhancement in DB2 V8 that allows you to Explain SQL statements from the dynamic statement cache. You can retrieve the information about the currently executing SQL statement via IFCID 124. That now contains the SQL statement ID of the statement in the dynamic statement cache, and then explains the SQL statement from the dynamic statement cache using that statement ID.

For more information on how to explain a statement from the dynamic statement cache, see section 4.20, “EXPLAIN STMTCACHE” in the book *DB2 UDB for z/OS Version 8: Everything You Ever Wanted to Know, ... and More*, SG24-6079.

Agent level workfile tracing

DB2 V8 introduces a new IFCID 342 to record workfile and TEMP database usage at an application level. This allows you to take corrective action, such as adding space or cancelling a runaway query that is using a lot of space in the databases and preventing other queries from running successfully even they only need very little space.

When active, IFCID 342 is written whenever space is allocated or deallocated in the workfile or TEMP database. It gives information about the total amount of space being used by the agent currently and the maximum space that the agent may have used at any given point in time in the workfile or TEMP database. The trace also records the total amount of space being used for indexes on Declared Temporary Tables.

This record is activated by:

-START TRACE(PERFM) CLASS(32) IFCID(342)

Monitors, for example, can trigger exceptions on this new IFCID to catch runaway queries before they monopolize the system and cause other applications to fail, because the workfile database and TEMP database are shared amongst all DB2 users.

Obtaining secondary authorization ID information

This enhancement allows applications to obtain the authorization identifiers that are associated with an application (such as QMF) that is executing. The primary authorization ID, the CURRENT SQLID, and secondary authorization IDs can be retrieved with a synchronous read of IFCID 234. This enhancement is also available in DB2 V6 and V7 via APAR PQ47973, PTF UQ57178 for V7 and PTF UQ57177 for V8.

Note however that IFCID 234 is always started implicitly by DB2 at startup time. It is not contained in any DB2 trace class and cannot be specified as an IFCID value on the -START TRACE command.

Auditing for multilevel security

Audit record (IFCID 142) is produced if a table with a security label is created, altered or dropped.

The user must be identified to Security Server with a valid SECLABEL. If not, an authorization error and audit record are produced (IFCID 140).

4.12.7 Miscellaneous trace enhancements

In this section we discuss other enhancements to monitoring, such as new DB2 messages.

Monitor system checkpoints and log offload activity

DB2 V8 provides new messages, as well as an IFCID 335 record, to help you to identify problems with log offloading faster. An IFCID 335 record is produced if the statistics class 3 trace is active. See “Monitoring system checkpoints and log offload activity” in *DB2 UDB for z/OS Version 8: Everything You Ever Wanted to Know, ... and More*, SG24-6079, for more information about the console messages that are produced.

This enhancement helps you identify potential problems with DB2 logging before they become real problems which may cause application outages. For example, when DB2 stops taking system checkpoints or gets stuck during its log offload activity, the system grinds to a halt.

4.13 Miscellaneous items

In this section we introduce miscellaneous enhancements which also impact DB2 subsystem availability, performance and scalability.

4.13.1 DB2 logging enhancements

As the throughput of a single DB2 subsystem increases, the amount of data DB2 must log also increases. Keeping the DB2 logging data available and online is very important, as it dramatically speeds up DB2 recovery and maximizes data availability.

DB2 V8 in new-function mode increases the number of active log data sets from 31 to 93, allowing DB2 to keep more log data online and therefore available for any recovery.

In addition, DB2 V8 in new-function mode increases the number of archive log data sets DB2 remembers from 1,000 to 10,000. This increases the amount of log data DB2 is able to access and use in recovery, therefore satisfying the larger window often required by Service Level Agreements (SLA).

4.13.2 Up to 65,041 open data sets

DB2 V8 increases the number of open data sets from 32k to 65,041. This enhancement is available starting with z/OS V1.5 base code.

Open data sets require storage for MVS control blocks below the 16 MB line. This has been the cause of virtual storage constraint problems below the 16 MB line in the past. Dynamic allocations can request these control blocks to be allocated above the 16 MB line. With z/OS V1.5, media manager, the interface that DB2 uses, can fully exploit this feature. This reduces the amount of storage that DB2 requires below the 16 MB line.

This enhancement does not only provide virtual storage relief. It also removes a potential scalability problem with growing systems and growing machine capacity.

In addition, DB2 V8 no longer uses system-generated DDNAMEs, as the current limit there is also 32k. DB2 generates its own DDNAMEs for dynamic allocation. This way DB2 can have better open data set performance as well, and, can avoid the problem of the task that generates the system DDNAMEs becoming a bottleneck for parallel data set open.

When DSMAX is reached, DB2 currently attempts to close up to 3% of the open data sets. However, 3% of 65,041 is still a large number, and this burst of data set close activity could be disruptive. Therefore DB2 now closes MIN(3%,300) data sets when DSMAX is reached.

PFT UQ96862 for APAR PQ96189 is related to the number of open data sets.

4.13.3 SMF type 89 record collection enhancements

SMF type 89-1 interval records provide IBM with the required "Software Usage Report" for reporting on zSeries Software License Requirements.

Prior to V8, DB2 automatically used detailed tracking of measured usage if SMF type 89 records were activated.

DB2 V8 provides a new DSNZPARM parameter, SMF89, which lets you specify whether DB2 is to do detailed tracking for measured usage pricing. The default value is NO, which means that DB2 does not do detailed measured usage tracking. If the SMF type 89 record is activated, only high-level tracking is recorded in the SMF type 89 record.

If you select YES, DB2 does detailed measured usage tracking if SMF type 89 records are activated. When SMF89 is set to YES, DB2 invokes a z/OS service on every entry or exit into or out of DB2 to ensure accurate tracking. A new SMF type 89 record is cut on each SQL statement.

If you select NO, DB2 CPU is reduced; however, the amount of time spent in DB2 as measured by SMF 89 is increased.

A "typical" CPU overhead for DB2 performing detailed tracking of CPU usage is in the range of 0 to 5%. Since there is a fixed overhead per SQL statement, the overhead tends to be higher for short-running SQL statements and negligible for long-running SQL statements. For

an online transaction with very simple SQL statements each accessing and retrieving one row, the overhead can be about 1% if there is no other transaction running. On the other hand, if there are many concurrent transactions, we have heard of up to 15% overhead. A 20% figure is for a projected maximum number for 32-way complex with many concurrent threads.

In DB2 V8, the overhead is thought to be completely negligible as there is no longer SMF 89 data collection for each SQL statement.

We therefore recommend you select SMF89 YES only if you use measured usage pricing.

4.13.4 Lock avoidance enhancements

We briefly describe the main lock avoidance enhancements.

Overflow lock avoidance

Sometimes when you update a variable length row, DB2 is not able to store the row on the original page. This happens when the size of the row has increased and it no longer fits on its original page. In order to avoid updating all the indexes that point to that row (each index entry contains a RID, which contains the original page number), a pointer record is created on the original page. The pointer record then points to the actual row. If a later update of the row decreases its size, it can be put back into its original page, again without any updates to the index.

You can have variable length rows when:

- ▶ Using variable length columns like VARCHAR and VARGRAPHIC
- ▶ Using DB2 data compression
- ▶ Altering a table to add a column, but you have not performed a REORG to materialize all the rows to have the new column
- ▶ Using the new V8 online schema enhancements to enlarge the a columns data type, but you have not run REORG to materialize all the rows to the latest format

The good thing about using the overflow pointer is that we avoid updating the indexes. The disadvantage is that we potentially double the number of data I/Os and getpage requests, and increase the number of lock and unlock requests. This can obviously impact performance.

You can check whether or not overflow records exist by looking at the FARINDREF and NEARINDREF information in SYSIBM.SYSTABLEPART. This information is updated by running RUNSTATS and provides information about the number of rows that have been relocated far from (> 16 pages) or near (< 16 pages) their “home” page. We recommend you run a REORG when (FARINDREF +NEARINDREF) exceed 5 to 10%. To reduce the amount of overflow rows, and hence the number of double I/Os, getpages, and lock and unlock requests, you can use a higher PCTFREE value (FREEPAGE does not help in this case).

In V7, there is no lock avoidance on both the pointer record and the overflow record itself. With DB2 Version 8, only the pointer is locked.

Table 4-15 Lock avoidance of overflow record

Isolation Level Cursor Stability with CURRENTDATA NO or YES	V7	V8
Lock and Unlock required for the pointer record	YES	YES
Lock and Unlock required for the overflow record	YES	NO

In addition, with V8, PQ89070 implements lock avoidance for a singleton SELECT statement with ISOLATION(CS) and CURRENTDATA(YES). Semantically, a singleton select with ISOLATION(CS) can avoid getting an S page or row lock regardless of the setting of the CURRENTDATA parameter, since the cursor is closed after the SELECT is processed.

Lock avoidance for embedded SELECT or SELECT INTO

Semantically, a SELECT INTO with ISOLATION(CS) should avoid getting an S page or row lock regardless of the setting of the CURRENTDATA(CD) option since the cursor is closed after the SELECT is processed. Currently, with ISO(CS) and CD(NO) we get LOCK AVOIDANCE but not with ISO(CS) and CD(YES).

If DB2 can determine that the data that it is reading has already been committed, it can avoid taking the lock altogether. For rows that do not satisfy the search condition, this lock avoidance is possible with CURRENTDATA(YES) or CURRENTDATA(NO). For a row that satisfies the search condition on such a SELECT, lock avoidance is possible with CURRENTDATA(YES) or CURRENTDATA(NO). For other rows that satisfy the search condition, lock avoidance is possible only when you use the option CURRENTDATA(NO).

In a test case, executing a simple select statement in a loop (100k times), we have seen up to a 25% better elapsed time in V8 when compared to the run without lock avoidance. When compared to V7, V8 is 7% better with lock avoidance.

Archived

Availability and capacity enhancements

Continuous availability is a major business requirement. New releases of DB2 keep introducing enhancements to meet this need.

DB2 V8 introduces new and enhanced function in this area.

- ▶ **System level point-in-time recovery:** V8 provides enhanced backup and recovery capabilities at the DB2 subsystem level or data sharing group level. The new BACKUP SYSTEM and RESTORE SYSTEM utilities rely on the new Fast Replication services of DFSMS in z/OS V1R5.
- ▶ **Automated space management:** This feature potentially enhances availability for many DB2 installations. DB2 will adjust the size of the secondary extent, when extending the data set, in such a way that the maximum allowable data set size is reached before the maximum number of extents, avoiding out of space conditions.
- ▶ **Online schema:** Online schema evolution allows for table space, table and index attribute changes while minimizing impact on availability.
- ▶ **Volatile table:** Statistics for tables which can vary greatly in size can deviate completely at runtime from values used at bind time. Performance can benefit from favoring index access for these types of tables.
- ▶ **Index changes:** Data-partitioned indexes, partitioning without a partitioning index, padded and not-padded indexes enhance indexing capabilities.
- ▶ **More availability enhancements:** Other DB2 V8 availability enhancements and also recent enhancements introduced via maintenance are discussed in this section.

5.1 System level point-in-time recovery

Prior to DB2 V8 you could make an external copy of your production system by delimiting the back up activity with the `-SET LOG SUSPEND` and `-SET LOG RESUME` commands. DB2's update activity and logging, while you were making the external copy of your DB2 system, was suspended.

DB2 V8 introduces the new `BACKUP SYSTEM` and `RESTORE SYSTEM` utilities. This provides an easier and less disruptive way to make fast volume-level backups of an entire DB2 subsystem or data sharing group with minimal disruption, and recover a subsystem or data sharing group to any point-in-time, regardless of whether or not you have uncommitted units of work. At DB2 restart, the uncommitted URs are resolved during the forward and backward recovery phase.

These new utilities rely on the new Fast Replication in DFSMS 1.5. DFSMS provides a volume-level fast replication function that allows you to create a backup that is managed by DFSMSHsm with minimal application outage. DFSMSHsm recovers a Fast Replication backup version from the volume or copy pool level, but not at the data set level. This function enables the backup and recovery of a large set of volumes to occur within a small time frame. The term Fast Replication refers to the FlashCopy function supported by IBM TotalStorage® Enterprise Storage Server® (ESS) disk and the SnapShot function supported by IBM RAMAC® Virtual Array (RVA) disk.

- ▶ The `BACKUP SYSTEM` utility provides fast volume-level copies (versions) of DB2 databases and logs.
- ▶ The `RESTORE SYSTEM` utility recovers a DB2 system to an arbitrary point-in-time. `RESTORE SYSTEM` automatically handles any creates, drops, and `LOG NO` events that might have occurred between the backup and the recovery point-in-time.

Prerequisites for this feature

In order to use the `BACKUP SYSTEM` and `RESTORE SYSTEM` utilities, all data sets that you want to back up and recover must be SMS-managed data sets. Additionally, you must have the following requirements:

- ▶ z/OS V1R5, DFSMSHsm, DFSMSdss™.
- ▶ Disk control units that support ESS FlashCopy API (we recommend FlashCopy V2).
- ▶ SMS copy pools that are defined by using the DB2 naming convention.
- ▶ Defined SMS copy pool backup storage groups.
- ▶ DB2 has to be running in new-function mode.

`RESTORE SYSTEM LOGONLY` option has no dependency on z/OS 1.5.

A copy pool is a set of SMS-managed storage groups that can be backed up and restored with a single command.

A copy pool backup is a new storage group type that is used as a container for copy pool volume copies. Each volume in the copy pool storage group needs to have a corresponding volume in the copy pool backup storage group. For each copy pool volume as many backup volumes as backup versions defined for the copy pool should be available in the copy pool backup.

Each DB2 system has up to two copy pools, one for databases and one for logs. The name of each copy pool that is to be used with DB2 must use the following naming conventions:

DSN\$locn-name\$cp-type

The variables that are used in this naming convention have the following meanings:

- ▶ DSN: Unique DB2 product identifier.
- ▶ \$: Delimiter.
- ▶ locn-name: DB2 location name.
- ▶ cp-type: copy pool type. Use DB for the database copy pool and LG for the log copy pool.

You have to set up both copy pools properly in order to successfully implement system level point-in-time recovery:

The “log” copy pool should be set up to contain the volumes that contain the BSDS data sets, the active logs, and their associated ICF catalogs.

The “database” copy pool should be set up to contain all the volumes that contain the databases and the associated ICF catalogs.

Important: Make sure that the **ICF catalog** for the data is on an other volume than the ICF catalog for the logs.

Three new DFSMSHsm commands are introduced to support this new function, and other commands are changed to provide new information related to the DFSMSHsm Fast Replication function:

- ▶ FRBACKUP: Create a fast replication backup version for each volume in a specified copy pool.
- ▶ FRRECOV: Use fast replication to recover a single volume or a pool of volumes from the managed backup versions
- ▶ FRDELETE: Delete one or more unneeded fast replication backup versions.

LIST and QUERY commands are modified to aid monitoring of the fast replication backup versions.

Prepare for fast replication

After establishing the copy pools, the storage administrator issues FRBACKUP PREPARE against each copy pool. The PREPARE allows you to validate your fast replication environment for DFSMSHsm. DFSMSHsm preassigns the required number of target volumes to each source volume in each storage group in the specified copy pool. When you specify the PREPARE keyword, DFSMSHsm verifies that there is a sufficient number of eligible target volumes in the specified copy pool backup storage group.

For instance, if you specified in the copy pool definition that DFSMSHsm should keep two backup versions, then DFSMSHsm preassigns two target volumes to each source volume in the copy pool.

You should use the PREPARE keyword whenever there is a change in the fast replication environment. Changing any of the following definitions will change the fast replication environment:

- ▶ The volumes in the copy pool or copy pool backup storage groups are changed or removed.
- ▶ The number of backup versions for DFSMSHsm to maintain is changed.
- ▶ The storage groups defined to the copy pool have changed.

Important: It is recommended to run the PREPARE after each copy pool assignment since it removes the time for target volume selection from the BACKUP execution window. Otherwise the prepare phase will be executed implicitly when the backup starts and will elongate its execution.

BACKUP SYSTEM

During backup DB2 records the “recovery base log point” (RBLP) in the header page of DBD01. The RBLP is identified as the most recent system checkpoint prior to a backup log point, and the point at which DB2 starts scanning logs during a RESTORE SYSTEM recovery operation. DB2 updates its BSDS (boot strap data set) with backup version information and can keep track of up to 50 backup versions. In the case of data sharing the submitting member records the backup version in its BSDS and also in the SCA. This information is also recorded in DFSMSHsm control data sets.

Note that in DB2 V8, the **SET LOG SUSPEND** command also writes the RBLP to the DBD01 header page.

Concurrency and compatibility

BACKUP SYSTEM can run concurrently with any other utility; however, it must wait for the following DB2 events to complete before the copy can begin:

- ▶ Extending data sets
- ▶ Writing 32 KB pages for page sets with a VSAM CISZE that is different from the DB2 page size
- ▶ Writing close page set control log records
- ▶ Creating data sets (for table spaces, indexes, and so forth)
- ▶ Deleting data sets (for dropping tables spaces, indexes, and so forth)
- ▶ Renaming data sets (for online reorganizing of table spaces, indexes, and so forth during the SWITCH phase)

Only one BACKUP SYSTEM job can be running at one time.

The log write latch is not obtained by the BACKUP SYSTEM utility, as is done by the SET LOG SUSPEND command. Therefore, using the BACKUP SYSTEM utility should be less disruptive in most cases. There is no need to suspend logging when using the BACKUP SYSTEM utility, since DB2 is now in control of taking the volume copies, where you are in charge of initiating copying the data in the case of using -SET LOG SUSPEND.

RESTORE SYSTEM

Use the RESTORE SYSTEM utility only when you want to recover a subsystem or data sharing group to an arbitrary point-in-time. The utility restores only the database copy pool of a data only or full system backup, and then applies logs until it reaches a point in the log equal to the log truncation point specified in a point-in-time conditional restart control record (SYSPITR CRCR) created with DSNJU003. You cannot explicitly name the backup to use for recovery. That is implicitly determined by the log truncation point used to create the SYSPITR CRCR. Similar to the Recovery utility, Restore can optionally use the Fast Log Apply (FLA) option. As restore will be the only task running in DB2, we strongly advise you to enable Fast Log Apply in order to accelerate the recovery process.

RESTORE SYSTEM LOGONLY can run without a BACKUP SYSTEM backup and can run with z/OS 1.3. By using this option, you indicate that the database volumes have already been restored using DFSMSHsm or other means, and the restore phase will be skipped. A backup made in a SET LOG SUSPEND/RESUME cycle is required because SET LOG SUSPEND updates the RBLP in the DBD01 header page.

DB2 V8 Fast Log Apply behavior

Fast Log Apply was introduced in DB2 V6. It is activated by specifying a non-zero value for the DSNZPARM parameter LOGAPSTG. The acceptable values are 0 to 100 MB with 100

MB as default. The FLA design consists of one log read task per recover job and one or more log apply tasks employing the use of multitasking whenever possible. It takes advantage of list prefetch and nearly eliminates duplicate synchronous read operations for the same page. It achieves these efficiencies by sorting groups of log records together that pertain to the same table space before applying those changes; it then applies those records in parallel using several log apply tasks, up to 99.

When LOGAPSTG=0 is specified, DB2 restart will still use a minimum buffer size of 100 MB for FLA during the forward log recovery phase. RESTORE SYSTEM and RECOVER will not use FLA.

When LOGAPSTG> 0 is specified then:

- ▶ DB2 will use that value for the FLA buffer at restart time with a default of 100 MB.
- ▶ RESTORE SYSTEM will use FLA during the log apply phase. RESTORE, with APAR PQ95164, will always try to allocate a FLA buffer of 500 MB. If it cannot get the storage, it retries with 250 MB, 125 MB, and so on.
- ▶ Each concurrent recover utility will use a buffer of 10 MB out of the storage defined by LOGAPSTG. If no more storage is available, the next concurrent recover utility will execute without FLA. This means that the maximum value of 100 MB for LOGAPSTG allows for 10 concurrent recover utilities executing with FLA, the 11th recover will execute without FLA.

For a detailed description on Point-in-Time Recovery, and other disaster recovery functions, see *Disaster Recovery with DB2 UDB for z/OS*, SG24-6370.

5.1.1 Performance

In this section we describe the System Level point-in-time recovery performance measurement.

Performance measurement description

For this measurement we used:

- ▶ z/OS V1R5 with DFSMS, DFSMSdss, and DFSMSHsm V1R5
- ▶ DB2 V8
- ▶ zSeries 2084 Model 316
- ▶ ESS 2105-800, FlashCopy V2
- ▶ Copy pool:
 - Database copy pool either 67 (27% full) or 195 volumes (10% full)
 - Log copy pool 13 volumes
- ▶ Copy pool backup:
 - 211 volumes for the database copy pool
 - Up to 3 versions available to the 67-volume copy pool
 - One version only for the 195-volume copy pool
 - 13 volumes for the log copy pool
 - 1 version for the 13-volume log copy pool

Performance measurement results

In this section we present the System Level point-in-time recovery performance measurement results.

FRBACKUP PREPARE measurements

See Figure 5-1 for the measurement results of FRBACKUP PREPARE. The results show a non-linear behavior between elapsed time to prepare and the number of volumes in the pools.

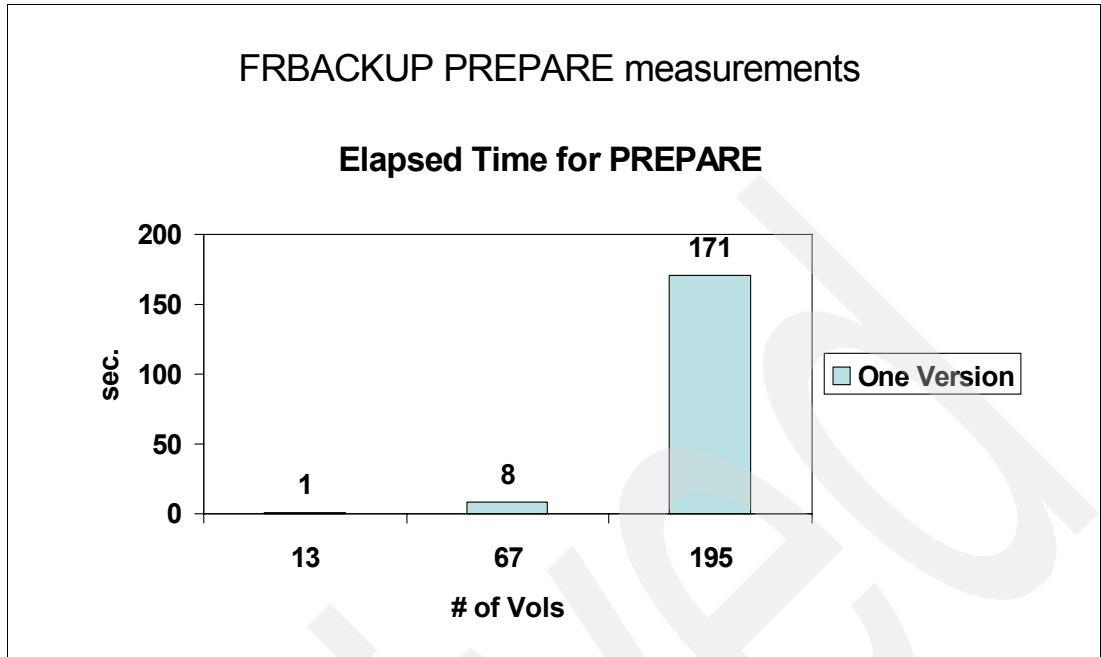


Figure 5-1 FRBACKUP PREPARE elapsed time vs. #volumes in the copy pool

Since the time to prepare the replication environment will increase with the number of volumes in the copy pool, this action should be executed before initiating the real backup.

Figure 5-2 shows the measurement results increasing the number of versions.

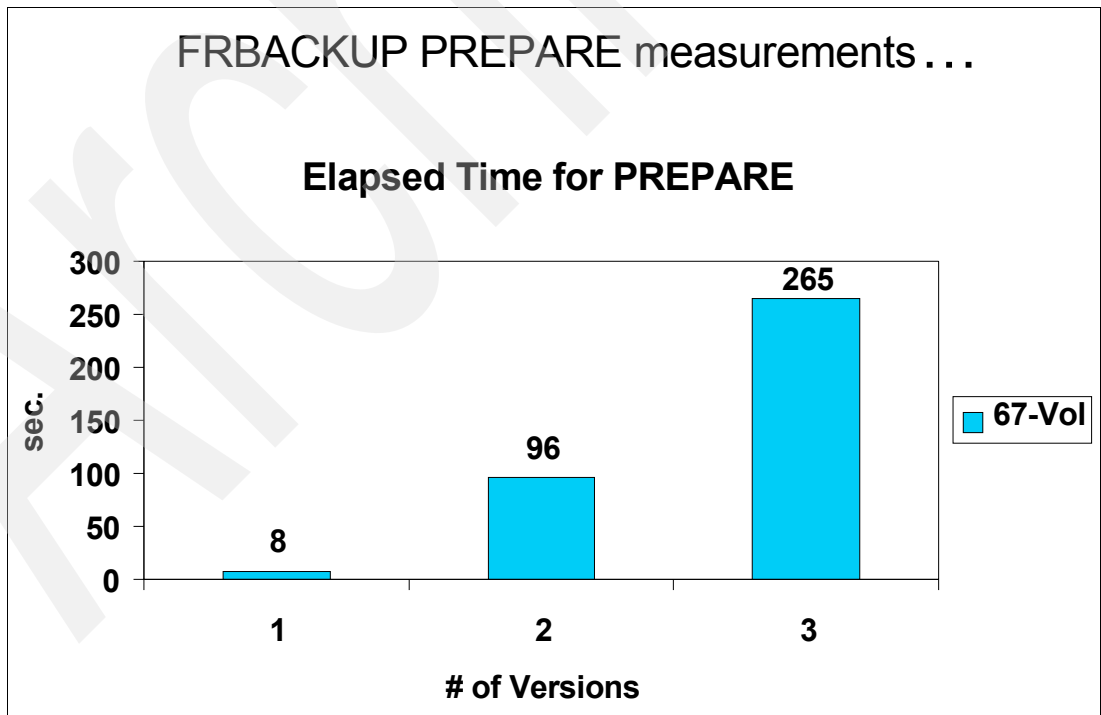


Figure 5-2 FRBACKUP PREPARE elapsed time vs. #versions for 67 volume copy pool

Increasing the number of versions means increasing the number of target volumes, this significantly increases the time to prepare the replication environment. Again we recommend

to execute the prepare outside the actual backup cycle since the time to prepare is a multiple of the time to take the actual backup.

BACKUP SYSTEM measurements

The BACKUP SYSTEM DATA ONLY utility invokes the command

`'FRBACKUP COPYPOOL(DSN$location-name$DB) EXECUTE.`

See Figure 5-3 and for the measurement results.

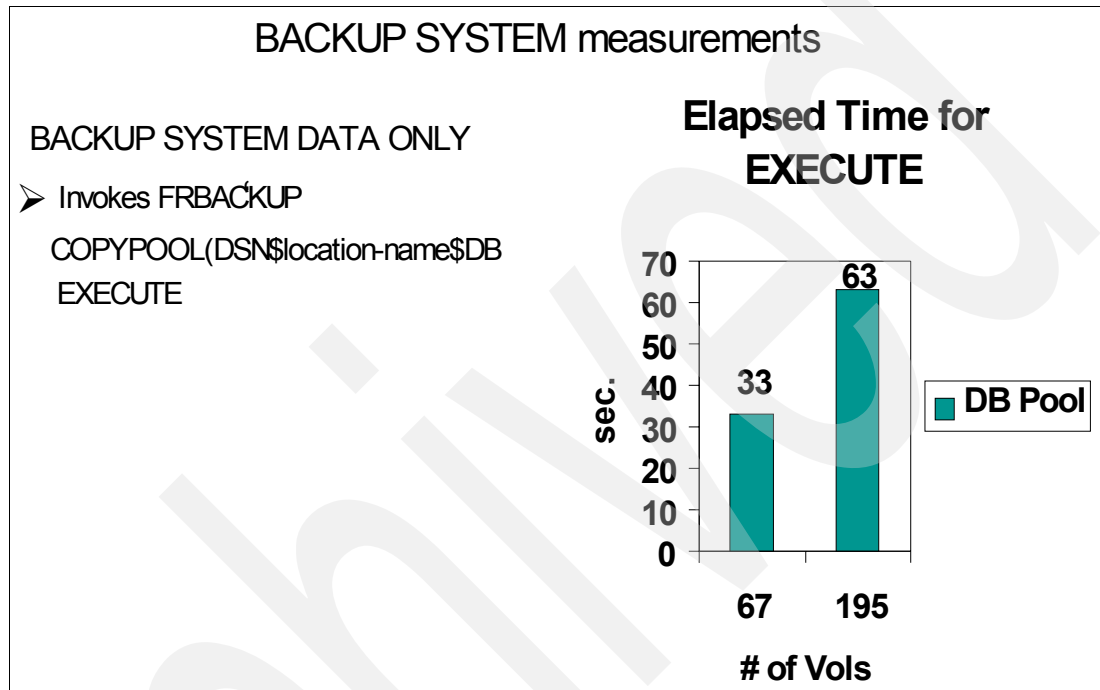


Figure 5-3 BACKUP SYSTEM DATA ONLY measurement

The BACKUP SYSTEM FULL utility invokes the commands

`'FRBACKUP COPYPOOL(DSN$location-name$DB) EXECUTE.` followed by
`'FRBACKUP COPYPOOL(DSN$location-name$LG) EXECUTE'`

See Figure 5-4 for the measurement results.

BACKUP SYSTEM measurements...

- BACKUP SYSTEM FULL
 - Invokes 'FRBACKUP COPYPOOL(DSN\$location-name\$DB) EXECUTE'
 - Then invokes 'FRBACKUP COPYPOOL(DSN\$location-name\$LG) EXECUTE'

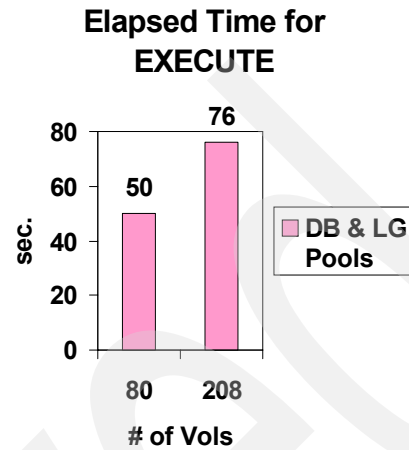


Figure 5-4 BACKUP SYSTEM FULL measurement

If FlashCopy is the fast replication technique used, then DFSMShsm returns control to the system after a FlashCopy relationship for each volume has been established. However, the background copies for those volumes last longer, depending on the number of background copies being processed and the amount of other higher priority activity that exists in the storage subsystem.

Another interesting measurement case is the concurrent execution of BACKUP SYSTEM DATA ONLY with the IRWW non-data sharing workload. See Table 5-1 for the results.

Table 5-1 Concurrent execution of BACKUP SYSTEM and IRWW workload

	IRWW non-data sharing	IRWW + BACKUP SYSTEM data only
ETR (commit/sec.)	386.42	386.38
CPU (%)	66.19	72.96
Class 3 DB I/O related suspend time (msec.)	14.40	19.17

Running the BACKUP SYSTEM utility concurrent with the IRWW workload shows no noticeable performance degradation. The external throughput rate (ETR) remains unchanged. CPU slightly increased but the DB2 transaction CPU was not affected. The database related I/O suspend time increases but this was not reflected in the ETR due to the think time.

RESTORE SYSTEM measurements

The RESTORE SYSTEM utility invokes the command

'FRRECOV COPYPOOL(DSN\$location-name\$DB)'

See Figure 5-5 for the measurement results.

RESTORE SYSTEM measurements

- **RESTORE SYSTEM**
 - Invokes 'FRRECOV
COPYPOOL(DSN\$location-name\$DB)'
 - Followed by LOGAPPLY phase
- **RESTORE SYSTEM
LOGONLY**
 - Enters LOGAPPLY phase
directly

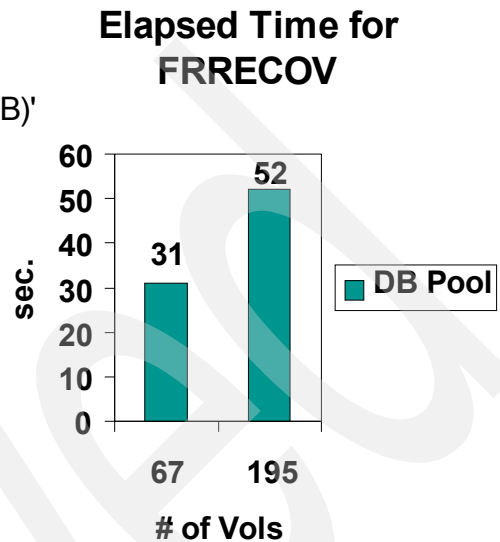


Figure 5-5 *RESTORE SYSTEM measurement*

Remember that **RESTORE SYSTEM** only restores the database copy pool. The logs are applied after that restore.

Make sure all of the background copies have completed from an invocation of the **BACKUP SYSTEM** utility before starting the restore utility.

The log apply performance can be improved by activating fast log apply (FLA).

Table 5-2 presents the log apply phase measurement results.

Table 5-2 *LOGAPPLY phase measurement*

IRWW non-data sharing	Without FLA	With FLA (100 MB)	With FLA (500 MB)
Log records processed (million)	3.800	3.642	3.772
Elapsed time (sec.)	1,320	325	262
Rate (log records/sec.)	2,900	11,200	14,400

A FLA buffer size of 100 MB resulted in a 4 times improvement in elapsed time. With APAR PQ95164, the DSNZPARM parameter LOGAPSTG will not determine the FLA buffer size for the **RESTORE** utility, only if it is used or not. DB2 will first try to allocate a 500 MB FLA buffer; if not successful DB2 will reduce the memory request to 250 MB, then 125 MB. During restore the **RESTORE SYSTEM** utility is the only process running and there should be no problem obtaining 500 MB of storage for the FLA feature at this time. The 500 MB FLA buffer further decreased the elapsed time by 28% compared to a 100 MB FLA buffer. Compared to the test case without FLA a 500 MB buffer resulted in nearly 5 times faster log apply phase.

List prefetch is the major contributor to FLA improvements, so the more random the logging had been, the more advantage FLA will bring.

5.1.2 Conclusion

The BACKUP SYSTEM utility provides an easier and minimally disruptive way to backup an entire DB2 subsystem or data sharing group. Executing the BACKUP SYSTEM utility concurrent with your workload should not impact the performance.

5.1.3 Recommendations

We recommend that you execute the FRBACKUP prepare command outside the actual backup cycle in order to minimize the impact on the backup elapsed time.

Fast log apply supports the system level point-in-time recovery and provides good performance. We highly recommend to activate fast log apply (LOGAPSTG DSNZPARM parameter), your system will benefit from it, not only the RESTORE SYSTEM utility but also during all RECOVER utility executions and at DB2 restart.

Individual recovery actions at the table space or index space level cannot use a BACKUP SYSTEM backup which implies these recoveries need a data set level copy. This data set level copy can be a DB2 image copy or a non DB2 backup (log only recovery), for example, a data set level FlashCopy backup (FlashCopy V2 required).

5.2 Automated space management

In general, a large percentage of your data sets are today managed with DFSMS storage pools, reducing the workload and the interaction of your DB2 database administrators and storage administrators. Only the most critical data, as defined with service level agreements or as revealed by monitoring, may require special attention.

Using DFSMS, the DB2 administrator gains the following benefits:

- ▶ Simplified data allocation
- ▶ Improved allocation control
- ▶ Improved performance management
- ▶ Automated disk space management
- ▶ Improved data availability management
- ▶ Simplified data movement

An important function, automated data set space management, was still lacking in this list. DB2 provides this functionality in Version 8.

The idea is that DB2 will adjust the *size of the secondary extent*, when extending the data set, in such a way that the maximum allowable data set size is reached, before the maximum number of extents for that data set is reached, therefore avoiding out of space conditions. Remember that today space allocation supports a maximum number of 255 extents per component of a VSAM data set for multivolume data sets and a maximum of 123 extents for a single volume allocation. A striped VSAM data set can have up to 255 extents per stripe. Also new in DFSMS for z/OS 1.5 is that the system consolidates adjacent extents for a VSAM data set when extending the data set on the same volume.

Users of DB2 receive out of extent errors during data set extend processing when the maximum number of extents is reached, thus preventing a DB2 managed page set from reaching maximum data set size. Although DB2 issues warning messages (DSNP001I) of impending space shortage, user procedures in many cases cannot react quickly enough to spot the problem, use SQL ALTER to modify PRIQTY and SECQTY values, and then schedule a REORG. This is a significant problem today for large users of DB2, particularly for

users of ERP and CRM vendor solutions and ported applications with not well known growth, and can lead to application outages.

Default PRIQTY and SECQTY values and often user provided PRIQTY and SECQTY values are too small, resulting in track allocation rather than cylinder allocation. As a result the performance of SQL SELECT and INSERT processing can be penalized by 2x or more.

The first objective of this enhancement is to avoid the situation where a DB2 managed page set reaches the VSAM maximum extent limit of 255 before it can reach the maximum data set size which may be less than 1 GB or 1 GB, 2 GB, 4 GB, 8 GB, 16 GB, 32 GB or 64 GB.

The second objective of the enhancement is to remove the need for a user to specify the secondary quantity for a DB2 managed page set, or even both the primary and secondary quantities, either when creating it initially or by using an ALTER statement later. The overall goal is to provide a solution which would benefit all users of DB2 in terms of improved DBA productivity and preventing application outages.

DB2 V8 adds a new DSNZPARM parameter called MGEXTSZ (with a global scope) which enables the use of a “sliding secondary space allocation” for DB2 managed data sets (STOGROUP defined). The values are: NO and YES. The default value for MGEXTSZ is NO.

The enhancement will help to reduce the number of out of space conditions, eliminate need for PRIQTY and SECQTY specification on SQL CREATE, improve user productivity, and avoid the performance penalty associated with small extent sizes.

This patented solution should benefit all users of DB2, not just for ERP and CRM solutions, ported and new applications. It delivers autonomic selection of data set extent sizes with a goal of preventing out of extent errors before reaching maximum data set size.

An increasing secondary quantity size up to 127 extents is allocated and a constant number of 559 cylinders for 32 GB and 64 GB data sets is used thereafter. Figure 5-6 shows the sliding scale mechanism for 64 GB data sets with 255 extents. It assumes an initial extent of 1 cylinder.

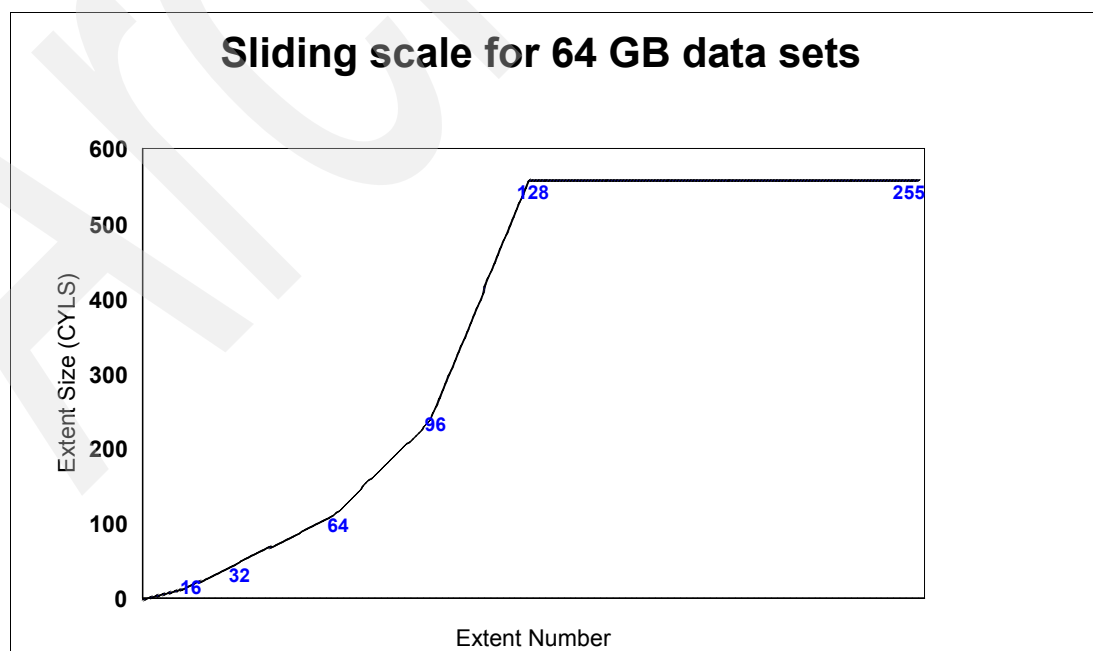


Figure 5-6 Sliding scale for 64 GB data sets

Figure 5-7 shows the sliding scale mechanism for a 16 GB data set with a maximum of 246 extents. It assumes an initial extent of 1 cylinder.

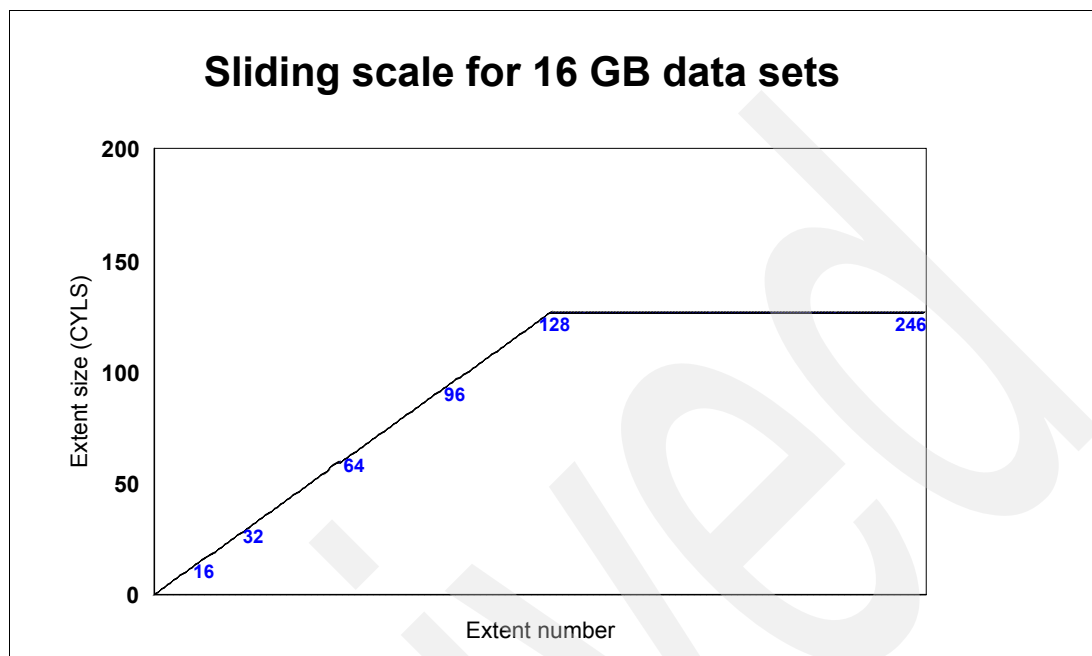


Figure 5-7 Sliding scale for 16 GB data set

An increasing secondary quantity size up to 127 extents is allocated and a constant number of 127 cylinders for data sets less than 1 GB or 1, 2, 4, 8 and 16 GB is used for the secondary allocation thereafter.

This approach of sliding the secondary quantity minimizes the potential for wasted space by increasing the extents size slowly at first, and it also avoids very large secondary allocations from extents 128-255 which will most likely cause fragmentation where multiple extents have to be used to satisfy a data set extension. The solution will address new data sets which will be allocated and also existing data sets requiring additional extents.

A new default PRIQTY value will be applied if not specified by the user and recorded in the Catalog. The actual value applied by DB2 for default PRIQTY will be determined by the applicable TSQTY or IXQTY DSNZPARMs which were introduced with DB2 V7. DSNZPARMs TSQTY and IXQTY will now have global scope. TSQTY will apply to non-LOB table spaces. For LOB table spaces a 10x multiplier will be applied to TSQTY to provide the default value for PRIQTY. IXQTY will apply to indexes. DSNZPARMs TSQTY and IXQTY will continue to have a default value of 0 (zero), but this value will indicate a new default value of 720 KB (1 cylinder) is to be applied. If TSQTY is set to 0 then 1 cylinder will be the default PRIQTY space allocation for non-LOB table spaces and 10 cylinders will be the default PRIQTY space allocation for LOB table spaces. If IXQTY is set to 0 then 1 cylinder will be the default PRIQTY space allocation for indexes.

The user can provide override values for TSQTY and IXQTY DSNZPARMs to avoid wasting excessive DASD space. For example on a development subsystem, TSQTY and IXQTY may be set to 48KB for track allocation. The use of the default for PRIQTY will be recorded in the associated PQTY column as -1 in the SYSTABLEPART or SYSINDEXPART catalog table.

DB2 will always honor the PRIQTY value specified by the user and recorded in the associated PQTY column in SYSTABLEPART or SYSINDEXPART catalog table.

If MGEXTSZ is set to YES, DB2 will always honor SECQTY value specified by the user in the Catalog if it is greater than the secondary quantity value calculated by the sliding scale methodology used by the solution. This allows the user to override with a larger SECQTY value to get to maximum data set size quicker.

If no SECQTY value is specified by the user, it will be recorded as -1 in the associated SQTY column in SYSTABLEPART or SYSINDEXPART catalog table, DB2 will calculate 10% of the PRIQTY value specified by the user in the Catalog but will cap by the maximum secondary quantity allocation according to the respective sliding scale.

If you specify SECQTY 0, it will be honored by DB2. This results in SQLCODE -904 and reason code 00D70027 when DB2 tries to extend the data set.

For a striped data set, the proposed extent size will have to be divided by the number of stripes and each stripe can have up to 255 extents.

When extent consolidation is in effect in z/OS V1.5 the secondary space allocation quantity can be smaller than specified or calculated by the sliding scale based on the physical extent number. PTF UQ89517 for APAR PQ88665 corrects this problem by using the logical extent number together with the sliding scale.

5.2.1 Conclusion

No primary nor secondary allocation needs to be specified at create time. DB2 determines the optimal values based on a combination of the DSSIZE, LARGE and PIECESIZE specifications and system parameters MGEXTSZ, TSQTY and IXQTY. The secondary allocation is automatically increased using a sliding scale until 127 extents and a constant size thereafter to avoid exhausting the maximum number of extents per data sets.

The enhancement has the potential to improve SQL SELECT and INSERT performance, more efficient preformat, prefetch, and deferred write processing.

This enhancement changes the PRIQTY default to achieve cylinder allocation, forces tiny SECQTY allocations to use these defaults, and slides secondary quantity allocations upwards.

Any penalty with having multiple extents goes down as the extent size increases. The size of the extents is actually more important than the actual number of extents. Above an extent size of 10 cylinders independent of the number of extents there is no noticeable difference in performance.

Note that no space availability nor disk fragmentation is addressed by this enhancement, and still remain the responsibility of DFSMS and the storage administrator.

5.2.2 Recommendations

We recommend to start with activating automatic space management for secondary extents (MGEXTSZ=YES). Availability will increase by avoiding out of space abends due to maximum number of extents reached and, as I/Os will never span extents, your installation will benefit from better extent sizing when performing

- ▶ Prefetch
- ▶ Deferred write
- ▶ Mass Insert
- ▶ Utility Processing

Maybe you have implemented a space allocation strategy at your site. You do not have to alter your current strategy in order to benefit from the automated secondary quantity space management. Specifying MGEXTSZ=YES triggers DB2 to manage the size of secondary allocations by the sliding scale mechanism. This guarantees that you will never run out of extents and still honors the SECQTY that you specified if greater than the DB2 calculated value.

Note: PTF UK08807 for APAR PK0564 provides a data set preformat quantity adjustment to take into account the larger sizes of secondary extents.

5.3 Online schema

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

Some schema changes could already be done without having to drop and recreate an object, or stopping and starting the object, such as adding a column to a table, renaming a table, altering the primary and secondary quantity of an object, or changing partition boundaries. But many changes to table, table space, and index schemas (DDL) prior to DB2 V8 require that you adhere to the following procedure to implement them:

- Unload the data, extract the DDL and authorizations of the object you are about to change, and all dependent objects, like tables, indexes, views, synonyms, triggers, and so on.
- Drop the object.
- Create the object with the new definition and reestablish authorizations for the object.
- Recreate all dependent objects, such as views and indexes, and so forth, and their authorizations.
- Reload the data.
- Rebind plans and packages.
- Test that all is OK.

Without the use of DB2 administration tools, this is a cumbersome and error prone process. DB2 V8 allows many more schema changes without having to drop and recreate the objects.

5.3.1 Changing attributes

Online schema evolution allows for table space, table and index attribute changes while minimizing impact on availability.

The following schema changes are allowed in DB2 V8:

Alter column data type

Column data types can be changed without losing data availability.

- ▶ Increase column size within a data type
- ▶ Change the data type

After the change:

- ▶ The table space is placed in Advisory Reorg Pending (AREO*) state.

- ▶ The data is accessible but until the table space is reorganized a performance penalty is paid.
- ▶ Plans, packages and cached dynamic statements referring to the changed column are invalidated. If auto-rebind is enabled, the plans and packages referencing the changed table space are automatically rebound during the next access if not manually rebound previously.
- ▶ Frequency statistics for columns are invalidated.
- ▶ The new column type must be large enough to hold the maximum value possible for original column.
- ▶ If indexed, changes for character data type columns result in immediate index availability. This includes columns defined as CHAR, VARCHAR, GRAPHIC, or VARGRAPHIC.
- ▶ If indexed, changes for numeric data type columns are immediate with delayed index availability. This includes columns defined as SMALLINT, INTEGER, DECIMAL or NUMERIC, FLOAT, REAL, or DOUBLE. Availability to the index is delayed until the index is rebuilt.
- ▶ Scope of unavailability for dynamic SQL
 - Deletes are allowed for table rows, even if there are indexes in RBDP.
 - Updates and inserts are allowed for table rows, even if their corresponding non-unique indexes are in RBDP state.
 - Inserting or updating data rows which result in inserting keys into an index that is in RBDP state is disallowed for unique or unique where not null indexes.
 - For queries, DB2 does not choose an index in RBDP for an access path.

Recommendations

When using this feature, make sure that you do not forget to assess which programs need to be changed. Host variables, for example, may need to be extended to cater to the extra length. To minimize any performance degradation, schedule a REORG as soon as possible after ALTER. This reestablishes fast column processing and eliminates the conversion from old row format to the new one when retrieved by the application.

Rebuild any affected indexes and rebind plans and packages. Otherwise DB2 might pick an inefficient access path during automatic rebind because the best suited index is currently not available.

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

Alter index

Several changes can now be implemented via the **ALTER INDEX** statement:

- ▶ Add a column to the end of an index

Adding the column to the table and index in the same UOW will place the index in AREO* state. The index is immediately available.

Adding an existing column to the index places the index in rebuild pending (RBDP) status.
- ▶ Scope of unavailability for dynamic SQL

Deletes are allowed for table rows, even if there are indexes in RBDP.

Updates and inserts are allowed for table rows, even if their corresponding non-unique indexes are in RBDP state.

Inserting or updating data rows which results in inserting keys into an index that is in RBDP state is disallowed for unique or unique where not null indexes.

For dynamically prepared queries, DB2 does not choose an index in RBDP for an access path.

- Alter an index VARCHAR columns to be non padded

Prior to DB2 V8, varying length columns in indexes are padded to the maximum length specified by VARCHAR(length). DB2 V8 provides a new default index option NOT PADDED. Varying length columns are no longer padded to the maximum length in index keys and index-only access becomes available for VARCHAR data.

This enhancement could also reduce storage requirements since only the actual data is stored and the index tree level could be reduced.

The index can be altered between PADDED and NOT PADDED but will be placed in RBDP.

Note that DB2 V6 introduced the DSNZPARM RETVLCFK. This parameter controls the ability of the Optimizer to use index-only access when processing VARCHAR columns that are part of an index. However, when one of the columns in the SELECT list of the query is retrieved from the index when using index-only access, the column is padded to the maximum length, and the actual length of the variable length column is not provided. Therefore, an application must be able to handle these “full-length” variable length columns. DB2 V7 enhances this feature by allowing index-only access against variable length columns in an index, even with RETVLCFK=NO, if no variable length column is present in the SELECT list of the query. DB2 V8 supports true varying-length key columns in an index. Varying-length columns are not padded to their maximum lengths.

Versioning

To support online schema evolution, DB2 has implemented a new architecture, called *versioning*, to track object definitions at different times during its life by using versions.

Altering existing objects may result in a new format for tables, table spaces, or indexes which indicates how the data should be stored and used. Since all the data for an object and its image copies cannot be changed immediately to match the format of the latest version, support for migrating the data over time in some cases is implemented by using versions of tables and indexes. This allows data access, index access, recovery to current, and recovery to a point-in-time while maximizing data availability.

Availability considerations

Today, when an index is in Rebuild Pending (RBDP), the optimizer is unaware of that and can still pick that index during access path selection when you are executing a query.

In Version 8, DB2 will try to avoid using indexes in RBDP for dynamically prepared queries.

DB2 will bypass indexes in Rebuild Pending (index is ignored):

- For all DELETE processing
- For all non-unique indexes for UPDATES and INSERTs
- A unique index in RBDP cannot be avoided for UPDATE or INSERT because DB2 needs the index to be available to be able to enforce uniqueness.

The DB2 optimizer treats RBDP indexes as follows:

- Dynamic PREPARE:
 - Indexes in RBDP are avoided.
- Cached statements:
 - If the statement is cached, the PREPARE is bypassed. But if the index is used by the statement that is cached, it is invalidated by the ALTER.

- Invalidation occurs during ALTER.
- ▶ Static BIND does not avoid an index that is in RBDP:
 - Indexes in RBDP can still be chosen, and will get a resource unavailable condition at execution time.

Dynamic partition management

The enhancements related to partitions are:

- ▶ Add, rotate, and rebalance partitions

Execute the ALTER TABLE ADD PARTITION statement to add a partition to a partitioned table space and you are ready to go. You can start inserting/loading rows into the new partition immediately.

Rotating partitions allows old data to “roll-off” while reusing the partition for new data with the ALTER TABLE ROTATE PARTITION FIRST TO LAST statement. When rotating, you can specify that all the data rows in the oldest (or logically first) partition are to be deleted, and then specify a new table space high boundary (limit key) so that the partition essentially becomes the last logical partition in sequence, ready to hold the data that is added. The emptied partition is available immediately, no REORG is necessary. Logical and physical partition numbering is now different. Refer to the columns DSNUM and LOGICAL_PART in catalog tables SYSTABLEPART and SYSCOPY for the actual physical to logical partition number correspondence. Recover to previous PIT is blocked as SYSCOPY and SYSLGRNX entries are deleted.

Rebalancing partitions is done by means of the REORG TABLESPACE utility. Specifying the REBALANCE option when specifying a range of partitions to be reorganized allows DB2 to set new partition boundaries for those partitions, so that all the rows that participate in the reorganization are evenly distributed across the reorganized partitions. However, if the columns used in defining the partition boundaries have many duplicate values perfect rebalancing may not be possible.

Note that adding and rotating partitions will automatically convert index controlled to table controlled partitioning.

- ▶ Alter partition boundaries

In DB2 V6, the ability to modify limit keys for table partitions was introduced. The enhancement in DB2 V8 introduces the same capability for table-based partitioning with the ALTER TABLE ALTER PARTITION ENDING AT statement. The affected data partitions are placed into Reorg Pending state (REORP) until they have been reorganized.

5.3.2 Performance

In this section we describe the online schema performance measurements.

Performance measurement description

The environment we used in this measurement:

- ▶ z/OS V1R3
- ▶ DB2 for z/OS V8 in NFM
- ▶ IBM z900 Series 2064 4-way processor
- ▶ ESS 2105-800 with FICON channels

Performance measurement results

Alter Column Data Types

In a non-partitioned table with 17 columns and 2 indexes the following alters were executed:

- ▶ CHAR(8) to VARCHAR(8)
- ▶ CHAR(8) to CHAR(10)
- ▶ INTEGER to DECIMAL(10,0)

The CPU time (microsec.) for FETCH and INSERT before ALTER, after ALTER and after REORG are measured. Figure 5-8 shows the results for FETCH.

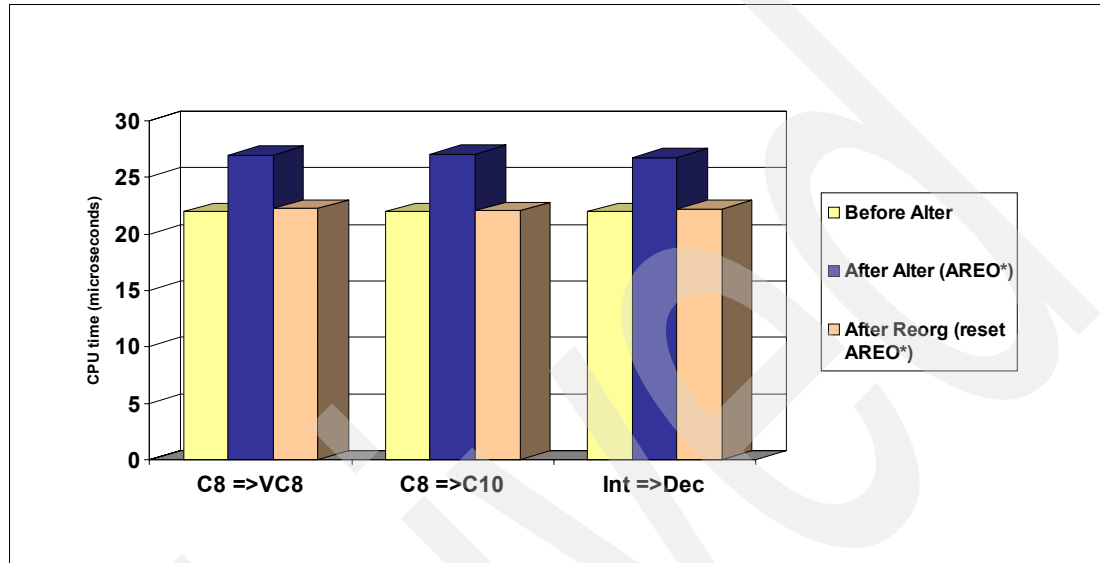


Figure 5-8 CPU time before ALTER, after ALTER and after REORG

A performance degradation of 21% to 23% is measured after ALTER and before REORG. After reorganization the performance overhead remains below 5%.

The performance fluctuation reflects the change of format and that “fast path for column processing” is disabled after ALTER and re-enabled again after reorganization of the table space. Even improved performance after reorganization is possible, depending on the new data type as some data types tend to be less CPU intensive, for example change from decimal to integer.

Fast path for column processing was introduced in DB2 V4. For SELECT, FETCH, UPDATE and INSERT, DB2 will process each column definition for the first three rows. For the fourth and subsequent rows DB2 generates and uses an optimized procedure to access further rows.

Figure 5-9 shows the results for INSERT.

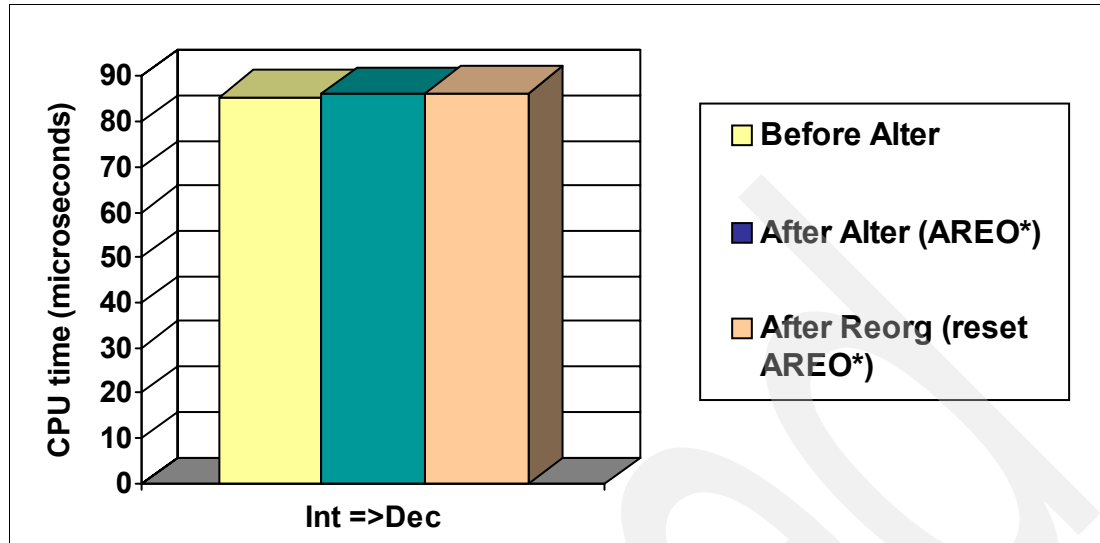


Figure 5-9 INSERT CPU time before ALTER, after ALTER and after REORG

Almost no difference in INSERT performance is noticed after the ALTER in this case. The row is inserted in the new format, no conversion cost is associated with the INSERT and the fast path for column processing is not disabled. The slight increase in CPU time is due to the decimal data type being a more expensive data type in terms of CPU than an integer. The difference in CPU time after ALTER and before REORG compared to after REORG is negligible.

INSERT after ALTER from CHAR to VARCHAR or CHAR(8) to CHAR(10) resulted in 13% increase.

Conclusion

After ALTER table the table space is placed in an AREO* status. The data remains accessible with some performance degradation. On SELECT the data is materialized to the latest format. On INSERT and UPDATE the entire row is saved using the new definition. Reorganization of the table space changes all rows to the latest format and access to the data shows no degradation. A degradation can occur only if altering to a more expensive data type. If altering to a less expensive data type, like from VARCHAR to CHAR, then an improvement would take place.

Recommendations

Schedule reorganization of the table space soon after ALTER to minimize performance degradation. We typically experience 20% degradation in FETCH, 10% in INSERT, prior to REORG.

Alter Index Add Column

We executed the following scenario on a 10 partitions table space, a 10 M rows table, 1 PI and 2 NPIs:

1. CREATE INDEX on column C1
2. SELECT C1, C2 WHERE C2='XX' - C2 is not in the index key
3. ALTER INDEX add column C2 to the index
4. REBUILD INDEX
5. SELECT C1, C2 WHERE C2='XX' - C1, C2 are in the index key

The elapsed and CPU time for step 2 and 5 were measured and compared, see Figure 5-10.

Before ALTER INDEX add column, a table space scan was used to access the data. After ALTER INDEX ADD of an existing column of the table to an index, the index was placed in a RBDP state and needs to be rebuilt. After rebuild, the access path changed from table space scan to non-matching index scan showing a 10 times elapsed time improvement.

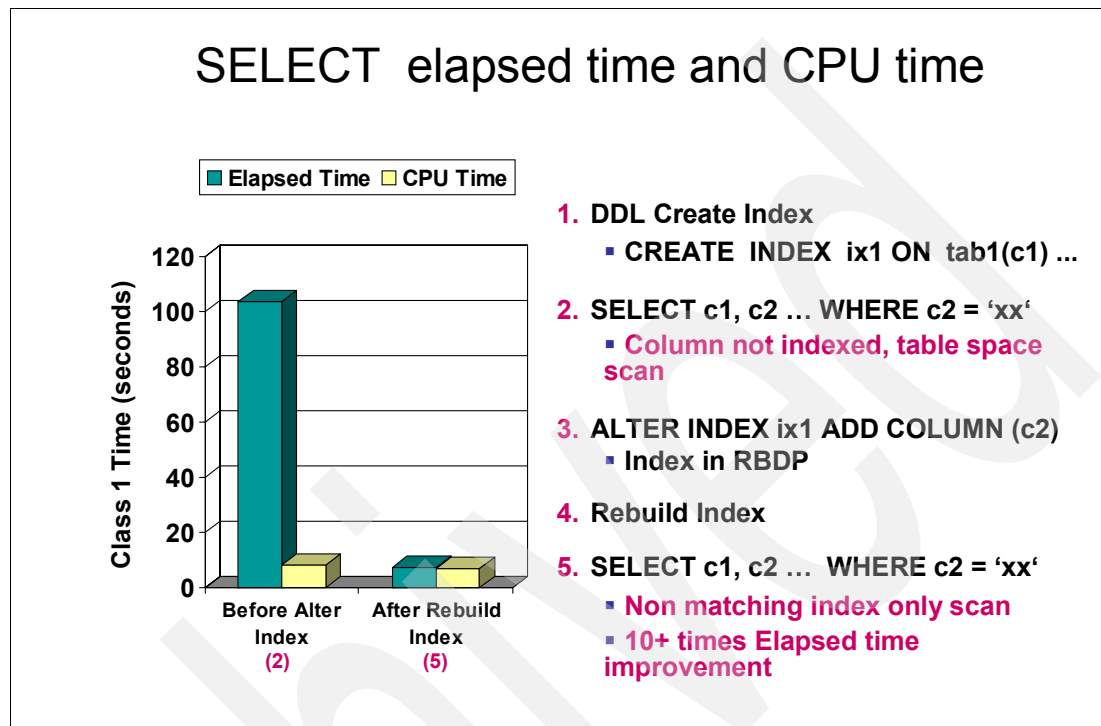


Figure 5-10 SELECT elapsed and CPU time before and after ALTER INDEX add column

Conclusion

A change of the access path from table space scan to an access path using the index improves the elapsed time. This is well known. What is worth noting is the ease of implementing the change. Remember that an index in RBDP state does not block access to the data. For non-unique indexes, static plans and packages that do not use the index to retrieve data, they will continue to function as before. For unique indexes placed in RBDP, those plans and packages that are dependent on them will get a -904 (unavailable resource) on an update of an existing row or insert of a new row. Deletes are no problem. For the time until rebuild, the optimizer simply ignores the index for dynamic SQL; for static SQL a rebind is necessary.

Recommendations

Once the index is available again, revert to optimal access paths by:

- ▶ Rebinding affected plans and packages for static SQL.
- ▶ Automatic invalidation of related statements in the dynamic statement cache when RBDP is reset. At next execution, the statement is prepared again, and can go back to the original access path.

Important: The enhancement is designed to allow maximum availability for the situations where a new column is added to a table, and this new column is also desired as part of an existing index. By making changes within the same unit of work there is no unavailability.

Alter Index Padding

In a first test with a non-partitioned table with 17 columns and 1 index the following scenario is executed and measured:

1. CREATE PADDED INDEX index on C1, C2 and C3, each column being VARCHAR(10)
2. ALTER INDEX to NOT PADDED and REBUILD INDEX
3. ALTER INDEX to PADDED and REBUILD INDEX
4. CREATE NOT PADDED INDEX on C1, C2 and C3, each column being VARCHAR(10)

After each step the CPU time is measured for the following SQL:

```
FETCH C1, C2, C3... WHERE C1='XXXXXXXX' AND C2='YYYYYYYY'
```

The results of this measurement are shown in Figure 5-11.

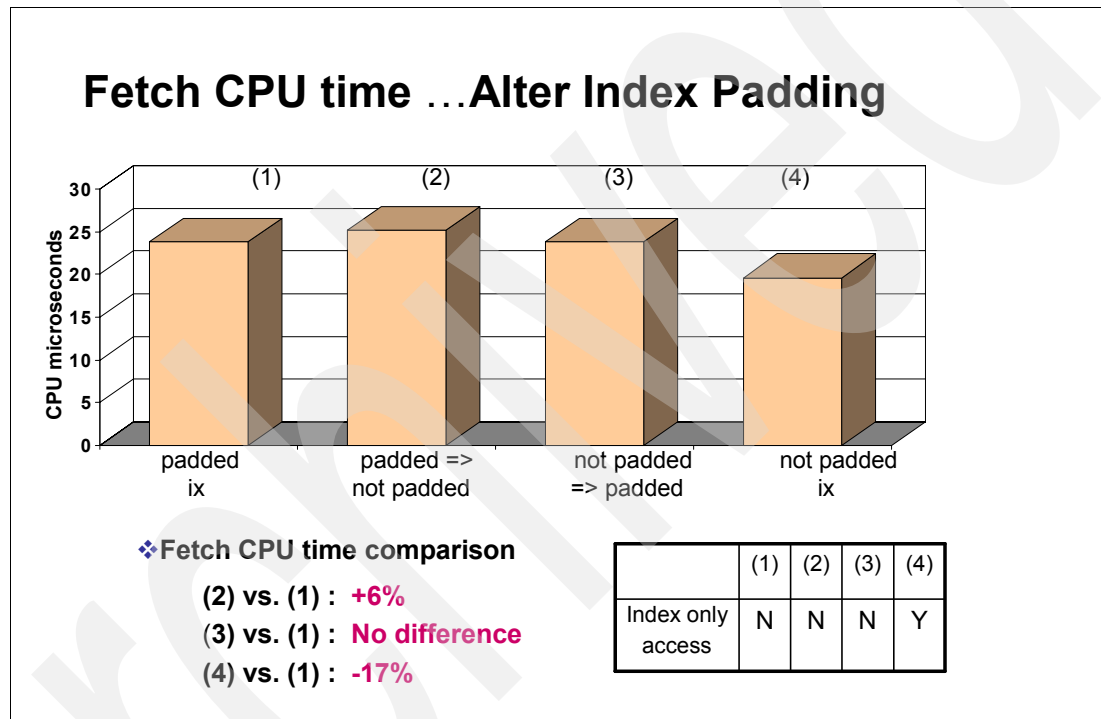


Figure 5-11 *FETCH CPU time - Alter index padded*

- ▶ After ALTER of the index to NOT PADDED and REBUILD the CPU time for FETCH increased by 6% (2 vs. 1)
- ▶ Altering the index back to PADDED shows no difference in FETCH CPU time compared to the original situation (3 vs. 1)
- ▶ Compared to an index that was created originally with the keyword NOT PADDED there is a FETCH CPU time increase of 23% (2 vs. 4)
- ▶ The impact of NOT PADDED versus PADDED evaluates to a CPU time decrease of 17% (4 vs. 1) due to index-only access which outweighs the more expensive VARCHARs index key processing

Note that the access path after the change from PADDED to NOT PADDED did not change to index-only access.

We repeated this measurement and included a rebind of the application after altering the index from PADDED to NOT PADDED. See Figure 5-12 for the results.

Fetch CPU time ...Alter Index Padding

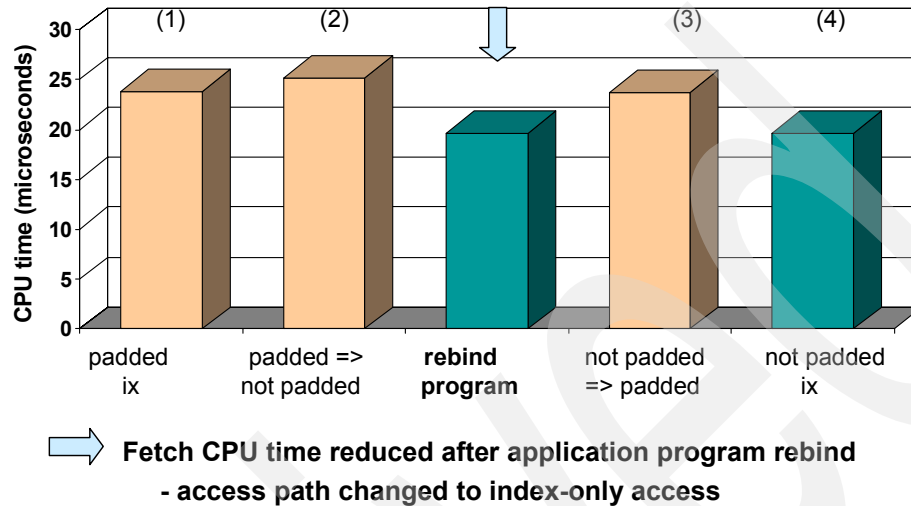


Figure 5-12 FETCH CPU time - Alter index not padded and rebind

The FETCH CPU time after rebind shows no difference compared to an originally created NOT PADDED index. Also an index-only access path is used after rebind.

A second test was run with a set of 15 queries doing joins on two VARCHAR(10) columns. The join columns are accessed via matching index scan. We measured the query performance for this set of queries using padded keys. Then we altered the indexes to NOT padded, ran rebuild index and RUNSTATS and measured the query performance for 15 queries using not padded keys. We repeated this test with VARCHAR(20) and VARCHAR(30) columns. In Figure 5-13 we show the CPU overhead using NOT PADDED indexes vs. PADDED indexes.

Average CPU overhead for 15 Queries

Query CPU overhead using NOT PADDED vs. PADDED keys

Two varchar keys in two indexes are altered from padded to not padded

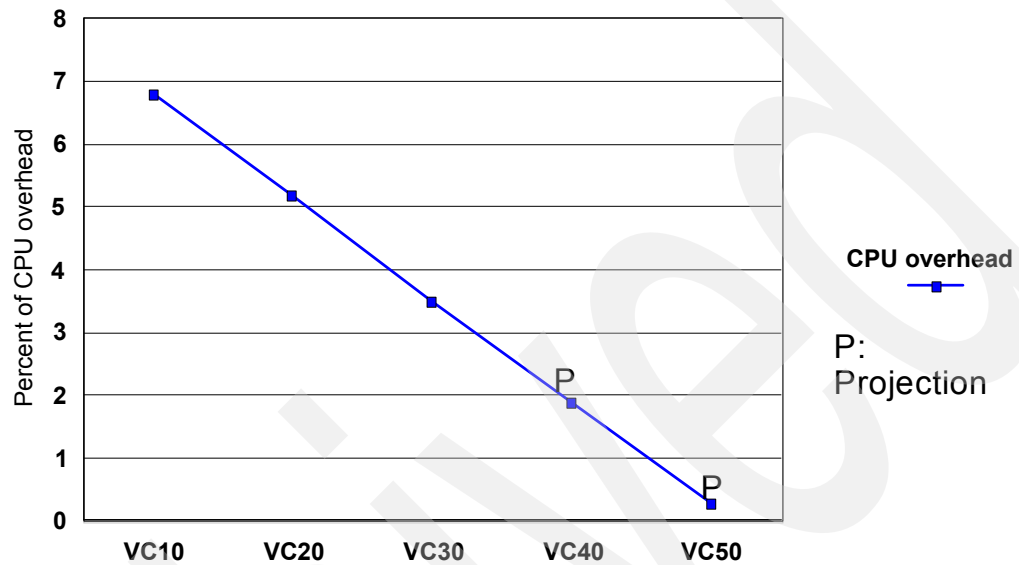


Figure 5-13 Query CPU overhead using NOT PADDED vs. PADDED index

There is a cost associated with the use of not padded keys as together with the key also the length bytes are saved. As the VARCHARs become longer the CPU overhead decreases.

The impact on index leaf pages and index getpages is shown in Figure 5-14.

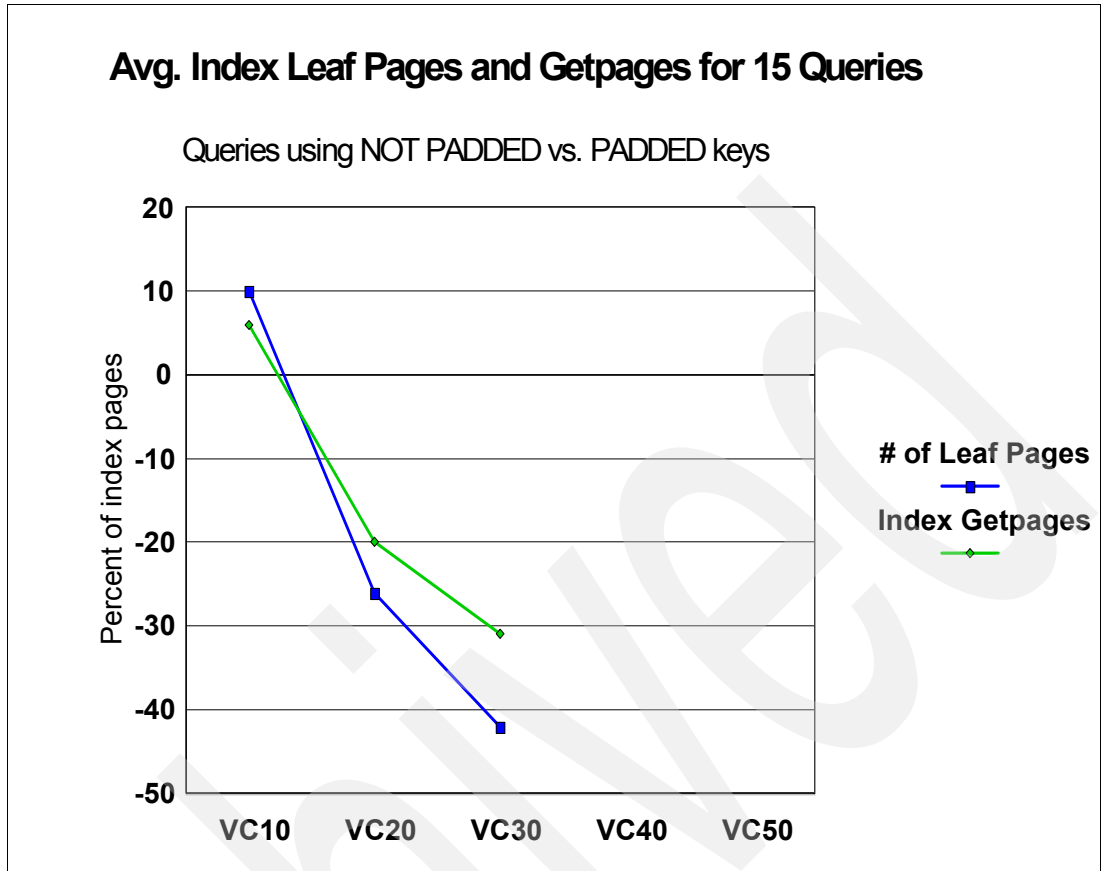


Figure 5-14 Average index leaf pages and index getpages

Initially there is no benefit from not padded index keys as together with the key also the length bytes are saved. The longer the VARCHAR becomes the more the benefit from not padding becomes visible. Index storage requirements could be reduced as only the actual data is stored and the index tree may have fewer levels.

In a third measurement the impact of NOT PADDED indexes on utilities was measured. We used a table with 10 M rows and one PADDED or NOT PADDED index. The following cases were measured:

1. 10 M rows table with a one-column indexed key of VARCHAR(14) with varying length from 1 to 14.
2. 10 M rows table with a one-column indexed key of VARCHAR(1)
3. 10 M rows table with a four-column indexed key of each VARCHAR(1)

The tests were run for LOAD, REORG table space, REORG index, REBUILD, CHECK index and RUNSTATS index.

See Figure 5-15 for the measurement result of test case 1.

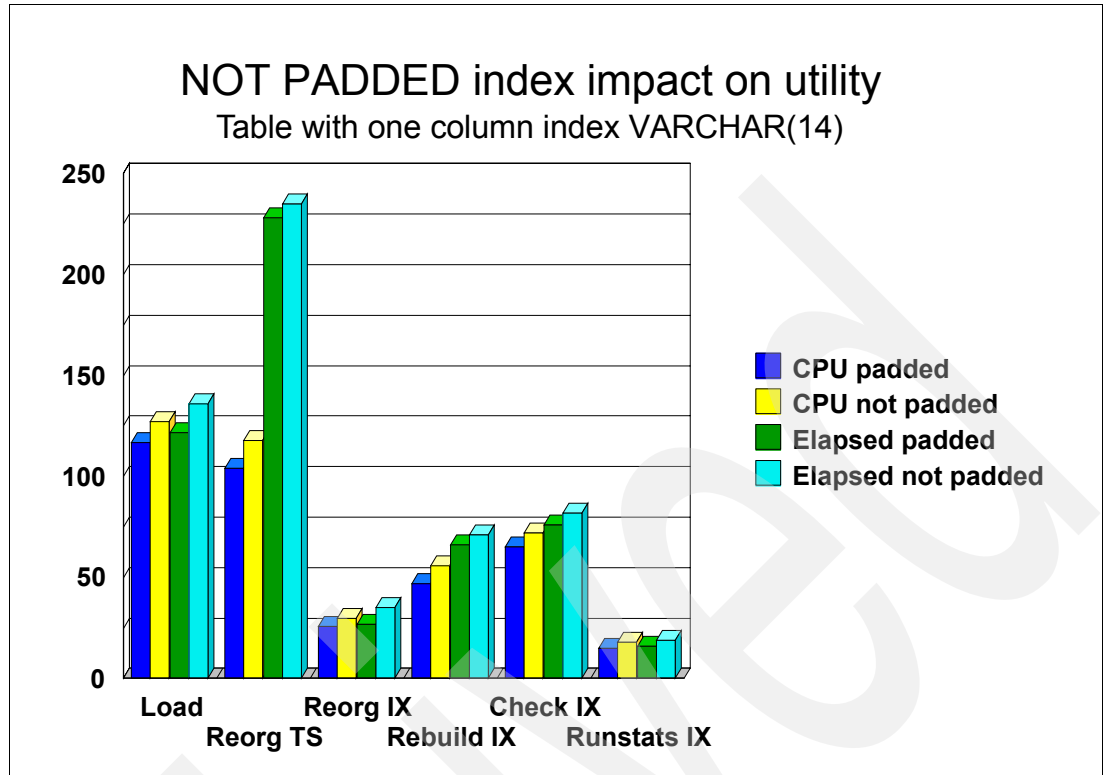


Figure 5-15 NOT PADDED index impact on utilities - Index VARCHAR(14)

See Figure 5-16 for the measurement result of test case 2.

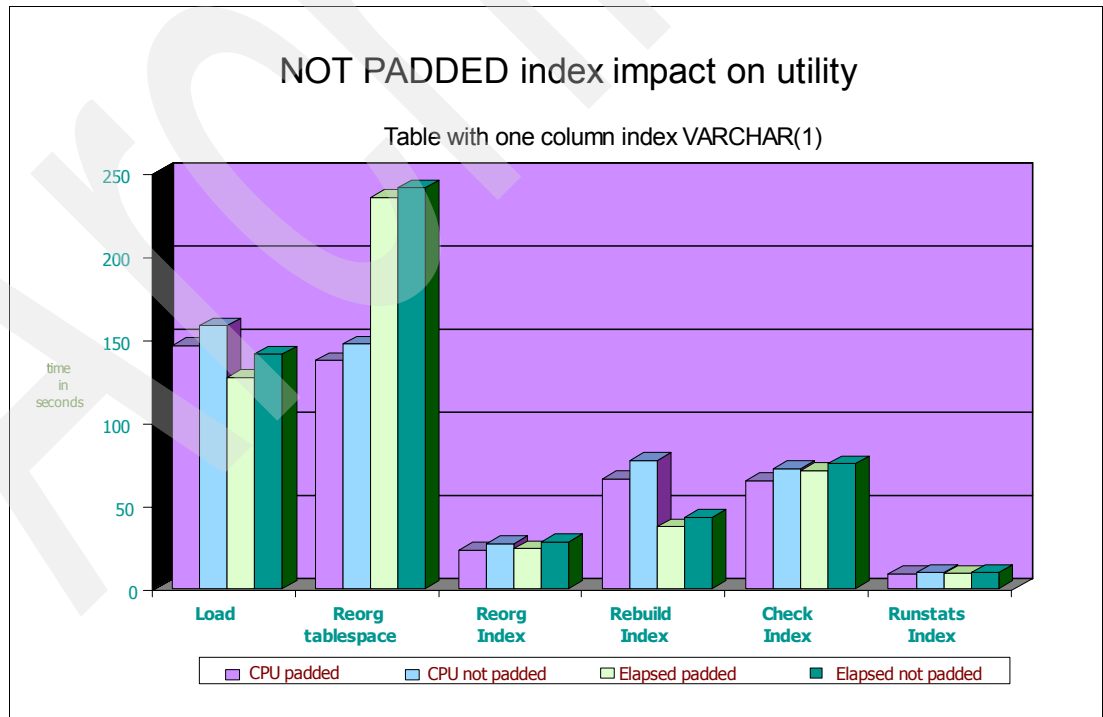


Figure 5-16 NOT PADDED index impact on utilities - Index VARCHAR(1)

See Figure 5-17 for the measurement result of test case 3.

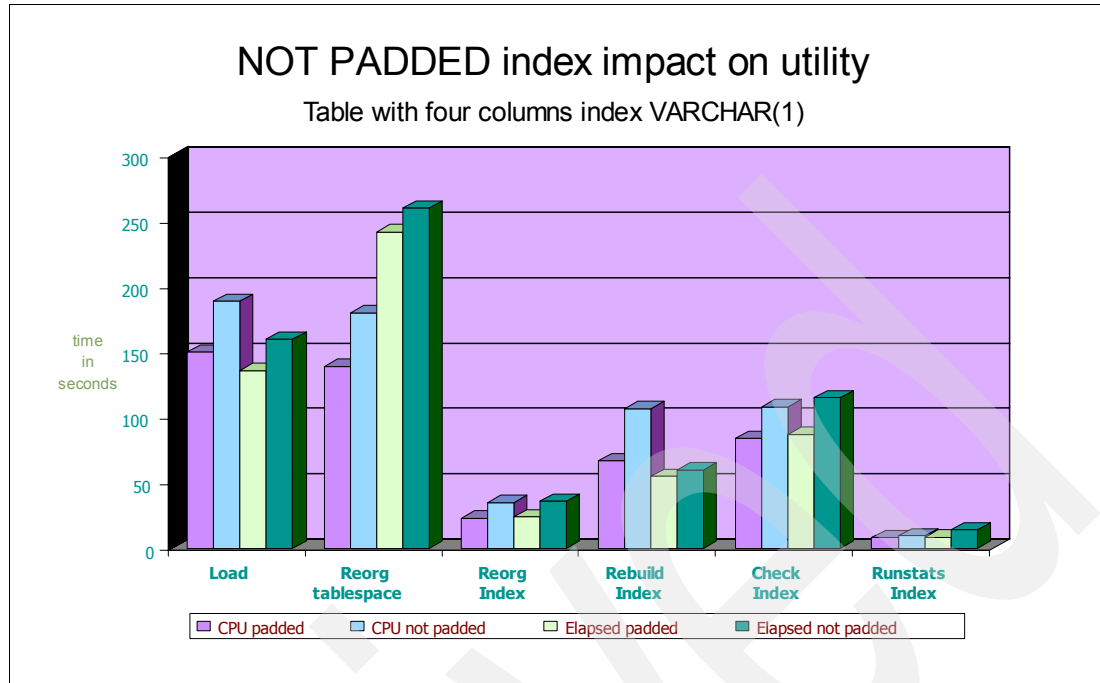


Figure 5-17 NOT PADDED index impact on utilities - Index 4 columns VARCHAR(1)

The results from these measurements show:

- ▶ Up to 20% degradation in CPU with one-column indexed key
- ▶ Up to 60% degradation in CPU with four-column indexed key
- ▶ LOAD and REORG table space: Degradation in LOAD and BUILD phases
- ▶ REORG INDEX and REBUILD INDEX and CHECK INDEX: Degradation in UNLOAD and BUILD or CHECK IX phases
- ▶ RUNSTATS INDEX degrades 20% with one-column indexed key of VARCHAR(14) but only 14% with four-column indexed key of VARCHAR(1)
- ▶ With NOT PADDED index key, two bytes are added to each key → less CPU degradation with bigger VARCHAR key length (>18 char)

Note that VARCHAR(1) is not just a theoretical example. It was used in this measurement in order to illustrate the worst case scenario, but it is often present in SAP environments, and whenever an index was likely to be altered.

Conclusion

After ALTER index to NOT PADDED, REBUILD INDEX and rebind the applications in order to achieve better performance.

Recommendations

For those installations that avoid the use of long VARCHAR columns in indexes, there is now less reason for doing so.

As a rule of thumb, start using not padded indexes for VARCHARs with length more than 18 bytes. If you have VARCHAR(>18) columns in indexes consider altering these indexes to NOT PADDED, rebuild the indexes and rebind dependent packages that may be able to exploit the index-only access.

Alter Table Rotate Partition

We executed and measured the following test cases (TC) on a 10 partition table space, a 50 M row table, 1 PI and 0, 1, 3 NPIs:

- ▶ TC1 - Delete all rows of partition 1 with SQL
- ▶ TC2 - Alter table rotate partition
- ▶ TC3 - Load replace with dummy input followed by alter table rotate partition

See Figure 5-18 for the results of the measurements.

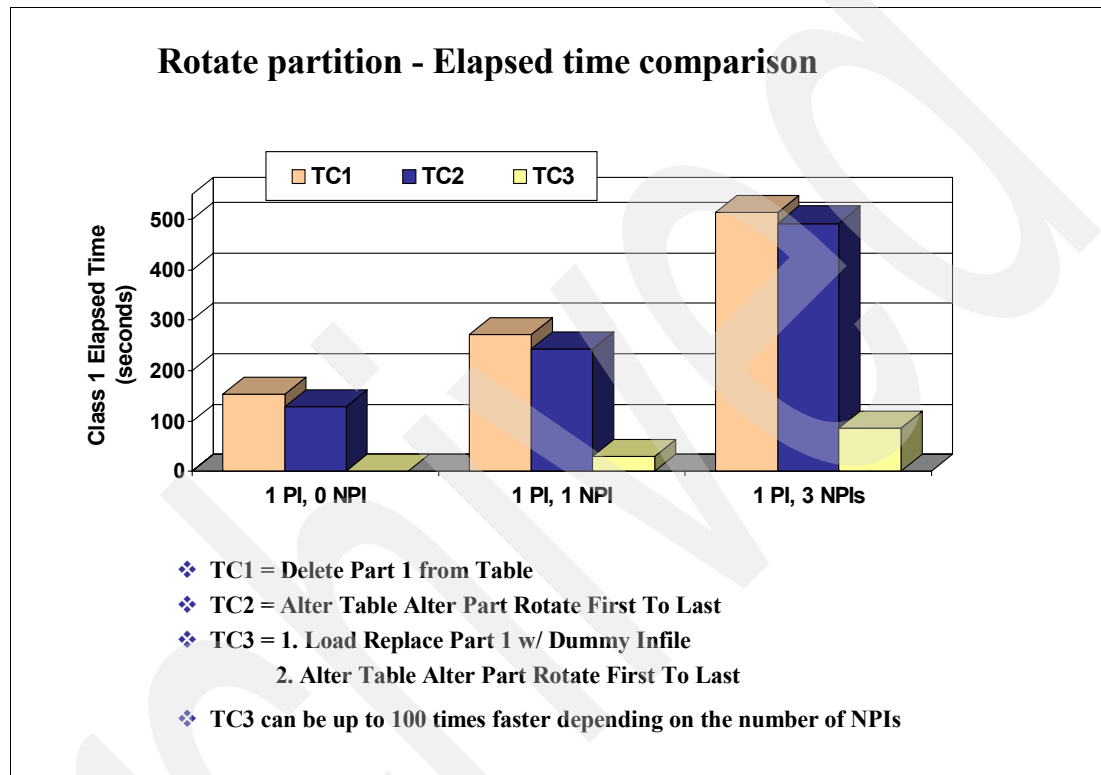


Figure 5-18 Rotate partition - Elapsed time comparison

In TC1 we delete all the rows from partition 1. The slight difference with TC2 is due to DB2 predicate evaluation. Also ROTATE uses delete on part 1 but does not have to evaluate a predicate. If part 1 is emptied first by a dummy LOAD REPLACE elapsed time improves dramatically. NPIs must be scanned to delete the keys related to rows of part 1, the more NPIs defined the higher the impact on elapsed time.

Conclusion

Rotate will issue deletes for the partition to be reused. Consider the impact on logging and performance. A DBD lock will be held for the duration of the ROTATE.

Recommendations

In order to reduce the impact on logging and speed up the delete process consider first archiving the data in the affected partition and running a LOAD table PART n REPLACE with a dummy input. Be aware that ROTATE cannot be undone. Backup your data before rotating.

REORG REBALANCE

We executed and measured the following test cases (TC) on a 10 partition table space, a 50 M row table, 8 partitions with 5 M rows, 1 partition with 1 M rows and 1 partition with 9 M rows, 1 PI and 0, 1, 3 NPIs:

- ▶ TC1 - REORG table space REBALANCE all partitions
- ▶ TC2 - REORG table space REBALANCE partition 9 and 10

See Figure 5-19 for the partition distribution before and after REORG REBALANCE.

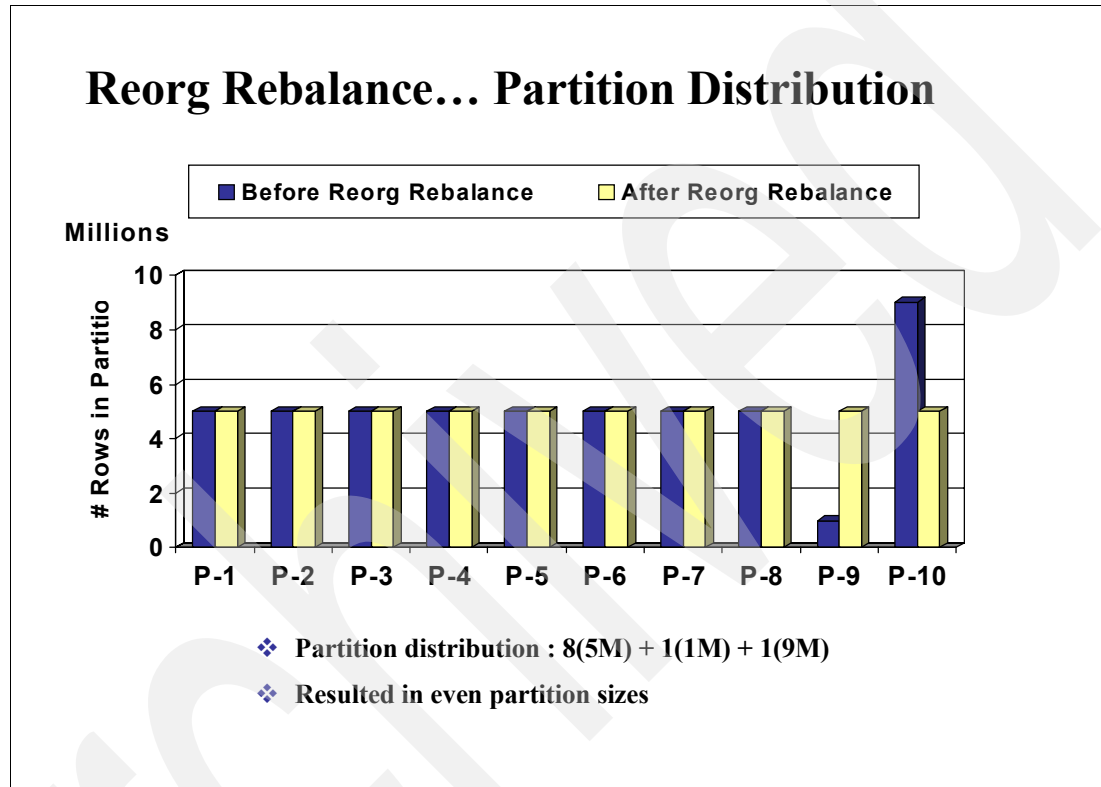


Figure 5-19 Partition distribution before and after REORG REBALANCE

REORG REBALANCE resulted in an even distribution of rows over all partitions.

See Figure 5-20 for the elapsed time measurement result.

Reorg Rebalance... Elapsed Time

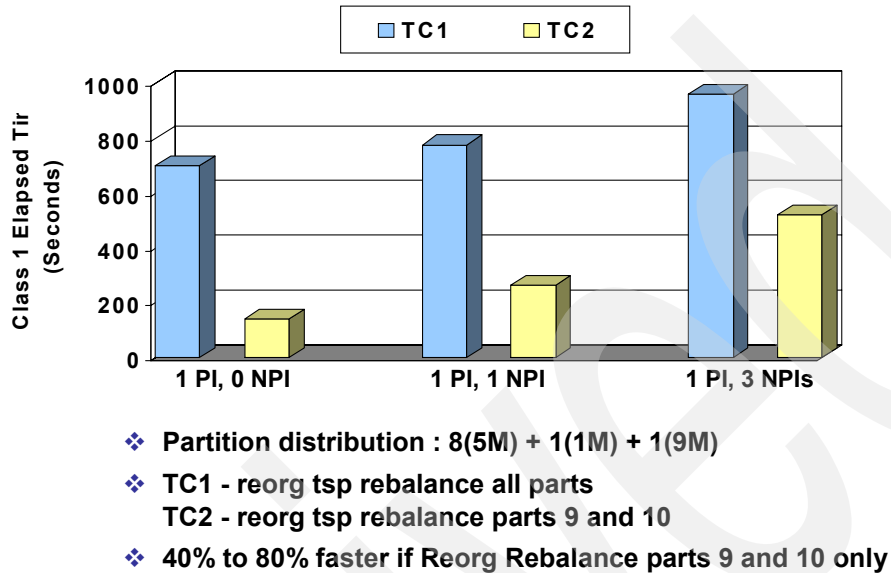


Figure 5-20 REORG REBALANCE elapsed time

In test case 1 all partitions are read and updated even if not all partitions needed to be rebalanced. Depending on the number of NPIs REORG REBALANCE runs 40% to 80% faster if only partition 9 and 10 are rebalanced.

Conclusion

Rebalancing could be executed in prior versions of DB2 by altering the partitioned index limit keys followed by a reorganization. But the affected partitions were placed in a REORGP state and the data was unavailable until the reorganization completed. Rebalancing can be done with SHRLEVEL REFERENCE and so the data remains available for readers almost all the time. Only during the switch phase, the data is unavailable. It is also during the switch phase that DB2 updates the limit keys to reflect the new partition boundaries.

Recommendations

The use of rebalancing instead of altering the partitioned index key limits will enhance the availability of your data.

Note: REORG REBALANCE might not be possible if the logical and physical partition numbers for the specified table space do not match. This situation can be created by a series of ALTER ROTATEs and ALTER ADD PARTs. The partition range that you specify on the REORG REBALANCE statement is a range of physical partitions. When rebalancing, DB2 unloads, sorts, and reloads the data based on logical partition numbers. Message DSNU1129I is issued and REORG terminates.

5.4 Volatile table

DB2 V6 introduced a new parameter, NPGTHRSH, which will cause the DB2 Optimizer to favor index access for tables whose statistics indicate fewer than a given number of pages. For a given table, if NPAGES is fewer than the NPGTHRSH value, index access for the table will be preferred over a table space scan. After the initial install of ERP application packages, there are many empty or small tables which could grow rapidly in size. There are also a number of tables which are very volatile, meaning the number of rows can change very quickly and in large amounts. If a RUNSTATS is run on these tables when they are small, the DB2 optimizer would favor a table space scan, which would be inappropriate when the table grows. Prior to this enhancement, the recommendation was to update the catalog statistics for volatile tables.

DB2 V8 introduces the VOLATILE keyword on the CREATE and ALTER statements. A volatile table is a table whose contents can vary from zero to very large at run time. DB2 often does a table space scan or non-matching index scan when the data access statistics indicate that a table is small, even though matching index access is possible. This is a problem if the table is small or empty when statistics are collected, but the table is large when it is queried. In that case, the statistics are not accurate and can lead DB2 to pick an inefficient access path. Forcing index access may be desired for tables whose size can vary greatly.

In some applications, and often with SAP, a DB2 table, called a *cluster* table, can be made up of logical rows, with each logical row consisting of multiple physical rows from that table. A logical row is identified by the primary key with a “sequence number” appended to provide the logical ordering of the physical rows. When accessing this type of table, the physical rows are intended to be accessed in this order (primary key + sequence number). This reduces the chance of deadlocks occurring when two applications attempt to lock the same logical row but touch the underlying physical rows in a different order. This means that certain types of access paths are disabled for these types of tables. They are: list prefetch hybrid join and multi-index access.

5.4.1 Conclusion

Although volatile tables and the NPGTHRSH subsystem parameter both cause DB2 to favor index access, they behave differently and have different granularities. NPGTHRSH favors index access on a subsystem level when the number of pages drops below the threshold. The NPGTHRSH subsystem parameter causes DB2 to prefer index access in all of these cases, even when statistics are not accurate. Volatile tables are designated at the table level instead of the subsystem level, which allows you to favor index access for specific tables and specific queries because only those tables that are defined as volatile favor index access. Finally, volatile tables differ from the NPGTHRSH subsystem parameter by further restricting the types of access to preserve the access sequence that is provided by the primary key.

5.4.2 Recommendations

Use the VOLATILE keyword on the CREATE table statement instead of the NPGTHRSH DSNZPARM parameter in order to favor index access for volatile and cluster tables at the table level.

5.5 Index enhancements

With the table-controlled partitioning concept in DB2 V8, clustering, being partitioned, and being the partitioning index are now separate concepts. When using table-controlled partitioning, a table does not require a partitioning index, since the partitioning is based on

the PARTITION BY clause in the CREATE TABLE statement. Since the partitioning index is no longer required, another index may also be used as a clustering index.

Remember that index-controlled partitioning is still supported in DB2 V8 but several new enhancements are only supported when using table-controlled partitioning.

We now review index terminology and classification.

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

- ▶ Based on whether or not an index is physically partitioned:
 - Partitioned index: The index is made up of multiple physical partitions (not just index pieces)
 - Non-partitioned index: The index is a single physical data set (or multiple pieces)
- ▶ Based on whether or not the columns in the index correlate with the partitioning columns of the table (the left most partitioning columns of the index are those specified in the PARTITION BY clause of the table):
 - Partitioning index: The columns in the index are the same as (and have the same collating sequence), or start with the columns in the PARTITION BY clause of the CREATE TABLE statement.
 - Secondary index: Any index in which the columns do not coincide with the partitioning columns of the table.
 - Data-partitioned secondary indexes: Any index defined with the PARTITIONED option (see later).
- ▶ Based on whether or not the index determines the clustering of the data. Note that when using table-controlled partitioning:
 - Clustering index: The index determines the order in which the rows are stored in the partitioned table.
 - Non-clustering index: The index does not determine the data order in the partitioned table.

5.5.1 Partitioned table space without index

Because the table definition of a table-controlled partitioned table is complete after executing the CREATE TABLE statement, no partitioning index is required. So you can have a table-controlled partitioned table without a partitioning index. Due to the non-unique nature of DPSIs it is advised to specify predicates on partitioning key columns in order to enable partition pruning from queries using predicates for DPSI columns.

5.5.2 Clustering index

In DB2 V8, any index on a table-controlled partitioned table can be the clustering index, including a secondary index. If predicates used to access the table controlled partitioned table do not refer to the partitioning columns, a clustering index can now be defined for these predicates.

5.5.3 Clustering index always

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

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

5.5.4 DPSI and impact on utilities and queries

With DB2 V7, secondary indexes on partitioned tables are non-partitioned indexes. Secondary indexes cannot be physically partitioned. They can only be allocated across multiple pieces to reduce I/O contention. DB2 V8 introduces the ability to physically partition secondary indexes. The partitioning scheme introduced is the same as that of the table space. That is, there are as many index partitions in the secondary index as table space partitions, and index keys in partition “n” of the index reference only data in partition “n” of the table space. Such an index is called a Data-Partitioned Secondary Index (DPSI).

Description

When an index is created and no additional keywords are specified, the index is non-partitioned. If the keyword PARTITIONED is specified, the index is partitioned. The index is both data-partitioned and key-partitioned when it is defined on the partitioning columns of the table. Any index on a partitioned table space that does not meet the definition of a partitioning index is a secondary index. When a secondary index is created and no additional keywords are specified, the secondary index is non-partitioned (NPSI). If the keyword PARTITIONED is specified, the index is a data-partitioned secondary index (DPSI).

You create a DPSI by specifying the new PARTITIONED keyword in the CREATE INDEX statement and defining keys on some columns that coincide with the partitioning columns of the table. When defining a DPSI, the index cannot be created as UNIQUE or UNIQUE WHERE NOT NULL. This is to avoid having to search each part of the DPSI to make sure the key is unique.

Benefits of DPSIs are:

- ▶ Allow partition-level operation for utilities operating on secondary indexes such as REORG INDEX, RUNSTATS INDEX
- ▶ Eliminate contention on index pages for utilities operating on partition level with DPSI.
- ▶ Utilize concurrent LOAD PART more efficiently while inserting DPSI keys
- ▶ Eliminate the BUILD2 phase during Online REORG of a partition
- ▶ Reduce data sharing overhead

For a detailed description of DPSIs see *DB2 UDB for z/OS Version 8: Everything You Ever Wanted to Know, ... and More*, SG24-6079.

Performance

In this section the performance impact of DPSIs on utilities and queries is evaluated.

Performance measurement description

For this measurement we used:

- ▶ z/OS V1R3
- ▶ DB2 for z/OS V8 in NFM
- ▶ IBM z900 Series 2064 4-way processor
- ▶ ESS model F20 with FICON channels

Performance measurement result for utilities and DPSIs

We used a partitioned table of 20 M rows with 10 partitions

- ▶ With one NPI or one DPSI
- ▶ With 3 DPSIs and 2 NPIs
- ▶ With 5 DPSIs or 5 NPIs

The following measurement cases were run:

- ▶ Load table
- ▶ Reorg table space
- ▶ Rebuild index
- ▶ Check index
- ▶ Runstats index

Figure 5-21 shows the result of a measurement where one partition was loaded of a 20 M row table with 10 partitions and 1 NPI vs. 1 DPSI.

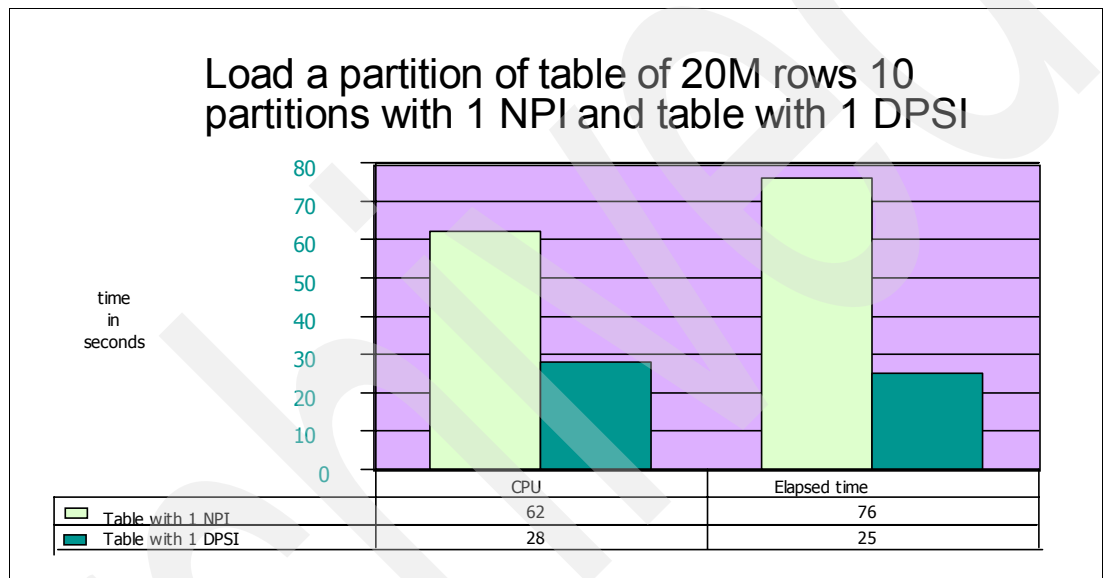


Figure 5-21 Load with 1 NPI vs. 1 DPSI

The reduction in CPU of the test case with a DPSI is mostly due to better code method: Load of the DPSI (a page at the time) instead of Insert in the NPI case.

In Figure 5-22 we compare the load of a single partition of a 20 M row table with 10 partitions with 5 NPIs or 5 DPSIs versus the load of the whole table.

Load table of 20M rows, 10 partitions with 5 NPI or 5 DPSI

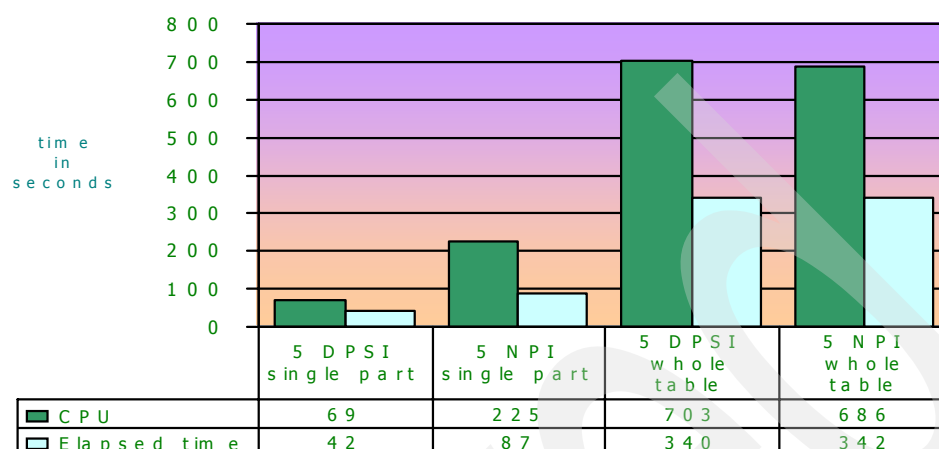


Figure 5-22 Load with 5 NPIs vs. 5 DPSIs

When we compare the single partition load to the previous test case with 1 NPI or 1 DPSI, a bigger reduction in CPU for 5 DPSIs compared to 5 NPIs is noticed because the index keys were loaded for all DPSI index physical partitions while keys were inserted into all 5 NPI index logical partitions during the Build phase. Contention on NPIs is possible.

In case the whole table is loaded the DPSI case shows a slight increase in CPU compared to the NPI case. In this case 5 NPI data sets vs. 5 DPSI data sets are loaded.

Figure 5-23 presents the measurement results of a reorganization of a partition of a 20 M rows table with 10 partitions and 1 NPI or 1 DPSI.

Reorg a partition of table of 20M rows 10 partitions with 1 NPI and table with 1 DPSI

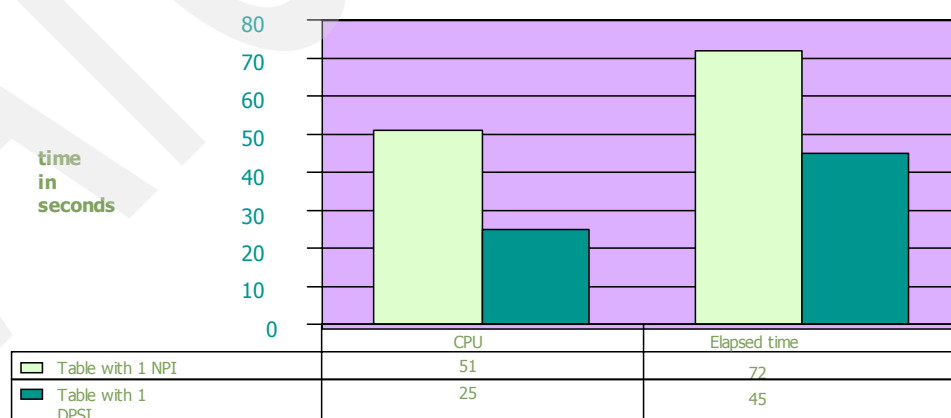


Figure 5-23 REORG with 1 NPI vs. 1 DPSI

The reduction in CPU of DPSI is due to better code method: Load instead of INSERT as in the NPI case. Note that in the DPSI test case the BUILD2 phase is eliminated.

In Figure 5-24 we compare the reorganization of a single partition of a 20 M rows table with 10 partitions with 5 NPIs or 5 DPSIs versus the reorganization of the whole table.

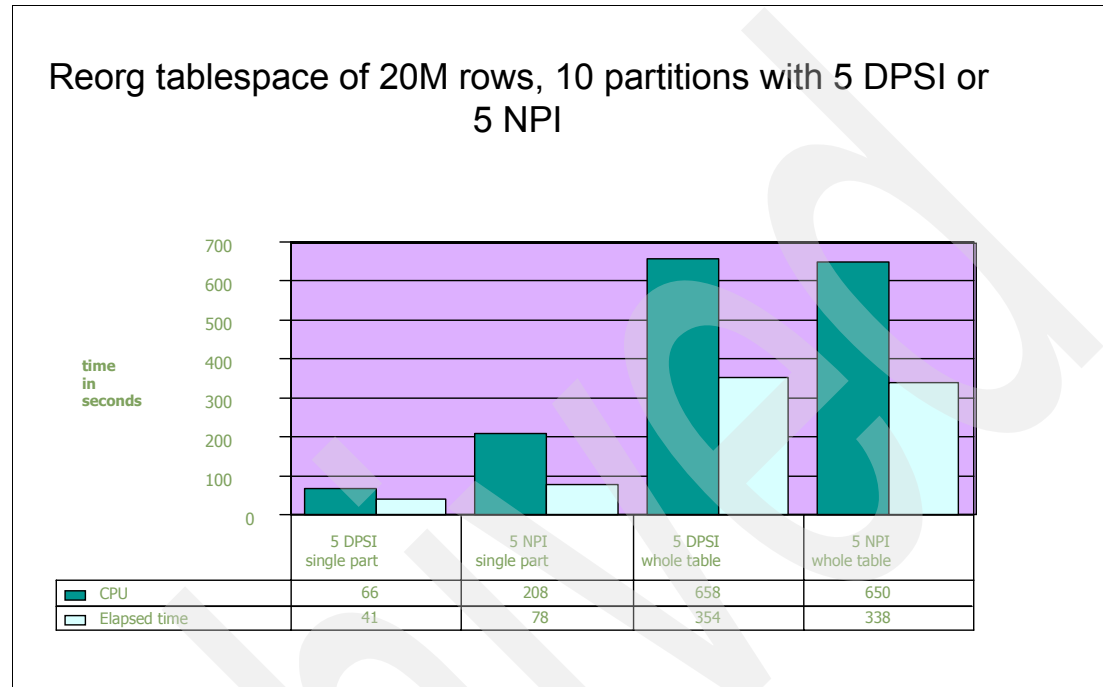


Figure 5-24 REORG with 5 NPIs vs. 5 DPSIs

The single partition reorganization compared to the previous reorganization test case with 1 NPI or 1 DPSI shows a bigger reduction in CPU for 5 DPSIs compared to 5 NPIs because the index keys were loaded for all DPSI index physical partitions while keys were inserted to all 5 NPI index logical partitions during the BUILD2 phase. Contention on NPIs is possible.

In case the whole table is reorganized, the DPSI case shows a slight increase in CPU compared to the NPI case. In this case 5 NPI data sets vs. 50 DPSI data sets are loaded in the index Build phase.

In Figure 5-25 we compare the rebuilding of a partition of a PI, NPI and DPSI.

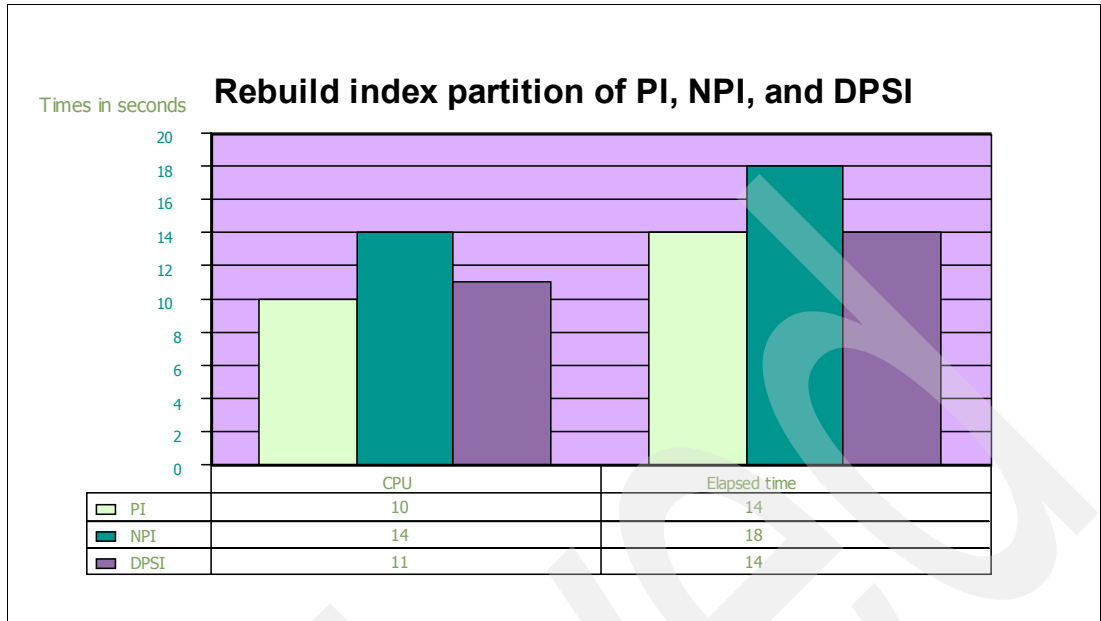


Figure 5-25 Rebuild index partition of PI, NPI and DPSI

Equivalent performance between DPSI and PI is noticed, performance of NPI is impacted by the key Insert processing vs. Load of PI and DPSI partitions.

In Figure 5-26 we compare the rebuilding of the complete PI, NPI and DPSI.

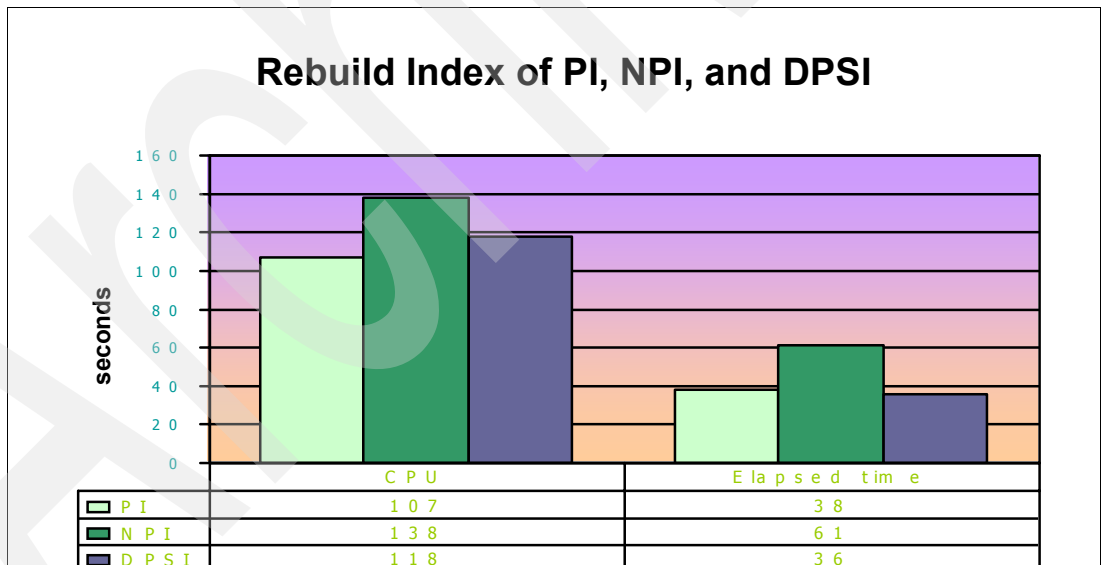


Figure 5-26 Rebuild index PI, NPI, and DPSI

For the PI and DPSI indexes a parallel Unload, Sort and Build is executed. For the NPI Unload and Sort are done in parallel followed by a single Merge task and a single Build task of the NPI. Performance is equivalent for DPSI and PI but better compared to NPI.

In Figure 5-27 check index on a partition of a PI, NPI and DPSI is compared.

Check Index a partition of PI, NPI, and DPSI

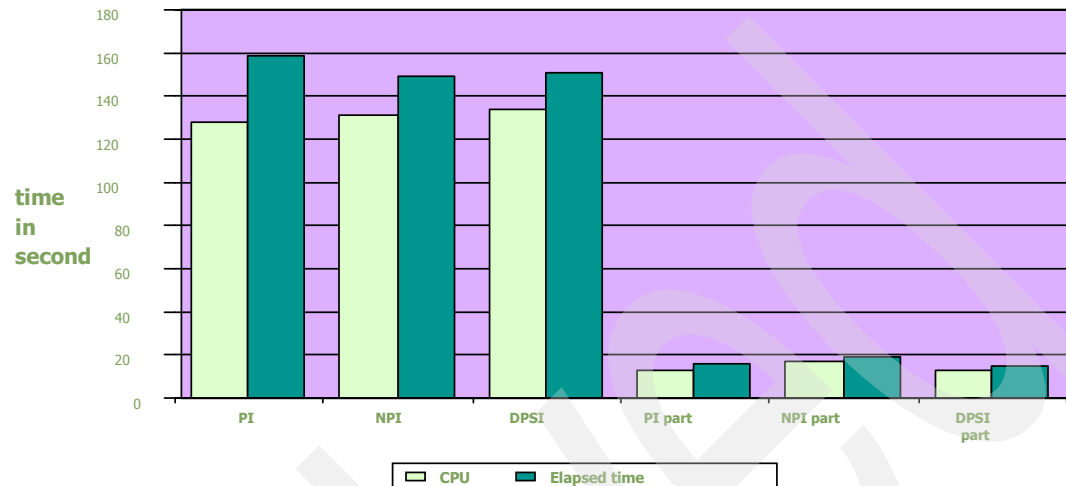


Figure 5-27 Check index PI, NPI and DPSI

Check index of a partition of DPSI showed equivalent performance as with partition of PI which had better performance compared to Check Index of a logical part of a NPI.

In Figure 5-28 we compared executing RUNSTATS INDEX on a PI, NPI and DPSI and RUNSTATS INDEX on a partition of a PI, NPI and DPSI.

Runstats Index a partition of PI, NPI, and DPSI

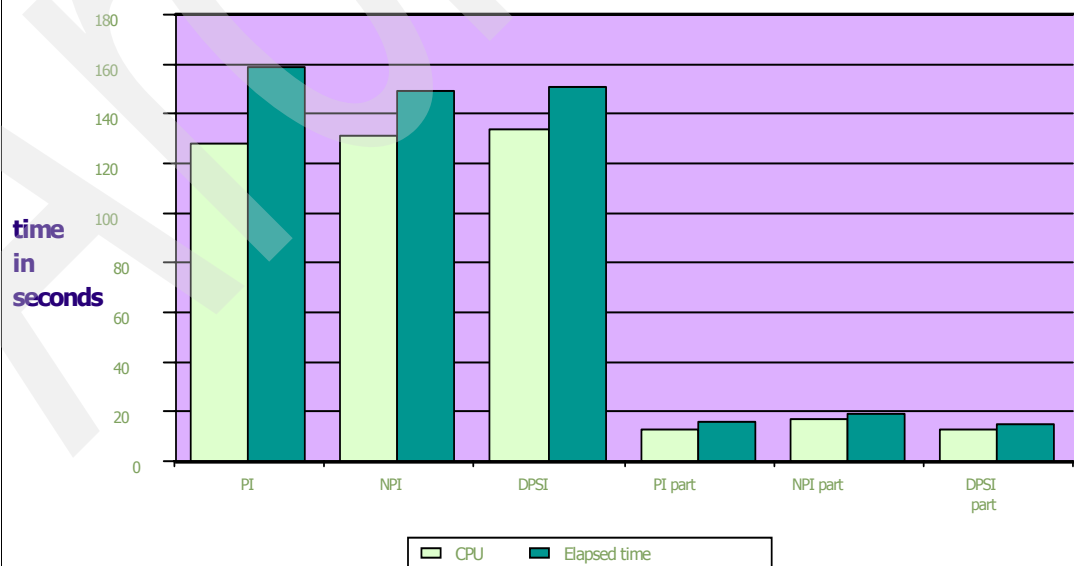


Figure 5-28 Runstats index PI, NPI and DPSI

RUNSTATS INDEX PARTITION of a DPSI showed equivalent performance to a partition of a PI. Comparing to RUNSTATS INDEX of a logical part of a NPI, the performance was better.

Performance measurement result for queries and DPSIs

We executed queries against 3 partitioned tables with an NPI and a DPSI on the same columns. The measurements were done with and without parallelism.

The tables used are:

- ▶ CVR - 1,654,700 rows, 10 partitions
- ▶ EVE - 931,174 rows, 6 partitions
- ▶ PLC - 624,473 rows, 8 partitions

Attention: Make sure PTF UQ93972 for APAR PQ92227, which addresses SQL performance problems with DPSIs, is applied.

Table 5-3 shows the measurement results for the following query:

```
SELECT COUNT(*) FROM CVR
```

Table 5-3 NPI vs. DPSI SELECT COUNT query

	NPI	DPSI	% difference
Access path	index only (2285 pages)	index only (2302 pages)	=
CPU (sec.)	0.759703	0.587190	-23%
Elapsed (sec.)	0.825141	0.668409	-19%

This is the case of an index scan. Almost the same number of index getpages are executed for DPSI as for NPI. The results shown are those of the second execution of the queries (data sets are all open) but the data is not preloaded. The query with predicates that only reference DPSI indexed columns experiences a reduction in CPU time of 23% compared to an NPI on the same columns. Since these queries are close to CPU bound, a similar reduction in elapsed time of 19% is experienced.

The same query was also executed with parallelism enabled. The results are presented in Table 5-4.

Table 5-4 NPI vs. DPSI SELECT COUNT query parallelism

	NPI	DPSI	% difference
Access path	index only (2312 pages)	index only (2342 pages)	=
CPU (sec.)	0.718432	0.657405	-9%
Elapsed (sec.)	0.637515	0.231345	-64%

Parallelism helps quite a lot in improving the elapsed time because of the probing in parallel of the DPSI. The reduction in CPU is somewhat offset by the activation of parallelism, but the elapsed time improvement is 64%.

Now we evaluate a query which qualifies data on NPI and DPSI columns in order to verify the overhead of DPSI over NPI:

```
SELECT COUNT(*) FROM EVE, PLC WHERE EVE_DPSI>=PLC_DPSI AND PLC_DPSI = constant
```

Table 5-5 shows the results for this query.

Table 5-5 DPSI overhead

	NPI	DPSI	% difference
Access path	PLC - index only EVE - index only	PLC - index only EVE - index only	
CPU (sec.)	0.021699	0.083861	+ 281
Elapsed (sec.)	0.037188	0.305458	+ 721
Index getpages	8	17,000	

The DPSI overhead is explained by DB2 having to probe each DPSI partition in order to evaluate the predicates. The access path remains the same in both test cases.

In order to illustrate an access path change from changing use of a NPI to a DPSI we evaluated the following query:

```
SELECT COUNT(DISTINCT CSCD) FROM CVR
```

Table 5-6 shows the measurement results.

Table 5-6 DPSI access path change

	NPI	DPSI	% difference
Access path	index only	table space scan + sort	
CPU (sec.)	1.21221	8.30175	+ 585
Elapsed (sec.)	1.317186	24.67209	+ 1773
Getpages	2k index getpages	139k data getpages + 29k work file getpages	

Conclusion

Utility processing benefits from the use of DPSIs. Query processing using only the DPSI in the access path has to probe every partition of the DPSI to evaluate the predicates due to the non-unique nature of the DPSI. In a data sharing environment, if some workloads are partition and member specific, reduced data sharing overhead is possible with DPSIs as intersystem read/write interest on physical partitions can be eliminated vs. non-partitioned indexes, where keys belonging to different data partitions are spread throughout the non-partitioned index.

Recommendations

Replacing all NPIs by DPSIs is not the objective. NPIs will still prove to be useful. Utilities will definitely benefit from DPSIs while contention at the logical partition level is eliminated. Queries that combine predicates on the partitioning columns and the DPSI will perform well. Explicitly coding the partitioning columns in the predicate allows the DB2 optimizer to eliminate not qualifying partitions from the query. This is called *partition pruning*. Pruning can be deferred until execution time if host variables are used. The pruning takes place once the contents of the host variables are known.

When using DPSIs part of the responsibility is shifted from DBA to development. Coding the correct predicates to allow for partition pruning is crucial to performance.

The use of DPSIs versus NPIs is dictated by availability considerations. If, for example, even the outage due to the BUILD2 phase of an online REORG cannot be tolerated DPSIs are the solution.

Note that when the qualified partitions cannot be determined at BIND time because the literal value in the predicate is unknown (for example, because a host-variable, parameter marker, or special register is used), then DB2 will determine what the qualified partitions are at execution time. By doing this, only the qualified partitions will be accessed, even if DB2 could not determine which partitions were qualified at BIND time. (This enhancement does NOT require the use of REOPT(VARS)). In addition, the DB2 access path selection has been enhanced to make use of all leading partitioning key columns when determining the qualified partitions. (Currently, only the first column of the partitioning key is used to determine which partitions qualify. This can result in more partitions being in the range of qualified partitions than actually do qualify.)

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

5.6 Other availability enhancements

Other DB2 V8 availability enhancements and also recent enhancements introduced via maintenance are discussed in this section.

5.6.1 More online DSNZPARMs

In order to minimize downtime when making DSNZPARM parameter changes effective, DB2 V7 introduced the online change for subsystem parameters. This function allows you to load a new subsystem parameter load module into storage without recycling DB2. To do this, you can use the normal installation parameter update process to produce a new load module, which includes any desired changes to parameter values. You can then issue the -SET SYSPARM command to load the new module in order to affect the change.

DB2 V8 adds more online changeable parameters.

Not all subsystem parameters are dynamically changeable. Refer to the Appendix of DB2 *UDB for z/OS Version 8 DB2 Command Reference*, SC18-7416, for a list of all parameters, the macros where they are defined, and whether or not they are changeable online.

5.6.2 Monitor long running UR back out

During restart processing, DB2 issues progress messages during the forward and backward phase of the restart. Message DSNR0311 is issued in 2 minute intervals, showing the current log RBA being processed and the target RBA to complete the restart phase. With DB2 V8, a new progress message is issued during the back out of postponed URs as well. This message is first issued after the process has at least exceeded one two minute interval and is repeated every two minutes until the back out process is complete.

5.6.3 Monitor system checkpoints and log offload activity

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

5.6.4 Improved LPL recovery

One source of unplanned outages that many DB2 customers have struggled with is LPL (logical page list) pages. DB2 puts pages into the LPL typically when I/O or coupling facility read or write errors are encountered. Once a page is entered into the LPL, that page is inaccessible until it is recovered. LPL recovery can be done using:

- ▶ -START DATABASE command with SPACENAM option
- ▶ RECOVER utility

The -START DATABASE command drains the entire table space or partition, therefore making the entire page set or partition unavailable for the duration of the LPL recovery process even if only one page is in the LPL for that table space or partition. During the execution of the recover utility the table space or partition is unavailable.

DB2 V8 always attempts to automatically recover the LPL pages at the time that the pages are added to LPL. Those pages are deleted from the LPL when the automatic LPL recovery completes successfully.

Even more important than automatic LPL recovery is that with DB2 V8, during LPL recovery, all pages that are not in the LPL are accessible by SQL. Up to DB2 V7, the entire table space or partition was unavailable during LPL recovery.

5.6.5 Partitioning key update enhancement

Some customers find the current implementation that allows update of values in partitioning key columns unusable because it does not allow concurrency if an update of a partitioning key changes the partition to which the row belongs.

If the update requires moving the data row from one partition to another, DB2 tries to take exclusive control of the objects to perform the update by acquiring DRAIN locks. Because of this, no other application can access the range of partitions affected by update of values in partitioning key columns.

Description

Up to DB2 V7, DB2 takes the exclusive control to perform the partitioned key update on:

- ▶ The partition of the table space from which the row is moving, the partition of the table space to which the row is moving, and all partitions in between,
- ▶ The partition of the partitioning index from which the partitioning key is moving, the partition of the partitioning index to which the partitioning key is moving, and all partitions in between
- ▶ Non partitioning indexes defined on the table space.

When updating values in columns that define the partitioning key, DB2 V8 will NOT take exclusive control of the objects to perform the update.

If a row moves from one partition to another in a table that is a dependent table in a Referential Integrity relationship, there is a possibility of missing the row when checking for RI. To prevent this, if any update changes the row's partition, a COMMIT duration S lock will be acquired on the parent table rows.

Change to DSNZPARM parameter PARTKEYU

The existing system parameter PARTKEYU specifies whether values in columns that participate in partitioning keys may be updated.

- ▶ "YES" - Values in such columns may be updated. This is the default value.

- ▶ “NO” - Values in such columns may not be updated.
- ▶ “SAME” - Values in such columns may be updated if and only if the update leaves the row belonging to the same partition.

5.6.6 Lock holder can inherit WLM priority from lock waiter

Assume that a transaction executes with a low WLM priority and makes updates to the database. Because it is updating, it must acquire X-locks on the rows of the pages that it modifies and then these X-locks are held until the transaction reaches a commit point. This is business as usual. If such a transaction holds an X-lock on a row in which another transaction is interested, the other transaction is forced to wait until the first transaction (with a low WLM priority) commits. If the lock waiter performs very important work and is thus assigned a high WLM priority, it would be desirable that it is not slowed down by other transactions that execute in a low priority service class.

To reduce the time the high priority transaction has to wait for that lock to be given up by the low priority transaction, it would be better if the waiting transaction could temporarily assign its own priority to the transaction that holds the lock until this transaction frees the locked resource.

DB2 V8 exploits WLM enqueue management. When a transaction has spent roughly half of the lock timeout value waiting for a lock, then the WLM priority of the transaction, which holds the lock, is increased to the priority of the lock waiter if the latter has a higher priority. If the lock holding transaction completes, it resumes its original service class. In case multiple transactions hold a common lock, this procedure is applied to all of these transactions.

This function is implemented by the PTF for APAR PK19303, currently open.

5.6.7 Detecting long readers

Prior to DB2 V8, DB2 issues warning messages DSNR035I and DSNJ031I for a long running unit of recovery (UR) based on a number of system checkpoints taken since the beginning of the UR or the number of log records written.

DB2 V8 takes this one step further by also alerting you when you have a long-running reader in the system. It is probably less well known that not only updates have to commit frequently. Also read-only transactions should commit regularly. This is to allow utilities (that need to drain the object) to take control of the object, for example, during the switch phase of online REORG, the utility needs full control of the object to switch between the shadow data set and the actual data set used by DB2. For parallel operations, DB2 records this information only for the originating task.

A new system parameter LONG-RUNNING READER THRESHOLD, LRDRTHLD, is provided to specify the number of minutes that a read claim can be held by an agent before DB2 will write an IFCID 313 record to report it as a long-running reader when statistics class 3 is active.

IFCID 313 records all the LONG-RUNNING URs during a checkpoint or after exceeding the LOG RECORD's written threshold without commit or all the LONG-RUNNING readers after exceeding the elapsed time threshold without commit.

5.6.8 Explain a stored procedure

This function allows you to run EXPLAIN without needing the authority to run the SQL to be explained.

Application programmers need to be able to make sure that SQL will perform correctly, but in many cases they should not have the authority to read the actual data the SQL will be selecting or updating, for example, sensitive payroll data. OWNER(SYSADM) DYNAMICRULES(BIND) allows a user to EXPLAIN SQL statements without having the privilege to execute those SQL statements. However, the drawback of this setting is that the owner becomes the default qualifier for unqualified tables and views inside the SQL.

PTFs UQ92322 (DB2 V7) and UQ92323 (DB2 V8) for the two APARs PQ90022 and PQ93821 add a new sample stored procedure called DSN8EXP that can be used to work around these constraints. DSN8EXP accepts an SQL statement of up to 32700 single-byte characters, passed as an input parameter from its caller. It first sets the current SQLID to locate EXPLAIN tables, then it issues the EXPLAIN statement for the input SQL. You can optionally let the stored procedure parse the SQL and add the qualifier ahead of those unqualified tables/views if there are any unqualified tables/views in the SQL. For insert-within-select and common table expressions, you need to disable the parsing function, and add the qualifier manually. DSN8EXP returns the SQLCODE, SQLSTATE, and diagnostic text from the EXPLAIN by output parameter.

Visual Explain for DB2 for z/OS Version 8 provides an easy-to-use interface to call the DSN8EXP stored procedure.

Installation

This PTF also provides a new sample job called DSNTJXP that you can use to prepare and bind the DSN8EXP stored procedure. DSNTJXP is not customized by the DB2 installation CLIST. See the prolog of DSNTJXP for guidance on customizing it to run at your site.

Running DSN8EXP from Visual Explain

In the Subsystem Parameters Panel there are fields to specify the stored procedure's schema name and procedure name. When all necessary fields are filled in the TuneSQL panel, note there is a field called "Table qualifier for Explain stored procedure:" When this field has a non-blank value, it is used as the input parameter QUALIFIER for the stored procedure and the parsing function will be enabled; if this field is blank, then the parsing is disabled. There is also a button called "Explain with stored procedure" shown in the TuneSQL panel, which actually invokes the EXPLAIN stored procedure. The original EXPLAIN button still issues normal Explain statements.

5.6.9 PLAN_TABLE access via a DB2 alias

Up to V7, you did not have the capability to access Explain tables with different OWNER and QUALIFIER names. OWNER refers to the creator of the EXPLAIN table; QUALIFIER refers to the user that issues the EXPLAIN statement. A limitation of the EXPLAIN command has been that only the owner of the EXPLAIN tables can issue the EXPLAIN statement to populate his/her tables. This has prevented you from being able to consolidate your plan tables under a single AUTHID.

Starting with V8, you can use the ALIAS mechanism to populate EXPLAIN tables created under a different AUTHID. The following external EXPLAIN tables can now have aliases defined on them:

- ▶ PLAN_TABLE
- ▶ DSN_STATEMNT_TABLE
- ▶ DSN_FUNCTION_TABLE

The alias name must have the format userid.table_name where tablename can be PLAN_TABLE, DSN_STATEMNT_TABLE or DSN_FUNCTION_TABLE.

5.6.10 Support for more than 245 secondary AUTHIDs

With PTF UQ92698 for APAR PQ90147, support for 1,012 secondary authorization ids is provided. The connection and sign on exits DSN3@ATH and DSN3@SGN can supply 1,012 secondary authorization IDs in the AIDL, which will be used in DB2 authorization checking.

Description

DB2 V8 will be updated to support 1,012 secondary authorization IDs. Customers may need to review their modified connection and sign-on exits, DSN3@ATH and DSN3@SGN, to see if changes are required to support 1,012 secondary authorization IDs. IBM supplied samples DSN3SATH and DSN3SSGN do not require changes to support 1,012 secondary authorization ids.

Utilities

The utilities have all been enhanced to support all other major enhancements that DB2 V8 has introduced. For instance, extensive updates have been required to provide support for long names, Unicode, 64-bit addressing, the new data partitioning techniques, online schema evolution, and system point-in-time recovery. Refer to “Related publications” on page 425 for details.

In this chapter we concentrate on availability and performance aspects of V8 utilities:

- ▶ **Online CHECK INDEX:** CHECK INDEX with option SHRLEVEL CHANGE increases availability and satisfies the requirement of customers who cannot afford the application outage. Its also dramatically cuts down elapsed time due to parallel processing and usage of FlashCopy V2.
- ▶ **LOAD and REORG:** With DB2 V8, LOAD and REORG utilities have the option SORTKEYS as a default, this provides performance improvement for multiple index key sort.
- ▶ **LOAD and UNLOAD delimited input and output:** This usability and DB2 family compliance enhancement makes it much easier to move data into DB2 with good performance.
- ▶ **RUNSTATS enhancements:** The RUNSTATS utility adds the new functionality of calculating the frequencies for non-indexed columns. It updates DB2 catalog tables and leads to better access path selection for queries.
- ▶ **Cross Loader:** This functionality enables you to use a single LOAD job to transfer data from one location to another location or from one table to another table at the same location. A new PTF for V7 and V8 provides a large performance boost to cross loader performance.
- ▶ For the new **BACKUP and RESTORE SYSTEM utilities**, introduced to ease the implementation of point-in-time recovery, see 5.1, “System level point-in-time recovery” on page 218.

All these functions are available in CM except for BACKUP and RESTORE SYSTEM.

6.1 Online CHECK INDEX

Online CHECK INDEX is a new capability of DB2 V8, added by APARs PQ92749 (PTF UK04683) and PQ96956 (still OPEN), which allows you to have read-write access to applications during CHECK INDEX and contributes to decreasing read-only time for users. This function satisfies the high availability requirement of users with very large tables who cannot afford the application outage caused by the extensive read-only time required to run the normal CHECK INDEX.

CHECK INDEX is an online utility which is used to test whether an index is consistent with its data. It is generally useful after a conditional restart or point-in-time recovery. It is also recommended to run CHECK INDEX before CHECK DATA, especially if you specify DELETE YES. Running CHECK INDEX before CHECK DATA ensures that the indexes that CHECK DATA uses are valid. In DB2 V8 the utility statement CHECK INDEX was enhanced to designate the current read-only behavior as SHRLEVEL REFERENCE, and designate a reduction of restricted availability with the new SHRLEVEL CHANGE option. In addition CHECK INDEX provides a parallel processing structure similar to REBUILD INDEX.

SHRLEVEL REFERENCE and SHRLEVEL CHANGE

There are two options to indicate the type of access that is allowed for objects to be checked during the procedure. Figure 6-1 shows the utility process with the two options.

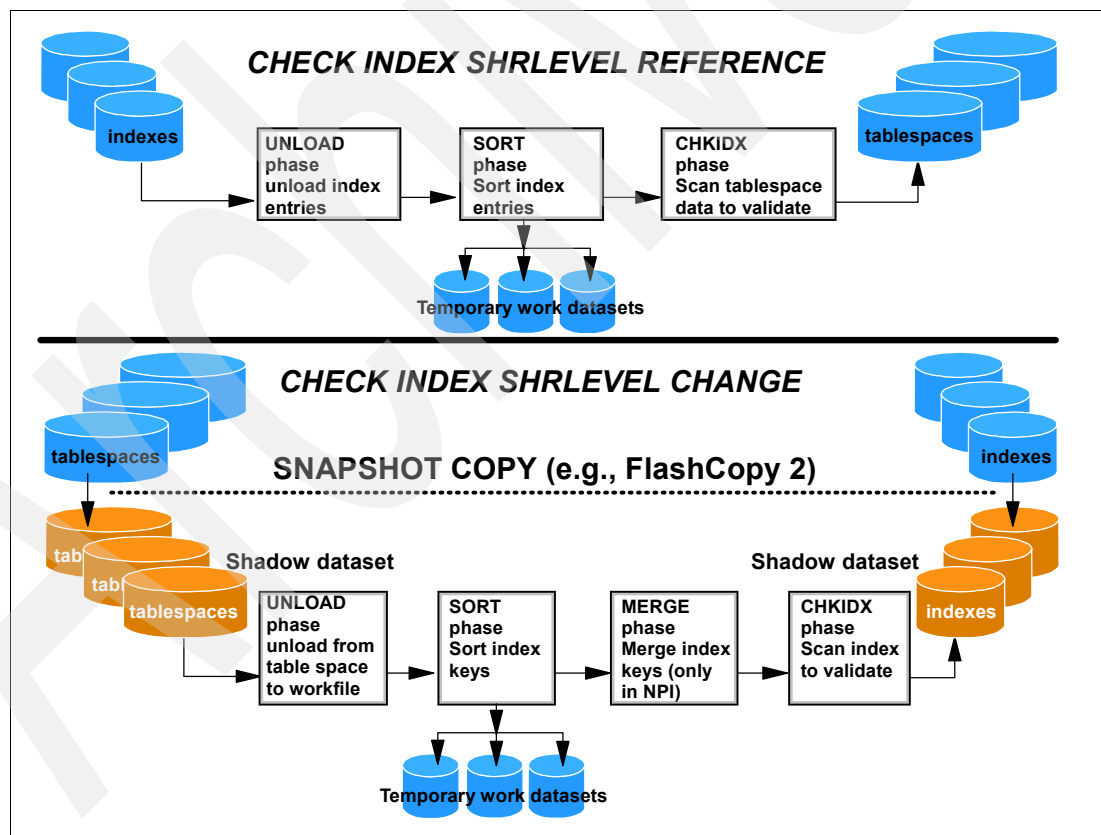


Figure 6-1 CHECK INDEX SHRLEVEL REFERENCE and SHRLEVEL CHANGE

If you specify SHRLEVEL REFERENCE, which is also the default, the application can read from but cannot write to the index, table space, or partition that is to be checked.

If you specify SHRLEVEL CHANGE, the applications can read from and write to the index, table space or partition that is being checked. The phases of processing inside CHECK INDEX differ in SHRLEVEL REFERENCE and SHRLEVEL CHANGE. In the case of SHRLEVEL REFERENCE, DB2 unloads the index entries, sorts the index entries, and scans the data to validate index entries. On the other hand, SHRLEVEL CHANGE first goes to the table space data set and unloads the data and sorts the index keys and merges (if the index is non-partitioned), and then scans the index to validate.

CHECK INDEX with option SHRLEVEL CHANGE performs the following actions:

- ▶ Issues QUIESCE WRITE YES for the specified object and all of its indexes (this drains the writers so no updates are allowed for a while)
- ▶ Invokes DFSMSdss at the data set level to copy the table space and all indexes to shadow data sets

Note: Online CHECK INDEX will use FlashCopy V2 if available, otherwise concurrent copy will be used, this can elongate the read-only time.

- ▶ Allows write access to the specified object and indexes
- ▶ Runs CHECK INDEX on the shadow data sets

Tip: FlashCopy V2 is needed since it supports data set copy (FlashCopy V1 works only at the volume level). Another advantage is that the source and target of a FlashCopy V2 can be a different logical subsystem within the Enterprise Storage System.

Index processing with SHRLEVEL CHANGE

SHRLEVEL CHANGE has a new architecture. SHRLEVEL REFERENCE unload phase unloads keys from the index and the check phase compares sorted keys against the data via tablespace scan. SHRLEVEL CHANGE unload phase does a table space scan extracting keys and the check phase compares sorted keys against the keys in the index.

CHECK INDEX in V8 can provide parallel processing like REBUILD INDEX, which minimizes elapsed time for checking the data and indexes. If you specify more than one index to check and the SHRLEVEL CHANGE option, CHECK INDEX checks the indexes in parallel unless constrained by available memory or sort work files. Checking indexes in parallel reduces the elapsed time for a CHECK INDEX job by sorting the index keys and checking multiple indexes in parallel, rather than sequentially.

It is important for parallel processing to specify appropriate values in the SORTDEVT and SORTNUM parameter. SORTDEVT specifies the device type for temporary data sets that are to be dynamically allocated by the DFSORT. SORTNUM can specify the number of temporary data sets dynamically allocated by the sort program (DFSORT). Example 6-1 is a sample JCL of CHECK INDEX.

Example 6-1 Sample JCL of CHECK INDEX SHRLEVEL CHANGE

```
//CHKIDXB EXEC PGM=DSNUTILB,REGION=4096K,PARM='SSTR,CHKINDX1'  
//SYSPRINT DD SYSOUT=A  
//SYSUDUMP DD SYSOUT=A  
//UTPRINT DD SYSOUT=A  
//DSNTRACE DD SYSOUT=A  
//SYSUT1 DD UNIT=SYSDA,SPACE=(CYL,(5,2)),VOL=SER=SCR03  
//SYSOUT DD UNIT=SYSDA,SPACE=(CYL,(5,2)),VOL=SER=SCR03  
//SORTLIB DD DISP=SHR,DSN=SYS1.SORTLIB
```

```
//SORTOUT DD UNIT=SYSDA,SPACE=(CYL,(5,2)),VOL=SER=SCR03
//SYSERR DD UNIT=SYSDA,SPACE=(CYL,(5,2)),VOL=SER=SCR03
//SYSIN DD *
LISTDEF CHKIDXB_LIST INCLUDE INDEXSPACE DBOT55*.* ALL
CHECK INDEX LIST CHKIDXB_LIST SHRLEVEL CHANGE
                WORKDDN SYSUT1
                SORTDEVT SYSDA
                SORTNUM 4

/*
```

You should estimate the size of the sort work file which you write in the SORTWKnn DD statement of the JCL. If a sort is required, you must provide the DD statement that the sort program requires for the temporary data sets.

Note: To find the size of the sort work, the following steps should be taken.

1. For each table, multiply the number of records in the table by the number of indexes on the table that need to be checked.
2. Add the products that you obtain in step 1.
3. Add 8 to the length of the longest key. For non padded indexes, the length of the longest key is the maximum possible length of the key with all varying-length columns in the key padded to their maximum length, plus 2 bytes for each varying-length column.
4. Multiply the sum from step 2 by the sum from step 3.

Availability enhancement

To improve availability, the DRAIN_WAIT, RETRY, and RETRY_DELAY parameters were added to CHECK INDEX. This is similar to online REORG. Rather than waiting the lock time out time, the utility multiplier for threads to relinquish the write claim, the utility will wait the DRAIN_WAIT value to get all writers drained and if the drain is not successful, wait the specified delay, and then retry the specified number of times. This will lessen the impact on applications when the utility is unable to drain all objects.

Recommendation in creating shadow data sets

When you execute the CHECK INDEX utility with the SHRLEVEL CHANGE option, the utility uses shadow data sets. For user-managed data sets, you must preallocate the shadow data sets before you execute CHECK INDEX SHRLEVEL CHANGE.

If a table space, partition, or index that resides in DB2-managed data sets and shadow data sets does not already exist when you execute CHECK INDEX, DB2 creates the shadow data sets. At the end of CHECK INDEX processing, the DB2-managed shadow data sets are deleted. We recommend you create the shadow data set in advance for DB2-managed data sets also, especially for the shadow data sets of the logical partitions of non-partitioned secondary indexes.

See *DB2 UDB for z/OS Version 8 Utility Guide and Reference*, SC18-7427, for more information about this topic.

6.1.1 Performance measurement

Several performance studies have been done about online CHECK INDEX. Here we describe the measurement environment and results.

Test environment and scenarios

- ▶ Configuration
 - DB2 for z/OS V8 NFM
 - Non-data sharing
 - z/OS Release 1.5
 - DFSMSdss V1R5
 - IBM z900 Series 2064 4-way processor
 - ESS 800 DASD
 - FlashCopy V2
- ▶ Workload for measurement
 - 20 M rows, 6 indexes, 1 table space with 10 partitions
- ▶ Measurement scenarios
 - CHECK INDEX SHRLEVEL(REFERENCE), CHECK INDEX SHRLEVEL(CHANGE) against
 - Partitioned Index
 - Non-partitioned index
 - All indexes (6) of the table space
 - Concurrent CHECK INDEX SHRLEVEL(REFERENCE) PART of all partitions of the table space versus single CHECK INDEX SHRLEVEL(CHANGE) against
 - Partitioned index
 - Non-partitioned index

CHECK INDEX measurement results and study

We look at several measurements using and not using FlashCopy, and executing CHECK INDEX on concurrent partitions.

SHRLEVEL CHANGE with FlashCopy vs. SHRLEVEL REFERENCE

This test case measures the CPU and elapsed time to compare CHECK INDEX with option SHRLEVEL CHANGE using FlashCopy V2 to CHECK INDEX with option SHRLEVEL REFERENCE. The results of this test case are shown in Figure 6-2. SHRLEVEL CHANGE using FlashCopy reduces the elapsed time dramatically.

The observations of this study are as follows:

- ▶ With the SHRLEVEL CHANGE option, DB2 uses parallel tasks in the execution of the utility. In the case of a partitioned index (PI), the UNLOAD, SORT and CHECKIDX tasks are processed in parallel in each partition of the table space. In the case of a non-partitioned index (NPI), the UNLOAD and SORT tasks are processed in parallel, like for PI, but the CHECKIDX and MERGE phases do not use parallel processing.
- ▶ Most CPU time improvement in SHRLEVEL CHANGE of partitioned index comes from the CHECKIDX phase. Also SHRLEVEL CHANGE of non-partitioned index improves CPU time in the CHECKIDX phase in comparison with SHRLEVEL REFERENCE, but the total CPU time increases in SHRLEVEL CHANGE because of SORT and MERGE phases. The MERGE phase exists only in the case of a non-partitioned index and SHRLEVEL CHANGE.

If we only look at the CHECKIDX phase, we can say that SHRLEVEL CHANGE improved CPU time significantly both in PI and NPI. SHRLEVEL CHANGE makes the total elapsed time decrease dramatically both in PI and NPI. A PI can benefit from the performance improvement of online CHECK INDEX more than an NPI.

- ▶ CHECK INDEX 6 indexes of the table space with SHRLEVEL CHANGE has 18 tasks.
 - UNLOAD 6 indexes in parallel

- SORT 6 indexes in parallel
- CHECK 6 indexes in parallel

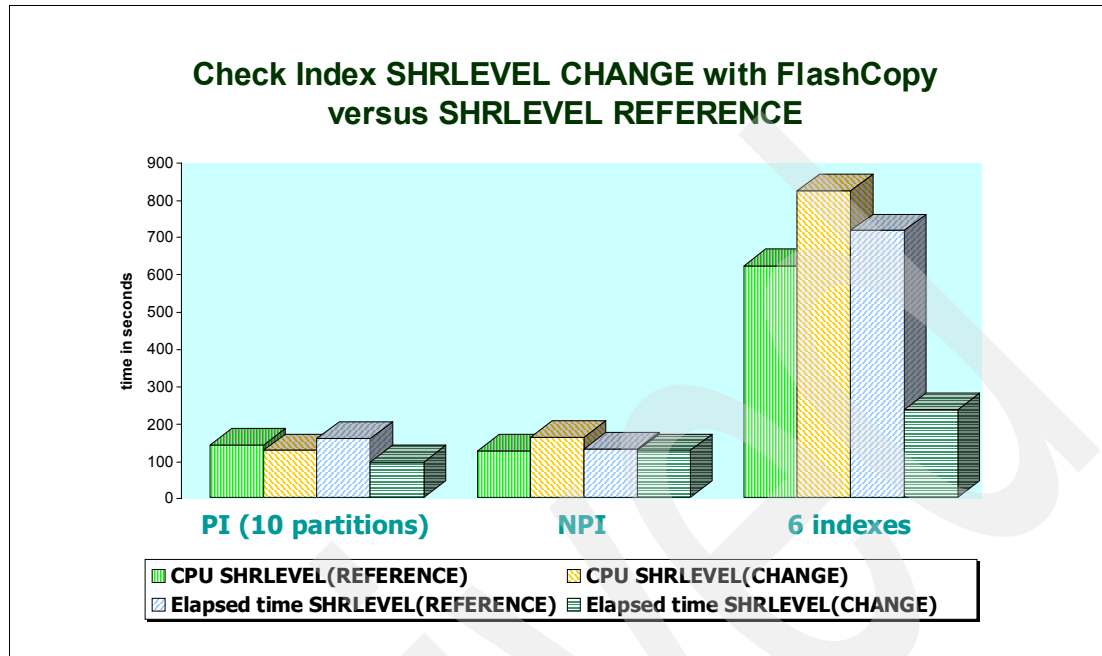


Figure 6-2 SHRLEVEL CHANGE vs. SHRLEVEL REFERENCE with FlashCopy

SHRLEVEL CHANGE w/o FlashCopy vs. SHRLEVEL REFERENCE

This test case measures the CPU and elapsed time comparing CHECK INDEX with option SHRLEVEL CHANGE without FlashCopy to CHECK INDEX with option SHRLEVEL REFERENCE. DFSMSdss does not find FlashCopy V2 enabled in the ESS and reverts to concurrent copy. In this case the elapsed time of SHRLEVEL CHANGE increases significantly. The results are shown in Figure 6-3.

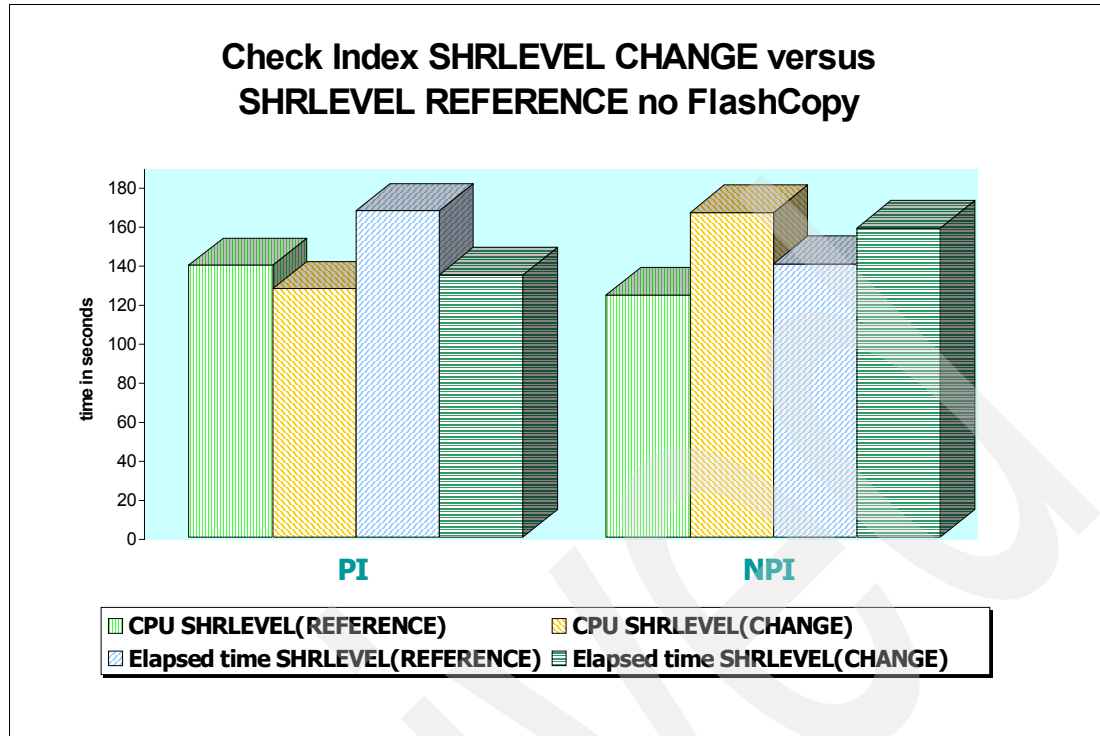


Figure 6-3 SHRLEVEL CHANGE vs. SHRLEVEL REFERENCE w/o FlashCopy

6.1.2 Conclusion

Online CHECK INDEX (SHRLEVEL CHANGE) dramatically cuts down elapsed time. The first reason is that the parallel processing structure minimizes elapsed time in cross checking the data and indexes. The second reason is using FlashCopy V2. Compared to online CHECK INDEX with a non-partitioned index, online CHECK INDEX with a partitioned index could provide better performance. The most important aspect from the availability point of view is that using CHECK SHRLEVEL(CHANGE) with FlashCopy 2 the table and the indexes are available for RW after a few seconds, while with SHRLEVEL(REFERENCE) they are both in UTRO for most of execution time.

Another study shows the better performance of CHECK INDEX with SHRLEVEL CHANGE in one task, when compared to CHECK INDEX with SHRLEVEL REFERENCE as concurrent jobs of each table space partition. It means you should always choose SHRLEVEL CHANGE for shorter CPU and elapsed time, rather than SHRLEVEL REFERENCE, when FlashCopy V2 is available.

The two options have a different behavior:

- ▶ SHRLEVEL REFERENCE Unload phase unloads keys from the indexes and then the Check phase compares sorted keys against the data via tablespace scan.
- ▶ SHRLEVEL CHANGE Unload phase does a tablespace scan extracting keys and then the Check phase compares sorted keys against the keys in the index.

6.1.3 Recommendations

Use online CHECK INDEX whenever availability is needed. You should use online CHECK INDEX in conjunction with FlashCopy V2, otherwise concurrent copy is invoked which increases elapsed time and elongates the time when the data and index are read-only. To

make DB2 take the maximum parallelism in sort tasks it is recommended to estimate the sort work data set size correctly.

Note: APAR PQ92749 and PQ96956 provide the CHECK INDEX availability enhancement.

6.2 LOAD and REORG

SORTKEYS for LOAD and REORG, and SORTDATA for REORG are keywords which generally contribute to better performance. In V7 SORTKEYS and SORTDATA were not the default, so you had to specify explicitly these parameters to activate the methods providing better performance. In V7 if SORTKEYS is not specified, and if data is 100% clustered and there is only one index on the table, LOAD utilities skip the SORT phase. If you want to exploit the functions, you need to change the utility jobs to add these keywords. In V8, SORTKEYS in the LOAD utility is the default and SORTDATA and SORTKEYS are the default in the REORG utility. The SORT phase is enforced in LOAD and REORG in V8.

SORTKEYS cuts down the elapsed time of the index key sort by reducing I/O to the work file, since using SORTKEYS index keys are passed in memory rather than written to the work file. In addition, SORTKEYS provides a method of parallel index build. If there is more than one index on your table space or partition, it reduces the elapsed time of LOAD jobs.

SORTDATA in the REORG utility specifies that the data is to be unloaded by a table space scan, and sorted in clustering order. This allows a consistent execution time for REORG and good performance whether the data is disorganized or not.

6.2.1 LOAD with default performance measurement

The following performance measurement has been done to observe LOAD performance with the default option of SORTKEYS. The purpose of the measurement is to assess the performance benefit of the SORTKEYS method provided as default. In this measurement we run a comparison of LOAD utilities in V7 and V8 over table spaces with 1 index, 2 indexes, and 6 indexes. The following is the description of our test environment.

- ▶ 1 partitioned table with 10 partitions
- ▶ 1,000,000 rows in one partition (total 10 million rows) and 118 byte row size
- ▶ 26 columns of CHAR, INTEGER, and DECIMAL data type

Figure 6-4 summarizes the results.

LOAD executed against the table space with just one index shows up to 11% CPU degradation and 7% elapsed time degradation in V8 compared to V7. When more than 1 index exists, V8 performs better in terms of CPU and elapsed time. The dramatic elapsed time reduction in V8 with 6 indexes comes from parallel processing. One thing to be considered is that more work file space is needed to accommodate parallel processing.

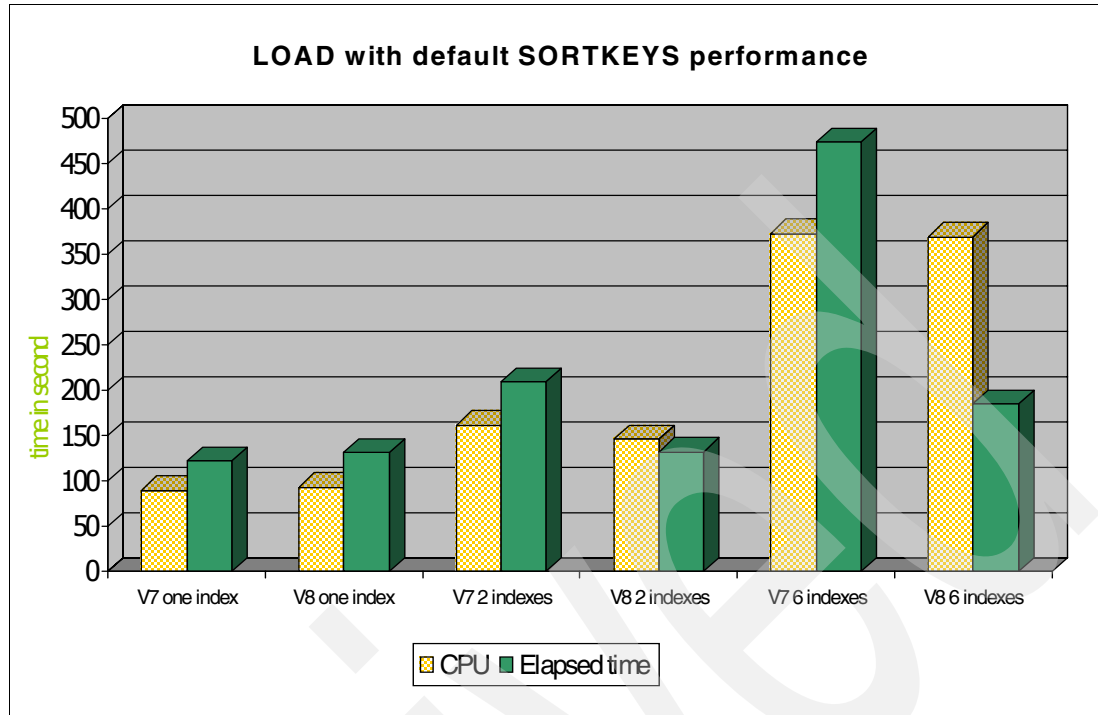


Figure 6-4 LOAD with default SORTKEYS

6.2.2 Conclusion

In V8, SORT parameters are always enforced, this provides performance improvement because of shorter I/O in index key sort and parallel index build. However, if the index keys are already sorted in perfect order or there is only one index in the table space, SORTKEYS does not provide much advantage.

6.2.3 Recommendations

You take advantage of the high performance of SORT parameters, if more than one index exists in a table space.

To improve SORT performance during LOAD, you can also specify the appropriate value in the SORTKEYS option. The SORTKEYS option is an estimate of the number of keys for the job to sort, so that the index keys are passed in memory and avoid I/Os to the work file. To read how to estimate the number of keys, refer to topic *Improved performance with SORTKEYS* in *DB2 UDB for z/OS Version 8 Utility Guide and Reference*, SC18-7427.

Also you should not specify sort work data set like SORTWKxx DD statements, while using SORTKEY, make sure DB2 uses memory instead of sort work data sets.

See the informational APAR II14047 for useful information on the changes of using DFSORT by DB2 V8 utilities.

Note: With PTF UK03983 for APAR PK04076, LOAD sort is avoided for LOAD SHRLEVEL NONE if data is sorted and only one index exists.

6.3 LOAD and UNLOAD delimited input and output

In V7, the LOAD utility input file is required to be of positional format. If you want to move data to DB2 for z/OS, you must write a program to convert the data into the positional format that the LOAD utility can understand, or use INSERT statements, thereby not exploiting the performance advantage that the LOAD utility offers for a large amount of data.

In DB2 V8, the LOAD utility is enhanced to accept data from a delimited file. The UNLOAD utility is also enhanced to produce a delimited file when unloading the data. These enhancements help to simplify the process of moving or migrating data into and out of DB2 for z/OS.

Reformatting data for DB2 (before V8)

Reformatting data for DB2 is a very time consuming procedure. Prior to DB2 V8, you can load data only from an input flat file, either a partitioned data set or a regular flat file. The file must include length information for those fields that become VARCHAR columns in a DB2 table. Reformatting the data for DB2 means adding 2 bytes of binary data to the beginning of such a field. These 2 bytes contain a binary number representing the length of the data within the field. The following steps were needed to reformat data for DB2 for z/OS.

1. Unload the data from a table residing on another relational database and transmit the file to z/OS and store it (either as member in a PDS, or as a sequential file).
2. Write a program, possibly using REXX, to determine the length of the variable character data and generate the 2 byte binary length fields. Also the program should output a file with the positions and length of the fields which can be useful to set up LOAD utility statements.
3. Prepare for an input control file used for a program, describing where is the beginning of each field and what kind of data will be encountered in input file records.
4. Run the program, input the data from the other RDBMS and the input control file.
5. Setup the LOAD utility control statement using the output of the program and specifying the generated input data in INDDN in the JCL statement, and then run the LOAD utility to load data into DB2. See Example 6-2 for an example of the LOAD statement.

Example 6-2 Sample LOAD statement

```
LOAD DATA INDDN(CPRSRECS) LOG NO RESUME YES
  INTO TABLE PAOLOR2.DATA_TYPE
      (DATA_TYPE POSITIN(1) VARCHAR,
       DESCRIPTION POSITION(9) VARCHAR)
```

Delimited file input and output

In V8, the LOAD utility accepts a delimited file, and the UNLOAD utility can produce a delimited file. Most relational databases including DB2 on Linux, UNIX, Windows, Oracle® and Sybase can unload data in delimited format, where each record is a row, and columns are separated by commas, for example. So this enhancement eases dramatically the import or export of a large amount of data from other operating system platforms to DB2 for z/OS and vice versa. And also this eliminates the requirement to write a program to convert non-z/OS platform data into positional format for the LOAD utility, so you can avoid the data reformatting process described above.

For instance, if you want to move data from a spreadsheet from a Windows workstation to DB2 for z/OS, you need to take the following steps:

- Save the spreadsheet data in comma separated value (CSV) format

- ▶ Upload the saved data to the host in one of the following formats:
 - text (the data is converted from ASCII to EBCDIC before it is stored on the host)
 - binary (the data is stored in ASCII)

To upload the data, you can either use your terminal emulator's file transfer program, or use FTP.

- ▶ Create a DB2 table with the appropriate number of columns and data types (consistent with the number of cells and the types of data stored in a spreadsheet), either as an EBCDIC (default option), or an ASCII table.
- ▶ Use the LOAD utility with FORMAT DELIMITED specified as follows:
 - If the spreadsheet data is uploaded to the host in text format, specify it as shown in Example 6-3.

Example 6-3 LOAD DATA text format

```
LOAD DATA INTO TABLE tablename
FORMAT DELIMITED
```

- If the spreadsheet data is uploaded to the host in binary format, specify it as shown in Figure 6-4.

Example 6-4 LOAD DATA binary format

```
LOAD DATA INTO TABLE tablename
FORMAT DELIMITED ASCII
```

If you want to move the data from DB2 to a spreadsheet, you take the following steps:

- ▶ Use the UNLOAD utility with DELIMITED specified as follows:
 - If you want the unloaded data to be in EBCDIC format, specify as shown in Example 6-5. The data and the delimiters are unloaded in EBCDIC. Transmit the data to the workstation to be used by the spreadsheet in text format.

Example 6-5 UNLOAD DATA in EBCDIC format

```
UNLOAD DATA FROM TABLE tablename
FORMAT DELIMITED EBCDIC
```

- If you want the unloaded data to be in ASCII format, specify as shown in Example 6-6. The data and the delimiters are unloaded in ASCII. Transmit the data to the workstation to be used by the spreadsheet in binary format.

Example 6-6 UNLOAD DATA in ASCII format

```
UNLOAD DATA FROM TABLE tablename
FORMAT DELIMITED ASCII
```

See *DB2 UDB for z/OS Version 8 Utility Guide and Reference*, SC18-7427, for more information about this topic.

6.3.1 Performance

We have tested loading the data from a workstation system with delimited format into DB2 and UNLOAD from DB2 to the data which is used by spreadsheet. The LOAD and UNLOAD of internal and external format data have been measured to compare with LOAD and UNLOAD delimited data.

We measured the CPU time for both LOAD and UNLOAD using three variations (internal format, external format, and delimited format) and our tested hardware was a z900 Turbo. There are 10 columns, including 4 CHAR, 1 VARCHAR, 3 DECIMAL, 1 SMALLINT and 1 DATE and no index exists in our test table. Delimited requires using external format for numeric. For example, INTEGER or SMALLINT is treated as if they were INTEGER EXTERNAL, and DECIMAL is treated as DECIMAL EXTERNAL. DATE is always treated as EXTERNAL, which causes the DATE fields to always have a large impact on UNLOAD performance, even though you might prefer internal format in several cases like migrating the DATE data to a different table.

The results are shown in Table 6-1.

Table 6-1 Delimited Load and Unload CPU comparisons

	internal	external	delimited	delta (del/ext)
LOAD (sec.)	11.41	16.3	26.5	+63%
UNLOAD (sec.)	13.3	30.57	34.75	+14%

External is more expensive than internal because DECIMAL and SMALLINT data in external format has to contain a character string that represents a number in specific forms. However, it should be noted that external performs significantly worse than internal, especially with UNLOAD.

6.3.2 Conclusion

When compared to LOAD and UNLOAD of external undelimited format, delimited format LOAD and UNLOAD have more impact on performance. It makes sense because in UNLOAD delimited format DB2 has to build the output data to be either in character string or numeric external format, and also has to add column delimiters (default is comma) and a double quotation mark (") for the character string and a period (.) for the decimal point character. LOAD costs more because of the need to scan the VARCHAR data to define the length.

In our test UNLOAD of external format significantly increased CPU time compared to internal format.

6.4 RUNSTATS enhancements

In this section we first show some performance improvements in executing RUNSTATS with different options, and then we discuss the new function which allows us to gather distribution statistics.

6.4.1 General performance improvements

Table 6-2 shows the measurements for RUNSTATS of a table with 10 million rows and 6 indexes.

The measurements were implemented on a IBM z900 Series 2064 processor.

Table 6-2 Summary of RUNSTATS improvements

RUNSTATS option	DB2 V7 (sec.)		DB2 V8 (sec.)		Delta CPU	Delta ET
	CPU	Elapsed	CPU	Elapsed	%	%
TABLESPACE	15.7	39	8.6	40	-45.23	2.57

RUNSTATS option	DB2 V7 (sec.)		DB2 V8 (sec.)		Delta CPU	Delta ET
	CPU	Elapsed	CPU	Elapsed	%	%
TABLE	137	140	115	118	-16.06	-15.72
TABLE and 6 INDEXES	189	193	168	173	-11.12	-10.37

6.4.2 DSTATS

The RUNSTATS utility has been enhanced in V8 and can collect distribution statistics on any column or groups of columns, indexed or non-indexed columns, specified by table level. This enhancement lets the DB2 optimizer calculate more accurately the filter factor of a query and helps to improve SQL performance.

For versions prior to V8 you can use a tool called DSTATS (Distribution Statistics for DB2 for OS/390) which you can download from the Web site:

<http://www.ibm.com/support/docview.wss?uid=swg24001598>

This tool is available for use with DB2 V5, V6, and V7. DSTATS is a standalone DB2 application program, written in PL/I, which contains dynamic and static SQL statements. The tool collects additional statistics on column distributions and column frequencies that were not collected by RUNSTATS before DB2 V8. With DB2 V8, equivalent function is provided by the RUNSTATS utility. The enhanced RUNSTATS utility collects additional information on column distribution and frequencies, and updates the DB2 catalog with the more accurate statistics. It greatly enhances the performance of the query if there are non-uniform data distributions in your tables or indexes.

6.4.3 Description of distribution statistics in V8

When there is a skewed distribution of values for a certain column, this can cause bad response time of a query, especially in ad hoc query applications. The collection of correlation and skew statistics for non-leading indexed columns and non-indexed columns can help the optimizer to compute accurate filter factors. This leads to better access path selection and improves query performance.

The enhancement of the RUNSTATS utility allows you to use RUNSTATS to collect distribution statistics on any column in your tables, whether or not the columns are part of an index, and whether or not the columns are the leading columns of an index. It updates the DB2 catalog with the specified number of highest frequencies and optionally with the specified number of lowest frequencies. The new functionality also optionally collects multi-column cardinality for non-indexed column groups and updates the catalog.

The summary of the new functionality of RUNSTATS is as follows:

- ▶ Collect distribution statistics on any column, or groups of columns, indexed or non-indexed, specified at the table level
- ▶ Frequency distributions for non-indexed columns or groups of columns
- ▶ Cardinality values for groups of non-indexed columns
- ▶ LEAST frequently occurring values, along with MOST frequently occurring values, for both index and non-indexed column distributions (DB2 V7 only gathers the most frequently occurring value)

Options for cardinality and distribution statistics

To support these new functionalities, several new options are introduced in the RUNSTATS utility. New keywords COLGROUP, LEAST, MOST and BOTH enable the collection of distribution statistics on any table column and group of columns. The existing keywords FREQVAL and COUNT also can be used. You can specify the following options for collecting cardinality and distribution statistics:

- ▶ **COLGROUP:** When you specify the COLGROUP keyword, the specified set of columns is treated as a group and RUNSTATS collects correlation statistics for the specified column group. You can specify the COLGROUP keyword repeatedly, if you want to collect statistics on a different set of columns.
- ▶ **FREQVAL:** This keyword controls the collection of frequent value statistics. These are collected either on the column group or on individual columns depending on whether COLGROUP is specified or not. You can specify the FREQVAL keyword together with the NUMCOLS keyword when you run RUNSTATS INDEX. But when you run table-level statistics, FREQVAL can only be specified together with the COLGROUP keyword. If FREQVAL is specified, then it must be followed by the keyword COUNT.
- ▶ **COUNT:** This indicates the number of frequent values to be collected. Specifying an integer value of 20 means to collect 20 frequent values for the specified columns. No default value is assumed for COUNT. This can be optionally followed by the keyword MOST (which is the default), LEAST, or BOTH.

Note: The following statement:

```
RUNSTATS TABLESPACE DBGBTEST.TSGBRUNS  
TABLE(ALL) INDEX(ALL) KEYCARD
```

does collect FREQVAL values on the indexes by default.

If you do not want to collect **any** FREQVAL data, you need to specify the following options:

```
RUNSTATS TABLESPACE DBGBTEST.TSGBRUNS  
TABLE(ALL) INDEX(ALL) KEYCARD  
FREQVAL NUMCOLS 1 COUNT 0
```

- ▶ **MOST:** The **most** frequent values are collected when the keyword MOST is specified.
- ▶ **LEAST:** The **least** frequent values are collected when the keyword LEAST is specified.
- ▶ **BOTH:** The **most** frequent values and the **least** frequent values are collected when the keyword **BOTH** is specified.

Options for using sort work data sets

When you are collecting distribution statistics for column groups or collecting statistics on a data-partitioned secondary index, for example, it is necessary for DB2 to use sort work to sort the statistics. As a new option of V8, you can specify the SORTDEVT and SORTNUM keywords for using dynamic allocation of the sort work data sets:

- ▶ **SORTDEVT:** This specifies the device type that DFSORT uses to dynamically allocate the sort work data set.
- ▶ **SORTNUM:** This specifies the number of required sort work data sets that DFSORT is to allocate.

If you do not use dynamic allocation for the SORT program, and if you collect the statistics for which sort data is required, you have to define data sets through JCL. You need to specify ST01WKnn and STATWK01 in DD statements depending on the statistics you collect:

- ▶ **ST01WKnn:** You need to specify the data set name for the temporary data sets of sort input and output when collecting statistics on at least one data-partitioned secondary index.
- ▶ **STATWKnn:** You need to specify the data set name for the temporary data sets of sort input and output when collecting distribution statistics for column groups.

Note: To calculate the approximate size (in bytes) of the ST01WKnn data set, use the following formula:

$$2 \times (\text{maximum record length} \times \text{numcols} \times (\text{count} + 2) \times \text{number of indexes})$$

The values in the formula are as follows:

- ▶ **Maximum record length:** The maximum length of the SYSCOLDISTATS record on which RUNSTATS is to collect frequency statistics. You can obtain this value from the RECLENGTH column in SYSTABLES.
- ▶ **Numcols:** The number of key columns to concatenate when you collect frequent values from the specified index.
- ▶ **Count:** The number of frequent values that RUNSTATS is to collect.

6.4.4 Performance

When you plan to use distribution statistics, make sure that the following maintenance is applied for correct functionality and good performance:

- ▶ PTF UQ88754 for APAR PQ88375 provides the right value of cardinality for partitioned table spaces.
- ▶ PTF UQ91099 for APAR PQ87509 is required for correctness of frequency values.
- ▶ PTF UQ93177 for APAR PQ90884 is required for better (200%) performance.

DSTATS performance comparison of V7 and V8 with PQ90884

In this test we first run RUNSTATS in V7 followed by the standalone DSTATS tool. We measure class 1 elapsed and CPU time and get the total time of both. In the second case we measure V8 RUNSTATS with distribution statistics where we collect equivalent statistics to those taken in V7 with the DSTATS tool. Before running this case we have applied the PTF for PQ90884.

We use an internal query workload and the test table contains about 931k rows. The description of the test environment is:

- ▶ DB2 V7 and V8 (non-data sharing)
- ▶ Table: 1 test table in 1 table space
- ▶ Number of pages: 77601
- ▶ Number of rows: 931,174
- ▶ Number of indexes: 4 (1 partitioned index, 3 non-partitioned indexes)
- ▶ Number of partitions of table space: 6
- ▶ 2 columns are non-indexed columns (used for statistics)
 - Column1 cardinality=15104
 - Column2 cardinality=24

To explain our test case more clearly, let us show you a part of our control statements for testing. We have done equivalent RUNSTATS in V7 and V8, and we have taken equivalent statistics in V7 with the DSTATS tool, and in V8 with RUNSTATS. We are specifying 2 columns for statistics and both are non-indexed columns. The description of the statistics is:

- ▶ Number of column groups: 1
- ▶ Number of columns in column group: 2 (2 columns in one table)
- ▶ Frequent values for distribution statistics: 10 most and 10 least frequent values

Example 6-7 shows the sample statement of RUNSTATS in V7.

Example 6-7 RUNSTATS statement in V7

```
//SYSIN DD *
  RUNSTATS TABLESPACE INSQDB01.EVENTS TABLE(EVEN)
  REPORT(YES)
```

Example 6-8 shows the sample statement of DSTATS tool used in V7.

Example 6-8 DSTATS tool in V7

```
//SYSIN DD *
VALUES 10,10
CARDINALITY MUST
COLCARDF1 SCAN
CARDF ALLOW 0
USRT001.EVEN.CLID, TXCD
```

Example 6-9 shows the RUNSTATS job used in V8 to collect distribution statistics.

Example 6-9 RUNSTATS with distribution statistics in V8

```
//SYSIN DD *
  RUNSTATS TABLESPACE INSQDB01.EVENTS TABLE(EVEN)
  COLGROUP(CLID, TXCD) FREQUAL COUNT 10 BOTH
  REPORT(YES)
```

The result of this measurement is shown in Figure 6-5. If we compare the V7 RUNSTATS plus DSTATS tool, with the V8 RUNSTATS distribution statistics, we notice that both CPU and elapsed time are equivalent or better in V8.

V8 RUNSTATS with distribution statistics shows a 15% improvement on elapsed time (51.5 sec. vs. 44.5 sec.) and a 10% improvement on CPU time (43.8 sec. vs. 39.6 sec.).

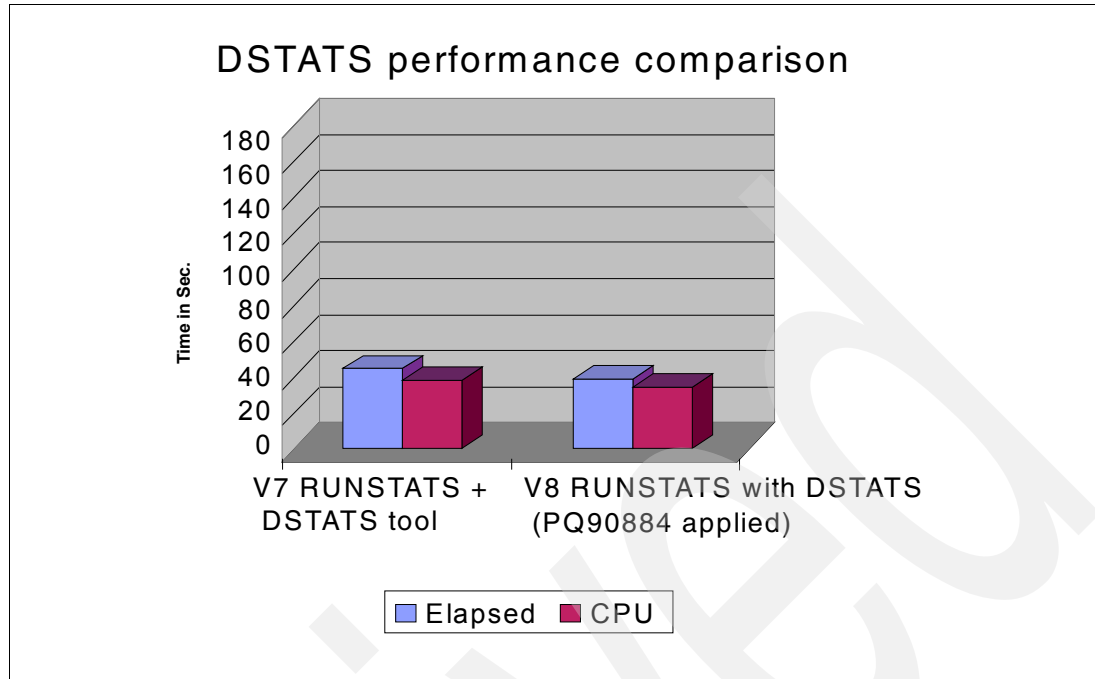


Figure 6-5 Distribution statistics performance comparison

DSTATS measurement with different number of columns

As an additional study of the cost of RUNSTATS with distribution statistics, we have tested in both V7 and V8 (with PQ90884) several cases collecting distribution statistics of different numbers of columns or column groups. As a base, we measure simple RUNSTATS against a table, and then measure RUNSTATS against a table and all related indexes. After that, we collect distribution statistics against different numbers of columns and column groups. We use the same environment as the previous test case and the following is the column information used for the distribution statistics:

- ▶ 3 columns are non-indexed columns (used for statistics)
- ▶ column1 cardinality=15104
- ▶ column2 cardinality=24
- ▶ column3 cardinality=41

We have measured the following six test cases:

1. RUNSTATS against one table
2. RUNSTATS against one table and *indexes*
3. RUNSTATS against one table and distribution statistics on *one column*
4. RUNSTATS against one table and distribution statistics on *one column group* specifying 2 columns
5. RUNSTATS against one table and *indexes*, and distribution statistics on *one column group* specifying 2 columns
6. RUNSTATS against one table, and distribution statistics on two column groups in which we specify 2 columns in the first column group and specify 1 column in the second column group.

Table 6-3 shows the results of the test cases run in V7. From Case 3 to Case 6, we ran the RUNSTATS utility followed by the DSTATS tool. So class 1 elapsed and CPU time in these

cases is the total time of RUNSTATS plus DSTATS. The overhead of DSTATS is derived from the percentage of DSTATS time over RUNSTATS time (DSTATS time / RUNSTATS time *100).

Table 6-3 RUNSTATS with DSTATS measurement data in V7

	Case 1 Runstats table	Case 2 RUNSTAT S table, idx	Case 3 RUNSTAT S table+ 1 column	Case 4 RUNSTAT S table+ 1 colgrp w/ 2 cols	Case 5 RUNSTAT S table idx+ 1 colgrp w/ 2 cols	Case 6 RUNSTAT S table+ 2 colgrp w/ 2 ,1 cols
Class 1 elapsed (sec.)	25.91	29.70	31.09	38.80	42.59	43.08
Class 1 CPU time (sec.)	19.31	20.94	21.97	28.85	30.49	30.80
DSTATS CL1 ELT (sec.)	N/A	N/A	5.18	12.88	12.88	17.17
DSTATS CL1 CPU (sec.)	N/A	N/A	2.66	9.54	9.54	11.49
DSTATS Overhead ELT (%)	N/A	N/A	20	49.7	43.4	66.3
DSTATS Overhead CPU (%)	N/A	N/A	13.8	49.4	45.6	59.5

Take a look at the DSTATS overhead in Case 3 (collecting distribution statistics on 1 column): DSTATS overhead is 20% for elapsed time. In V7 the overhead of using the DSTATS tool shows a high percentage in general, because of the cost of running and binding DSTATS as a standalone program on top of the cost of RUNSTATS. The largest overhead appears when going from 1 column to 2 in one column group (case 3 to 4). In this case there is 24.8% degradation in elapsed time and 31.3% degradation in CPU time.

When the number of column groups goes from 1 to 2 (case 4 to 6), there is 11% performance degradation in elapsed time and 6.8% degradation in CPU time.

Table 6-4 shows the results of the same test cases with V8 RUNSTATS with distribution statistics. DSTATS times are now the difference between RUNSTATS with distribution statistics and RUNSTATS only time (for example, DSTATS in Case 3 is the difference of Case 3 and Case 1 in terms of class 1 time).

Table 6-4 RUNSTATS with DSTATS measurement data in V8

	Case 1 RUNSTATS table	Case 2 RUNSTATS table idx	Case 3 RUNSTATS table 1 column	Case 4 RUNSTATS table 1 colgrp w/ 2 cols	Case 5 RUNSTATS table idx 1 colgrp w/ 2 cols	Case 6 Runstats table 2 colgrp w/ 2 ,1 cols
Class 1 elapsed (sec.)	26.04	29.26	28.00	27.99	30.59	39.67
Class 1 CPU time (sec.)	19.10	21.62	24.50	24.32	26.62	32.22
DSTATS CL1 ELT (sec.)	N/A	N/A	1.96	1.94	1.32	13.63
DSTATS CL1 CPU (sec.)	N/A	N/A	5.39	5.22	5.00	13.12
DSTATS Overhead ELT (%)	N/A	N/A	7.5	7.5	4.5	52.3
DSTATS Overhead CPU (%)	N/A	N/A	28.3	27.3	23.1	68.7

In V8 we can observe the CPU overhead when running RUNSTATS specifying distribution statistics. In case 3, which is the base scenario of DSTATS, we see a 28.3% overhead in CPU time, and in case 4 (adding 1 column to the COLGROUP) we see 27.3%.

However, in Case 6 (2 column groups) it is 68.7%. We can conclude that, in V8, increasing the number of column groups drives the CPU cost higher than increasing the number of columns.

There is almost no difference when 1 column is added in 1 column group (Case 3 to 4). On the other hand, there is 29.7% degradation in elapsed time and 21% degradation in CPU time of the total RUNSTATS time, when the column group of statistics goes to 2 (Case 4 to 6).

In general, V8 RUNSTATS cuts down the elapsed time well compared to V7.

Access path improvement

To verify the usefulness of DSTATS we have tested the access path improvement of 6 queries out of 25 queries of the query workload after running distribution statistics. We have gathered distribution statistics on 10 columns manually picked. After that, most of the queries have shown positive access path changes.

Bind cost

You need to consider that RUNSTATS executions, by adding the frequency values in the SYSIBM.SYSCOLDIST catalog table, cause high bind cost. Especially high CPU consumption occurs for dynamic prepare when FREQUAL COUNT in SYSCOLDIST is greater than 100. In this case a large portion of CPU time is used to remove duplicate frequency values. PTF UK00296 for APAR PQ94147 is meant to reduce such large bind cost.

Statistics Advisor tool

This tool is a function of the new Visual Explain and it can help you cut down the effort in determining what kind of distribution statistics you need to run and set up the RUNSTATS control statements. For information about the Statistics Advisor tool refer to 3.15.3, “Statistics Advisor” on page 116.

6.4.5 Conclusion

V8 RUNSTATS with distribution statistics achieves a better performance compared to V7 RUNSTATS and the DSTATS tool. You no longer need to run the separate DSTATS program to collect distribution statistics, in V8 RUNSTATS will do it and cut down CPU and elapsed time. Measurements show that increasing the number of columns or column groups to collect statistics may significantly impact CPU and elapsed time. However, access path improvements provided by distribution statistics can be quite remarkable.

6.4.6 Recommendations

To get good performance when using RUNSTATS distribution statistics in V8, it is important to be current with maintenance, apply the fixes for PQ90884 and PQ94147 (dynamic prepare). We also recommend you use the *Statistics Advisor* tool to define the right statistics for your application: The Statistics Advisor suggests what RUNSTATS and DSTATS are needed for your queries. The CPU overhead observed when there are hundreds of frequency values in SYSCOLDIST can be reduced by deleting unnecessary information from the catalog table.

6.5 Cross Loader

The Cross Loader function enables you to use a single LOAD job to transfer data from one location to another location or from one table to another table at the same location. You can use either a local server or any DRDA-compliant remote server as a data input source for populating your tables. Your input can even come from other sources besides DB2; you can use IBM Information Integrator for access to data from other relational or non-relational data sources. For more information about the cross loader function of the LOAD utility, see *DB2 UDB for z/OS Version 8 Utility Guide and Reference*, SC18-7427.

6.5.1 Performance APAR

APAR PQ84160, opened on V7, has optimized the code reducing the path length by a large percentage. Prior to this fix, the cross loader took longer than the normal LOAD and UNLOAD utilities. With two times performance improvement provided by this rework, the cross loader is now at the same level as LOAD and UNLOAD in terms of performance.

PTFs are available for both V7 and V8:

- ▶ V7: UQ91416
- ▶ V8: UQ91422

Networking and e-business

This chapter describes enhancements related to distributed processing and e-business applications.

- ▶ **DDF enhancements:** A requester database alias, a server location alias and member routing in a TCP/IP network are provided in DDF. The DRDA architecture allows larger query blocks: the number of network exchanges between the workstation and a DB2 for z/OS V8 server is about half that when accessing the DB2 for z/OS V7 server.
- ▶ **Multi-row FETCH and INSERT in DRDA:** Multi-row FETCH and INSERT have a significant positive impact on performance in a DRDA client/server environment. However, that performance impact differs when DB2 for z/OS V8 acting as a DRDA application server is being accessed from a DB2 Connect Client running on Linux, Unix and Windows or another DB2 for z/OS V8 acting as a DRDA application requester.
- ▶ **WebSphere and DB2 Universal Driver for SQLJ and JDBC:** In the past, access to a Linux, UNIX and Windows server and a z/OS server used different database connection protocols. To provide transparent access across the DB2 Family, these protocols are now standardized, and all of them use the Open Group's DRDA Version 3 standard. This new architecture is called the Universal Driver. The first deliverable of this new architecture is the IBM DB2 Universal Driver for SQLJ and JDBC Version 1.0, also known as the IBM *Java Common Connectivity*.
- ▶ **ODBC enhancements:** Several DB2 ODBC enhancements such as Unicode support, improved authentication, long name support, 2 MB SQL statements and SQLcancel are briefly presented in this section.
- ▶ **XML support in DB2 for z/OS:** In DB2 V7, you could use the XML extender feature for manipulating XML data in and out of DB2. The SQL/XML Publishing Functions become available in DB2 V8 and provide a set of SQL built-in functions that allow applications to generate XML data from relational data with high performance. It reduces application development efforts in generating XML data for data integration, information exchange, and web services, thus enhancing the leadership position of DB2 for z/OS as the enterprise database server. This is the first step DB2 for z/OS takes to more tightly integrate XML into the database engine.
- ▶ **Enterprise Workload Manager™:** Enterprise Workload Manager (EWLM) enables you to automatically monitor and manage multi-tiered, distributed, heterogeneous or homogeneous workloads across an IT infrastructure to better achieve defined business

goals for end-user services. We show the preliminary tests to evaluate the impact on throughput of an EWLM implementation on a server combination WebSphere/AIX with DB2 for z/OS V8.

7.1 DDF enhancements

A requester database alias, a server location alias and member routing in a TCP/IP network are provided in DDF.

7.1.1 Requester database alias

During this discussion, in which we want to differentiate between DB2 for z/OS and DB2 for Linux, UNIX, and Windows, we call the latter DB2 for Multiplatforms.

A DB2 for Multiplatforms database is known in the network by its database name. This database name is used by applications to connect to an instance of a for Multiplatforms database. A DB2 for z/OS requester uses a location name to connect to an instance of a DB2 for Multiplatforms database. A location name maps to a DB2 for Multiplatforms database name. When a DB2 for Multiplatforms database is deployed to a large number of servers, in some cases in the thousands, the database is commonly deployed with the same name in different locations.

The current Communications Database catalog tables require a one to one mapping with a location and a network address. A location can only be defined to one network address. To allow a z/OS requester to access multiple DB2 for Multiplatforms databases with the same database name, a new column is added to SYSIBM.LOCATIONS table called DBALIAS. This column allows a database administrator to specify multiple locations for a DB2 for Multiplatforms database deployed on multiple locations. A z/OS application would then specify a location name to access an instance of a DB2 for Multiplatforms database and the SYSIBM.LOCATIONS would map the location name to the DB2 for Multiplatforms database name which is then used to connect to the DB2 for Multiplatforms database.

7.1.2 Server location alias

A DB2 server is known in the network by its location name. This name is used by applications to identify an instance of a DB2 subsystem or a group of data sharing DB2 subsystems. When two or more DB2 subsystems are migrated to a single data sharing group, multiple locations must be consolidated into a single location. When the locations are consolidated, all applications using the old location name need to be changed to access the new data sharing group location name.

When there is a large number of remote applications deployed across the network, in some cases in the thousands, it is impossible to effectively change each application simultaneously to use the new location name. In order to support the migration from multiple locations to a single location, eight location alias names can be defined for a DB2 subsystem or group of data sharing DB2 subsystems.

A location alias is another name that a requester can use to access a DB2 subsystem. DB2 accepts requests from applications that identify the DB2 server with its location name or any of its location alias names. The Change Log Inventory utility allows a user to define up to eight alias names in addition to the location name. The Print Log Map utility prints any location alias names defined for the DB2 subsystem.

7.1.3 Member routing in a TCP/IP Network

In a sysplex environment, setting up rows in the new SYSIBM.IPLIST table at a requester in conjunction with defining location aliases at the server provides the ability for a DBA to override the default TCP/IP workload balancing. Currently, connections are automatically

balanced across all members. With the new SYSIBM.IPLIST table, a DB2 for z/OS application requester can define a specific member or a subset of members in a group to route requests.

At the server, location alias names do not need to be defined to allow these applications to access the DB2 server with the alternate location names. Aliases are only needed for member subsetting when the application requester is non-DB2 for z/OS (like DB2 Connect). The location DB2PLEX might represent three DB2 subsystems in a group: DB2A, DB2B, and DB2C. As in previous versions, the location DB2PLEX is set up to use the group domain name to route requests to all available members. If a DBA wants to provide member routing to DB2A and DB2B but not to DB2C, a location name DB2PLEXA is defined in SYSIBM.LOCATIONS and a row is inserted for DB2A and DB2B in the SYSIBM.IPLIST table with their member domain name. If an application wants to route requests to either DB2A or DB2B, it connects using location DB2PLEXA. When an application wants to route requests to the least loaded member, it connects using location DB2PLEX.

With this new service, applications can now route requests using a name different than the group location name. If the application contains SQL statements that qualify object names with a name other than the group location name, a location alias needs to be defined. In the above example, DB2A and DB2B need to define location alias DB2PLEXA using the Change Log Inventory utility. When an application connects to DB2PLEXA and issues a

```
SELECT * FROM DB2PLEXA.COLLECTION.TABLE;
```

the DB2 member recognizes the fully qualified object as a local object.

7.1.4 DRDA allows larger query blocks

The DRDA architecture has been enhanced to allow query blocks up to 10 MB.

Description

A query block is a group of rows that fits into a (query) block and is sent as a single network message. The default query block size used by DB2 in Version 7 is 32 KB. The number of rows that DB2 puts into a query block depends on the row length, as well as the OPTIMIZE FOR n ROWS clause. Blocking is used by both DB2 Private Protocol and DRDA. When blocking is not used, DB2 will send a single row across the network.

To support the larger network bandwidths and customer requirements to control the amount of data returned on each network request, the DRDA architecture has been enhanced to allow query blocks up to 10 MB. This way, a DB2 for z/OS V8 requester can ask for a query block size of up to 10 MB from a DB2 for z/OS V8 server. This allows the requester to better manage and control the size of blocks returned from DB2.

DB2 Connect V8 and DB2 V8 clients can specify a query block size of up to 65,535 in size but only the DB2 for z/OS V8 DDF server will support this. Thus, one will notice that the number of network exchanges between the workstation and DB2 for z/OS V8 server is about half that when accessing the DB2 for z/OS V7 server. DB2 V7 can only support a query block size up to 32,767.

Performance

Performance measurement description

In this measurement we used:

- ▶ DB2 for z/OS, V7 and V8
- ▶ z/OS Release 1.4
- ▶ z900
- ▶ DB2 Connect EE, Client V8 on 1.3 GHz P690 with AIX

The following query was utilized:

```
SELECT X, Y, Z FROM CUSTOMER OPTIMIZE FOR 240001 ROWS FOR FETCH ONLY
```

All column data is CHAR or VARCHAR

240000 rows at 133 bytes per row were returned resulting in a result set size of 31,920,000 bytes.

Performance measurement results

See Table 7-1 for the results.

Table 7-1 DB2 connect evaluation DB2 V7 vs. DB2 V8

	Elapsed time (sec.)	Transfer rate (MB/sec.)
DB2 V7	7.111	4.28
DB2 V8	6.176	4.93

DB2 Connect and client V8 performed better for large result sets with DB2 V8 as the DRDA application server compared to DB2 V7. The retrieval of data from DB2 V8 is 15.2% faster than from DB2 V7. This improvement is achieved through the support of a 64 KB query block size and the automatic use of multi-row fetch by DDF in DB2 V8.

Conclusion

The 64 KB query block size can reduce the network transfers by 50% in DB2 V8 compared to DB2 V7.

Recommendation

In order to benefit from the larger query block size supported by DB2 V8 when dealing with large result sets, you should update the DB2 client configuration parameter RQRIOBLK value to 65,635 since the default is 32,767.

7.1.5 DB2 Universal Driver for SQLJ and JDBC

DB2 UDB for Multiplatforms Version 8.1 was the first member of the DB2 Family to introduce the new JDBC Driver, called the IBM DB2 Universal Driver for SQLJ and JDBC. This new Java driver architecture is the future basis of all DB2-related Java efforts. It supports what is called the JDBC type 2 and type 4 Driver.

These drivers are currently supported on DB2 UDB for Linux, Unix, and Windows and DB2 for z/OS Version 8. They are also available for DB2 for z/OS and OS/390 Version 7, through the maintenance stream via UQ85607.

Note that with PTF UQ94429 (DB2 V7) and UQ94430 (DB2 V8) for APAR PQ93458, release 2.3 of the DB2 Universal Driver became available.

Refer to 7.3, "WebSphere and the DB2 Universal Driver" on page 292 for more information.

7.2 Multi-row FETCH and INSERT in DRDA

Client applications that use the DB2 CLI/ODBC API could already use multi-row fetch and insert on the client side. The function SQLExtendedFetch() performs an array fetch of a set of rows. Also for multiple insert, delete, or update, DB2 CLI/ODBC provides an array input

method. Prior to DB2 V8 those array fetch and insert requests only affect the processing between the application program and the DB2 CLI/ODBC Driver on the workstation.

DB2 V8 introduces multi-row fetch and insert. This function is by default enabled for DRDA requests and client application array requests. In Figure 7-1 we show how the usage of multi-row operations saves cross-memory trips between the distributed address space and the database engine. This improvement can significantly reduce elapsed time and CPU cost on the server.

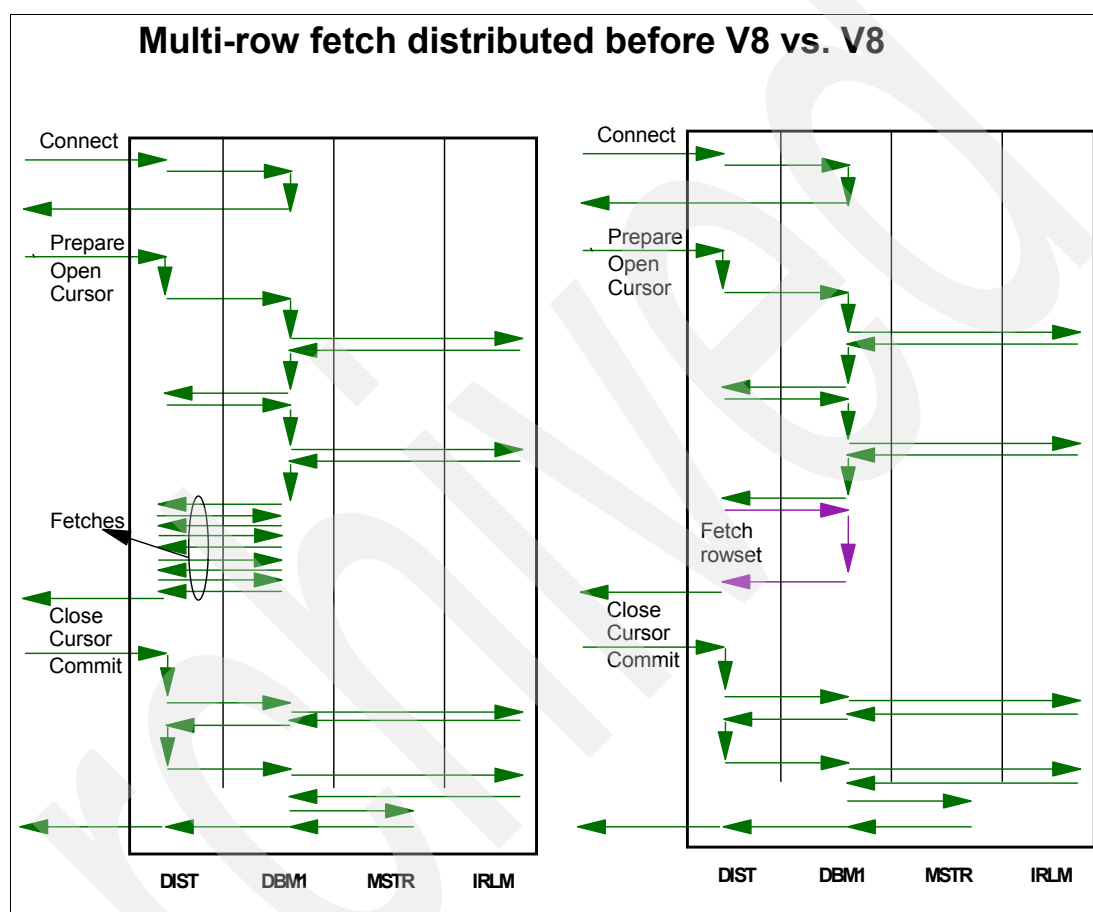


Figure 7-1 Multi-row FETCH distributed DB2 V7 vs. DB2 V8

Multi-row in DRDA becomes already active in CM for client CLI/ODBC and host to host SQL requests.

In the following sections the performance of multi-row FETCH and INSERT is evaluated.

7.2.1 Performance - Distributed host to host

In this section we present the performance measurements in a distributed host to host environment.

Performance measurements description

In these measurements we used:

- ▶ DB2 for z/OS, V7 and V8
- ▶ z/OS Release 1.3

- ▶ IBM z900 Series 2064 2-way processor
- ▶ ESS 800 DASD (with FICON channel)

Performance measurement results

In these measurements DB2 V8 is acting as a DRDA application server, accessed by another DB2 V8 acting as a DRDA application requestor. We fetch 100,000 20 column rows from a cursor without and with rowset positioning. The size of one row is 100 bytes.

We look at three cases: using multi-row fetch implicitly, explicitly and insert.

The results for implicit multi-row are presented in Figure 7-2. Note that the impact of multi-row fetch depends on network performance, the number of columns and rows fetched in one fetch request, the row size and the complexity of the SQL call. This means that your results are likely to differ from what is presented here.

The requestor class 1 elapsed time is derived from requestor (class 2 elapsed time + (class 1 CPU time - class 2 CPU time)) in order to avoid wide fluctuation reported in accounting class 1 elapsed time.

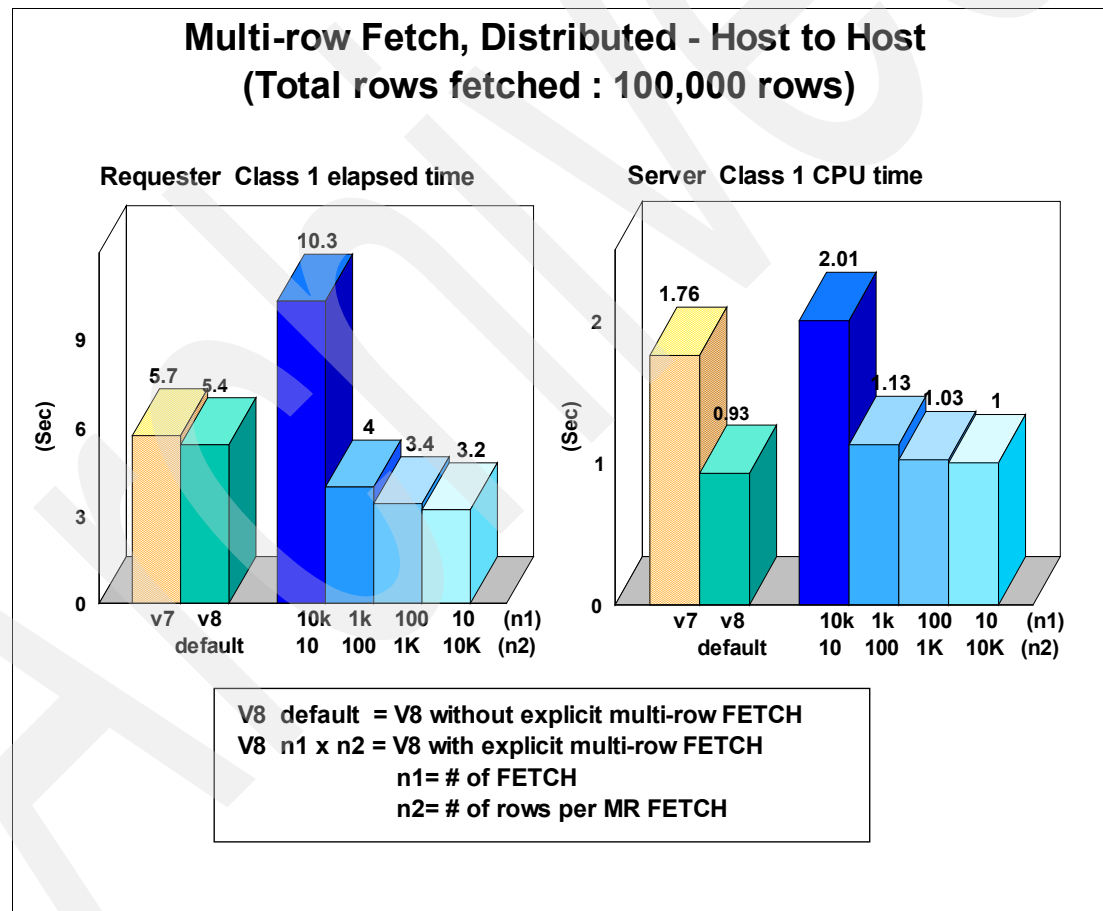


Figure 7-2 Multi-row FETCH - Distributed host to host

Note is that in the case of implicit multi-row fetch, the requestor application does not need to be changed.

In DB2 V8 *implicit multi-row FETCH* (default), the results listed in Table 7-2 show a 5% elapsed time improvement at the requestor and a 47% CPU time improvement at the server

when compared to DB2 V7. At the server side the DIST task requests the DBM1 to FETCH automatically a query block instead of rows. At the requester the rows are still fetched one by one. With implicit multi-row FETCH the query block size is 32 KB. Accounting class 2 does not have any impact here since class 2 data collection is done for each block, not for each row fetched.

Table 7-2 DB2 V8 with implicit multi-row FETCH vs. V7

	DB2 V7	DB2 V8	% difference
Requestor elapsed time (sec.)	5.70	5.39	-5
Server CPU time (sec.)	1.768	0.93	-47
Network transfers	326	326	0

With *explicit multi-row FETCH* the elapsed time improved up to 43% at the requestor for a rowset size of 10k due to an increase in the message block size from 32 KB up to 10 MB, see Table 7-3.

Table 7-3 DB2 V8 with explicit multi-row FETCH vs. V7

	DB2 V7	DB2 V8	% difference
Requestor elapsed time (sec.)	5.703	3.225	-43
Server CPU time (sec.)	1.758	0.999	-43
Network transfers	326	10	-97

The requester had only to issue 10 FETCH requests to the server to retrieve the result set. At the server a 43% CPU time improvement is realized. To benefit from this improvement the requestor application must be changed to use cursors with rowset positioning.

Network transfers go down proportionally to the number of fetch SQL calls issued. One fetch of a rowset is one network transfer.

If the rowset size is reduced, the corresponding performance numbers degrade. A rowset size of 10 performs worse compared to V7 and V8 default. Note that the rowset size also determines the number of rows in each query block. For a 10 rows rowset 10000 network transfers are needed compared to about 326 network transfers for V7 and V8 default. But more network transfers do not result always in degraded performance. In the case of rowset size 100, 1000 network transfers were needed but the savings at the application requestor, due to fewer cross memory requests between the application address space and the DBM1 address space, and the savings at the application server due to handling rowsets between the DIST and DBM1 address spaces, compensate for the less efficient network transfer.

The server CPU improvement for explicit 10k rowset (43%) is less than the V8 default, 320 rows (47%). This delta can be attributed to measurement noise, they should be equivalent.

In the next measurement *INSERT processing* in a DRDA host to host environment is evaluated. DB2 V8 acting as a DRDA application server is accessed by another DB2 V8 acting as a DRDA application requestor. In this test case a total of 10,000 20 column rows were inserted. The results are presented in Figure 7-3.

By default INSERTs only process a single row at a time. For DRDA each single row insert causes 1 network transfer from the requestor to the server. Compared to DB2 V7 the elapsed

time for an insert increases by 3.7% and the CPU time by 17% due to the architectural changes in DB2 V8.

If we change the requesting application to insert an array of 10, 100, 1000 or 10000 rows each with 20 columns we notice that the corresponding elapsed time at the DRDA application requestor and CPU time at the DRDA application server drop significantly compared to the 1 row insert in DB2 V7. The best result is obtained with a rowset size of 10000 where the DRDA application server CPU time is reduced by -72% and the DRDA application requestor elapsed time by 89%.

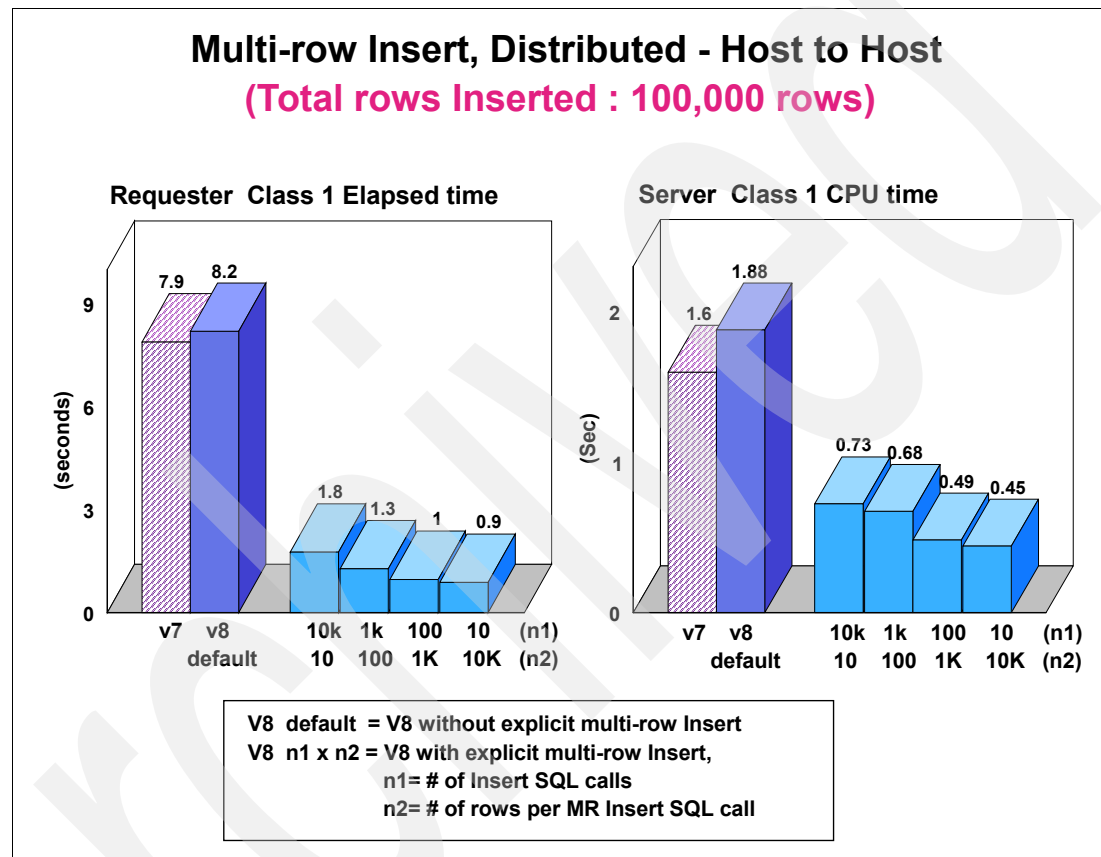


Figure 7-3 Multi-row INSERT - Distributed host to host

Conclusion

With implicit multi-row FETCH your applications will benefit immediately from the multi-row FETCH enhancement in a host to host DRDA environment without having to change the applications. Explicit multi-row support requires an application change. Host variable arrays are used to receive multi-fetch rowsets or serve as input for multi-row INSERT.

Recommendation

In host to host DRDA multi-row operations the best performance is obtained when the size of the host variable array matches the query block size which is 10 MB or 32767 rows whichever limit is first reached. Program virtual storage usage increases with the size of the arrays used. Explicit multi-row operations require DB2 V8 in new-function mode (NFM).

In general, 100 rows is just about the maximum needed to get the optimal performance. There is very little return beyond 100.

7.2.2 Performance - Distributed client to Host

In this section we present the performance measurements in a distributed client to host environment.

Performance measurement description

- ▶ DB2 for z/OS, V7 and V8
- ▶ z/OS Release 1.3
- ▶ IBM z900 Series 2064 2-way processor
- ▶ ESS 800 DASD (with FICON channel)
- ▶ DB2 Connect/client V8 acting as a DRDA application requester

Performance measurement results

DB2 for Multiplatforms V8 acting as client still does not support explicit multi-row FETCH when accessing DB2 for z/OS V8. Multi-row fetch is supported when using dynamic scrollable cursors in CLI coming from a distributed platform. However the client application and the DB2 for z/OS V8 server will benefit from the DB2 for z/OS V8 multi-row enhancement. By default the DIST address space will issue multi-row FETCH requests to DBM1 for read-only cursors with the rowset size determined by the query block size.

Figure 7-4 presents the result of a test case in which 100,000 20 column rows were fetched.

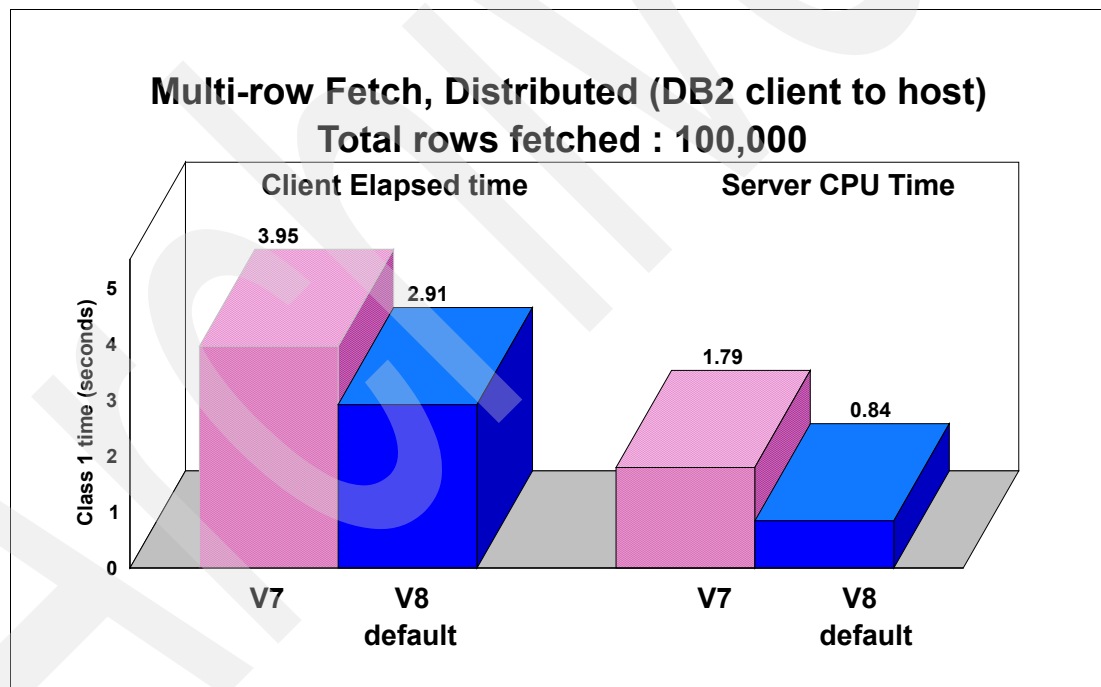


Figure 7-4 Multi-row fetch - DB2 client to host

Compared to DB2 V7, we notice a decrease in CPU at the DRDA application server of 53% and a decrease in elapsed time of 26% at the client application due to the larger query block size, up to 64 KB is supported by DB2 for z/OS V8.

The 64 KB query block size can reduce the network transfers by 50% in DB2 V8, compared to DB2 V7.

In DB2 V8 performance of CLI/ODBC applications will benefit from array insert. If we compare single row insert in DB2 V8 to single row in DB2 V7 elapsed time at the client increases with

2.3% and CPU time at the server with 8.1% again due to the architectural changes in DB2 V8, see Figure 7-5.

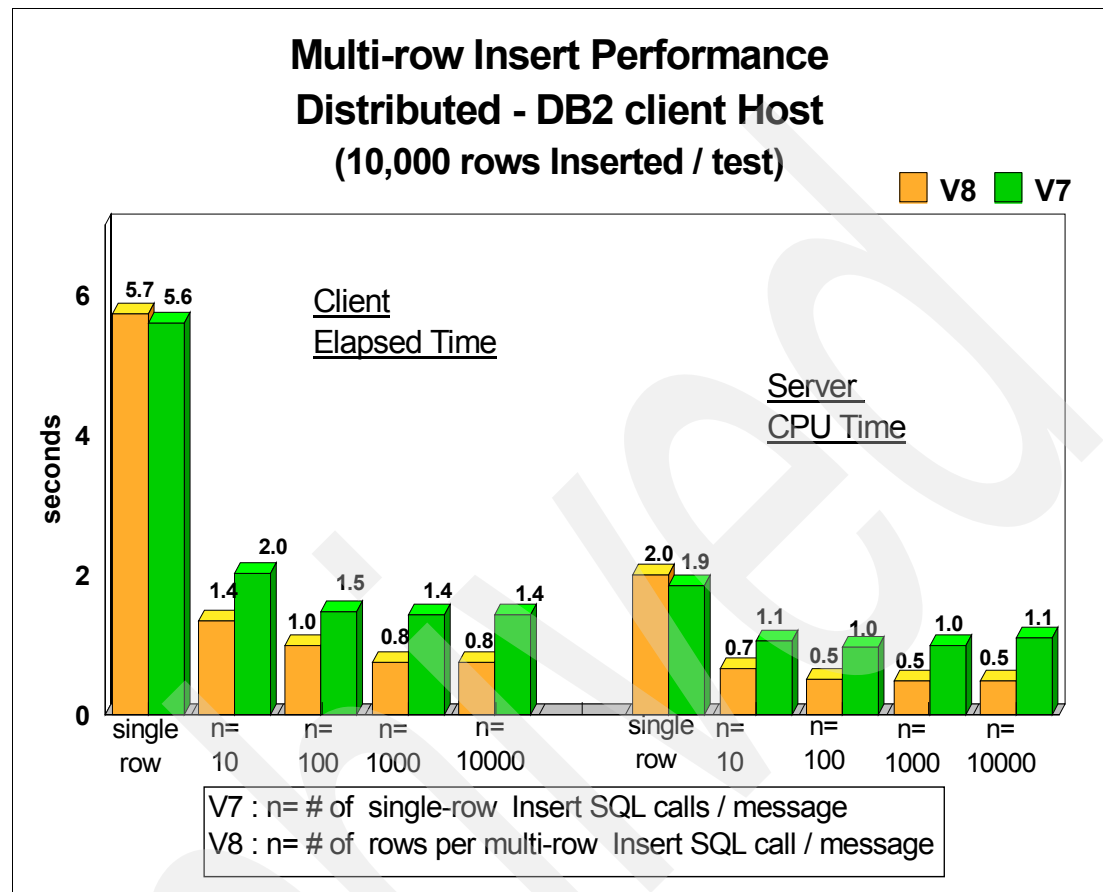


Figure 7-5 Multi-row insert - DB2 client to host

If the client application is coded to use array insert the difference with DB2 V7 becomes significant. In DB2 V7 this results still in single row inserts (the array insert only affects the processing between the application and the CLI/ODBC Driver) but in DB2 V8 full advantage is taken from multi-row insert between the DIST and DBM1.

For an application that inserts 10 times an array of 1,000 rows this results in a CPU time improvement of 51% at the DRDA application server side, and an elapsed time improvement of 47% for the client application, compared to DB2 V7.

If the ODBC Driver detects DB2 V8 as the server it translates the insert into a network transfer of 1 insert of 1,000 rows. If the ODBC Driver detects a pre-V8 server then each network transfer becomes 1,000 inserts of 1 row each.

Conclusion

Although there is no DB2 client support for native multi-row fetch, the CLI/ODBC `SQLExtendedFetch()` function fetches arrays from the query block result sets. At the server side the DIST address space will start using multi-row fetch by default and fetch a number of rows that will fit the query block size which results in significant CPU savings.

If your CLI/ODBC applications currently use array insert, you will immediately benefit from this enhancement as this SQL syntax will now be honored by DB2 V8 in NFM and not be translated by the CLI/ODBC Driver into a series of individual row inserts.

Recommendation

If you currently do not use array fetch and insert in your CLI applications we recommend that you start using array operations. There is no default optimal rowset size. The improvement that can be realized will depend on the number of columns, the size of the row, the network bandwidth,... As you can conclude from this measurement the larger the rowset the less significant the improvement. As a rule of thumb we recommend that you experiment with a rowset size that fits in a 64 KB communication buffer. Note that column-wise binding of the application array variables is more efficiently supported by the CLI/ODBC Driver than row-wise binding.

7.3 WebSphere and the DB2 Universal Driver

In this section we discuss the IBM DB2 UDB Universal Driver for SQLJ and JDBC, also known as the Java Common Connectivity (JCC), from a performance perspective and the DB2 enhancements that will benefit Java in general.

JDBC is a vendor-neutral SQL interface that provides data access to your application through standardized Java methods.

These methods and interfaces are packaged as DB2 JDBC Drivers and numbered from 1 to 4:

- ▶ Type 1

This is the oldest type of driver. It was provided by Sun to promote JDBC when no database-specific drivers were available. With this driver, the JDBC API calls an ODBC Driver to access the database. This driver type is commonly referred to as a JDBC-ODBC bridge driver. Its shortcoming is that ODBC must be implemented and installed on all clients using this driver type. This restricts it to platforms that have ODBC support, and as such is mostly Windows centric. Since it also has to translate every JDBC call into an equivalent ODBC call, the performance of a Type 1 Driver is not great. IBM does not provide a Type 1 Driver.

This type is no longer commonly used, because virtually every database vendor nowadays supports JDBC and provides their own (vendor-specific) driver.

- ▶ Type 2

A JDBC type 2 Driver relies on platform- and database-specific code to access the database. The application loads the JDBC Driver and the driver uses platform- and database-specific code to access DB2. This driver is also known as the “app” driver which refers to the implementation class name `com.ibm.db2.jdbc.app.DB2Driver`. This name and driver class name only applies to the old driver (not the universal) and applies only to DB2 for Multiplatforms. This is the most common driver type used, and offers the best performance. However, as the driver code is platform-specific, a different version has to be coded (by the database vendor) for each platform.

The new Universal Driver has a Type 2 implementation available.

- ▶ Type 3

With this driver type, also referred to as the “net” driver (implementation class `com.ibm.db2.jdbc.net.DB2Driver`), the JDBC API calls are routed through a middleware product using standard network protocols such as TCP/IP. The driver itself is written in Java. The middleware translates these calls into database-specific calls to access the target database and returns data to the calling application. In the case of DB2, this task is performed by a program called the JDBC applet server. DB2 for z/OS and OS/390 does not supply a Type 3 Driver. DB2 for Linux, Unix and Windows still has a so-called network

or net driver, but if you are not using it already, you are strongly encouraged to use the Type 4 Driver instead.

- ▶ **Type 4**

A Type 4 Driver is fully written in Java, and provides remote connectivity using DRDA within IBM products. As the driver is fully written in Java, it can be ported to any platform that supports that DBMS protocol without change, thus allowing applications to also use it across platforms without change.

This driver type has been implemented through the IBM DB2 Universal Driver for SQLJ and JDBC. This driver is also known as the DB2 Universal Driver for Java Common Connectivity or JCC. In addition, the Universal Driver also provides Type 2 connectivity.

For more information on these Drivers, see the book *DB2 for z/OS and OS/390: Ready for Java*, SG24-6435.

The *DB2 Universal Driver for SQLJ and JDBC* is an entirely new driver, rather than a follow-on to any other DB2 JDBC drivers. The DB2 Universal Driver has the following advantages over the previous solutions:

- ▶ One single driver for the Unix, Windows, and z/OS platforms, improving portability and consistent behavior of applications.
- ▶ Improved integration with DB2.
- ▶ Type 4 Driver functionality for thin clients, as well as Type 2 functionality.
- ▶ Easier installation and deployment. As a Type 4 Driver is completely written in Java, it is easy to deploy; you only have to ship the jar files containing the Driver code to the required machine, change the classpath, and you are done. In addition, the use of a Type 4 Driver does not require any setup in DB2, such as cataloging entries in the Database, Node, and DCS directories. For DB2 on z/OS you only have to use the following URL syntax:

```
jdbc:db2://<hostname>:<port number>/<location name>
```

- ▶ 100 percent Java application development process for SQLJ
- ▶ 100 percent JDBC 3.0 compliance. Since DB2 UDB V8.1.3 (FixPak 3), the Universal Driver is JDBC 3.0 compliant. The version that is made available with z/OS is also JDBC 3.0 compliant.
- ▶ Significant performance improvements for both JDBC and SQLJ applications on the Linux, Unix, and Windows platforms.
- ▶ Trace improvements.

In general, you should use Universal Driver type 2 connectivity for Java programs that run on the same z/OS system as the target DB2 subsystem. Use Universal Driver type 4 connectivity for Java programs that run on a different system from the target DB2 subsystem.

For Java applications, application connectivity from a z/OS LPAR to a remote DB2 for z/OS system does not need a DB2 instance anymore. The z/OS Application Connectivity to DB2 for z/OS optional feature can be installed to provide Universal Driver type 4 connectivity.

7.3.1 Performance measurement

The performance objectives were to verify that:

- ▶ The DB2 Universal JDBC type 2 Driver provides better or equivalent performance for workloads using container managed persistence (CMP) and better performance for traditional workloads that use the type 2 IBM DB2 App Driver.

- ▶ The DB2 Universal JDBC type 4 Driver provides better performance than the current JDBC type 3 Driver. Note that the type 3 JDBC Driver or “net” driver is deprecated.
- ▶ The balancing features of the DB2 Universal Driver offer good performance and a good choice of functions.

Starting with V5.0.2 WebSphere is supporting the new DB2 Universal Driver.

WebSphere V5.0.2 z/OS - DB2 for z/OS V8 - Servlet workload

Different test cases were measured and are presented in this section.

DB2 universal type 2 Driver vs. legacy JDBC 2.0 Driver - local environment

WAS V5.0.2 and DB2 V8 are running in the same LPAR. The throughput of this configuration with the Type 2 legacy Driver is compared to the configuration with the new Type 2 Driver, see Figure 7-6.

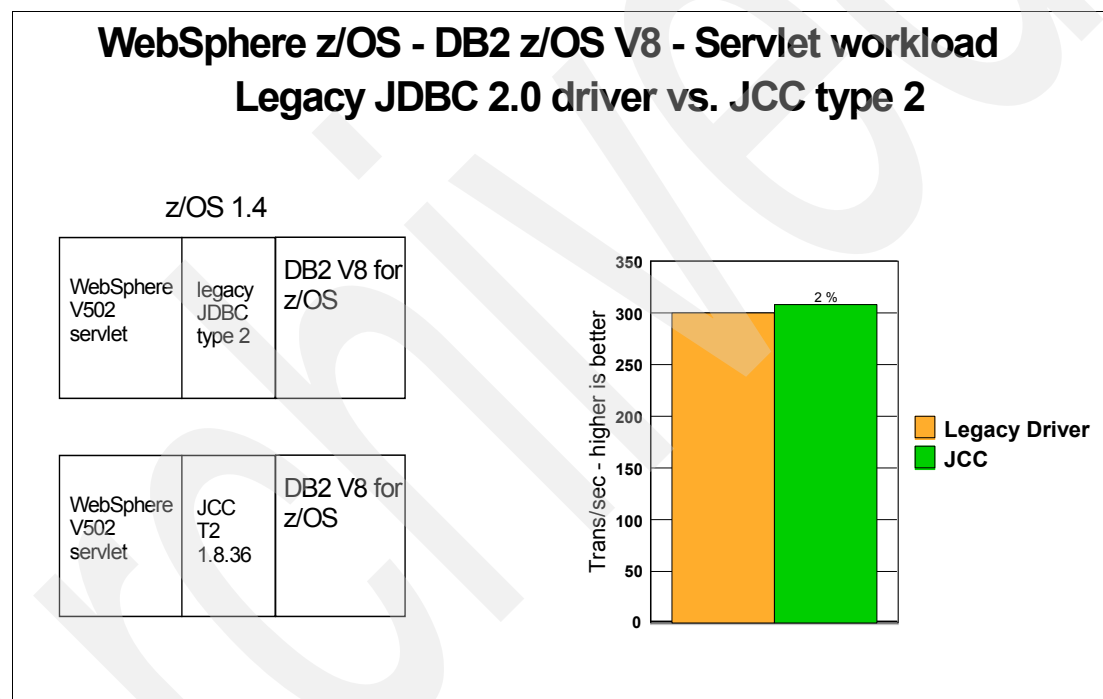


Figure 7-6 DB2 Universal Type 2 Driver vs. legacy JDBC 2.0 Driver

An increase of 2% throughput is noticed with the DB2 Universal Type 2 Driver. There is no reason to stop you from using the new JCC Type 2 Driver.

If you still have the legacy Drivers in your CLASSPATH, you must make sure that the db2jcc.jar file is in the CLASSPATH, ahead of the legacy Driver files, because some classes have the same name in the legacy and the DB2 Universal Driver for SQLJ and JDBC. However, to avoid confusion it is probably best to only have one set of drivers (old or new) in the CLASSPATH.

DB2 universal type 2 Driver - Gateway DB2 vs. DB2 universal type 4 Driver

Although it seems unusual to establish a connection to a remote DB2 for z/OS with a fully implemented DB2 subsystem, it was your only choice in the past since there was no DB2 client on the z/OS platform available. Besides the license cost for the DB2 software, the DB2 subsystem used also memory and CPU resources on the WAS LPAR.

With the z/OS Application Connectivity to DB2 for z/OS feature that provides DB2 universal Driver type 4 connectivity, the use of a gateway DB2 is avoided and a 73% increase in throughput on the WAS LPAR is realized, see Figure 7-7.

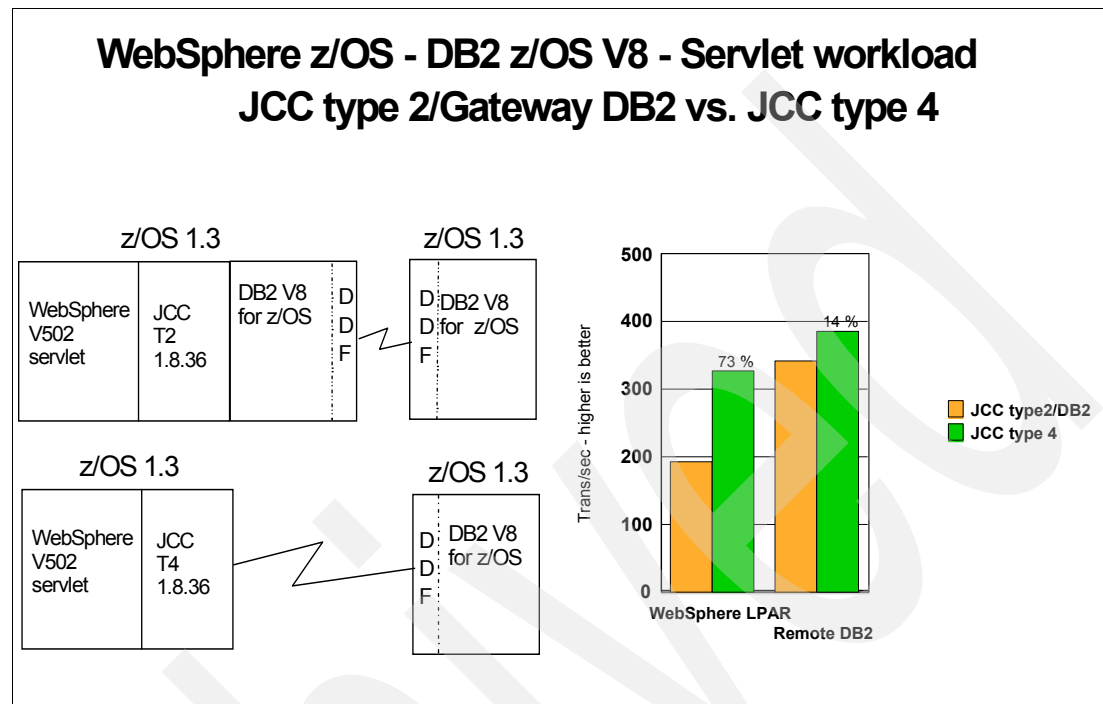


Figure 7-7 DB2 Universal Type 2 Driver - Gateway DB2 vs. DB2 Universal Type 4 Driver

DB2 Universal Type 2 Driver vs. DB2 Universal Type 4 Driver - Local environment

In Figure 7-8 we compare the JCC Type 2 and Type 4 Driver in a local environment. The intention is to demonstrate the difference in performance between the Type 2 and Type 4 Driver in a local environment in the same LPAR as DB2.

Since the WebSphere Application Server is local to the DBMS (same LPAR), the Type 2 Driver can do “native” calls to DB2; this configuration normally provides the best performance. Your Java application that is running under the control of WAS can talk to DB2 through the Type 2 JDBC Driver, and a local RRS attachment.

Using the Type 4 Driver in this configuration is not a recommended setup for a production environment, since all SQL requests go through DRDA over TCP/IP (so through the TCP/IP stack) into DB2’s DDF address space. Even though DRDA is a very efficient database communication protocol, it is normally more expensive than going through the local RRS attachment as will become clear from this test case.

WebSphere z/OS - DB2 z/OS V8 - Servlet workload JCC type 2 vs. JCC type 4 in local environment

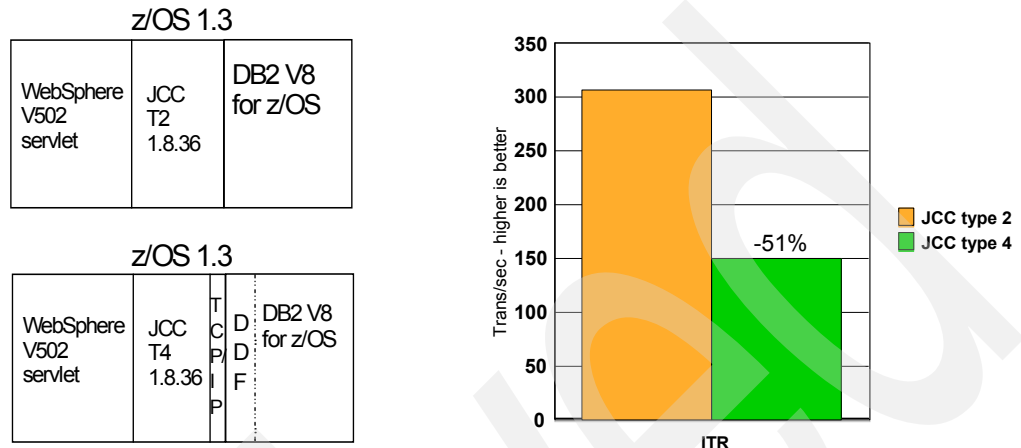


Figure 7-8 DB2 Universal Type 2 Driver vs. DB2 Universal Type 4 Driver

Using the setup with the DB2 Universal Type 4 Driver (DRDA), the throughput is reduced by 51% when compared to the Type 2 Driver RRSF connection to DB2. The Type 2 Driver communicates with DB2 via cross-memory calls where the Type 4 Driver uses the TCP/IP stack.

The setup with the Type 4 Driver is not that unusual. If access to several DB2 subsystems in the same LPAR is needed, you will need the Type 4 Driver as this is not supported by the Type 2 Driver with an RRSF connection.

This measurement was executed using JDK 1.3.1. The performance of the setup with the Type 4 Driver improves with JDK 1.4.2.

JDK 1.4.2 has seen the JIT compiler optimized, and the Java TCP/IP implementation improved. A separate performance comparison is shown in Figure 7-9.

Notice that JDK versions go in pairs with WAS versions:

- ▶ WAS 5.0.2 and JDK 1.3.1
- ▶ WAS 5.1.0 and JDK 1.4.2

WebSphere z/OS - DB2 for z/OS V8 - Servlet workload WAS 5.0.2/JDK 131 vs. WAS 5.1.0/JDK 142

- 16 % increase in normalized throughput caused by JDK 142 improvement
- Universal Driver Type 4 benefits from JDK 142 TCP/IP optimization

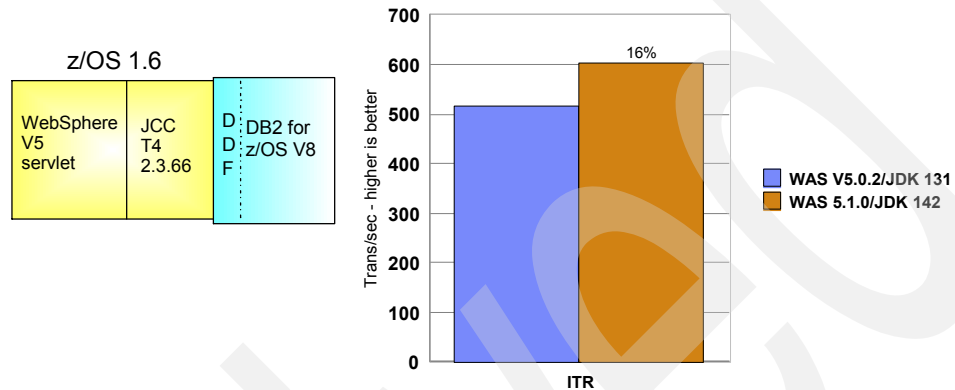


Figure 7-9 JDK improvements

WebSphere AIX - DB2 for z/OS V8

For this configuration tests were run with the Type 4 Driver as the Type 2 Driver is a local driver. With the Type 2 Driver you need something in the middle and this “thing in the middle” is DB2 Connect. With the Type 4 Driver and WAS, we do not need DB2 Connect with its characteristics of connection pooling and concentrator because WAS provides the same functionality.

JDBC .app/DB2 Connect vs. JCC Type 4 - Servlet workload

As shown in Figure 7-10, JCC Type 4 outperforms the .app/DB2 Connect with 10% increased throughput on WebSphere V5 AIX. On DB2 for z/OS the improvement is marginal.

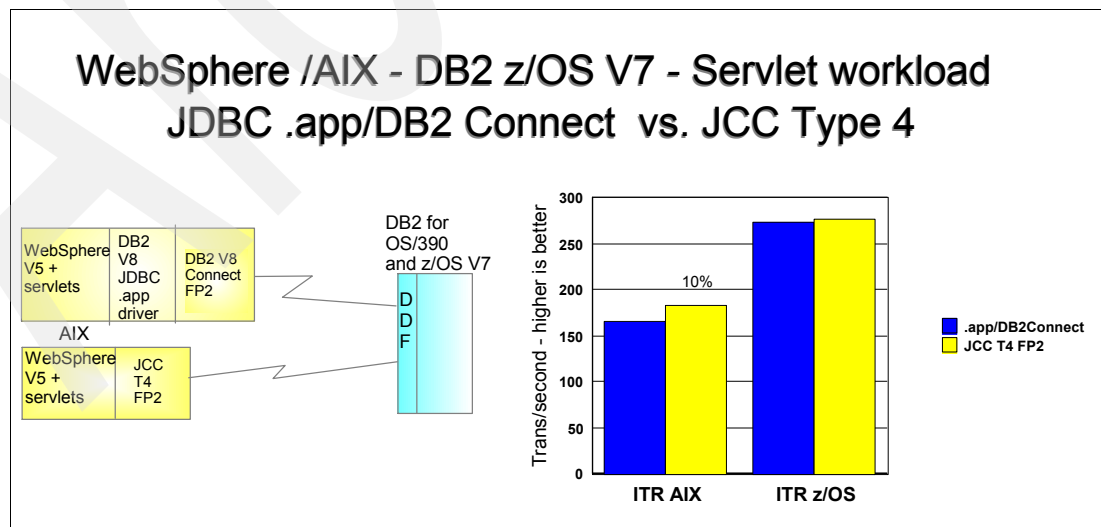


Figure 7-10 JDBC .app/DB2 Connect vs. JCC Type 4 - CMP workload

DB2 Universal Type 4 Driver - JDBC vs. SQLJ - CMP workload

As shown in Figure 7-11, SQLJ outperforms JDBC with 25% increased throughput on DB2 for z/OS and 11% on WebSphere V5 AIX. Besides the benefits of static SQL, persistence in WebSphere with SQLJ requires fewer API calls compared to JDBC.

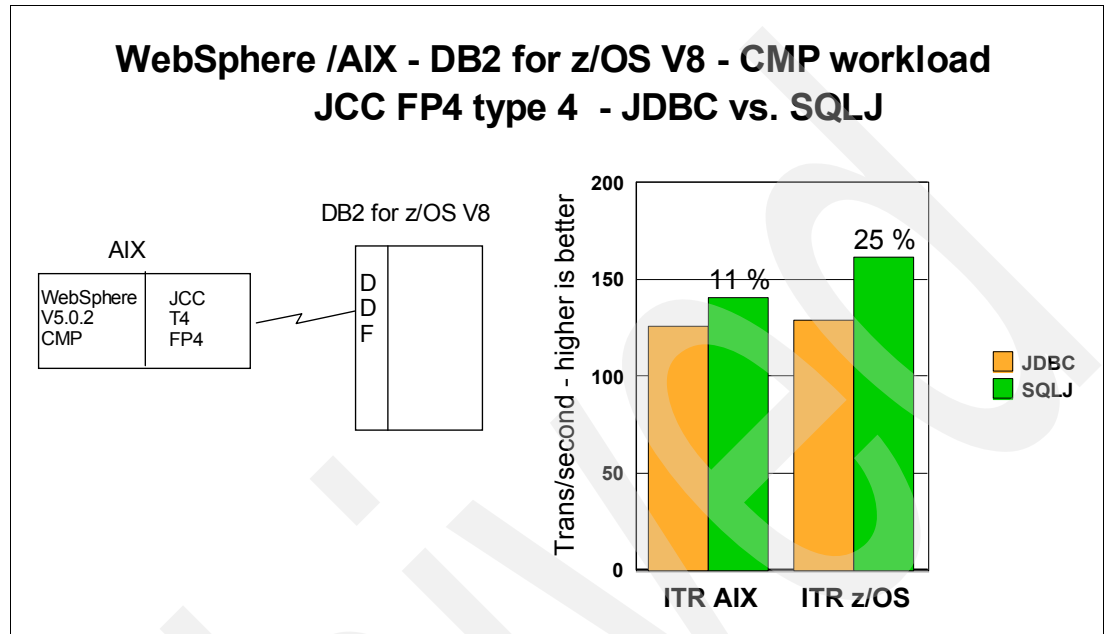


Figure 7-11 DB2 Universal Type 4 Driver - JDBC vs. SQLJ - CMP workload

DB2 Universal Type 4 Driver - JDBC vs. SQLJ - Servlet workload

Figure 7-12 shows that the best performance is achieved using SQLJ when the SQLJ files are compiled with the option -db2optimize.

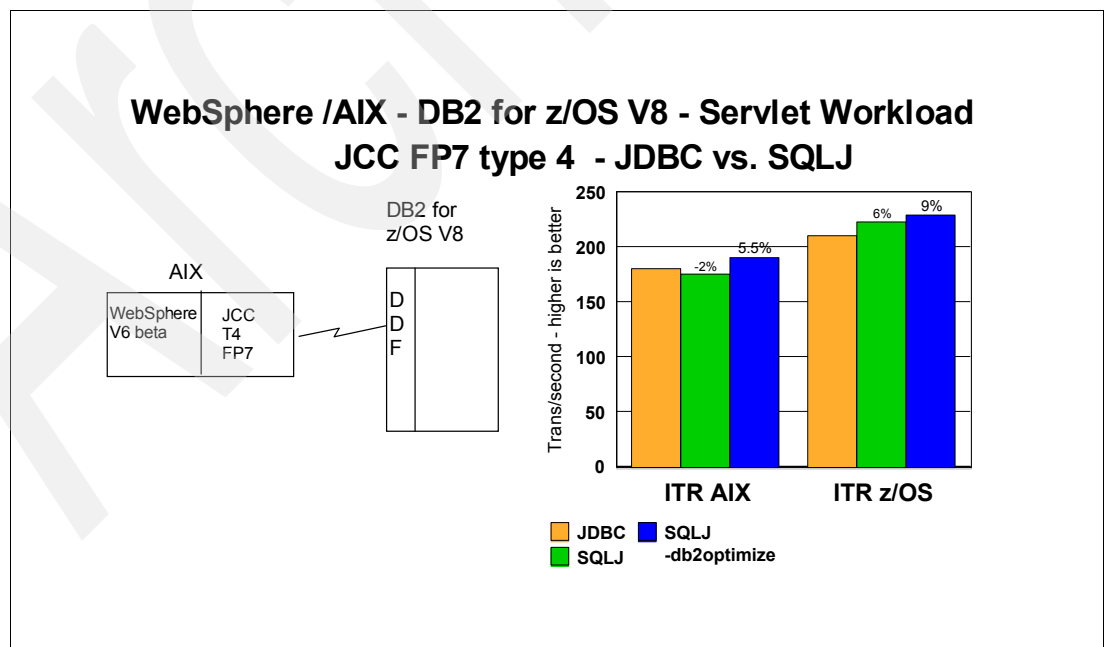


Figure 7-12 DB2 Universal Type 4 Driver - JDBC vs. SQLJ - Servlet workload

SQLJ programs need to use explicitly defined connection contexts as always has been recommended.

The test cases presented used WebSphere and the universal Type 4 Driver. The difference in performance with using the Type 2 Driver and DB2 Connect instead is negligible as WAS will perform connection pooling with the Type 4 Driver.

JCC Type 4 vs. JCC Type 4 with DB2 Connect - IRWW workload

Figure 7-13 illustrates the performance results of running IRWW workload using JCC Type 4 connection going from distributed platform to DB2 for z/OS in comparison of using JCC Type 4 to connect to DB2 for z/OS through DB2 Connect in gateway mode. The measurements show a 2.3% $[(382.14 - 373.73) / 382.14 * 100\%]$ internal throughput degradation by going through DB2 Connect rather than going directly to DB2 for z/OS.

In the DB2 Connect scenario, both JCC driver and DB2 Connect reside on the same machine, and DB2CONNECT_IN_APP_PROCESS is defined as NO in order to have the application connect to the DB2 Connect EE Server and then have the host connection run within an agent.

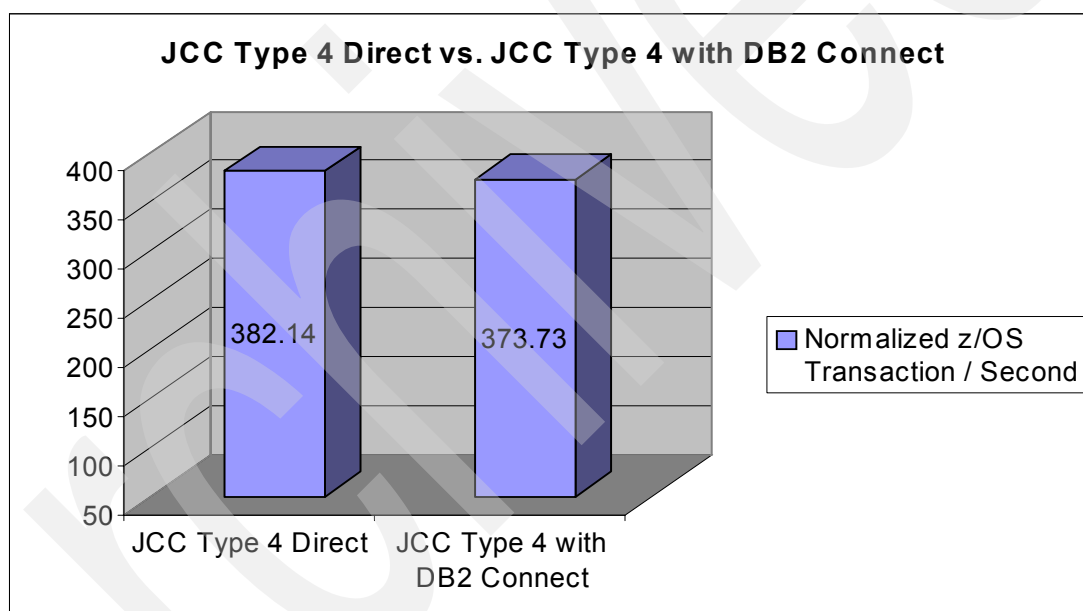


Figure 7-13 JCC Type 4 vs. JCC Type 4 with DB2 Connect

Conclusion

The new DB2 Universal Type 2 Driver performs as well and better than the legacy JDBC App Driver and the Type 4 Driver performs better than the Type 3 Net Driver.

Recommendations

We recommend the usage of DB2 universal Type 2 Driver in a local DB2 environment. The Type 4 Driver is preferred when accessing a remote DB2 server in a distributed environment.

Use the DB2 Universal Type 4 Driver to access a DB2 server directly when connections are already pooled by other application servers, for example, WebSphere. FixPak 10 has been recently made available and it supports connection concentration and sysplex workload balancing.

Use DB2 Connect when a high amount of client connections is expected, for example, from applets, in its quality as connection pool/connection concentrator. The measured overhead, shown in Figure 7-13, is about 2%.

Where possible, use SQLJ. Throughput is higher as well on the server as on the requester with SQLJ. Static SQL is faster than dynamic SQL, because at run time only the authorization for packages and plans must be checked prior to the execution of the program. In contrast to that, dynamic SQL statements require the SQL statements to be parsed, table and view authorization to be checked, and the access path to be determined. Besides better performance, SQLJ offers availability and usability benefits over JDBC:

- ▶ Less and easier code, easier to maintain
- ▶ Authorization on package level
- ▶ Easier deployment with the fully portable serialized profile when using the Universal Driver if compared to the old driver.

Balancing features of JCC with WebSphere V5.1.1 AIX - DB2 for z/OS V8

Starting from the GA version of DB2 Universal Driver FixPak 10, DB2 has introduced connection concentrator and sysplex workload balancing features for all Java type 4 connections to DB2 for z/OS servers. These two new features are built upon the JCC global object pooling architecture, and are similar in functions to DB2 Connect's connection concentrator and sysplex support.

The objective of these measurements is to evaluate the overhead of reusing JCC transports with connection concentrator and sysplex workload balancing enabled, and compare it to the overhead of DB2 Connect and JCC T2 driver.

The IRWW workload was run with the JCC T4 configuration shown in Figure 7-14.

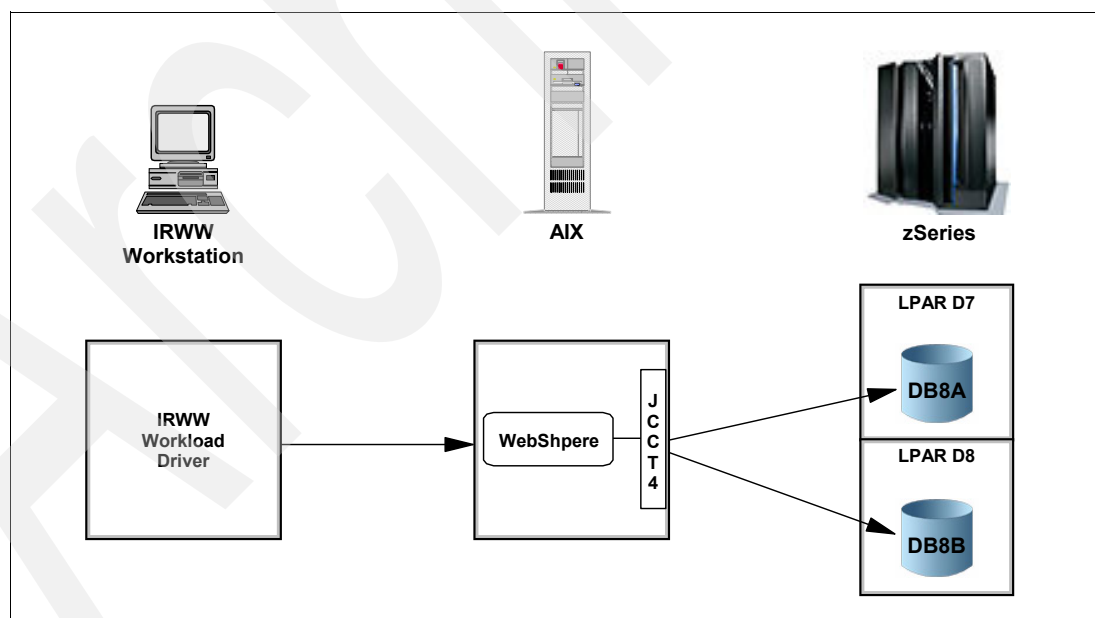


Figure 7-14 JCC T4 configuration for balancing options

Connection concentrator and sysplex workload balancing are two separate JCC features. This is different from DB2 Connect where connection concentrator must be enabled before you can exploit sysplex load balancing and failover routing at the transaction boundary. It is possible to enable connection concentrator or sysplex workload balancing in the JCC data source custom properties.

The JCC connection concentrator feature is available only when DB2 on z/OS is the target server. If you need concentration function for connections to DB2 UDB databases for distributed platforms, you need DB2 Connect today. Also, JCC connection concentrator only works with T4 direct connection to DB2 servers. Both JCC connection concentrator and JCC sysplex workload balancing features are automatically disabled if JCC driver detects that it is connected to a DB2 server via a DB2 Connect gateway. In this case the DB2 Connect gateway will be responsible for handling connection concentration and workload balancing.

Connection concentrator was designed for environments where a large number of clients are connecting to the host and there is the need to limit the number of connections resources on the host side. There is some overhead with switching applications to transports connected to the host at the transaction level, but much of this overhead is incurred on client machines where CPU and memory are less expensive and less constrained.

The same IRWW workload was run with the DB2 Connect configuration shown in Figure 7-15.

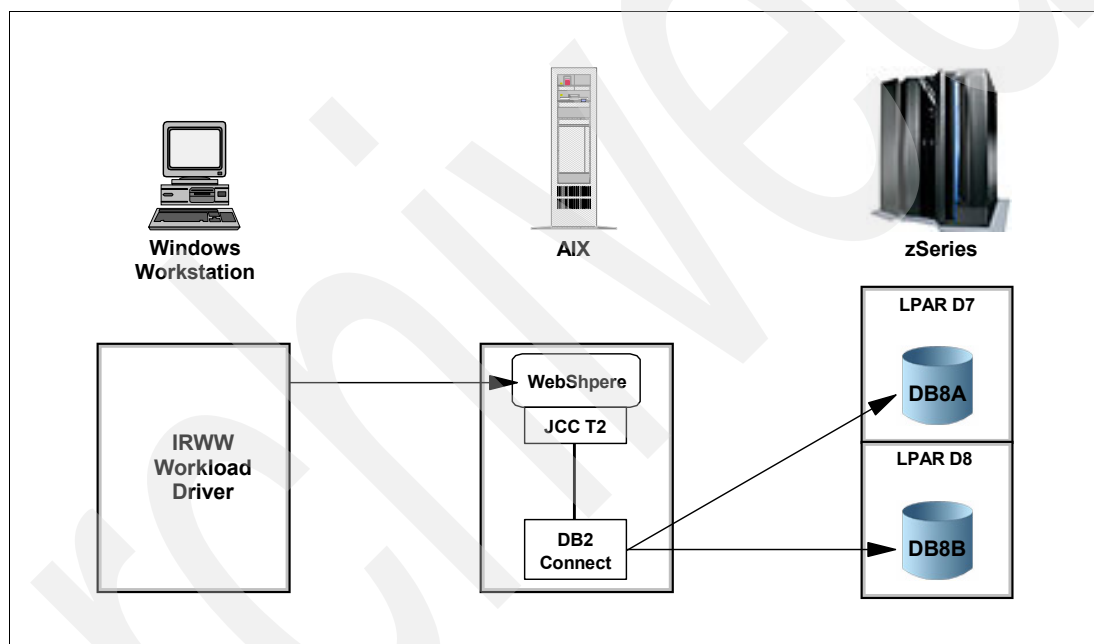


Figure 7-15 JCC T2 and DB2 Connect configuration

The graph shown in Figure 7-16 compares the performance degradations of JCC T4 and DB2 Connect before and after both are configured to concentrate transports or agents with the same connection-to-transport/agent ratio. This approach gives us a fair comparison of their respective connection concentration and load balancing efficiency. The performance impact on DB2 for z/OS is measured in normalized DB2 transactions per second (internal throughput).

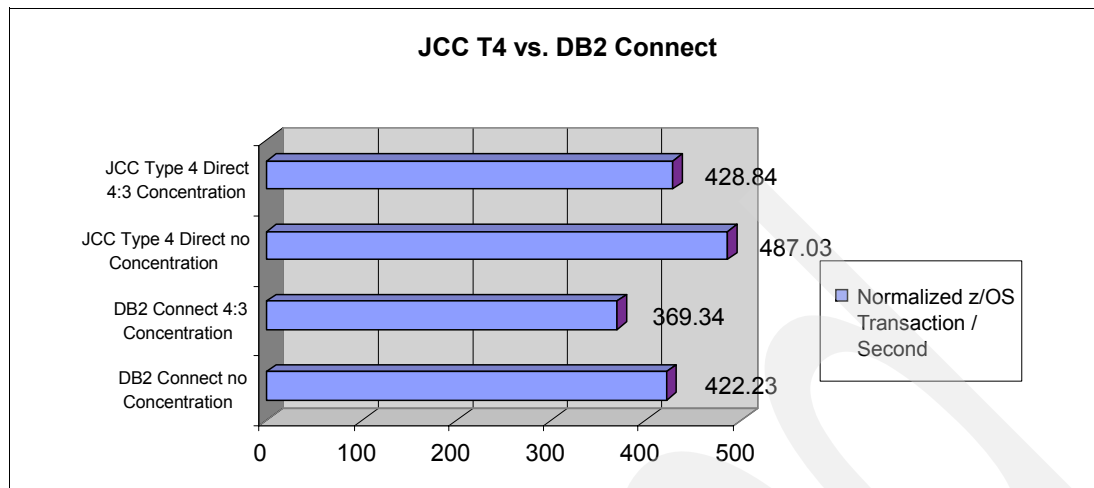


Figure 7-16 Performance of JCC T4 vs. DB2 Connect

We can observe that the DB2 Connection concentration degradation is 13.5%, versus the JCC Type 4 concentration degradation of 12%. The value is very similar.

In determining the choice of function to implement concentration and balancing, keep in mind that the two options have different characteristics and the requirements of the application must take into account the considerations summarized in Figure 7-4

Table 7-4 Comparing the key points of the two solutions

DB2 Connect	JCC Driver
Wider scope DB2 connect is a gateway. It is aware of all DB2 requests. it is capable of supporting connection concentration across multiple JVMs on one or more machines. JCC driver only runs within the scope of a single JVM, it is not aware of the connections of another JVM and it can only concentrate connections originated from its own JVM	Better granularity DB2 Connect can only specify the maximum number of coordinator agents allowed at the server level. it cannot control connection agents allocated to each JVM. JCC driver allows users to specify the maximum number of allowed transport objects at datasource level and at the driver level. This gives JCC driver much better granularity to manage transport resources.
Connection pooling Connection pooling allows agents and connections to be reused by other users and programs. JCC T4, at the time writing, doesn't have connection pooling, and it must either rely on external application server such as WebSphere or extra programming in standalone Java applications to support connection pooling.	Better Performance The performance measurements show that JCC T4 has slightly better performance than DB2 Connect V8 when running with the same connection to transport/agent ratio
Ease of software upgrade From a software management perspective, it is easier to have the software in one place, any upgrades only have to be done to the DB2 Connect Enterprise Edition (EE) server (a centralized gatekeeper for DB2 servers on z/OS), not to every JCC driver spread throughout the entire enterprise.	

7.3.2 DB2 Universal Driver X/Open XA support

One of the key reasons for using the Universal Driver in a type 2 architecture is for local application performance and for distributed transaction support (XA). XA support was first provided in the DB2 Universal Type 2 Driver and became available in the Type 4 Driver with DB2 V8.1 for Multiplatforms FP4. MQSeries® classes for Java Message Service include the JMS XA interface. These allow MQ JMS to participate in a two-phase commit that is coordinated by a transaction manager that complies with the Java Transaction API (JTA).

Description

If the application will be involved in a distributed transaction, the `COM.ibm.db2.jdbc.DB2XADataSource` class should be used when defining DB2 data sources within the WebSphere Application Server.

Performance

The internal throughput was measured for a servlet workload using the Type 2 non-XA versus the Type 2 XA compliant driver. The environment used in this measurement was z/OS V1.3, DB2 for z/OS V8 and WAS V5.0.2. See Figure 7-17.

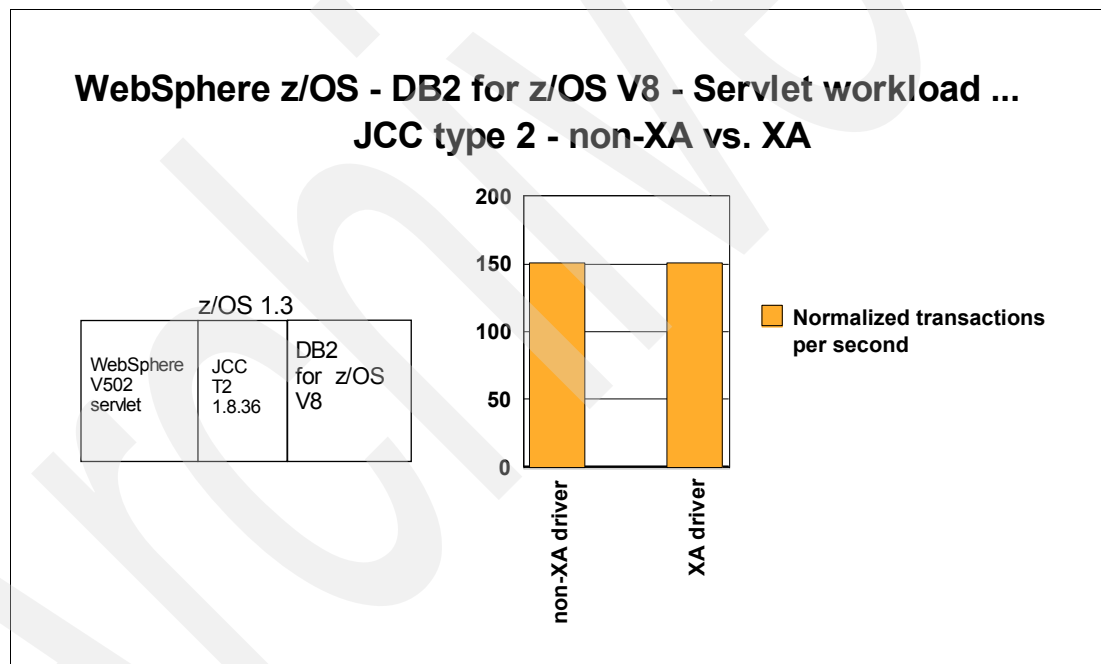


Figure 7-17 Servlet workload - Type 2 non-XA vs. Type 2 XA compliant driver

The results are less than 1% of internal throughput ratio (ITR= Units of Work/Processor Busy) reduction when using the type 2 XA compliant driver. This means that there is practically no extra overhead in choosing the driver with distributed transaction support from WebSphere.

DB2 client application programs that run under the DB2 Universal JDBC Driver can use distributed transaction support in DB2 V7 and DB2 V8. DB2 V8 servers include native XA mode support. However, DB2 V7 servers do not have native XA mode support, so the DB2 Universal JDBC Driver emulates the XA mode support using the existing DB2 DRDA two-phase commit protocol. This XA mode emulation uses a DB2 table named `SYSIBM.INDOUBT` to store information about indoubt transactions. DB2 uses a package named `T4XAIndbtPkg` to perform SQL operations on `SYSIBM.INDOUBT`.

The SYSIBM.INDOUBT table and the package can be created by invoking the JCC In-Doubt utility. It is a prerequisite to manually run this utility as a user having SYSADM privileges against the DB2 V7 location in order to accomplish XA (global /distributed) transactions. The syntax is as follows:

```
java com.ibm.db2.jcc.DB2T4XAIndoubtUtil -url jdbc:db2://<hostname>:<port number>/<location>
-user <userid sysadm> -password <password>
```

This has recently changed with PQ95881: Now you can run the "XAindoubt.." and the create of the indoubt table separately.

7.3.3 Miscellaneous items

Java Data Types

The Java language does not have a fixed length character string data type; every string is variable length. In DB2, on the other hand, in most cases, fixed length character columns defined as CHAR(n) are used. In DB2 V8 all predicates comparing string CHAR and VARCHAR data types with the same CCSID are stage 1 and indexable; even for unlike CCSIDs the predicate will still be stage 1 and indexable if the "column" side of it is in Unicode since all predicates that compare unlike CCSIDs are evaluated in the Unicode encoding scheme.

KEEPDYNAMIC YES

KEEPDYNAMIC specifies whether DB2 keeps already prepared dynamic SQL statements in the local dynamic statement cache after commit points. Bind a copy of the DB2 Universal Driver packages with a different collection id:

- ▶ KEEPDYNAMIC=YES
- ▶ collection=<collection name>

The keepDynamic connection and JdbcCollection property values for the DataSource specified must match the values that were specified when the bind was run. These values can be set using:

- ▶ The setXXX methods
- ▶ In a java.util.Properties value in the info parameter of a DriverManager.getConnection call
- ▶ The java.lang.String value in the url parameter of a DriverManager.getConnection call

The following measurement was executed:

WebSphere V5.0.2 on AIX was used to access DB2 for z/OS V8. The workload was using container managed persistence conforming to the SpecjApp2002 benchmark.

The results are presented in Figure 7-18.

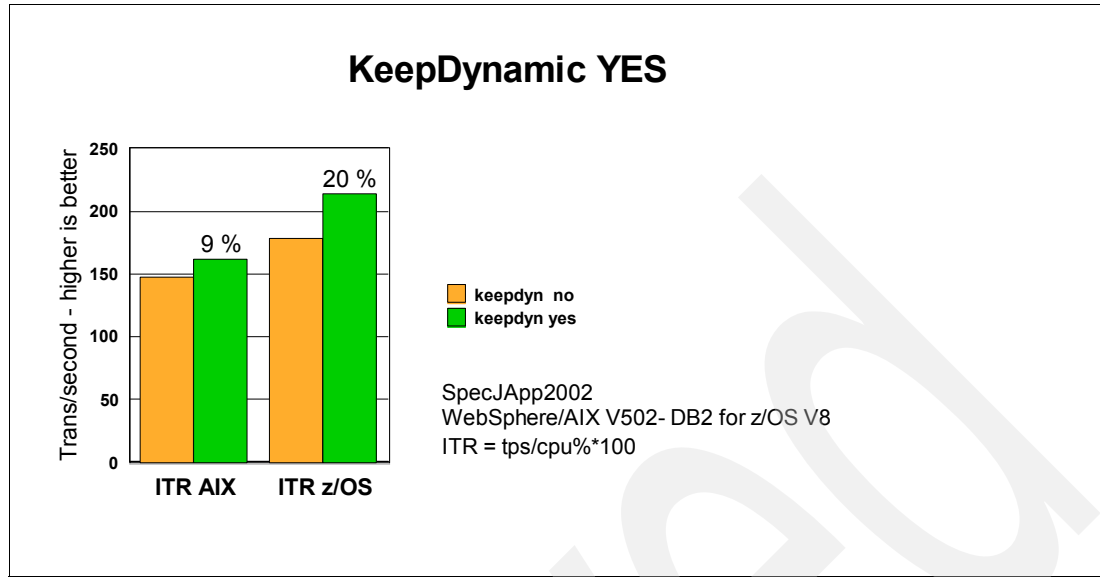


Figure 7-18 DB2 Universal Driver - KEEP_DYNAMIC YES

CPU is saved on the client (WebSphere) and in DB2 on the server. Throughput improved with 9% on the client and 20% on the DB2 server.

In order to benefit from performance improvements the following PTFs/APARs should be applied: PQ86787/UQ87049, PQ87126/UQ88971 and PQ87129/UQ87633.

Conclusion

The statements are saved on thread level and can consume a significant amount of storage. Type 2 distributed threads cannot turn inactive and DB2 resources for distributed threads are not freed at commit. Therefore this is not recommended for storage constrained systems.

Recommendation

KEEP_DYNAMIC(YES) is recommended for applications with a limited amount of SQL statements that are executed very often. It is not recommended for applications with a large number of SQL statements that are executed infrequently.

LOB streaming

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

LOB streaming, activated by the fullyMaterializeLobData property, allows the DB2 universal client or a DB2 for Multiplatforms server to not know the total length of a LOB value prior to sending a LOB value. This improves performance by eliminating the need to read the entire LOB to determine its length prior to sending the LOB value. DB2 continues to provide the total length of a LOB value, but it is now able to read in the LOB immediately without knowing the exact length up-front.

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

Batch updates

A batched update is a set of multiple update statements that are submitted to the database for processing as a batch. Sending multiple update statements to the database together as a unit can, in some situations, be much more efficient than sending each update statement separately. It reduces the number of times the application has to cross over to the JDBC Driver. This ability to send updates as a unit, referred to as the batched update facility, is one of the features of the JDBC 2.0 API, and is now supported with the DB2 Universal Driver.

7.4 ODBC enhancements

ODBC enhancements are mentioned for completeness.

7.4.1 ODBC Unicode support

ODBC now supports Unicode formats UTF-8 and UCS-2. In addition, a new ODBC initialization keyword, CURRENTAPPENDSCH, lets you specify the encoding scheme that you want the ODBC Driver to use for input and output of host variable data, SQL statements, and all character string arguments of the ODBC application programming interfaces. You can specify Unicode, EBCDIC, or ASCII encoding scheme.

7.4.2 ODBC long name support

ODBC for z/OS has been enhanced to be able to support the long names in V8 new-function mode. This means changes to the following ODBC functions.

Support for longer names in the INI file variables

The keyword strings for the following initialization keywords will be extended to support longer names:

- ▶ CLISHEMA
- ▶ COLLECTIONID
- ▶ CURRENTFUNCTIONPATH
- ▶ CURRENTSQLID
- ▶ SCHEMALIST
- ▶ SYSSHEMA

ODBC catalog functions

As a result of the name changes in the catalog, the ODBC catalog API needs to change the lengths of the host variables and data types in the SQLDA when fetching result sets from catalog tables. However, most of these changes are transparent to the external API. Only the data type of the REMARKS column returned by SQLColumns() and SQLTables() is changed to VARCHAR(762).

7.4.3 ODBC connection authentication

With V7, the DB2 ODBC Driver validates the user ID and password, but their argument values are not used for end user authentication. DB2 V8 implements the RACF verification support for USER/USING SQL CONNECT statement, and the ODBC Driver makes use of the user ID and password values provided on the APIs to perform authentication.

7.4.4 ODBC support for 2 MB SQL

Currently, the maximum size in bytes of an SQL statement string (including white spaces) allowed on the SQLPrepare, SQLExecDirect and SQLNativeSql APIs is 32765. DB2 V8 running in NFM supports a maximum statement size of 2MB (2097152). DB2 V8 ODBC will now support SQL statements up to 2 MB in size when connected to DB2 V8 servers running in NFM.

This enhancement requires APAR PQ88582/PTF UQ91257.

7.4.5 SQLcancel

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

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

Currently only dynamic SQL statements are interruptible. Stored procedures cannot be interrupted. Transaction level SQL statements like CONNECT, COMMIT and ROLLBACK cannot be interrupted. Even BIND PACKAGE cannot be interrupted.

7.5 XML support in DB2 for z/OS V8

For the past few years, XML has increasingly become the de facto data format on the internet, on corporate intranets, and for data exchange.

Up to DB2 V7, you could use LOB or CLOB columns to store XML documents in relational tables. It was recommended that character objects be used to enable use of SQL character functions, such as concatenate, on the XML CLOBs.

If you needed XML data from traditional relational databases, you had to create an application to convert the DB2 data to the XML format or use the XML extender.

DB2 V8 provides SQL/XML publishing functions to help with XML document composition directly from DB2 data.

7.5.1 XML extender

Up to DB2 V7, you could use the XML extender feature for manipulating XML data into and out of DB2.

Description

The XML extender is a set of external programs which are defined to execute as stored procedures and user-defined functions, invoked using SQL statements. The functionality provided by the XML extender is very rich, including intact storage, shredding to relational tables, indexing and searching. However, this function is still seen as an “add on” and the external routines do not perform as well as code inside the database engine.

Example

In DB2 V8 SQL/XML publishing functions become available. Storing and shredding or decomposing XML documents is still only provided in the XML extender. As an example we present in Figure 7-19 the results of shredding an XML document as a function of the document size with the XML extender.

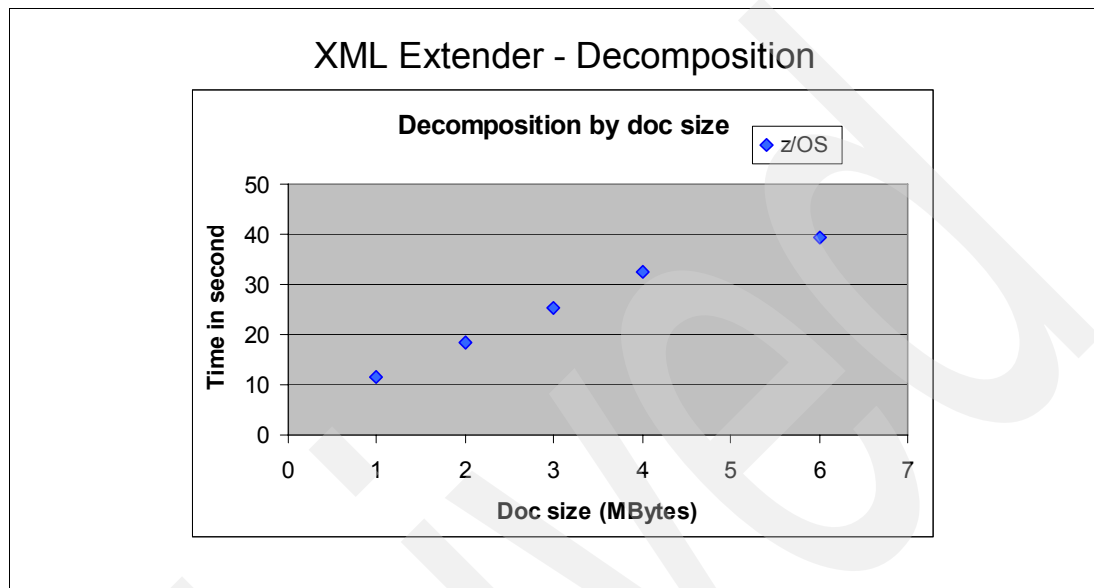


Figure 7-19 XML extender decomposition

Decomposing an XML document into 3 tables of 20 columns on a z990 system takes 10 to 40 seconds for document sizes ranging from 1 to 6 MB.

7.5.2 XML publishing functions

DB2 V8 now provides seven new built-in XML publishing functions to help with XML document composition from DB2 data.

Description

The XML publishing functions are built-in DB2 functions. They run inside the DB2 address spaces, unlike external User Defined Functions (UDFs), for example from the XML extender, that run in a WLM managed address space outside of DB2. The fact that the XML publishing functions are built into the DB2 engine gives them better performance.

XML data type

An XML data type is an internal representation of XML, and can be used only as input to built-in functions that accept this data type as input. XML is a transient data type that cannot be stored in the database or returned to an application. Transient means that this data type only exists during query processing. There is no persistent data of this type and it is not an external data type that can be declared in application programs. In other words, the XML data type cannot be stored in a database or returned to an application.

Valid values for the XML data type include the following:

- ▶ An element
- ▶ A forest of elements
- ▶ The textual content of an element
- ▶ An empty XML value

There are restrictions for the use of the XML data type:

- ▶ A query result cannot contain this data type
- ▶ Columns of a view cannot contain this data type
- ▶ XML data cannot be used in SORT, that is GROUP BY or ORDER BY and predicates
- ▶ The XML data type is not compatible with any other data type

The only supported operation is to serialize (by using the XML2CLOB function) the XML value into a string that is stored as a CLOB value.

XML publishing functions

The new publishing functions are:

- ▶ **XML2CLOB**

The XML2CLOB function returns a CLOB representation of an XML value.

- ▶ **XMLELEMENT**

The XMLELEMENT function returns an XML element from one or more arguments.

- ▶ **XMLATTRIBUTES**

This function constructs XML attributes from the arguments.

- ▶ **XMLFOREST**

The XMLFOREST function returns a bunch of XML elements that all share a specific pattern from a list of expressions.

- ▶ **XMLCONCAT**

The XMLCONCAT function returns a concatenation of XML elements that are generated from a concatenation of two or more arguments.

- ▶ **XMLAGG**

The XMLAGG function returns an aggregation of XML elements from a collection of XML elements.

- ▶ **XMLNAMESPACES**

The function XMLNAMESPACES returns the declaration of XML namespaces.

For a detailed description of the new XML publishing functions, see *XML for DB2 Information Integration*, SG24-6994.

Performance

In this section we present the XML performance measurements.

Performance measurement description

In this measurement we used:

- ▶ DB2 for z/OS V8
- ▶ z/OS Release 1.4
- ▶ XML extender V8
- ▶ z990
- ▶ ESS 800 DASD (with FICON channel)

This measurement compares the performance of the XML built-in publishing functions to the equivalent RDB_NODE composition with the XML extender user defined functions. See Example 7-1 for the SQL used in this measurement.

Example 7-1 XML publishing example

```

INSERT INTO PUB_TAB(DOC)
SELECT XML2CLOB(
  XMLELEMENT(NAME "AUCT_DB",
    XMLAGG(
      XMLELEMENT(NAME "CATEGORIE",
        XMLATTRIBUTES(C.CATEGORY_ID AS "NO" ),
        XMLCONCAT (
          XMLELEMENT(NAME "INFORMATIONS",
            XMLATTRIBUTES(P.NAME AS "PERSON_NAME" ),
            XMLCONCAT(
              XMLELEMENT(NAME "STATISTIQUES",
                XMLFOREST(P.GENDER AS "SEXE",
                  P.AGE AS "AGE",
                  P.EDUCATION AS "EDUCATION",
                  P.INCOME AS "REVENUE")
            ),
            XMLELEMENT(NAME "COORDONEES",
              XMLFOREST(P.AD_STREET AS "RUE",
                P.AD_CITY AS "VILLE",
                P.AD_COUNTRY AS "PAYS")
            ),
            XMLELEMENT(NAME "RESEAU",
              XMLFOREST(P.EMAILADDRESS AS "COURRIER",
                P.HOMEPAGE AS "PAGEPERSON")
            )
          )
        ),
      XMLELEMENT(NAME "CARTEPAIEMENT", P.CREDITCARD )
    )
  )
  ORDER BY C.CATEGORY_ID)
)
)

AS "result" FROM PERSON P, CATEGORY C, INTEREST I
WHERE P.PERSON_ID = I.PERSON_ID
AND C.CATEGORY_ID = I.CATEGORY
GROUP BY C.CATEGORY_ID ;

```

Performance measurement result

In Figure 7-20 the results of the measurement are presented in seconds.

Composition - Extender and Publishing

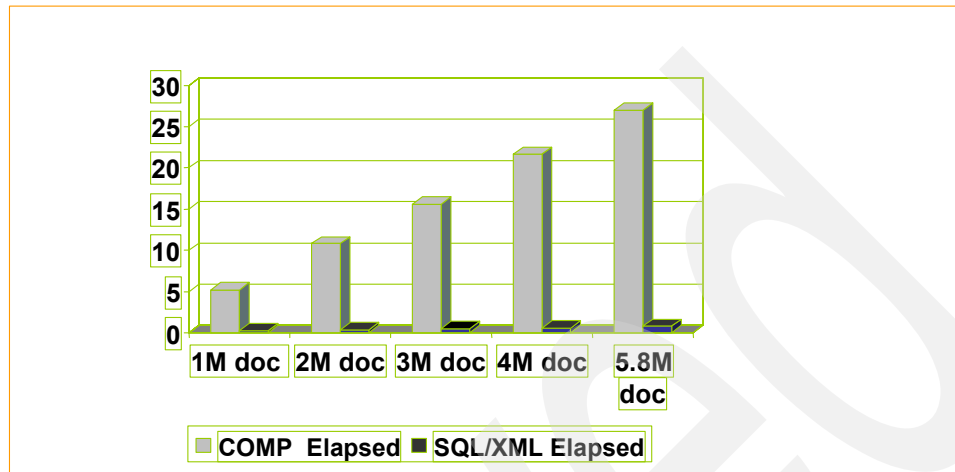


Figure 7-20 XML extender vs. SQL publishing functions

The XML publishing functions outperform the XML composition by a factor 20 and more mostly due to the fact that the XML publishing functions are built into the DB2 engine. The measurement was done for documents up to 5.8 MB as the limit of 10240 rows was reached. The XML Extender cannot insert more than 10240 rows into any table when composing an XML document.

In Figure 7-21 the results for document composition for CLOBs up to 6 MB are presented.

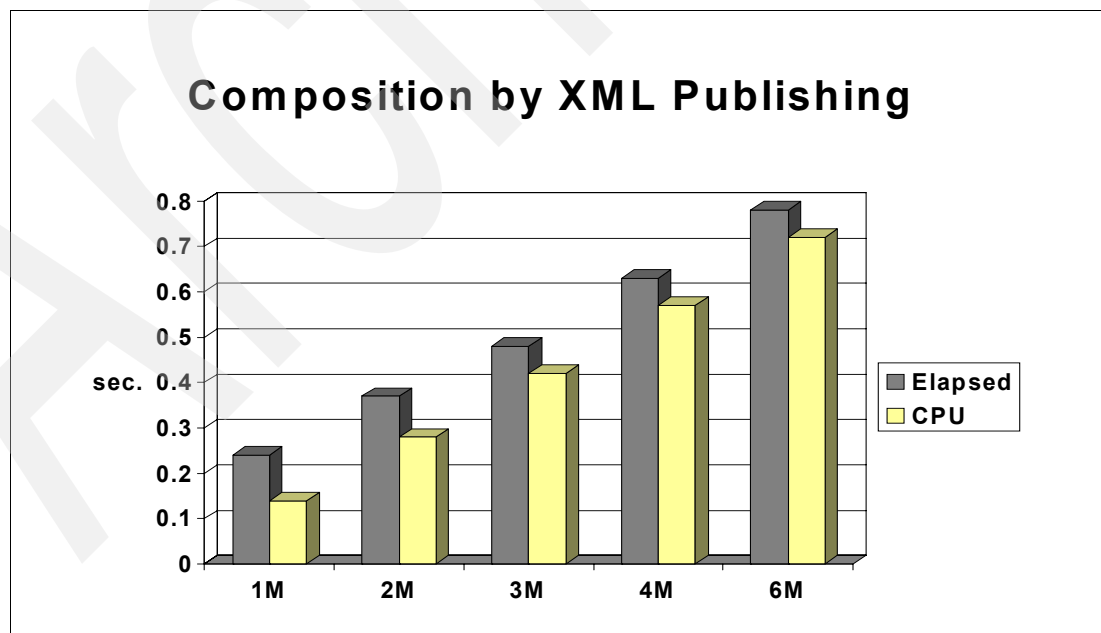


Figure 7-21 Composition by XML publishing functions

The elapsed and CPU time used by the XML publishing functions to compose XML documents scales nearly linear.

7.5.3 Conclusion

The new XML publishing functions outperform significantly the corresponding UDFs from the XML extender.

7.5.4 Recommendation

For the purpose of composing XML documents from DB2 data the use of the XML publishing function is preferred over the corresponding DB2 XML extender UDFs. They are generally more efficient, flexible and versatile than the DB2 XML Extender publishing functions.

7.6 Enterprise Workload Manager

Look at Enterprise Workload Manager (EWLM) as z/OS WLM applied to a heterogeneous environment. EWLM introduces the goal-oriented z/OS WLM philosophy to all platforms.

See the book *IBM Enterprise Workload Manager*, SG24-6350, for a detailed description of EWLM.

7.6.1 Introduction

Enterprise Workload Manager enables you to automatically monitor and manage multi-tiered, distributed, heterogeneous or homogeneous workloads across an IT infrastructure to better achieve defined business goals for end-user services. These capabilities allow you to identify work requests based on service class definitions, track performance of those requests across server and subsystem boundaries, and manage the underlying physical and network resources to set specified performance goals for each service class. By bringing this self-tuning technology to the set of servers, routers, and other devices enabled with Java virtual machine support, Enterprise Workload Manager can help enable greater levels of performance management for distributed systems.

In the first release, the Enterprise Workload Manager provides goal-based end to end performance monitoring of an application and influences network traffic routing decisions by network load balancers for improved application performance and server effectiveness. A systems administrator will be able to monitor multiple servers, answering such questions as:

- ▶ How well is the work progressing?
- ▶ Are the business commitments for end-user service being met?
- ▶ If the performance goals are not being met, which components of which servers are causing the problem?
- ▶ Which type of user is being affected by the problem?

And performing such tasks as:

- ▶ Easily identifying where time is being spent in a distributed application that spans different operating systems, applications, networks and routers.
- ▶ Specifying a performance goal that applies to each class of user of a distributed application.
- ▶ Identifying which servers are actively supporting a given application and what applications are running on a given server.

7.6.2 Description

EWLM is an implementation of policy-based performance management. EWLM Correlator Support is for server side only and it is only for TCP/IP. It is implemented with PTF UK03835 for APAR PQ91509 for DB2 V8, and it allows you to do performance measurement using ARM instrumentation. PTF UK03058 for APAR PQ99707 is also recommended for performance improvement for JCC clients. PTF UA15189 for APARs OA07196 and PTF UA15456 for APAR OA07356 implement this function from the z/OS side. APAR OA12005 (currently open) for z/OS 1.6 and z/OS 1.7 is also recommended for performance improvements.

The scope of management is a set of servers that you logically group into what is called an EWLM management domain. The set of servers included in the management domain have some type of relationship; for example, z/OS server with DB2 V8, AIX server with WAS. See Figure 7-22.

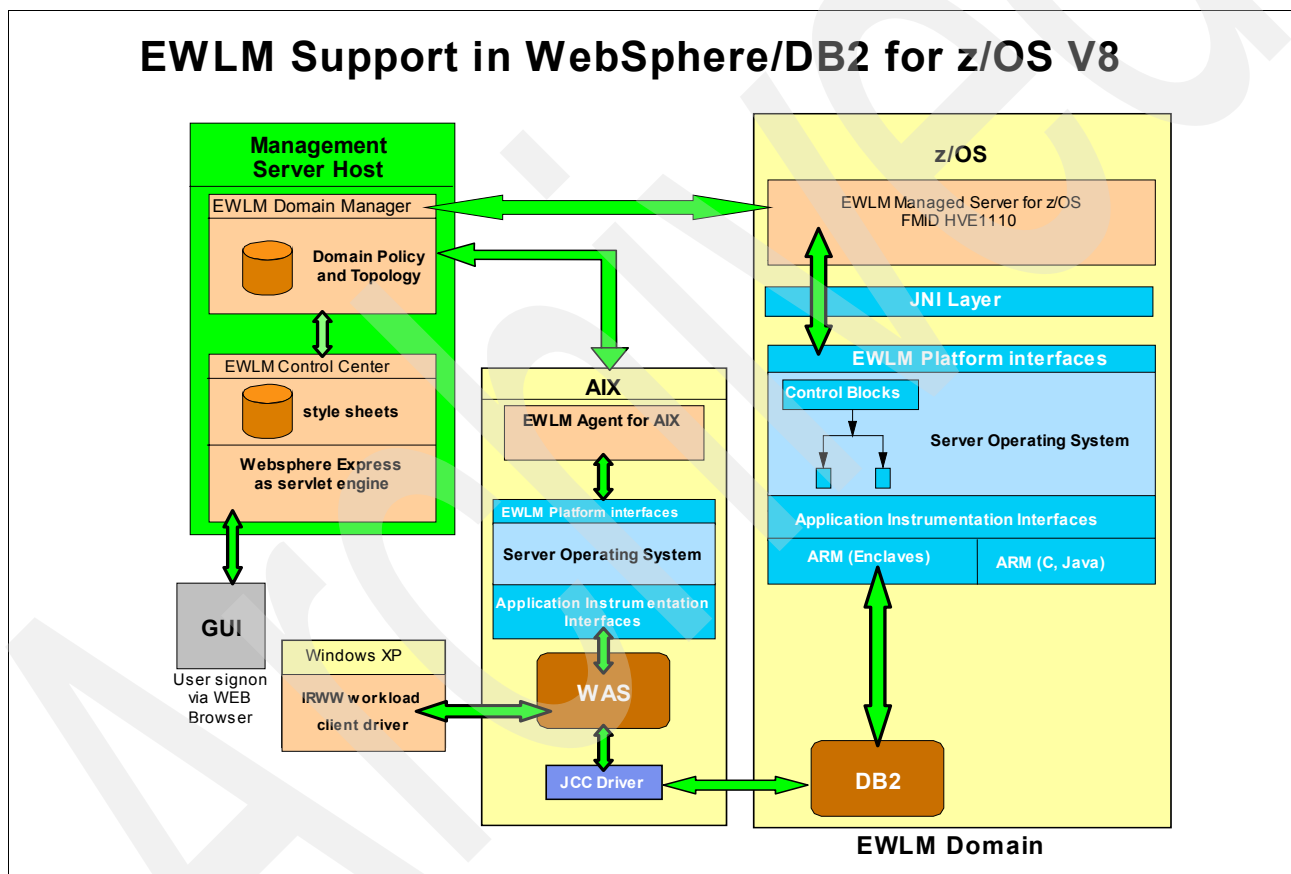


Figure 7-22 EWLM support with WAS/AIX and DB2 for z/OS V8

There is a management focal point for each EWLM management domain, called the EWLM domain manager. The domain manager coordinates policy actions across the servers, tracks the state of those servers, and accumulates performance statistics on behalf of the domain.

On each server (operating system) instance in the management domain there is a thin layer of EWLM logic installed called the EWLM managed server. The managed server layer understands each of the supported operating systems, gathering resource usage and delay statistics known to the operating system.

A second role of the managed server layer is to gather relevant transaction-related statistics from middleware applications and route these statistics to the EWLM domain manager. The application middleware implementations, such as WebSphere Application Server, understand when a piece of work starts and stops, and the middleware understands when a piece of work has been routed to another server for processing, for example, when a WAS server routes a request to a DB2 database server.

An important aspect of the EWLM approach is that all data collection and aggregation activities are driven by a common service level policy, called the EWLM Domain Policy.

The final link is the EWLM Control Center. This is where all the EWLM concepts come together; where you can create an EWLM domain policy and activate that policy on all servers of the domain with a single click. See the following screen shots in order to get a look and feel of the EWLM Control Center.

Figure 7-23 shows the Managed Server view of your EWLM Control Center. It provides an overview of the status of all the managed servers in the EWLM management domain. In this example, there is a Communication error between domain manager and managed servers on AIX and z/OS that requires attention. The managed server on Windows appears to be running fine as indicated by the “Active” state.

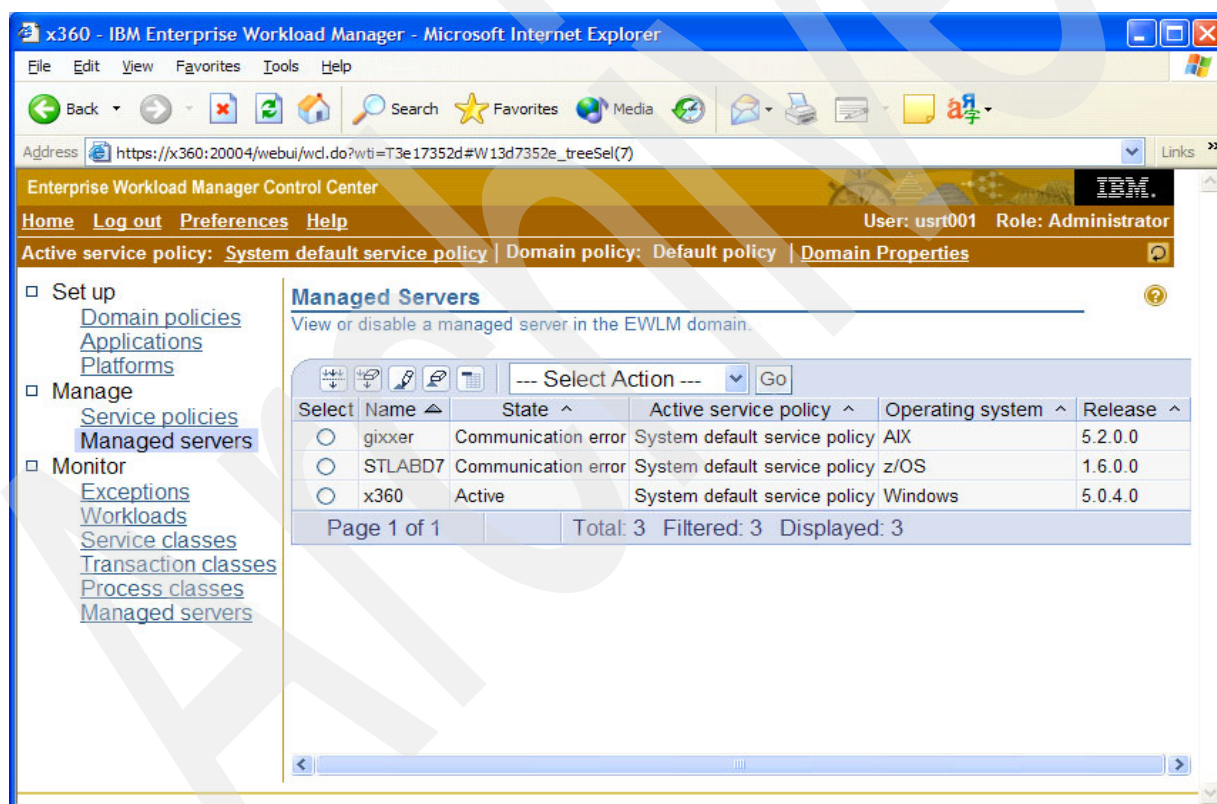


Figure 7-23 Enterprise Workload Manager Control Center - Managed servers

Figure 7-24 is an example of a server topology report. Note the location name (STLABD7) is displayed in the diagram when DB2 for z/OS is EWLM enabled.

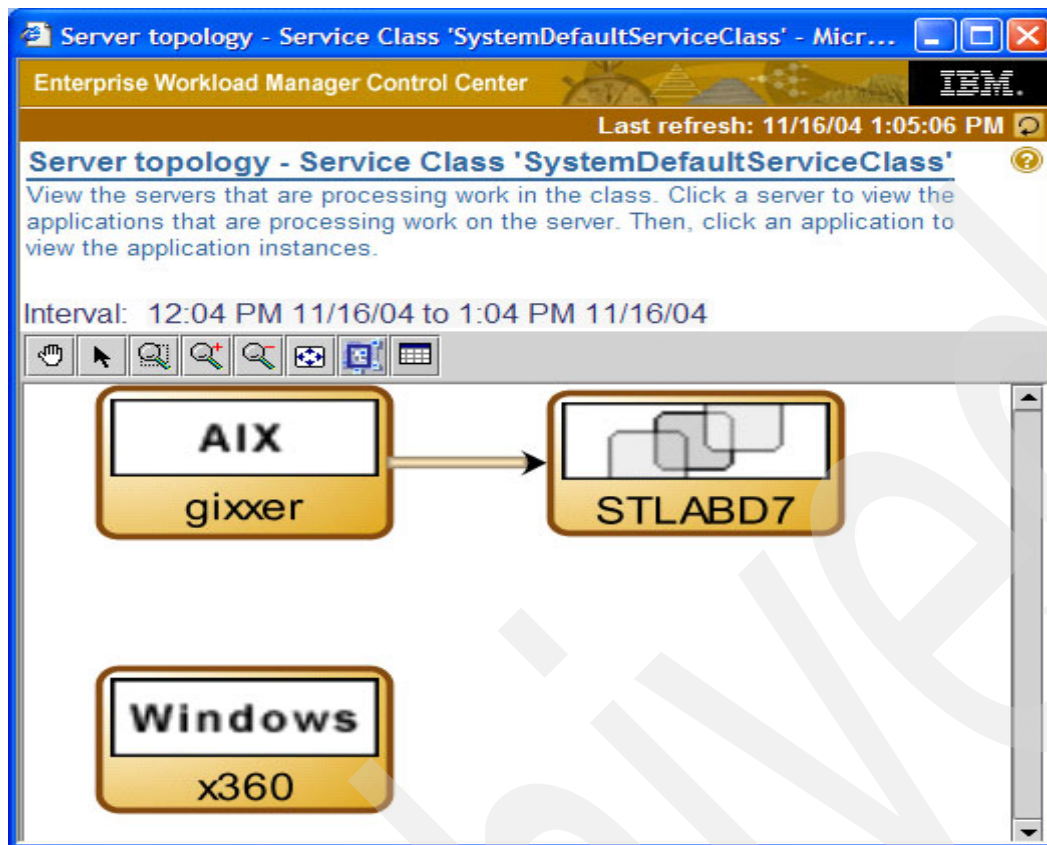


Figure 7-24 Enterprise Workload Manager Control Center - Server topology

When you click on the table view button from Server Topology, you will see a table representation of the topology like the above along with transaction statistics. See Figure 7-25.

Hop	Name	Stopped transactions	Failed transactions	Successful transactions	I/O
0	x360	0	0	0	0%
0	WebSphere [server1]	0	0	0	0%
0	x360.server1	0	0	0	0%
0	gixxer	0	0	0	0%
0	WebSphere [server1]	0	0	0	0%
0	gixxer.server1	0	0	0	0%
0	WebSphere [server1]	0	0	0	0%
0	gixxer.server1	0	0	0	0%
0	IBM DB2 Universal Database [db2inst1]	0	0	0	0%
0	7	0	0	0	0%
1	STLABD7	0	0	0	0%
1	DDF [DSND81A]	0	4	0	0%
1	D81A	0	4	0	0%
1	DDF [DSND81A]	0	0	0	0%
1	D81A	0	0	0	0%

Figure 7-25 EWLM control center - Service class statistics

The following information can be seen in the table view by scrolling to the right:

- ▶ Average response time
- ▶ Active time (milliseconds)
- ▶ Processor time (microseconds)
- ▶ Processor delay%
- ▶ Processor using%
- ▶ Stopped transaction
- ▶ Failed transactions
- ▶ Successful transactions
- ▶ I/O delay%
- ▶ Page delay%
- ▶ Idle%
- ▶ Other%
- ▶ Topology uncertainty count
- ▶ Reporting gap count
- ▶ Hard page faults
- ▶ Queue time (milliseconds)
- ▶ Standard deviation

7.6.3 Performance

Companies implementing EWLM will be able to reduce the amount of time their administrators spend to locate performance bottlenecks and optimize workloads. Their current hardware utilization will increase as EWLM instrumentation will provide data collection and aggregation for optimally changing resource settings across the existing platforms.

Preliminary tests to evaluate the impact on throughput of an EWLM implementation on a server combination WebSphere/AIX with DB2 for z/OS V8 are shown in Figure 7-26.

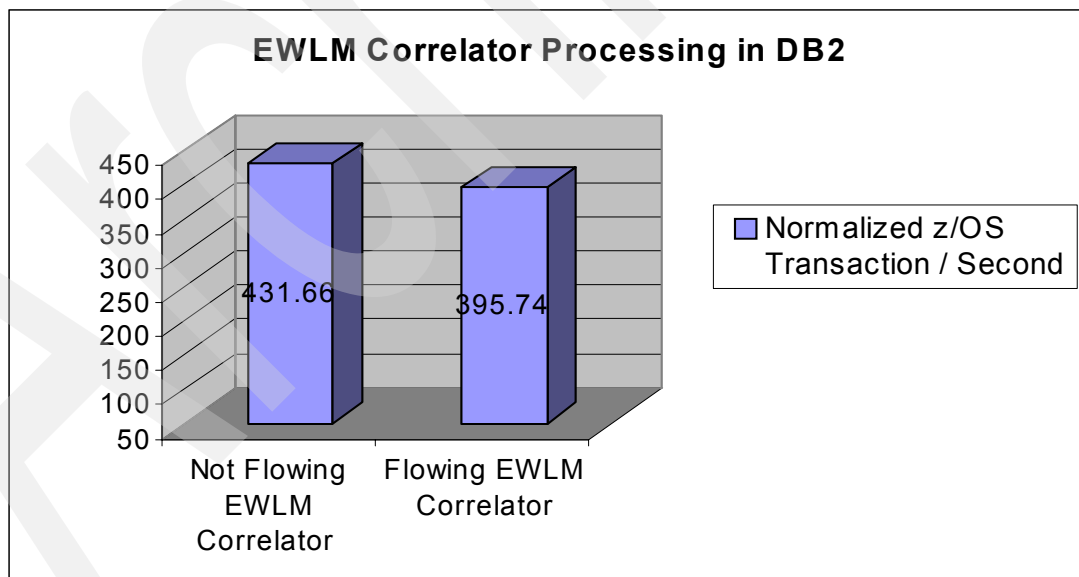


Figure 7-26 EWLM implementation cost

The diagram reflects the latest EWLM for DB2 for z/OS V8 performance numbers. With the implementation of an improved EWLM interface, EWLM correlator processing cost in normalized transactions per seconds is 8.33% measured with IRWW servlet workload. Since part of the overhead is due to CPU utilization by the EWLM Client, this utilization could be

offloaded to a zSeries Application Assist Processor (zAAP), if available. The zAAP processor delivers a dedicated zSeries processor for Java workloads, free from ISV and OS costs.

Archived

Archived

Data sharing enhancements

DB2 data sharing allows you the highest level of scalability, performance, and continuous availability by taking advantage of the z Parallel Sysplex technology to provide applications with full concurrent read and write access to shared DB2 data. For an introduction, refer to DB2 for z/OS: Data Sharing in a Nutshell, SG24-7322.

DB2 data sharing was introduced in 1994 with DB2 V4, and continues to evolve exploiting key functions of the IBM System z™ servers.

Data sharing DB2 V8 provides a number of significant enhancements to improve availability, performance and usability of DB2 data sharing:

- ▶ **CF request batching:** This enhancement utilizes new functionality in z/OS 1.4 and CF level 12 to enable DB2 to:
 - Register and write multiple pages to a group buffer pool in a single request
 - Read multiple pages from a group buffer pool for castout processing

This reduces the amount of traffic to and from the coupling facility for writes to group buffer pools and reads for castout processing, thus reducing the data sharing overhead insert, update and delete-intensive workloads.
- ▶ **Locking protocol level 2:** Protocol Level 2 remaps IX parent L-locks from XES-X to XES-S locks. Data sharing locking performance benefits because this allows IX and IS parent global L-locks to be granted without invoking global lock contention processing to determine that the new IX or IS lock is compatible with existing IX or IS locks.
- ▶ **Change to IMMEDIATE default BIND option:** In prior releases of DB2, any remaining changed pages in a data sharing environment are written to the group buffer pool during phase 2 of commit, unless otherwise specified by the IMMEDIATE BIND parameter. This enhancement changes the default processing to write changed pages during commit phase 1. DB2 will no longer write changed pages during phase 2 of commit processing.
- ▶ **DB2 I/O CI limit removed:** Prior to DB2 V8, DB2 did not schedule a single I/O for a data set with CIs that span a range of more than 180 CIs. V8 removes this limit for list prefetch and castout I/O requests. This is true in CM and NFM.
- ▶ **Miscellaneous items:**

Impact on coupling facility: You do not need to resize the group buffer pools, lock structure or SCA in the coupling facility as a result of moving to DB2 V8, even though the virtual buffer pools have now moved above the bar and have a longer memory address.

Improved LPL recovery: Currently LPL recovery requires exclusive access to the table space. DB2 V8 introduces a new serialization mechanism whereby LPL recovery is much less disruptive. In addition, DB2 V8 automatically attempts to recover LPL pages as they go into LPL, when it determines the recovery will probably succeed. Instrumentation has also improved whereby DB2 now attempts to indicate the reason pages are added to the LPL.

Restart LIGHT enhancements: If indoubt units of recovery (UR) exist at the end of restart recovery, DB2 will now remain running so that the indoubt URs can be resolved. After all the indoubt URs have been resolved, the DB2 member that is running in LIGHT(YES) mode will shut down and can be restarted normally.

Change to close processing for pseudo close: Table spaces defined as CLOSE NO are still closed during pseudo close processing at the end of the PCLOSET interval. This caused degraded performance because the data sets must be physically opened again on the next access. DB2 V8 now really honors the CLOSE NO attribute during pseudo close processing. See PTF UQ74866 for APAR PQ69741 for DB2 V7.

Buffer pool long term page fix: DB2 V8 allows you to fix the buffer pool pages once in memory and keep them fixed in real storage. This avoids the processing time that DB2 needs to fix and free pages each time there is an I/O.

Batched index page splits: Prior to V8, index page splits for indexes with inter-DB2 read/write interest required multiple synchronous log writes and multiple separate writes to the group buffer pool. This was required to ensure the integrity of the index for other DB2 members during the actual page split. DB2 V8 is able to accumulate the index page split updates and process them as a single entity, thereby reducing log writes and coupling facility traffic.

8.1 CF request batching

z/OS 1.4 and CF Level 12 bring two new coupling facility “commands”, Write And Register Multiple (WARM) and Read For Castout Multiple (RFCOM).

DB2 V8 data sharing will use these new coupling facility commands, if available, to reduce the amount of traffic to and from the coupling facility for writes to group buffer pools and reads for castout processing, thus reducing the data sharing overhead for insert, update and delete-intensive workloads.

When the group buffer pool is allocated in a coupling facility with CFLEVEL= 12 or higher, DB2 can register a list of pages that are being prefetched with one request to the coupling facility. This can be used for sequential prefetch (including sequential detection) and list prefetch.

For those pages that are cached as “changed” in the group buffer pool, or those that are locked for castout, DB2 still retrieves the changed page from the group buffer pool one at a time. For pages that are cached as “clean” in the group buffer pool, DB2 can get the pages from the group buffer pool (one page at a time), or can include the pages in the DASD read I/O request, depending on which is most efficient.

z/OS 1.4 and CF level 12 introduce two new CF “commands” to:

- ▶ Write And Register Multiple (WARM) pages to a group buffer pool with a single command.
- ▶ Read multiple pages from a group buffer pool for castout processing with a single CF read request. The actual command is called Read For Castout Multiple (RFCOM).

CF level 13 also introduces performance improvements related to DB2 castout processing. See the Web page:

<http://www.ibm.com/servers/eserver/zseries/pso/coupling.html>

DB2 V8 uses the CF request batching commands to read and write pages to and from the group buffer pool if more than one page needs to be read or written.

DB2 instrumentation (statistics IFCID 2 and accounting 3 and 148) records are enhanced to reflect the usage of these new commands. The DB2 PE accounting and statistics reports are enhanced to externalize the new counters that indicate the number of WARM and RFCOM requests in the GBP section:

- ▶ WRITE AND REGISTER MULTI
The number of Write and Register Multiple (WARM) requests.
- ▶ PAGES WRITE & REG MULTI
The number of pages written using Write and Register Multiple (WARM) requests.
- ▶ READ FOR CASTOUT MULTI
The number of Read For Castout Multiple (RFCOM) requests.

8.1.1 Performance

A number of performance studies have been done to understand the impact of using CF request batching. For these studies, a two member data sharing group was used, running the IRWW workload. z/OS 1.3 was compared to z/OS 1.4, to isolate the impact of CF request batching. CF request batching is not available in z/OS 1.3.

Table 8-1 shows extracts from DB2 PE statistics reports for an Online Transaction Processing workload (OLTP). The two columns of numbers within each z/OS represent values per transaction for data sharing member 1 and member 2.

Table 8-1 CF Request Batching - DB2 PE extract (OLTP)

	z/OS 1.3		z/OS 1.4	
	Member 1	Member 2	Member 1	Member 2
Pages Castout	3.91	4.35	3.71	4.61
Write & Register	6.28	6.29	2.22	2.22
Write & Register Multi	0.00	0.00	0.59	0.59
Pages Write & Reg Multi	0.00	0.00	4.09	4.09
Read for Castout	3.91	4.35	0.00	0.01
Read for Castout Multi	0.00	0.00	0.48	0.60

It can be clearly seen from Table 8-1 that CF request batching was not used in z/OS 1.3. The counters for “Write & Register Multi”, “Pages Write & Reg Multi” and “Read for Castout Multi” are all zero in z/OS 1.3.

CF request batching was used under z/OS 1.4:

- An average of 6.9 pages were written per WARM command

$$\text{“Pages Write \& Reg Multi”} / \text{“Write \& Register Multi”} = 4.09 / 0.59 = 6.9$$
- An average of 7.7 pages were read per RFCOM command.

$$(\text{“Pages Castout”} - \text{“Read for Castout”}) / \text{Read for Castout Multi} = (4.61 - 0.01) / 0.60 = 7.7$$

Table 8-2 shows extracts from RMF reports for the same OLTP workload.

Table 8-2 CF Request Batching - RMF extract (OLTP)

		z/OS 1.3		z/OS 1.4	
Requests / sec		14,082		9,880	
Sync requests					
Serv time (usec)	% of all req	17.7	95.5	19.5	86.6
Async requests					
Serv time (usec)	% of all req	173.9	4.4	192.8	13.3
CF utilization (%)		21.7		21.0	

As we can see from Table 8-2, the test using z/OS 1.4 results in significantly fewer requests to the coupling facility (9,880 compared to 14,082 for this workload). This is the direct result of CF request batching. We can also see the service times for both synchronous requests and asynchronous requests to the coupling facility have increased slightly. On average, the coupling facility is doing more work per each request, as a number of page movements may need to be processed with each request. We also notice the coupling facility utilization has

decreased slightly. The CF Link utilization also decreased slightly, as fewer messages are passed to the coupling facility.

The table also shows a jump in asynchronous requests when CF request batching was used. The actual number of asynchronous requests doubled from 620 to 1314 requests. This increase can be attributed to the heuristic conversion by the CFCC of synchronous requests to asynchronous requests due to more work being processed in each batch request.

The next series of studies investigate the impact of CF request batching for a batch workload. For these tests, one batch program performs a fetch-update of a single row for the first 2.5 million rows of a 5 million row table. A second batch program, running on the other DB2 member, performs a fetch-update of the other half of the 5 million row table. During the execution of these two batch programs, the table space is GBP dependent but with minimal global contention.

Table 8-3 shows extracts from DB2 PE statistics reports, revealing the impact of CF request batching on a batch workload.

Table 8-3 CF Request Batching - DB2 PE extract (batch)

	z/OS 1.3		z/OS 1.4	
Pages Castout	2,121.8K	2,253.4K	2,583.4K	2,749.8K
Write & Register	2,206.1K	2,206.1K	205	163
Write & Register Multi	0	0	209.5K	210.2K
Pages Write & Reg Multi	0	0	2,202.7K	2,203.0K
Read for Castout	2,121.8K	2,253.4K	363.5K	364.1K
Read for Castout Multi	0	0.	750.2K	790.7K

The impact of CF request batching appears to be much more pronounced. In this test, every page must be updated.

Without CF request batching, DB2 must write and register every page individually to the coupling facility. This results in over 2.2 million individual requests for each DB2 member.

With CF request batching, each DB2 member only needs a little over 200,000 requests to the coupling facility to support the same workload.

In addition, castout processing is dramatically improved. Rather than requiring over 4.3 million individual reads for castout processing without CF request batching, DB2 is able to reduce the number of requests to the coupling facility for castout processing, to a little over 2 million requests.

Rather than seeing only one DB2 member performing all the castout processing for the single page set we used during the tests, we can see both DB2 members performing castout processing for the same page set, both with and without CF request batching. This is normal. In the normal case, when DB2 decides that castout needs to be done for an object, the castout request is sent to the castout owner. However in times of high activity, the DB2 member that detected castout must be done can decide to perform the castout locally.

It is also normal to see a mixture of CF request batching and non-batching castout reads. Recall that DB2 uses CF request batching when only multiple pages are to be castout.

CF request batching is also used when DB2 writes pages to the secondary GBP when BP Duplexing is used. However, DB2 will only use CF request batching when both the primary

and secondary GBP structures have the CF request batching capability. Since CF request batching will only be used for reading castout pages from the primary GBP, there is no additional CPU saving for castout with or without GBP Duplexing. However, if you do not consider prefetch, then the additional CPU saving with CF request batching in a duplexed GBP environment will be half of the CPU saving without GBP duplexing.

Table 8-4 shows extracts from RMF reports for the same batch workload.

Table 8-4 CF Request Batching - RMF extract (batch)

		z/OS 1.3		z/OS 1.4	
Requests / sec		14,627		6,715	
Sync Requests					
Serv time (usec)	% of all req	10.5	98.0	12.0	87.7
Async Requests					
Serv time (usec)	% of all req	209.8	2.0	241.4	12.3
CF Utilization (%)		15.7		14.4	

Once again, we can see z/OS 1.4 results in significantly fewer requests to the coupling facility (6,715 compared to 14,627). We can also see the service times for both synchronous requests and asynchronous requests to the coupling facility have increased slightly, as well as notice the coupling facility utilization has decreased slightly.

The table also shows quite a jump in asynchronous requests when CF request batching was used, over 12% of all requests, compared with only 2% when CF request batching was not used. This trend is similar to what we noticed when we earlier reviewed the RMF extracts for the OLTP workload. The increase can be attributed to the heuristic conversion by the CFCC of synchronous requests to asynchronous requests due to more work being processed in each batch request.

Table 8-5 compares the CPU used by the DBM1 address space for the OLTP workload compared with the batch workload.

Table 8-5 CF Request Batching - DB2 CPU

	z/OS 1.3		z/OS 1.4		Delta (1.4 / 1.3)
DBM1 - (OLTP) (msec / commit)	0.536	0.545	0.427	0.473	-17%
Overall CPU time	3.229	3.224	3.111	3.368	+0%
DBM1 - (Batch) (msec / commit)	119.88	130.11	81.5	89.68	-32%
Overall CPU time	656.40	709.74	618.98	680.89	-5%

This table shows CF request batching has potential for significant saving in CPU charged to DBM1 address space, in both OLTP and batch workloads, but particularly batch.

The overall CPU times in this table include the transaction class 2 CPU times, the MSTR, DBM1 and IRLM CPU times. So, we can see the overall impact of CF request batching was

not all that significant for our OLTP workload, while the overall impact on our batch workload was indeed significant.

CF request batching also benefits DB2 performance in other ways. DB2 commit processing performance is improved, where any remaining changed pages must be synchronously written to the group buffer pool during commit processing. For GPBCACHE(ALL) page sets, DB2 is able to more efficiently write prefetched pages into the group buffer pool as it reads them from DASD.

8.1.2 Conclusion

CF request batching allows DB2 V8 to reduce the amount of traffic to and from the coupling facility for both writes to and reads from the group buffer pool for castout processing, thus reducing the data sharing overhead for most workloads.

The most benefit is anticipated for workloads which update large numbers of pages belonging to GBP-dependent objects, for example, batch workloads.

This enhancement also has potential to reduce the CPU used by the DBM1 address space, as fewer messages are passed to and from the coupling facility. Coupling facility link utilization may also decrease slightly. As a consequence, coupling facility utilization may be slightly higher as message service times increase. This is a direct result of the coupling facility having to perform more work to service each batched request. The data sharing overhead caused by castout processing is also reduced.

8.1.3 Recommendations

There is nothing you need to do in DB2 to enable DB2 to use CF request batching. However, you need to have the prerequisite software in place, z/OS 1.4 and CFCC Level 12.

We recommend you implement a strategy to keep reasonably current with software maintenance and coupling facility microcode levels, in order to take early advantage of enhancements such as CF request batching. This is particularly important in a data sharing environment.

8.2 Locking protocol level 2

Protocol Level 2 remaps parent IX L-locks from XES-X to XES-S locks. Data sharing locking performance benefits because this allows IX and IS parent global L-locks to be granted without invoking global lock contention processing to determine that the new IX or IS lock is compatible with existing IX or IS locks.

The purpose of this enhancement is to avoid the cost of global contention processing whenever possible. It will also improve availability due to a reduction in retained locks following a DB2 subsystem or MVS system failure.

Refer to section 11.1 of *DB2 UDB for z/OS Version 8: Everything You Ever Wanted to Know, ... and More*, SG24-6079, for a general review of locking in a data sharing environment.

XES contention caused by parent L-Locks is reasonably common in a data sharing environment. On page set open (an initial open or open after a pseudo close) DB2 normally tries to open the table space in RW. (DB2 rarely opens a table space in RO.) To do this, DB2 must ask for an L-Lock. This L-lock is normally an IS or IX L-lock, depending on whether or not a SELECT or UPDATEable SQL statement caused the table space to open. If any other

DB2 member already has the table space open for RW, global lock contention generally occurs.

DB2 V8 attempts to reduce this global contention by remapping parent IX L-locks from XES-X to XES-S locks. Data sharing locking performance benefits because parent IX and IS L-locks are now both mapped to XES-S locks and are therefore compatible and can now be granted locally by XES. DB2 no longer needs to wait for global lock contention processing to determine that a new parent IX or IS L-lock is compatible with existing parent IX or IS L-locks.

The majority of parent L-lock contention occurs when the table space is opened by at least one DB2 member in RW:

- ▶ IS-IS: We want to execute some read-only SQL against a table space and there are some other members who currently have some read-only SQL active against the same table space.
- ▶ IS-IX: We want to execute some read-only SQL against a table space and there are some other members who currently have some update SQL active against the same table space.
- ▶ IX-IS: We want to execute some update SQL against a table space and there are some other members who currently have some read-only SQL active against the same table space.
- ▶ IX-IX: We want to execute some update SQL against a table space and there are some other members who currently have some update SQL active against the same table space.

Parent lock contention with parent S L-locks is less frequent than checking for contention with parent IS and IX L-locks. Parent S L-locks are only acquired when DB2 is about to issue read-only SQL against a table space opened for RO.

To ensure that parent IX L-locks remain incompatible with parent S L-locks, S table and table space L-locks are remapped to XES-X locks. This means that additional global contention processing will now be done to verify that an S L-lock is compatible with another S L-lock, but this is a relatively rare case (executing read-only SQL against a table space, where at least one other DB2 member has the table space open in RO and currently has some read-only SQL active against the same table space).

Another impact of this change is that child L-locks are no longer propagated based on the parent L-lock. Instead, child L-locks are propagated based on the held state of the table space P-lock. If the table space P-lock is negotiated from X to SIX or IX, then child L-locks must be propagated.

It may be that some child L-locks are acquired before the page set P-lock is obtained. In this case child L-locks will automatically be propagated. This situation occurs because DB2 always acquires locks before accessing the data. In this case, DB2 acquires that L-lock before opening the table space to read the data. It can also happen during DB2 restart.

An implication of this change is that child L-locks will be propagated for longer than they are needed, however this should not be a concern. There will be a short period from the time where there is no intersystem read/write interest until the table space becomes non-GBP-dependent, that is, before the page set P-lock reverts to X. During this time, child L-locks will be propagated unnecessarily.

It is now important that L-locks and P-locks are maintained at the same level of granularity. Remember now that page set P-locks now determine when child L-locks must be propagated to XES.

For partitioned table spaces defined with LOCKPART NO, prior versions of DB2 lock only the last partition to indicate we have a lock on the whole table space. There are no L-locks held on each of the partition page sets. So, when should we propagate the child L-locks for the various partitions that are being used? We cannot tell by looking at the table space parent L-locks that we need to propagate the child locks since there no longer is a lock contention conflict that can trigger child lock propagation, and we cannot determine how each partition page set is being used by looking at the page set P-locks that are held at the partition level, while the parent L-lock is at the table space level with LOCKPART NO.

To overcome this problem, DB2 V8 obtains locks at the part level. LOCKPART NO behaves the same as LOCKPART YES.

In addition, LOCKPART YES is not compatible with LOCKSIZE TABLESPACE. However, if LOCKPART NO and LOCKSIZE TABLESPACE are specified then we will lock every partition, just as every partition is locked today when LOCKPART YES is used with ACQUIRE(ALLOCATE). With this change you may see additional locks being acquired on individual partitions even though LOCKPART(NO) is specified.

This change to the LOCKPART parameter behavior applies to both data sharing and non-data sharing environments.

Since the new locking protocol cannot co-exist with the old, the new protocol will only take effect after the first group-wide shutdown and restart after the data sharing group is in new-function mode (NFM). You do not have to delete the lock structure from the coupling facility prior to restarting DB2, to trigger DB2 on group restart to build a new lock structure. In addition, Protocol Level 2 will not be enabled if you merely ask DB2 to rebuild the lock structure while any DB2 member remains active.

The new mapping takes effect after the restart of the first member, after successful quiesce of all members in the DB2 data sharing group. So, to enable this feature, a group-wide outage is required.

No other changes are required to take advantage of this enhancement.

You can use the -DISPLAY GROUP command to check whether the new locking protocol is used (mapping IX IRLM L-locks to an S XES lock). Example 8-1 shows the output from a -DISPLAY GROUP command which shows Protocol Level 2 is active.

Example 8-1 Sample -DISPLAY GROUP output

```

DSN7100I  -DT21 DSN7GCMD
*** BEGIN DISPLAY OF GROUP(DSNT2  ) GROUP LEVEL(810) MODE(N)
          PROTOCOL LEVEL(2)  GROUP ATTACH NAME(DT2G)
-----
DB2      DB2 SYSTEM  IRLM
MEMBER   ID  SUBSYS  CMDPREF  STATUS  LVL NAME  SUBSYS  IRLMPROC
-----
DT21     1  DT21    -DT21    ACTIVE  810 STLABB9  IT21    DT21IRLM
DT22     3  DT22    -DT22    FAILED  810 STLABB6  IT22    DT22IRLM

```

The use of locking Protocol Level 2 requires that the PTFs for the following APARs are applied: PQ87756, PQ87168, and PQ86904 (IRLM).

8.2.1 Performance

We ran a number of benchmarks to try to understand the impact of the new locking protocol. Each test was performed using an IRWW benchmark executing against a two member data sharing group.

The tests were performed with DB2 V7 defaults and DB2 V8 defaults. V7 uses LOCKPART NO as the default, while V8 behaves as if the table spaces are defined with LOCKPART YES. So, for some workloads DB2 V8 would need to propagate more locks to the coupling facility than the same workload running in DB2 V7. This is true if the workload running does not require all of the partitions to be GBP dependent. For these tests, all partitions were GBP dependent. So, the impact of LOCKPART(NO) in V7 compared with V8 should be minimal.

We first review the impact on lock suspensions. Table 8-6 shows extracts from the Data Sharing Locking section of DB2 PE statistics reports. The two columns under DB2 V7 and V8 represent each DB2 member.

Table 8-6 Protocol Level 2 - Statistics Report extract

Suspends per commit	DB2 V7		DB2 V8	
IRLM Global Contention	0.01	0.01	0.01	0.01
XES Contention	0.54	0.52	0.00	0.00
False Contention	0.13	0.11	0.00	0.00

You can see from Table 8-6 that the benchmark produced significantly less XES contention as well as less false contention. This is because IX-L locks are now mapped to S XES locks which are compatible locks. The lock requests do not need to be suspended while XES communicates with the IRLMs to determine the IX-L locks are in fact compatible and can be granted. False contention is reduced because fewer lock table entries are occupied by X-XES locks. This is explained in more detail later in this section.

Table 8-7 shows extracts from DB2 PE accounting reports for the same test.

Table 8-7 Protocol Level 2 - Accounting Report extract

Class 3 suspend time (msec / commit)	DB2 V7		DB2 V8	
Lock / Latch (DB2 + IRLM)	6.219	5.708	0.080	0.109
Global Contention	8.442	7.878	0.180	0.197

This table reveals that, on average, each transaction enjoyed significantly fewer suspensions for lock and global contention. This is due to less XES contention, as a result of the new locking protocol.

Now, we have a look at the impact on the coupling facility and XCF links. Table 8-8 shows extracts from RMF XCF reports.

Table 8-8 Protocol Level 2 - RMF XCF Report extract

	DB2 V7		DB2 V8	
Req In (req / sec)	2,300	2,200	17	18

	DB2 V7		DB2 V8	
Req Out (req / sec)	2,300	2,200	17	18
CHPID Busy (%)	97.43	86.53	22.27	12.74

We can see from Table 8-8 that Protocol Level 2 generates a lot less XCF messaging traffic to and from the coupling facility.

Table 8-9 shows extracts from RMF CF activity reports for the test workload.

Table 8-9 Protocol Level 2 - RMF CF Activity Report extract

Lock Structure Activity	DB2 V7	DB2 V8
Total Requests (/ sec)	8,418.33	13,541.67
Deferred Requests (/ sec)	790.00	6.92
Contention (/ sec)	790.00	6.89
False Contention (/ sec)	115.00	2.58
CF Utilization (%)	5.4	6.9

This table is rather interesting. It shows many more requests to the lock structure as a result of using Protocol Level 2. This increase is in synchronous requests, as the number of deferred or asynchronous requests have decreased.

Under DB2 V7, the IX L-locks are mapped to X-XES locks and therefore would occupy a lock table entry in the lock structure. Once an XES essentially “owns” a lock table entry, by owning the X-XES lock, that XES becomes the global lock manager for the lock table entry and has the responsibility to resolve any contention that exists, (false, XES or real contention), for that entry.

Once potential contention has been identified by two lock requests hashing to the same lock table entry, the global lock manager XES must resolve the contention. The other XESs in the group are instructed to send the lock information they have that pertains to the lock table entry to the global lock manager XES so it can determine whether the contention is real or false. Resolution is possible as each XES holds information about locks that its IRLM has passed to it. Some optimization is in place so that once an XES “owns” a lock table entry, other XESs in the group do not have to interact with the lock structure in the coupling facility.

Under DB2 V8 the IX L-locks are mapped to S-XES locks and do not occupy a lock table entry. The end result is there will be fewer lock table entries “owned” by X-XES locks. The good side is that this reduces XES contention (IX-L locks are now mapped to S-XES locks which are compatible) and reduce false contention (S-XES locks do not “own” lock table entries, so fewer lock table entries are occupied). The down side is that more lock requests must be propagated to the coupling facility to determine if any contention may exist. The majority of these requests are synchronous requests as contention does not exist (the hashed lock table entry is not in use) and the request does not need to be suspended to

resolve any contention. This extra activity to the lock structure also increases the coupling facility CPU utilization.

However these tests show that the increase in synchronous requests to the lock structure and the resulting increase in coupling facility CPU utilization do not appear to be excessive for our workload. Nothing comes for free!

Now we have a look at the impact on the CPU consumed by DB2. Table 8-10 shows extracts from DB2 PE statistics reports for the same test.

Table 8-10 Protocol Level 2 - DB2 PE Statistics Report extract

	DB2 V7		DB2 V8		Delta (V8 / V7)
IRLM (msec / commit)	0.330	0.317	0.012	0.015	-96%
Total DB2 CPU time (msec / commit)	3.447	3.708	3.109	3.357	-10%
Group ITR	823.56		897.00		+9%

For our IRWW workload, we see a dramatic reduction in IRLM SRB time. This is because there is less XES contention to resolve. Remember that XES must defer to the IRLMs to resolve any XES contention and this activity drives IRLM CPU.

Our workload also achieved a significant increase in transaction throughput. This is a result of the workload suffering fewer class 3 suspensions, waiting for XES contention to be resolved.

In the past, a number of customers have opted to BIND their high use plans and packages using RELEASE(DEALLOCATE) rather than RELEASE(COMMIT). RELEASE(DEALLOCATE) has a number of performance advantages. DB2 does not have to release and re-acquire table space level locks after a commit. Some customers have also used RELEASE(DEALLOCATE) to reduce the amount of XES contention in their data sharing environments. However there is a cost in availability. It may be more difficult to execute DDL as the DB2 objects are allocated longer to the plans and packages.

So, let us have a look at the impact the new locking protocol has on the BIND RELEASE parameter. Table 8-11 compares the impact on performance of RELEASE(COMMIT) compared to RELEASE(DEALLOCATE) in both DB2 V7 and DB2 V8.

Table 8-11 Protocol Level 2 - RELEASE(DEALLOCATE) vs. RELEASE(COMMIT)

	DB2 V7	DB2 V8
Transaction CPU time	-18%	-7%
Global ITR	+18%	+5%

We can see that for DB2 V7, RELEASE(DEALLOCATE) has on average an 18% reduction in CPU time per transaction, compared to RELEASE(COMMIT). There is less reduction, 7%, in DB2 V8 when the new locking protocol is use. RELEASE(DEALLOCATE) still uses less CPU time, but the delta is much smaller with V8 and protocol 2.

The table also shows an 18% improvement in transaction throughput by using RELEASE(DEALLOCATE) in DB2 V7. This gain is reduced to only a 5% improvement in transaction throughput when the new locking protocol is used.

8.2.2 Conclusion

The purpose of the locking Protocol Level 2 enhancement is to avoid the cost of global contention processing whenever possible. DB2 no longer needs to wait for global lock contention processing to determine that a new parent IX or IS L-lock is compatible with existing parent IX or IS L-locks which is reasonably common in a data sharing environment.

For our workload, our testing has shown that the new locking protocol dramatically reduces both XES and false contention. This results in shorter lock and global contention suspension times. We also observed less IRLM SRB time with an improved overall transaction throughput. However, we see more synchronous XES requests to the coupling facility and more lock structure traffic with slightly higher coupling facility utilization. These increases do not appear to be excessive and should not impact overall performance.

The new locking protocol also reduces the performance gap between `RELEASE(COMMIT)` and `RELEASE(DEALLOCATE)` for data sharing workloads.

Another consequence of this enhancement is that, since child L-lock propagation is no longer dependent upon the parent L-lock, parent L-locks will no longer be held in retained state following a system failure. This means, for example, that an IX L-lock will no longer be held as a retained X-lock after a system failure. This can provide an important availability benefit in a data sharing environment. Because there is no longer a retained X-lock on the page set most of the data in the page set remains available to applications running on other members. Only the pages (assuming page locking is used) with a retained X-lock will be unavailable.

With the change to the `LOCKPART` parameter behavior, you may see additional locks acquired on individual partitions even though `LOCKPART NO` is specified.

When DB2 decides it must propagate its locks to the coupling facility for a given page set, DB2 must collect and propagate all the locks it currently owns for that page set to the coupling facility. This can cause some overhead, particularly when a page set is not used often enough for lock propagation to occur all the time. Page set P-locks are long duration locks and tend to be more static than L-locks, so the chances are higher that lock propagation will continue for longer.

8.2.3 Recommendations

The new mapping in the coupling facility lock structure to support the Protocol 2 Level locking, takes effect after the restart of the first member, after successful quiesce of all members in the DB2 data sharing group. To enable this feature, a group-wide outage is required after you have entered NFM. This obviously has some impact on availability.

However, we recommend you plan to schedule a group-wide restart of DB2 soon after you enter NFM to enable the new locking protocol as soon as possible.

If you currently are in the practice of starting table spaces in RO to avoid locking contention in a data sharing environment, you may wish to revise this strategy.

In data sharing, the recommendation for `RELEASE(DEALLOCATE)` (and thread reuse) to reduce XES messaging for L-locks is no longer required. This is good news because using `RELEASE(DEALLOCATE)`:

- ▶ Can cause increased EDM pool consumption, because plans and packages stay allocated longer in the EDM pool.
- ▶ May also cause availability concerns due to parent L-locks being held for longer. This can potentially prevent DDL from running, or cause applications using the `LOCK TABLE` statement and some utilities to fail.

There is also no need to use `RELEASE(DEALLOCATE)` if you currently use this `BIND` parameter to reduce XES contention.

8.3 Change to `IMMEDWRITE` default `BIND` option

Prior to V8, any remaining changed pages in a data sharing environment are written to the group buffer pool during phase 2 of commit, unless otherwise specified by the `IMMEDWRITE BIND` parameter. This enhancement changes the default processing to write changed pages during commit phase 1. DB2 no longer writes changed pages to the coupling facility during phase 2 of commit processing.

An implication of this change to commit processing is that DB2 V8 now charges all writes of changed pages to the allied TCB, not the MSTR address space.

Consider the situation where one transaction updates DB2 data using `INSERT`, `UPDATE`, or `DELETE`, and then, before completing phase 2 of commit, spawns a second transaction that is executed on another DB2 member and is dependent on the updates that were made by the first transaction. This type of relationship is referred to as “ordered dependency” between transactions. In this case, the second transaction may not see the updates made by the first transaction.

DB2 V5 introduced a new `BIND` parameter to overcome this problem. `IMMEDWRITE(YES)` allows the user to specify that DB2 should immediately write updated GBP dependent buffers to the coupling facility instead of waiting until commit or rollback. This option solves the application issue at the expense of performance since it forces each individual changed page immediately to the CF, ahead of commit, with each immediate write implying also a write to the DB2 log, for as many changes are made to same page.

DB2 V6 extended this functionality. `IMMEDWRITE(PH1)` allows the user to specify that for a given plan or package updated group buffer pool dependent pages should be written to the coupling facility at or before phase 1 of commit. This option does not cause a performance penalty. If the transaction subsequently rolls back, the pages will be updated again during the rollback process and will be written again to the coupling facility at the end of abort.

In prior versions of DB2, changed pages in a data sharing environment are normally written during phase 2 of the commit process, unless otherwise specified by the `IMMEDWRITE BIND` parameter, or `IMMEDWRI DSNZPARM` parameter.

DB2 V8 changes the default processing to write changed pages during phase 1 of commit processing. The options you can specify for the `IMMEDWRITE BIND` parameter remain unchanged. However, whether you specify “NO” or “PH1”, the behavior will be identical, changed pages are written during phase 1 of commit processing.

In DB2 V8, changed pages are only written at or before phase 1, never at phase 2 of the commit processing.

8.3.1 Performance

The book *DB2 for z/OS and OS/390 Version 7 Performance Topics*, SG24-6129, documents a study of the impact of writing pages to the group buffer pool during phase 2 of commit, `IMMEDWRITE(NO)`, compared to writing pages to the group buffer pool during phase 1 of commit, `IMMEDWRITE(PH1)`. The testing revealed the Internal Throughput Rates are very close to each other; meaning writing pages to the group buffer pool during phase 1 of commit has little or no performance impact.

The impact of IMMEDIATE YES remains unchanged. Changed pages are written to the group buffer pool as soon as the buffer updates are complete, before committing.

With IMMEDIATE NO (or PH1) and YES options, the CPU cost of writing the changed pages to the group buffer pool is charged to the allied TCB. Prior to DB2 V8, this was true only for PH1 and YES options. For the NO option, the CPU cost of writing the changed pages to the group buffer pool was charged to the MSTR SRB, since the pages were written as part of phase 2 commit processing under the MSTR SRB.

DB2 V8 now charges all writes of changed pages to the allied TCB, not the MSTR address space. This enhancement provides a more accurate accounting for all DB2 workloads. DB2 is now able to charge more work back to the user who initiated the work in the first place.

8.3.2 Conclusion

The impact of changing the default IMMEDIATE BIND option means that with V8 DB2 no longer writes any page to the group buffer pool during phase 2 of commit processing. This change has very little impact on Internal Throughput Rates of a transaction workload and therefore has little or no performance impact.

However, you may see some increase in CPU used by each allied address space, as the cost of writing any remaining changed pages to the group buffer pool has shifted from the MSTR address space to the allied address space.

8.3.3 Recommendations

The IMMEDIATE enhancements are immediately available when you migrate to DB2 V8. You do not have to wait until new-function mode.

Customers who use the allied TCB time for end-user charge back may see additional CPU cost with this change.

8.4 DB2 I/O CI limit removed

Prior to DB2 V8, DB2 did not schedule a single I/O for a page set with CIs that span a range of more than 180 CIs. This restriction was in place to avoid long I/O response times.

With advances in DASD technology, especially Parallel Access Volume (PAV) support, this restriction can now be lifted.

DB2 V8 removes the CIs per I/O limit for list prefetch and castout I/O requests. The limit still remains for deferred write requests, to avoid any potential performance degradation during transaction processing. Recall that a page is not available for update while it is being written to DASD. So, we do not want an online transaction to have to wait for a potentially longer deferred write I/O request to complete.

8.4.1 Performance

A number of performance studies have been done to understand the benefit of removing the CI limit for DB2 I/O requests. For these studies, the IRWW workload was used in both a data sharing environment and a non-data sharing environment.

Table 8-12 shows extracts from DB2 PE statistics reports for a non-data sharing environment running the IRWW workload.

Table 8-12 DB2 I/O CI Limit Removal - Non-data sharing

Activity	With I/O CI Limit	I/O CI Limit Removed
List Prefetch Requests	57,889	58,165
List Prefetch Reads	317,000	56,122
List Prefetch Pages Read / List Prefetch Reads	4.76	26.85
Pages Written per Write I/O	1.84	1.83

The values listed verify that the limit on CIs per DB2 I/O has been removed. The number of List Prefetch Reads has reduced dramatically, with a corresponding increase in the number of pages read per List Prefetch request.

Table 8-13 shows extracts from DB2 PE statistics reports for a data sharing environment with two members running the same IRWW workload.

Table 8-13 DB2 I/O CI limit removal - Data sharing

Activity	With I/O CI Limit		I/O CI Limit Removed	
List Prefetch Requests	42,876	42,783	43,186	43,187
List Prefetch Reads	209,600	208,200	40,894	40,711
List Prefetch Pages Read / List Prefetch Reads	3.58	3.56	18.44	18.39
Pages Written per Write I/O	6.20	4.37	27.11	26.49

You can see the positive impact of removing the DB2 CI I/O limits. The number of List Prefetch Reads has reduced, with a corresponding increase in the number of pages read per List Prefetch request.

Note also that the number of pages written per write I/O has also increased significantly. This is unique to data sharing and shows the extra benefits of removing this limitation in a data sharing environment. The DB2 CI I/O limits is also removed for castout processing in data sharing environments. The number of pages written per write I/O increased significantly because more pages can now be written in castout. We would also see a corresponding decrease in Unlock Castout requests as reported in the Group Buffer pool Activity section of a DB2 PE Statistics report.

Now we have a look at the impact on CPU. Table 8-14 shows CPU times from DB2 PE statistics reports for a non-data sharing environment, running the IRWW workload.

Table 8-14 DB2 I/O CI limit removal - Non-data sharing CPU

	With I/O CI Limit	I/O CI Limit Removed	Delta (Removed / With)
DBM1 (msec/commit)	0.342	0.322	-6%
Total DB2 CPU time (msec/commit)	2.281	2.238	-2%
ITR (commits/sec)	592.47	601.42	+2%

The values listed show a significant reduction in overall CPU used by the DBM1 address space, (-6%), with a modest increase in transaction throughput, (+2%). This is a result of DB2 performing fewer list prefetch requests for the given workload, as more pages are eligible to be retrieved by each prefetch I/O request.

Table 8-15 summarizes CPU times from DB2 PE statistics reports for a data sharing environment, running the same IRWW workload.

Table 8-15 DB2 I/O CI limit removal - Data sharing CPU

	With I/O CI Limit		I/O CI Limit Removed		Delta (Removed / With)
DBM1 (msec / commit)	0.518	0.590	0.456	0.457	-18%

This table combines the effect of both the Locking Protocol changes as well as the removal of the DB2 CI I/O limits on the CPU consumed by the DBM1 address space. Refer to 8.2, "Locking protocol level 2" on page 325, for a discussion on data sharing locking protocol changes in DB2 V8.

From Table 8-10 on page 330 we can see the locking protocol changes can account for about a 10% improvement in CPU used by the DBM1 address space. So, by removing the DB2 CI I/O limits in DB2 V8 you can see another modest reduction in CPU used by the DBM1 address space. This is due to fewer list prefetch requests and more efficient castout processing.

8.4.2 Conclusion

The removal of the DB2 CI I/O limits for list prefetch requests and castout I/O has the potential to significantly reduce the CPU used by the DBM1 address space.

Further benefits can be realized in a data sharing environment through more efficient castout processing. In the past castout processing generally only was able to castout a few pages on each request as pages to be castout can be quite random. With this enhancement castout processing can process more pages for a table space with each request and not be restricted to only pages that are within the 180 CIs within the table space.

The extent of CPU gains from this enhancement will vary significantly, depending on application processing, access paths used, data organization and placement. Applications which perform a lot of list prefetch with a large distance between pages will benefit the most from this enhancement.

In data sharing, applications which generate a lot of castout processing for not contiguous pages, should also benefit from this enhancement. OLTP performance can definitely experience a positive performance impact from 180 CI limit removal. One customer reported a significant CPU spike every 15 seconds caused by excessive unlock castout and delete namelist requests. This problem was eliminated after the 180 CI limit removal in castout.

8.4.3 Recommendations

This enhancement is transparent and is available in CM as soon as the PTFs for APARs PQ86071 and PQ89919 have been applied.

8.5 Miscellaneous items

In this section we describe some miscellaneous enhancements that impact DB2 data sharing.

8.5.1 Impact on coupling facility

You do not need to resize the group buffer pools, lock structure or SCA in the coupling facility as a result of moving to DB2 V8, even though the virtual buffer pools have now moved above the bar and have a longer memory address. This is true provided of course, you have not made any significant changes when you migrate to DB2 V8.

However, you may wish to review your coupling facility structure sizes after you have migrated to DB2 V8. For example, you may like to revise your virtual buffer pool sizes now that there are no hiperpools or buffer pools in data spaces.

We do recommend you review your coupling facility structure sizes as you prepare to implement CFLEVEL 12. This is required to take advantage of the CF Request Batch performance enhancements, described earlier. When migrating CFLEVELs, coupling facility structure sizes might need to be increased to support the new function. For example, when you upgrade from CFLEVEL 9 to CFLEVEL 11 or from CFLEVEL 10 to CFLEVEL 12, the required size of the structures might increase by up to 768 KB.

The following "Rule of Thumb" formulas may be used to estimate the required coupling facility structure sizes, based on migrating from CFLEVEL 10 or 11 to CFLEVEL 12. The thumb rules are only rough approximations.

CFLEVEL 12 structure size = CF level 10 (or 11) structure size PLUS the following based on type of structure and entry:element ratio:

Lock structure:

Without record data: 0

With record data: 20% of level 10 or 11

List or cache structure:

No data (no elements): 0

Entry:element ratio = 1:1: 2MB + 10%

Entry:element ratio >= 100:1(scale % factor for ratios in between): 2MB + 50%
and should only be used if methods 2 and 3 are not applicable.

For example, given a DB2 group buffer pool structure of 100 Mb, and an entry:element ratio of 5:1 and you are at CFLEVEL 10, then CFLEVEL 12 will require an extra:

$$100 + 2 + 12\% \times 100 = 114 \text{ MB}$$

If the higher CFLEVEL is accessible in your configuration and is defined in the CFRM policy, and OW54685 is applied, then structure rebuilds from lower CFLEVELs to higher CFLEVELs will resize the structure automatically based on its current structure object counts.

Alternatively, we suggest you use the CFSIZER to check your coupling facility structure sizes. The CFSIZER can be found at:

<http://www.ibm.com/servers/eserver/zseries/cfsizer/>

8.5.2 Improved LPL recovery

Prior to DB2 V8, when you issue the -START DATABASE command to recover LPL pages, the command must drain the entire table space or partition. The "drain" means that the command must wait until all current users of the table space or partition reach their next commit point.

All users requesting new access to the table space or partition are also suspended and must wait until the recovery completes (or until the user times out). Therefore, the drain operation can be very disruptive to other work that is running in the system, especially in the case where only one or a few pages are in LPL.

In DB2 V8, the locking and serialization schemes in the `-START DATABASE` command have changed when doing the LPL recovery. In prior versions of DB2, the `-START DATABASE` command acquires a DRAIN ALL lock on the table space or partition when doing the LPL recovery. DB2 V8 makes a WRITE CLAIM on the table space or partition. By acquiring a WRITE CLAIM instead of a DRAIN ALL, the “good” pages can still be accessed by SQL while the `-START DATABASE` is recovering the LPL pages. A new “LPL recovery” lock type is also introduced to enforce that only one LPL recovery process is running at a time for a given table space or partition.

This less disruptive locking strategy potentially enhances both the performance and availability of your applications, as more data can potentially be available to the application while the pages in the LPL are being recovered.

DB2 V8 also automatically attempts to recover pages that are added to the LPL at the time they are added to the LPL, if DB2 determines that automatic recovery has a reasonable chance of success. Automatic LPL recovery is not initiated by DB2 in the following situations:

- ▶ DASD I/O error
- ▶ During DB2 restart or end_restart time
- ▶ GBP structure failure
- ▶ GBP 100% loss of connectivity

Automatic LPL recovery improves the availability of your applications, as the pages in the LPL can be recovered sooner. In many cases you do not have to issue the `-START DATABASE` commands yourself to recover LPL pages.

In addition, DB2 V8 provides more detailed information in message DSNB250E why a page has been added to the LPL. The different reason types are:

- ▶ DASD: DB2 encountered a DASD I/O error when trying to read or write pages on DASD.
- ▶ LOGAPPLY: DB2 cannot apply log records to the pages.
- ▶ GBP: DB2 cannot successfully read or write the pages from or to the group buffer pool due to link or structure failure, GBP in rebuild, or GBP was disconnected.
- ▶ LOCK: DB2 cannot get the required page latch or page P-lock on the pages.
- ▶ CASTOUT: The DB2 Castout processor cannot successfully cast out the pages.
- ▶ MASSDEL: DB2 encountered an error in the mass delete processor during the phase 2 of commit.

These extra diagnostics help you to quickly identify and hopefully resolve why pages are being placed into the LPL thereby increasing the availability of your applications.

8.5.3 Restart Light enhancements

DB2 V7 introduced the concept of DB2 Restart Light which is intended to remove retained locks with minimal disruption in the event of an MVS system failure. When a DB2 member is started in restart light mode (`LIGHT(YES)`), DB2 comes up with a small storage footprint, executes forward and backward restart log recovery, removes the retained locks, and then self-terminates without accepting any new work.

DB2 V8 improves Restart Light. If indoubt units of recovery (UR) exist at the end of restart recovery, DB2 now remains running so that the indoubt URs can be resolved. After all the indoubt URs have been resolved, the DB2 member that is running in LIGHT(YES) mode will shut down and can be restarted normally.

A DB2 member that has been started with LIGHT(YES) that remains active to resolve indoubt URs will still not accept any new connection requests, except those that originate from connection names that have indoubt URs.

If DDF is normally active, Restart Light will also start DDF to facilitate the resolution of any distributed indoubt URs. However, no new DDF connections are allowed. Only resynch requests will be allowed from DB2 clients.

This enhancement to Restart Light has the potential to benefit both the availability and performance of applications that are still running on other active DB2 members.

For example, you do not have to bring DB2 up a second time to resolve any indoubt threads and remove any retained locks held by those indoubt threads. (Remember that under V7, DB2 would terminate immediately after a successful restart, when it is started in light mode.) Potentially more retained locks are released faster, making the data protected by those retained locks available sooner for applications that are running on other active DB2 members.

8.5.4 Change to close processing for pseudo close

In data sharing, any table space or index which has remained in a read-only state with no activity for a period of time longer than the pseudo-close interval (PCLOSET DSNZPARM) will be physically closed. This is done in an attempt to reduce data sharing overhead (the object can become non-GBP-dependent if it is closed on all but one member), but it has the disadvantage of requiring that the next access to the object must go through dynamic allocation and data set open again. This can be a significant amount of overhead, particularly if a large number of objects are accessed at the same time after a period of inactivity (for example, at the beginning of the business day).

DB2 V8 now honors the CLOSE YES or NO attribute of the table space or index. The physical close will now only take place for CLOSE YES objects, while CLOSE NO objects may remain open indefinitely. This allows you to control the behavior on an object-level basis; formerly, the only means of control was through the pseudo-close DSNZPARMs, and lengthening the pseudo-close interval to keep data sets open could have other undesirable effects.

There is no performance impact in a data sharing environment when you specify CLOSE YES. However CLOSE NO does have an impact. No really means no now, in a data sharing environment. If you want things to behave as they did prior to DB2 V8, (in a data sharing environment), you need to alter your CLOSE NO objects to CLOSE YES.

Consider CLOSE NO for objects that you estimate to be GBP-dependent most of the time and you do not want the extra open/close overhead associated with CLOSE YES. For objects that are infrequently GBP-dependent, CLOSE YES will likely provide better performance. Once GBP-dependent, CLOSE NO objects will stay GBP-dependent until either DSMAX is hit and all CLOSE YES objects have been closed or until the CLOSE NO objects are manually closed as a result of a DB2 command or a DB2 shutdown.

This enhancement is also made available in DB2 V6 and V7 by PTF UQ74866 for APAR PQ69741.

8.5.5 Buffer pool long term page fixing

DB2 V8 allows you to fix the buffer pages once in memory and keep them fixed in real storage. This is implemented by new - ALTER BUFFERPOOL parameter called PGFIX. The ability to long term page fix buffer pool pages in memory and keep them in real storage, avoids the processing time that DB2 needs to fix and free pages each time there is an I/O.

The ability to page fix buffer pool pages in memory has a significant improvement in DB2 performance, both for data sharing environments and non-data sharing environments. Although there are still significant CPU savings to be realized in data sharing, the CPU savings are generally not as much as in non-data sharing. This is because the castout buffers are NOT long term page fixed.

The primary reason why an overall saving is not as high in data sharing is a higher fixed overhead of data sharing. A long term page fix applies to GBP read and write. It does not apply to castout work area, since cast work area is not a part of buffer pool and a long term page fix is a buffer pool option.

Refer to 4.5, “Buffer pool long term page fixing” on page 169 for a more detailed analysis of long term page fixing DB2 buffers, both in data sharing and non-data sharing environments.

8.5.6 Batched index page splits

In previous versions of DB2, index page splits for indexes with inter-DB2 read/write interest, required multiple synchronous log writes and multiple separate writes to the group buffer pool. This was to ensure that when a leaf page split occurs, the index pages involved are written out in the correct order, to prevent another member from seeing a reference to an index page that does not yet exist in the group buffer pool.

This caused significant overhead for GBP-dependent indexes, resulting in additional synchronous group buffer pool writes, and even more important, in high wait times for synchronous log I/O.

With this enhancement, DB2 accumulates the index page split updates and processes them as a single entity, thereby reducing log writes and coupling facility traffic. The writes to the coupling facility of the accumulated updates use the new z/OS CF Batching commands described earlier.

However even without CFCC level 12 and z/OS 1.4, (the prerequisites for CF Batching), you can still take advantage of some performance improvements through this enhancement. In this case the coupling facility writes are not batched, but the number of forced writes to the log are reduced to just one or two.

Batching the updates for index page splits will boost performance by reducing the cost of page set group buffer pool dependency, but even more by reducing the log I/O wait time, is a more efficient mechanism to process index page splits. Applications which have heavy insertion activity will see the most benefit in data sharing.

This enhancement does not impact non-data sharing environments or data sharing environments where the index is not group buffer pool dependent. In these cases, DB2 does not have to externalize the updated index pages in any particular sequence to be seen correctly by any other DB2 members.

In both V7 and V8, we see 2 or 3 forced log writes now.

Archived

Installation and migration

During the installation and/or migration to DB2 V8, the system programmer should be aware of the way some installation parameters impact performance. Often the system parameters are chosen based on defaults and are carried along, but sometimes they change across versions and might need adjusting.

In this chapter we review several factors that have an impact on the performance of the installation of DB2 V8. If you want a more comprehensive and detailed description of the installation and migration processes, refer to the documentation listed in “*Related publications*” on page 425, specifically *DB2 UDB for z/OS Version 8 Installation Guide*, GC18-7418, and *DB2 UDB for z/OS Version 8 Data Sharing Planning and Administration Guide*, SC18-7417.

In this chapter, we discuss the following topics:

- ▶ Unicode catalog
- ▶ IVP sample programs
- ▶ Installation
- ▶ Migration
- ▶ Catalog consistency query DSNTESQ

9.1 Unicode catalog

DB2 for z/OS supports three types of encoding schemes:

- ▶ EBCDIC
- ▶ ASCII
- ▶ UNICODE

Traditionally IBM mainframes have been based on EBCDIC, while Unix and Windows applications are based on ASCII. Beginning with Windows NT®, everything stored in Windows was stored in Unicode (UTF-16). To handle larger character sets, which are absolutely necessary in Asian languages, double byte characters and mixtures of single byte and double byte characters were necessary.

Each national language developed its own code pages in EBCDIC or ASCII. The character set for a language is identified by a numeric CCSID. An encoding scheme consists of a single byte character set (SBCS), and optionally a double byte character set (DBCS) along with a mixed character set. For example, the EBCDIC CCSID used by the z/OS operating system itself is 37, but Japanese can use 8482 for SBCS, 16684 for DBCS and 1390 for mixed.

Terminology: Translation is what we do going from language to another, while conversion is what we do to character strings. CCSIDs (and code pages) are never converted, they are just definitions. The individual character strings are converted.

9.1.1 Character conversion

The variety of CCSIDs made it very difficult for multinational corporations to combine data from different sources. Unicode has arisen to solve the problem. By storing data in a single Unicode CCSID, text data from different languages in different countries can be easily managed. However, to make the transition to Unicode can be difficult and a lot of data conversion is unavoidable. When storing into and retrieving data from DB2, DB2 will convert data where necessary. Obviously, it is preferable that no conversion be carried out, because conversion impacts performance. However, considerable work has been done to improve conversion performance on zSeries. See 4.7, “Unicode” on page 174.

Character conversion is necessary whenever there is a mismatch between the CCSID of a source and target string, such as between a host variable and its associated column. Such conversion support in DB2 has existed since DB2 began to support client/server connections. In DB2 V2.3 such translations started out between different EBCDIC CCSIDs, as well as between EBCDIC and ASCII. To perform such character conversion (not involving Unicode), DB2 uses a translate table which is stored in SYSIBM.SYSSTRINGS. We will refer to such conversions as SYSSTRINGS conversions, which have particular performance characteristics.

9.1.2 Unicode parsing

In DB2 V8, because the catalog has changed to Unicode, data being sent or received by the application must be verified and possibly converted by the DBM1 address space.

Figure 9-1 depicts a legacy COBOL application running in z/OS using CCSID 37. The database has been converted to Unicode. The table contains one CHAR column called COLC (CCSID 1208) and one GRAPHIC column called COLG (CCSID 1200). When the application inserts host variables HV1 and HV2 into the table, the strings are converted by the DBM1 address space into Unicode. The CPU time for these conversions is added to class 2 CPU time.

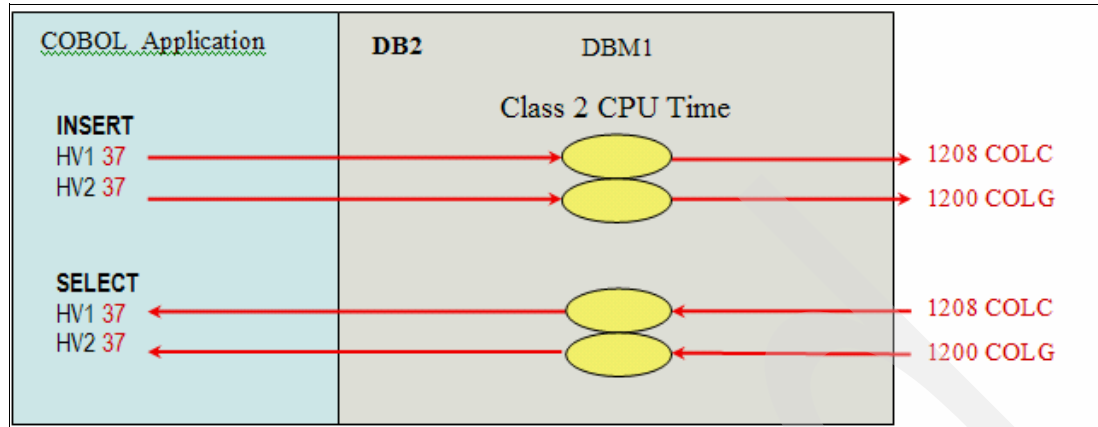


Figure 9-1 Unicode conversion example

DB2 V8 brings significant enhancements in the Unicode area lifting restrictions that were in DB2 V7. These enhancements include Unicode parsing and the conversion of the DB2 Catalog to Unicode. DB2 V8 formalizes a two stage migration process that used to be followed by customers implicitly, but which is now enforced by DB2.

When you first start DB2 V8, you are running in CM where the DB2 catalog is still in EBCDIC and you are prevented from exploiting new function. However, all SQL parsing in DB2 V8 is done in Unicode, that is, UTF-8, even in compatibility mode. SQL statements that are input in EBCDIC, ASCII or UTF-16 must be converted to UTF-8. In compatibility mode, metadata that is derived (in other words, parsed) from the SQL statement must be converted to EBCDIC in order to compare them to DB2 catalog data.

When you migrate the catalog to new-function mode (NFM), the DB2 catalog is converted to Unicode, and fallback to V7 is no longer possible. New DB2 V8 installations always run in NFM.

SQL statement conversion is unaffected by the switch to NFM, but metadata derived from SQL is no longer converted since the metadata is already in the CCSID of the DB2 catalog.

On the other hand, Unicode conversion is introduced for column names specified within an SQLDA in EBCDIC or ASCII. DB2 V8 also introduces the capability to join two tables of different CCSIDs. Needless to say, Unicode conversion is necessary to do this and the extra CPU time is likely to be substantial.

DB2 V8 major and minor conversion

Conversion of SQL statements and metadata generally involves English alphanumeric characters. To improve performance, DB2 V8 developed a faster way to convert such characters without calling the z/OS Conversion Services. This fast method is called *minor conversion*, while *major conversion* refers to those conversions performed by the z/OS Conversion Services. DB2 does this by maintaining a translate table associated with the EBCDIC SBCS/single bytes of the mixed CCSIDs that are specified during DB2 installation. The same translate table is used for accesses to the DB2 catalog and user data. Minor conversion is supported for all European EBCDIC character sets, and some Asian EBCDIC character sets. Minor conversion does not apply to UTF-16 (GRAPHIC) or ASCII; it may only apply to conversions between EBCDIC and UTF-8.

In a multinational corporate environment, we may see applications originating from different national languages accessing a common database. A Unicode database allows such a multinational corporation to store all of its data in a single database. Since we cannot assume

that all applications are converted to Unicode, Unicode conversion is supported for every language that is used to access the common database. However, only one nation (code character set) is specified during DB2 installation. This nation's character set is used as the default character set when a database is stored as EBCDIC or ASCII, and is also the default EBCDIC application encoding scheme. This nation will be referred to as the *host nation*. Minor conversion is only supported for the host nation's EBCDIC encoding scheme. All other nations will use major conversion.

How does minor conversion work? The zSeries instruction set has always included a TR instruction that translates one byte for one byte, based on a 256-byte conversion table. A TRT instruction has also existed which could be used to test a string for certain one-byte characters. As long as a single byte Latin alphanumeric character can be represented in UTF-8 by a single byte, a TR instruction will suffice. DB2 V8 has built-in translate tables for most common single byte CCSIDs and if the TRT instruction is “successful”, DB2 V8 can translate the string using a TR instruction. Such translations are called “minor conversions”. The presence of shift-in/shift-out characters in a mixed string will always cause the TRT instruction to fail. If the TRT fails, then DB2 must invoke the z/OS Unicode conversion service. Such conversions in DB2 are called “major conversions”. Whereas minor conversions cannot change the length of the string (in terms of bytes), major conversions may change the length, running the risk of possible truncation or padding when using fixed length columns or host variables.

The performance of minor conversion is much better than the performance of major conversion, hence, minor conversion is one of the significant advantages of DB2 V8 compared to V7, and represents an advantage of UTF-8 over UTF-16. Even when DB2 V8 must resort to major conversion, V8 is still much faster than V7.

9.1.3 DB2 catalog changes

The DB2 catalog continues to grow with every release of DB2. In addition to the new catalog objects required to support the new functions in DB2 (tables, columns, and so forth), V8 introduces a number of other significant changes to the catalog:

- ▶ Changing the definition of many existing columns to support long names
- ▶ Converting the character columns in the catalog from EBCDIC to Unicode

Refer to the *DB2 Release Planning Guide*, SC18-7425, and Appendix A of the *DB2 SQL Reference*, SC18-7426, for a complete list of changes to the DB2 catalog.

Table 9-1 shows how the DB2 catalog has grown across its Versions.

Table 9-1 DB2 catalog growth

DB2 Version	Table spaces	Tables	Indexes	Columns	Table check constraints
V1	11	25	27	269	N/A
V3	11	43	44	584	N/A
V4	11	46	54	628	0
V5	12	54	62	731	46
V6	15	65	93	987	59
V7	20	82	119	1206	105
V8	22	84	132	1265	105

Summary of DB2 Catalog changes

The most important changes are:

► Introduction of long names

Long names (varchar 128) are supported for most of the objects. Catalogs are defined using Unicode, so if you have existing queries on the catalog, you will receive the results in a different ordering.

► New tables

- SYSIBM.IPLIST: Multiple IP addresses to one LOCATION can be defined for data sharing group.
- SYSIBM.SYSSEQUENCEAUTH: Created in DSNDB06.SYSSEQ2. It records the privileges that are held by users over sequences.
- SYSIBM.SYSOBDS: Created in DSNDB06.SYSALTER, it contains the version of a table space or an index space that is still needed for recovery.

► New table spaces:

- SYSEBCDC: EBCDIC table space, not used until NFM
- SYSIBM.SYSDUMMY1: has been moved from SYSSTR to SYSEBCDC
- SYSALTER: New catalog table SYSIBM.SYSOBDS
To store prior version information of ALTERed tables
- SYSIBM.IPLIST: Created in DSNDB06.SYSDDF
Allows multiple IP addresses to be specified for a given LOCATION

► Column changes

- Existing binary columns will be ALTERed to be FOR BIT DATA columns to ensure they are handled properly during ENFM processing
- Approximately 60 columns added to existing tables
- Approximately 15 changes to values of existing columns
- Approximately 45 column definitions have been changed
- Approximately 10 changes to RI and table check constraints
- Column length changes to support long names are only done during ENFM processing

In summary, 17 catalog and 1 directory table space have been converted to Unicode, two tables have been dropped (SYSLINKS - No index data sets to be deleted, SYSPROCEDURES - VSAM data sets for Index DSNKCX01 can be deleted after ENFM), the table SYSIBM.SYSDUMMY1 has been moved from DSNDB06.SYSSTR to DSNDB06.SYSEDCBC, many columns have changed to VARCHAR to support long names, system-defined catalog indexes changed to NOT PADDED and 7 catalog and 1 directory table space moved out of BP0.

► Default changes

The initial defaults for the - ALTER BUFFERPOOL command have changed:

- DWQT: Specifies the buffer pool's deferred write threshold as a default percentage of the total virtual buffer pool size. The initial default is decreased from 50% to 30%.
- VDWQT: Specifies the virtual deferred write threshold for the virtual buffer pool. This parameter accepts two arguments. The first argument is a percentage of the total virtual buffer pool size. The default is decreased from 10% to 5%.

The initial defaults for the ALTER GROUPBUFFERPOOL command have changed:

- CLASST: Specifies the threshold at which class castout is started. It is expressed as a percentage of the number of data entries. The default is decreased from 10% to 5%.
- GBPOOLT: Specifies the threshold at which the data in the group buffer pool is cast out to DASD. It is expressed as a percentage of the number of data entries in the group buffer pool. The default is decreased from 50% to 30%.
- GBPCHKPT: Specifies the time interval in minutes between group buffer pool checkpoints. The default is lowered from 8 minutes to 4 minutes.

9.2 IVP sample programs

DB2 continues to enhance the IVP samples in order to demonstrate the usage of new functions, to provide more features and enhance the usability of the samples themselves. In this section we are interested in the performance of multi-row fetch from the programs DSNTPE2, DSNTPE4 and DSNTIAUL. This function can be activated only in NFM. These programs are often used for testing during the migration to a new DB2 release.

Performance measurements

In this section we describe how the tests were made, what kinds of scenarios were used, how we get the default, and how much improvement each program gets when using multi-row FETCH.

Native measurements environment

- ▶ DB2 for z/OS V8 and V7 (non-data sharing)
- ▶ z/OS Release 1.3.0
- ▶ IBM z900 Series 2064 2-way processor
- ▶ ESS 800 DASD with FICON channels

Measurement scenarios

- ▶ Non-partitioned table space, 10,000 row table, 26 columns, 1 index
- ▶ Dynamic SQL with table space scan
- ▶ Measured on both V7 and V8
- ▶ Retrieve 5 columns, 10k rows
- ▶ Retrieve 20 columns, 10k rows

DSNTEP2 and DSNTEP4

DSNTEP2 is a PLI program shipped with DB2 to demonstrate the support for dynamic SQL. The DSNTEP2 has been enhanced for:

- ▶ GET DIAGNOSTICS
DSNTEP2 now uses GET DIAGNOSTICS to retrieve error information.
- ▶ Large SQL statement:
The sample program DSNTEP2 can now handle SQL statements larger than 32 KB in size.
- ▶ Greater than 18 character table/column names:
The sample program DSNTEP2 has been modified to handle the longer table and column names.
- ▶ New MAXERRORS value:
A new MAXERRORS parameter has been added in DSNTEP2. It allows you to dynamically set the number of errors that DSNTEP2 will tolerate. In previous versions of DB2, DSNTEP2 stopped processing after it encountered 10 SQL errors. The

MAXERRORS value can be modified during run time by coding the following functional comment in the SQL statements:

```
SET MAXERRORS
```

► **SYSPRINT blocking in DSNTEP2:**

A change to SYSPRINT blocking in DSNTEP2 will speed up the rate in which DSNTEP2 outputs its results. The blocking size was very small before, thus impacting the performance when processing large result sets. DSNSTEP2 can now also use the system default or user assigned JCL block size.

A new sample program, DSNTEP4, has been added. It is like DSNTEP2, but it uses multi-row FETCH.

In Figure 9-2 we show measurements of the new DSNTEP4 using different multi-row settings, and we compare them to DSNTEP2 (not using multi-row).

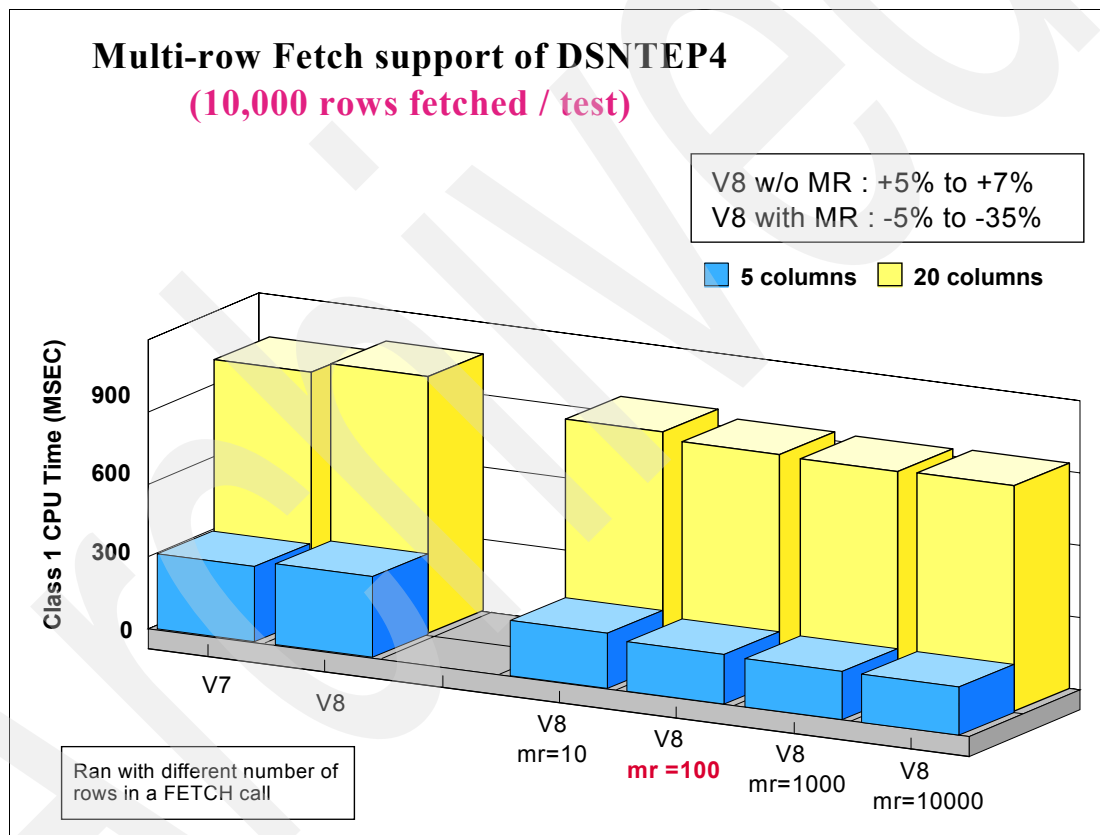


Figure 9-2 Multi-row fetch support of DSNTEP4

As mentioned, DB2 V8 ships DSNTEP2 and DSNTEP4. DSNTEP4 uses multi-row FETCH. You can specify a new parameter, MULT_FETCH, to specify the number of rows that are to be fetched at one time from the result table. The default fetch amount for DSNTEP4 is 100 rows, but you can specify from 1 to 32,676 rows. It can be coded as a functional comment statement as follows:

```
//SYSIN DD *
--#SET MULT_FETCH 5895
SELECT * FROM DSN8810.EMP;
```

In Figure 9-3 we show a comparison of DSNTEP2 with DSNTEP4.

From these measurements we observe that DSNTPE2 under V8 uses 3% more CPU than DSNTPE2 under V7. DSNTPE4, by exploiting multi-row fetch, uses 10 to 30% less CPU depending on the number of rows specified.

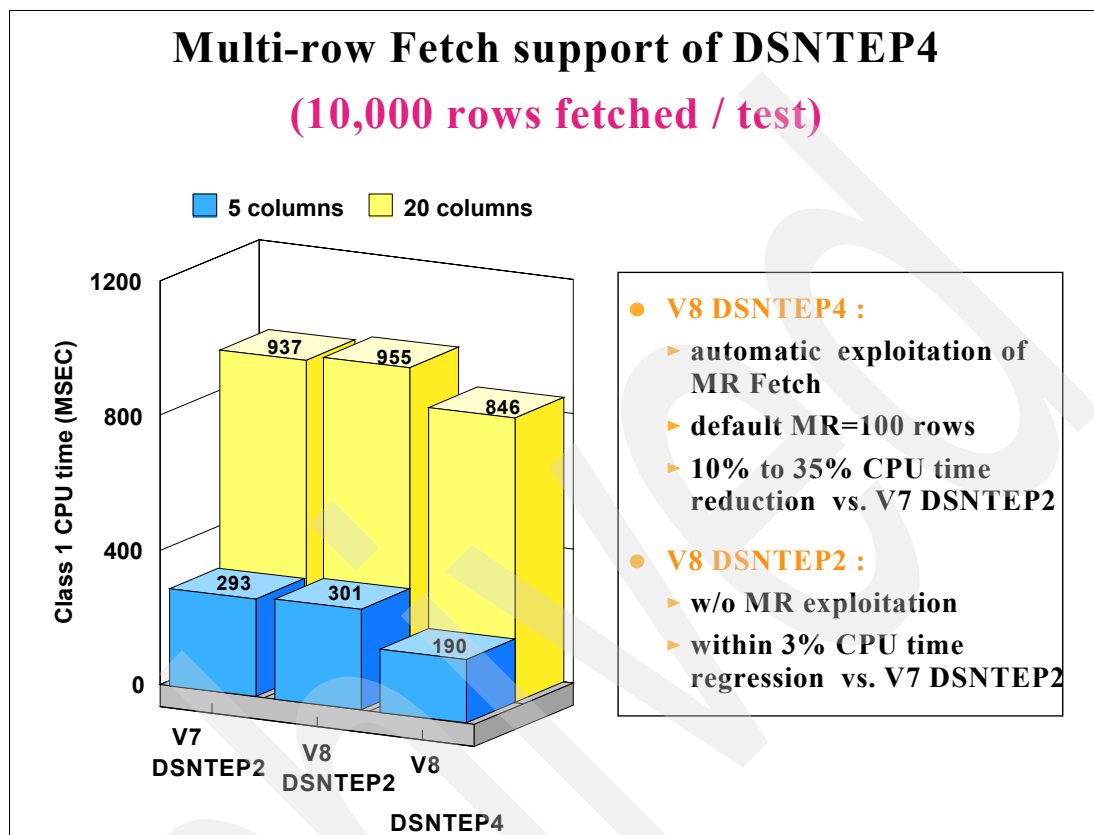


Figure 9-3 Multi-row fetch support of DSNTPE4

DSNTIAUL

DSNTIAUL is a sample assembler unload program. With DB2 V8, it has been enhanced to:

- ▶ Handle SQL statements up to 2 MB in size
- ▶ Use multi-row FETCH

You can specify an additional invocation parameter called “number of rows per fetch”. It indicates the number of rows per fetch that DSNTIAUL is to retrieve. You can specify a number from 1 to 32,767.

In Figure 9-4 we show the impact of using multi-row fetch in V8, vs. single row fetch in V7, for 5 and for 20 columns. In this test, on average, the DSNTIAUL improvement is 40%.

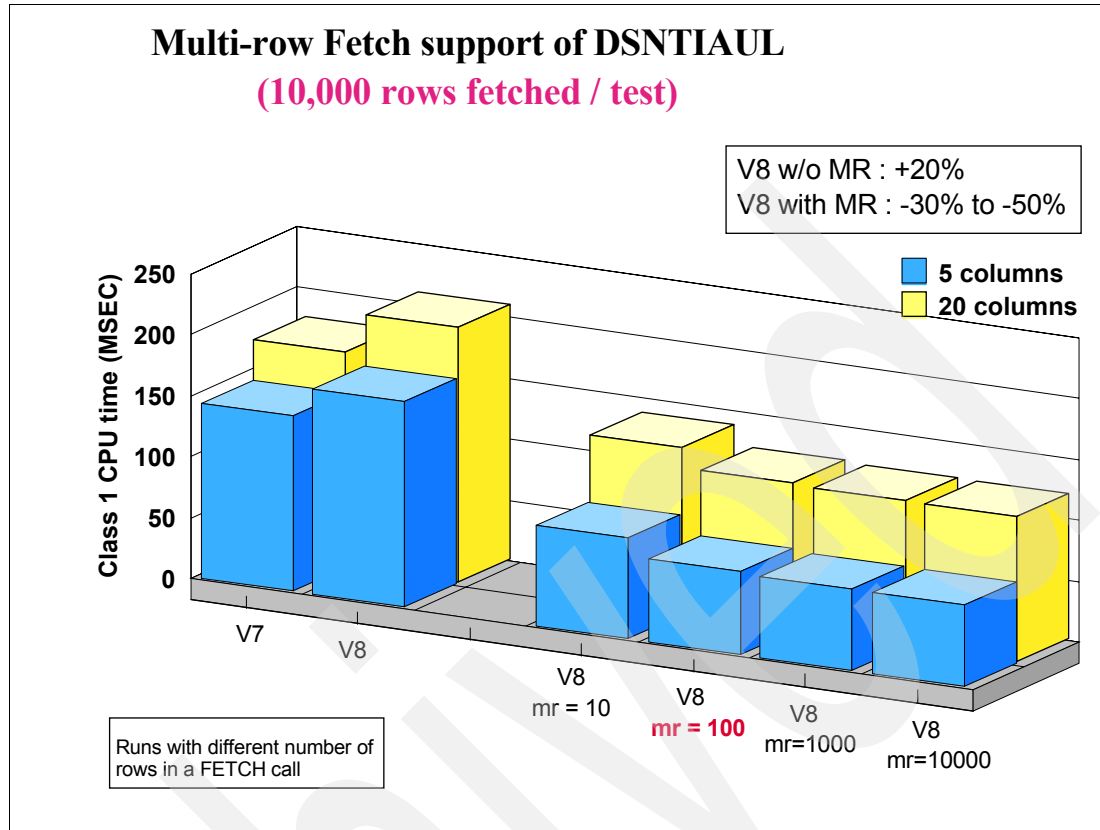


Figure 9-4 Multi-row fetch support of DSNTIAUL

As shown on Figure 9-4, the default value for DSNTIAUL retrieves 100 rows per fetch. When you retrieve 1000 or 10000 rows, the CPU time stays almost the same.

If you want to change the number of rows, the parameter can be specified together with the SQL parameter, as shown in Example 9-1.

Example 9-1 DSNTIAUL with multi-row parameter

```
//SYSTSIN DD *
DSN SYSTEM(DSN)
RUN PROGRAM(DSNTIAUL) PLAN(DSNTIB81) PARMS('SQL,250') -
LIB('DSN810.RUNLIB.LOAD')
...
//SYSIN DD *
LOCK TABLE DSN8810.PROJ IN SHARE MODE;
SELECT * FROM DSN8810.PROJ;
```

In Figure 9-5, we compare the performance of DSNTIAUL running under V7 and V8 in terms of CPU. Elapsed times are very similar.

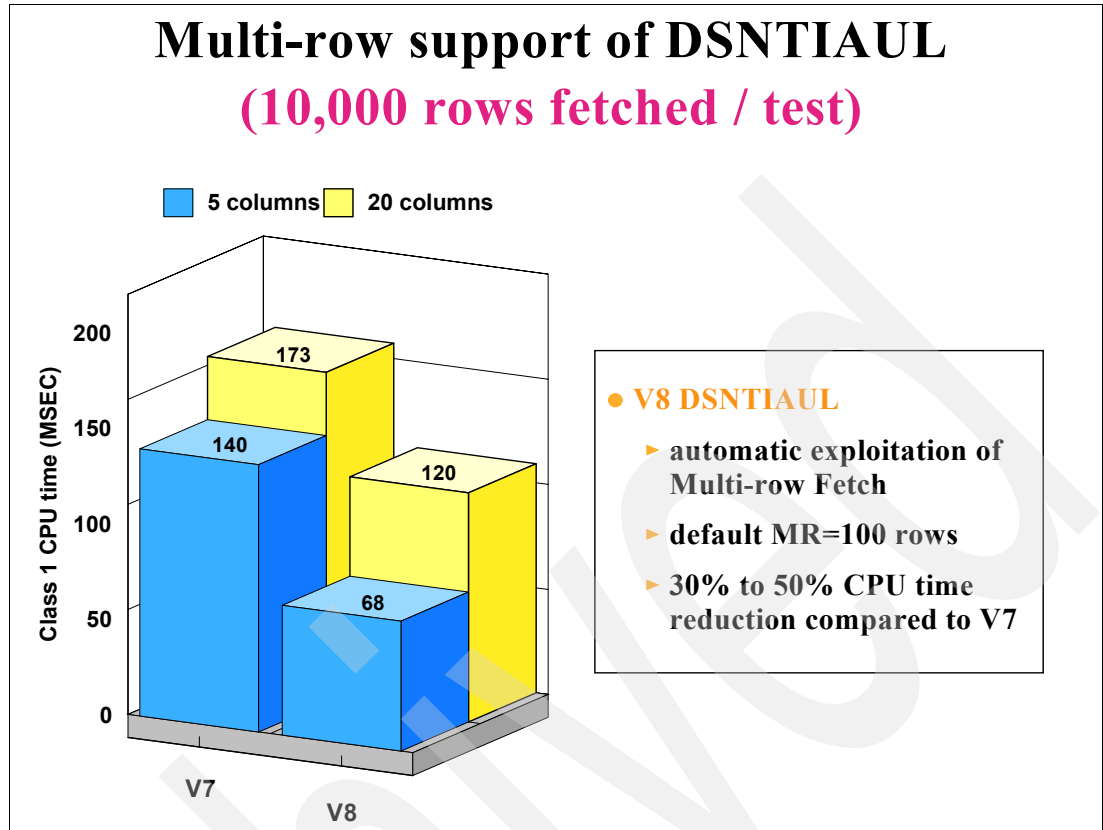


Figure 9-5 Multi-row support of DSNTIAUL comparing V7 with V8

9.3 Installation

DB2 V8 brings major changes to the installation and migration processes. With a newly installed DB2 V8 subsystem, you can immediately take advantage of all the new functions in V8. For more information about the changes, refer to the *DB2 UDB for z/OS Version 8 Installation Guide*, GC18-7418, and the *DB2 UDB for z/OS Version 8 Data Sharing Planning and Administration Guide*, SC18-7417.

The key changes to the installation and migration processes are:

- ▶ Valid CCSIDs must be defined for ASCII, EBCDIC, and Unicode.
- ▶ You must supply your own tailored DSNHDECP module. You can no longer start DB2 with the DB2-supplied DSNHDECP.
- ▶ DSNHMCID is a new data-only module in V8. It is required by DB2 facilities that run in DB2 allied address spaces, such as attaches and utilities.
- ▶ Buffer pools of sizes 4 KB, 8 KB, 16 KB, and 32 KB must be defined.
- ▶ Only WLM-established stored procedures can be defined.

Installation CLIST changes

In this section we summarize the changes to installation CLISTs.

- ▶ Installation panels changes

Table 9-2 shows the most important installation panels changes. For a description of all of the installation panels and their contents, refer to the *DB2 UDB for z/OS Version 8 Installation Guide*, SC18-7418.

Table 9-2 *Clist changes in the installation panels*

Panel Id	Panel parameter	DB2 V7 default	DB2 V8 default
DSNTIP7	User LOB value storage	2048 (kb)	10240 (kb)
DSNTPIE	Max users	70	200
DSNTIPE	Max remote active	64	200
DSNTIPE	Max remote connected	64	10000
DSNTIPE	Max TSO connect	40	50
DSNTIPE	Max batch connect	20	50
DSNTIPF	Describe for static	NO	YES
DSNTIPN	SMF statistics	YES (1,3,4,5)	YES (1,3,4,5,6)
DSNTIP8	Cache dynamic SQL	NO	YES
DSNTIPP	Plan auth cache	1024 bytes	3072 bytes
DSNTIPP	Package auth cache	NO	YES
DSNTIPP	Routine auth cache	32 KB	100 KB
DSNTIPL	Log apply storage	0	100 MB
DSNTIPL	Checkpoint frequency	50 000 records	500 000 records
DSNTIPA	Block size (archive log)	28672	24576
DSNTIPR	DDF threads	ACTIVE	INACTIVE
DSBTIPR	IDLE thread time-out	0	120
DSNTIPR	Extended security	NO	YES
DSBTIP5	TCP/IP KEEPALIVE	ENABLE	120 (seconds)
DSNTPIC	Max open data sets	3000	10000
DSNTIPC	EDMPOOL Storage Size	7312 KB	32768 KB
DSNTIPC	EDM Statement Cache	n/a	102400 KB
DSNTIPC	EDM DBD cache	n/a	102400 KB

The DB2 V8 Installation CLIST generates the job DSNTIJTC, which must also be run. This job executes the CATMAINT utility. In previous versions of DB2 you only ran the CATMAINT utility during migration, in order to update the DB2 catalog for the new version. With DB2 V8, job DSNTIJTC must be run also for new installations in order to update the DB2 catalog table spaces which are not moving to Unicode, keeping the default EBCDIC encoding schemes from DSNHDECP.

The CATMAINT execution in this case will be very quick. For more information, refer to *DB2 UDB for z/OS Version 8 Installation Guide*, GC18-7418.

- Cache dynamic SQL now enabled by default

This option specifies whether to cache prepared dynamic SQL statements for later use by eligible application processes. These prepared statements are cached in the EDM

dynamic statement cache. If activated, consider this usage when calculating your EDM pool size. See “Calculating the EDM pool space for the prepared-statement cache” of the *DB2 UDB for z/OS Version 8 Installation Guide*, GC18-7418, for details about estimating storage. If you specify YES, you must specify YES for USE PROTECTION on panel DSNTIPP.

- Fast log apply enabled by default

LOG APPLY STORAGE is an option on panel DSNTIPBA. The acceptable values are: 0 to 100 MB. The macro is DSN6SYSP LOGAPSTG.

The value in this field represents the maximum ssnmDBM1 storage that can be used by the fast log apply process. The default value is 0 MB, which means that the fast log apply process is disabled except during DB2 restart. During DB2 restart, the fast log apply process is always enabled and a value of 100 MB is assumed.

The value you specify, if not equal to zero, which disables the function, is used for RECOVER utility executions. Since RECOVER jobs are likely to run in parallel, specify 100 MB so you can also use 10 MB of log apply storage for each concurrent RECOVER job (up to 10) that you want to have faster log apply processing.

FLA is also utilized by the new RESTORE utility. In this case the default assumed by DB2 will be 500 MB, unless disabled by the value zero in LOG APPLY STORAGE.

- The data set size panel DSNTIP7 has four new fields which change the way DB2 manages VSAM data sets:

- Vary DS control interval (2 fields were changed)
- Optimize extent sizing
- Table space allocation and index space allocation

- The Performance and Optimization panel, DSNTIP8, provides the defaults for two new special registers which have been created to support Materialized Query Tables:

- CURRENT REFRESH AGE

Specifies the default value to be used for the CURRENT REFRESH AGE special register when no value is explicitly set using the SQL statement SET CURRENT REFRESH AGE. The values can be 0 or ANY. The default of 0 disables query rewrite using materialized query tables.

- CURRENT MAINT TYPES

Specifies the default value to be used for the CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION special register when no value is explicitly set using the SQL statement SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION.

Acceptable values are NONE, SYSTEM, USER, and ALL. The default (SYSTEM) allows query rewrite using system-maintained materialized query tables when the CURRENT REFRESH AGE is set to ANY. Alternatively, specifying USER allows query rewrite using user-maintained materialized query tables when CURRENT REFRESH AGE is set to ANY, and specifying ALL means query rewrite using both system-maintained and user-maintained materialized query tables.

For materialized query tables refer to 3.2, “Materialized query table” on page 39.

There are several other changes including:

- Checkpoint frequency increased from 50,000 to 500,000 log records
- Archive log block size reduced from 28672 to 24576 (for better DASD occupancy)
- Remove hipool definitions and VPTYPE settings. They are no longer supported in V8
- DDF panels have new terminology

- Use “Inactive DBAT” instead of “Type 1 Inactive Thread”
- Use “Inactive Connection” instead of “Type 2 Inactive Thread”
- ▶ CCSIDs must be specified for EBCDIC, ASCII and Unicode even when you do not use those encoding schemas in your application
- ▶ The DSNHDECP module must be defined by the user. It is not defined by DB2 by default.
- ▶ Buffer pools BP8K0, BP16K0, and BP32K must be defined in advance since some catalog table spaces use 8 KB, 16 KB and 32 KB pages.
- ▶ For data sharing groups: GBP8K0, GBP16K0 and GBP32K also need to be allocated.
- ▶ The location name must be specified, even if DDF is not used. See PTF UQ90701 for APAR PQ91009.

9.4 Migration

Migration is the process of converting an existing DB2 subsystem, user data, and catalog data to a new version. This process has changed from DB2 V7 to V8 in order to minimize the possible impact of regression and fallback incompatibilities. For more information refer to *DB2 UDB for z/OS Version 8 Installation Guide*, GC18-7418.

The migration to DB2 V8 is only allowed from V7. As mentioned, a DB2 V8 subsystem will migrate going through three different modes:

- ▶ Compatibility mode (CM)
- ▶ Enabling-new-function mode (ENFM)
- ▶ New-function mode (NFM)

We are interested in the performance of the catalog migration, how long the migration will take, and how big the catalog will be. In this section we are using general information from several real migrations to illustrate the catalog migration. After reading this section you will be able to determine how long it is going to take to migrate your catalog and how large it will be after the migration.

During the migration process, you must go through CM and ENFM, before reaching the NFM. When in CM, your Catalog and Directory have already successfully been converted to V8, new columns have been added to existing tables, new indexes and table spaces have been created and so on.

However, there are two other major changes to the DB2 catalog necessary to make most of the new functions provided with DB2 V8 possible. These changes are:

- ▶ Change existing catalog columns so that they can store long names.
- ▶ Convert the catalog to Unicode.

In CM, these two changes have not been done. Furthermore, the only new functions available for use in this mode of operation are those which allow fallback to V7.

In ENFM, you are in the process of converting your Catalog table spaces so that after completion of the conversion, they now accept long names and exist in the Unicode encoding scheme. Catalog conversion is performed by running job DSNTIJNE. You can spread conversion of your catalog table across several maintenance intervals. Use job DSNTIJNH to direct DSNTIJNE to halt after conversion of the current table space has completed. To resume, simply rerun DSNTIJNE from the top. It will automatically locate the next table space to be converted. The process of conversion is not disruptive, that is, you can continue operation while being in ENFM. Once you have migrated all table spaces of your DB2 catalog to Unicode, your DB2 subsystem will operate in NFM once you run the DSNTIJNF to indicate

that everything is ready. This step can be helpful for instance in making sure that all your members of a data sharing group, or all subsystems DRDA related, have been migrated. Now all new functions which have been introduced with DB2 V8 are available.

9.4.1 Migration plan recommendations

It is important to have an effective operational test environment and application regression workloads. The test environment should be a meaningful representation and closely mirror production characteristics, so that regression and stress tests would flush out any implementation issues. Unless several environments are available, it is not effective to run for an extended period with different code and function levels across “test” and “production” environments. Application changes, quick fixes and DB2 APARs (HIPERs, PEs, corrective) have to be tested and promoted to production. Observe a clear separation between *toleration* and *exploitation* of features in the new release.

The exploitation phase should follow after a toleration phase where the new release has been running stable in production for a reasonable period (typically at least to complete one processing month cycle) and there is little chance of a need to fallback to the previous version of DB2. The exploitation project will focus on a prioritized list of new features offering business value. During the toleration period you will run with release N-1 of the DB2 Precompiler to prevent new SQL features being used or you run the V8 Precompiler, specifying (or implying) NEWFUN = NO.

Keep in mind the distinction between coexistence, referred to data sharing, and compatibility mode. The period of data sharing coexistence should be kept reasonably short to reduce the possibility of running into problems which are unique to coexistence.

Note: To see the DB2 mode you just need to execute a - DIS GROUP and look at the

MODE (C*** BEGIN DISPLAY OF GROUP(.....) GROUP LEVEL(...)

MODE (E)

MODE(N))

C is compatibility mode, E is enabling-new-function mode, N is new-function mode.

Recommendations

This is a set of general recommendations:

1. Before the catalog migration to CM it is recommended that you clean up your catalog. Run MODIFY on all table spaces to clean up the SYSCOPY, SYSLGRNX entries. Also FREE the old versions of packages that are not used any more. You can also REORG the catalog in V7 to compact it.
2. Migrate a V7 *test* data sharing group to V8 CM and hold. If you want to roll in the new release member by member, migrate the first two members, test, then roll through the remaining members quickly.
3. After (2) has been proven for an extended period, migrate the *production* data sharing group to V8 compatibility mode. Migrate the first two members, wait, then roll through the remaining members quickly. We recommend a reorg on the catalog while in CM and before going to NFM. Online Reorg of the Catalog is available in CM.
4. Run through a complete monthly processing cycle in production to flush out implementation or egression issues (can fallback or go forward again if necessary).
5. After (4) is successful, start to migrate the V8 test data sharing group to V8 NFM.

6. After (5) is successful, start to migrate V8 *production* data sharing group to V8 NFM.

For your development system, this is another set of recommendations:

1. Migrate the development system to CM and run that way until you are satisfied with V8 on your development system,
2. Run DSNTIJNE on your development system, but not DSNTIJNF. This will convert your development system catalog to support NFM, but will leave your development system in ENFM, thus not allowing any application developer to use any new function that cannot be deployed on your production system since it is still in CM.
3. Migrate the corresponding production system to CM and run that way until you are satisfied with V8 on your production system.
4. Run DSNTIJNE on your production system to prepare it for NFM.
5. Run DSNTIJNF on both your production and development systems at roughly the same time.

Note that steps 2 and 3 can be in the opposite order.

9.4.2 Performance measurements

Before we start showing the measurements, we take a look at a table that shows us what kind of data is going to be migrated. Note that we did a clean up in our catalog before the migration. If you have never done it, run a MODIFY and a REORG of your catalog after migrating to the CM and before going to ENFM.

DB2 V7 catalog

In Table 9-3 you can see the most important tables and the number of rows from a DB2 V7 catalog.

Table 9-3 Row counts for catalog tables - 698 MB catalog in V7

Table	Number of rows	Table	Number of rows	Table	Number of rows
SYSCOPY	19032	LUMODES	0	SYSSTMT	225846
SYSCOLUMNS	78948	LUNAMES	1	SYSCOLDIST	12235
SYSFIELDS	0	MODESELECT	0	SYSCOLDISTSTATS	2822
SYSFOREIGNKEYS	130	USERNAMES	0	SYSCOLSTATS	64534
SYSINDEXES	6019	SYSRESAUTH	56	SYSINDEXSTATS	1397
SYSINDEXPART	7382	SYSSTOGROUP	35	SYSTABSTATS	1397
SYSKEYS	27097	SYSVOLUMES	35	SYSSTRINGS	752
SYSRELS	65	SYSPACKAGE	8713	SYSCHECKS	107
SYSSYNONYMS	32	SYSPACKAUTH	4412	SYSCHECKDEP	109
SYSTABAUTH	49018	SYSPACKDEP	93944	SYSUSERAUTH	1375
SYSTABLEPART	5368	SYSPROCEDURES	0	SYSVIEWDEP	173
SYSTABLES	4217	SYSPACKLIST	232	SYSVIEWS	131
SYSTABLESPACE	4005	SYSPACKSTMT	273762		

Table	Number of rows	Table	Number of rows	Table	Number of rows
SYSDATABASE	750	SYSPLSYSTEM	14		
SYSDBAUTH	756	SYSDBRM	6519		
IPNAMES	0	SYSPLAN	5515		
LOCATIONS	0	SYSPLANAUTH	6969		
LULIST	0	SYSPLANDEP	43949		

CATMAINT performance measurements show very similar CPU and elapsed times between data sharing and non-data sharing environments. This is true when migrating from V5 to V6, V6 to V7, V5 to V7 and V7 to V8 CM.

DB2 V8 catalog

The information related to the performance migration to the DB2 V8 catalog is listed in Table 9-4. We show the performance of the utility Catmaint, and also job DSNTIJNE, Reorg of the catalog. We also describe how much larger they can grow. The statistics refer to three different case studies from customer scenarios.

Table 9-4 CATMAINT statistics in non-data sharing

	Company 1 (28.3 GB)	Company 2 (15.2 GB)	Company 3 (698 MB)	
V 7 to V 8 CM CATMAINT UPDATE	255 sec.	67 sec.	6 sec.	CPU
	692 sec.	255 sec.	27 sec.	Elapsed time
V8 CM to V8 NFM DSNTIJNE	55 min. 5 sec.	20 min. 53 sec.	2 min. 6 sec.	CPU
	2 hrs 17 min. 40 sec	1 hr 28 min. 48 sec	8 min. 3 sec.	Elapsed time
DB2 Catalog Increase V8 NFM compared with V7	0.46%	1.03%	8%	Increase in table size
	2.29%	8.21%	9%	Increase in index size

The migration from V8 CM to V8 NFM does not use CATMAINT. When you migrate V7 to V8 CM, the size of the catalog does not change.

Note: The size of the catalog does not increase when you migrate from V7 to V8 CM. The size of the catalog just increases marginally when you are migrating from V8 CM to V8 NFM.

Dominant element in the catalog size

Just to illustrate the size considerations with one example, we refer to a real life customer scenario of a bank with a production data sharing group holding 19 members. Their DB2 V7 catalog size is 3 MB and the directory is 7.3 MB. In this case the directory is 142% bigger than the catalog with the largest VSAM file from the directory being SPT01 with 3 MB (the same size of the catalog) because of the huge number of packages. This is a good case for

spring cleaning of the directory: Before migrating it would be very appropriate to free all the packages that are not being used, reduce SPT01, and migrate faster.

SPT01 is used to store skeleton package tables (SKPTs). These tables store the access paths to DB2 data. When a package is executed, DB2 uses this information to access the data it needs. Because a single SKPT can be longer than the maximum record length that can be used in DB2, SKPTs are stored as a sequence of SKPT sections. The skeleton package table parent record (SPTR) contains as much of the SKPT section as the record can fit. The entire SKPT section is stored in this record if it fits; if it does not fit, it is stored in one or more SPTRs. Each SPTR is identified by a unique section and sequence number.

How long will it take to migrate?

Based on the measurements, you can estimate the migration time first to **CM**, and then to NFM.

When migrating to V8 CM, if your catalog is less than 30 GB, your performance can be predicted using Figure 9-6. Assuming that the catalog has no major anomalies, the most important structure to influence the duration of CATMAINT is SYSDBASE, because CATMAINT does a scan on that structure.

The elapsed time migrating to V8 is certainly longer than it was when migrating to V7 because DB2 has to prepare the data for later conversion to Unicode at ENFM time. This is the step that involves a scan of SYSDBASE. DB2 is not updating every page, as was done in the past, but it still scans the SYSDBASE.

Estimation based on this graph is only valid for a particular CPU and DASD model upon which this measurement was done.

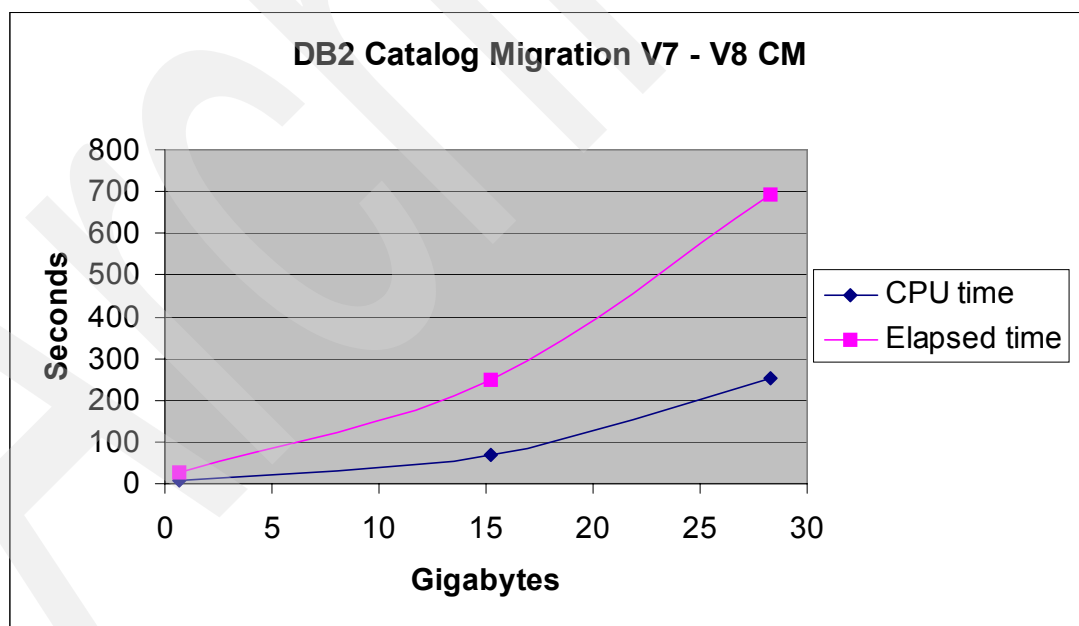


Figure 9-6 Migration times to CM

Note: The measurements in data sharing showed no difference with non-data sharing and in the case of migrating from V7 to V8 CM, data sharing was a little bit better in CPU.

In Figure 9-7 we describe the performance of the migration from V8 CM to V8 **NFM**, with that you can find out how long it will take you to migrate your catalog from CM to NFM. You just need to find the size of your catalog.

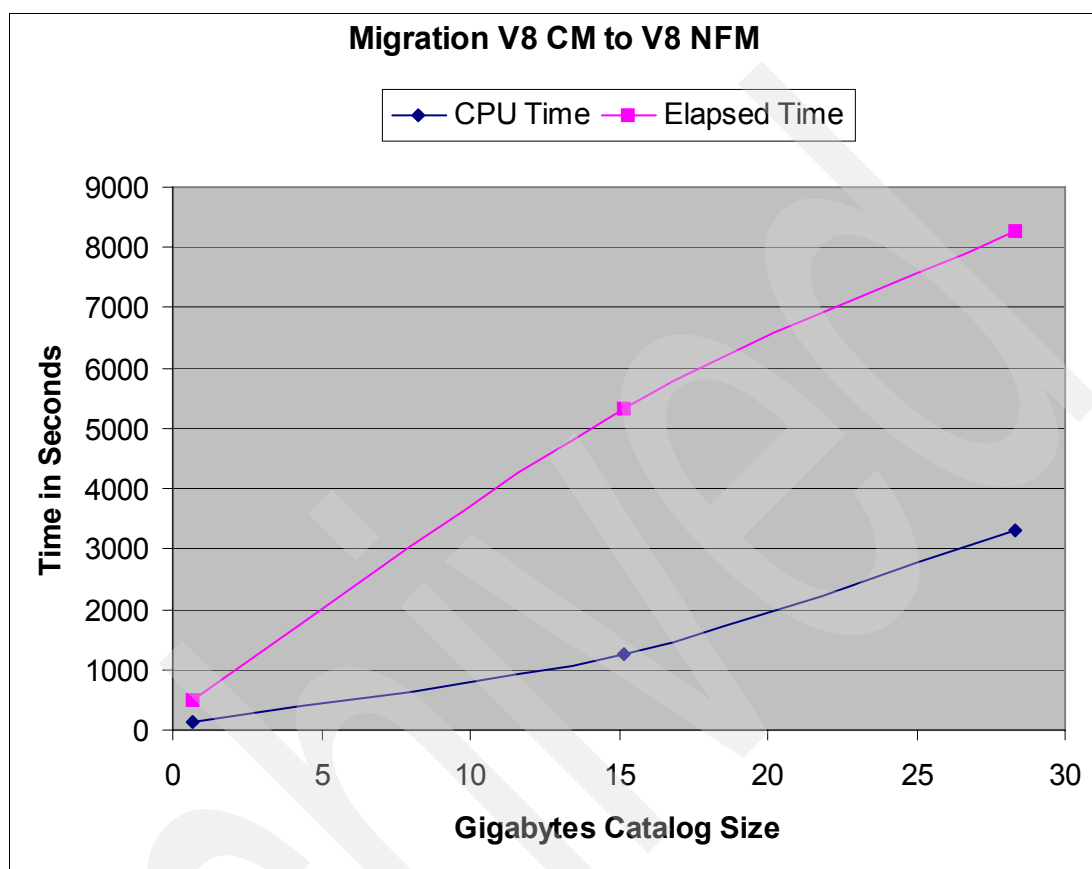


Figure 9-7 Migration from CM to NFM

Note: The major component of job DSNTIJNE is the Online Reorg of SPT01 and the 17 catalog table spaces.

Job DSNTIJNE performs several functions:

- ▶ Saves the current RBA or LRSN in the BSDS
- ▶ Changes types and lengths of existing catalog columns
- ▶ Converts catalog data to Unicode
- ▶ Changes buffer pools for several catalog table spaces
- ▶ Changes catalog indexes with varying length columns to NOT PADDED
- ▶ Changes page size of several catalog table spaces

This job can be run at any time. To stop it, you cannot use a **- TERM UTIL** command, you need to use job DSNTIJNH. When you want to resume running DSNTIJNE, you do not need to make any change, just run the job again. Just execute the steps as defined and you can stop and start it as needed. Do not take long though to migrate your catalog, you will use the new functions only after completing migration and you do not want to keep your DB2 subsystems not aligned for too long.

Summary and recommendations

Migration from V7 to V8 CM takes from 0.2 to 10 minutes depending on the size of the catalog and directory (medium to large). Before the catalog migration, it is recommended that you clean up your catalog doing a MODIFY on all table spaces: The result will be a cleanup on the SYSCOPY.SYSLGRNX entries. It is also useful to free the old version of packages that are not being used any more.

Note: When you execute a MODIFY, do not forget to take a look at the tables that will be in copy pending, because your application might not be expecting the DB2 return code.

Reorg Catalog and Directory before migrating to NFM.

Migration from V8 CM to V8 NFM takes from 0.1 to 2 hours depending on the size of the catalog and directory (medium to large). The longest step is the catalog reorg DSNTIJNE, and the execution of that job can be divided.

The size of the catalog, both data and index, has been observed to increase between 1% and 10%.

Online REORG SHRLEVEL REFERENCE of SPT01 and 17 catalog tables is the most time-consuming component. A very rough Rule-of-Thumb on estimating the time for a medium to large catalog and directory in non-data sharing is:

6 min. + 3 to 7 min. per GB of SPT01, SYSPKAGE, SYSDBASE, etc.

The time is heavily dependent on the DASD and channel model used.

Allocate larger work files for online REORG of large tables such as SYSDBASE, SYSPKAGE.

Allocate work files on separate DASDs, LCUs for online REORG of tables with parallel sort/build indexes such as SYSHIST, SYSPKAGE, SYSSTATS, and SPT01.

9.5 Catalog consistency query DSNTESQ

The DSNTESQ sample job is provided to DB2 users so that they can check the consistency of their DB2 catalogs. This sample job consists of DDL, INSERT and 65 queries to verify the integrity of the catalogs and directories. The SQL can be run from SPUFI or DSNTDP2 and contains DDL which creates copies of the catalog using segmented table spaces. In some cases, the queries in DSNTESQ run faster when run on copies of the catalog instead of the actual catalog because the copies have additional indexes.

Some of these catalog consistency queries can run for extended periods of time on subsystems with large amounts of catalog data. In order to improve the performance of most of these DSNTESQ catalog consistency queries, they were rewritten to use non correlated subqueries on table columns that are in index keys.

In the past, some queries in DSNTESQ have had very poor performance. In DB2 V6, a customer introduced a subset of queries rewritten from DSNTESQ which performed very well against IBM supplied DSNTESQ queries. In DB2 V8, some queries from DSNTESQ were rewritten for performance and retrofitted to DB2 V7. On Table 9-7 we have an example of retrofitted queries followed by other rewritten queries.

Table 9-5 Queries 5 and 18 rewritten for better performance

Query (DB2 V8 - New Function Mode)	Old (V8 NFM)	New (V8 NFM)
------------------------------------	--------------	--------------

Old - Query 5	New - Query 5	CPU (sec.)	Elapsed time (sec.)	CPU (sec.)	Elapsed time (sec.)
SELECT DBNAME, NAME FROM SYSIBM.SYSTABLESPAC E TS WHERE NTABLES <> (SELECT COUNT(*) FROM SYSIBM.SYSTABLES TB WHERE TYPE IN ('T', 'X','M') AND TB.DBNAME = TS.DBNAME AND TB.TSNAME = TS.NAME AND TS.NAME <> 'SYSDEFLT');	SELECT DBNAME, NAME FROM SYSIBM.SYSTABLESPA CE TS WHERE NTABLES" = (SELECT COUNT(*) FROM SYSIBM.SYSTABLES TB WHERE TYPE IN ('T', 'X') AND TB.DBNAME = TS.DBNAME AND TB.TSNAME = TS.NAME AND TB.DBID = TS.DBID AND TS.NAME = 'SYSDEFLT');	1926	2104	7.7506	10.366
Old - Query 18 (DB2 V8 NF Mode)	New - Query 18 (DB2 V8 NF Mode)	CPU (sec.)	Elapsed time (sec.)	CPU (sec.)	Elapsed time (sec.)
SELECT TBCREATOR, TBNAME, NAME FROM SYSIBM.SYSCOLUMNS CL WHERE FLDPROC = 'Y' AND NOT EXISTS (SELECT * FROM SYSIBM.SYSFIELDS FL ! WHERE FL.TBCREATOR = CL.TBCREATOR AND FL.TBNAME = CL.TBNAME AND FL.NAME = CL.NAME AND FL.COLNO = CL.COLNO);	SELECT CL.TBCREATOR, CL.TBNAME, CL.NAME FROM SYSIBM.SYSCOLUMN S CL LEFT OUTER JOIN SYSIBM.SYSFIELDS FL ON FL.TBCREATOR = CL.TBCREATOR AND FL.TBNAME = CL.TBNAME AND FL.NAME = CL.NAME AND FL.COLNO = CL.COLNO WHERE CL.FLDPROC = 'Y' AND FL.TBCREATOR IS NULL;	0.3745	0.4046	0.37382	0.3924

Table 9-6 shows the old and new text of Query 31.

Table 9-6 Catalog consistency - Query 31

Old - Query 31	New - Query 31
SELECT NAME FROM SYSIBM.SYSPLAN PL WHERE PLENTRIES = 0 AND NOT EXISTS (SELECT * FROM SYSIBM.SYSDBRM WHERE PLNAME = PL.NAME);	SELECT NAME FROM SYSIBM.SYSPLAN WHERE PLENTRIES = 0 AND NAME NOT IN (SELECT PLNAME FROM SYSIBM.SYSDBRM);

Table 9-7 shows the old and new text of Query 33.

Table 9-7 Catalog consistency - Query 33

Old - Query 33	New - Query 33
SELECT NAME FROM SYSIBM.SYSPLAN PL WHERE PLENTRIES = 0 AND NOT EXISTS (SELECT * FROM SYSIBM.SYSSTMT WHERE PLNAME = PL.NAME);	SELECT NAME FROM SYSIBM.SYSPLAN WHERE PLENTRIES = 0 AND NAME NOT IN (SELECT PLNAME FROM SYSIBM.SYSSTMT);

As shown in Figure 9-8, the performance of the two queries has improved noticeably.

Query	V8 Compatibility Mode				V8 New Function Mode			
	Old DSNTESQ		New DSNTESQ		Old DSNTESQ		New DSNTESQ	
	CPU	E.T	CPU	E.T	CPU	E.T	CPU	E.T
31	38 min 53 sec	10 hr 32 min 36 sec	1.2 sec	17 sec	25 min 35 sec	10 hr 2 min 39 sec	14 sec	23 sec
33	49 min 2 sec	9 hr 58 min 55 sec	3.5 sec	17.5 sec	42 min 12 sec	10 hr 3 min 37 sec	18 sec	25 sec

Figure 9-8 Catalog consistency queries - Measurement results

The old consistency Query 35 and Query 36 have merged into a new query, as shown in Table 9-8.

Table 9-8 Catalog consistency - Queries 35 and 36 from SDSNSAMP.

Old Query 35	Old Query 36	New Query 35 and 36
SELECT PLNAME, NAME FROM SYSIBM.SYSDBRMDR WHERE NOT EXISTS (SELECT * FROM SYSIBM.SYSSTMTST WHERE ST.PLNAME = DR.PLNAME AND ST.NAME = DR.NAME);	SELECT DISTINCT PLNAME, NAME FROM SYSIBM.SYSSTMTST WHERE NOT EXISTS (SELECT * FROM SYSIBM.SYSDBRMDR WHERE DR.PLNAME = ST.PLNAME AND DR.NAME = ST.NAME);	SELECT DBRM.PLNAME, DBRM.NAME, STMT.PLNAME, STMT.NAME FROM SYSIBM.SYSDBRM DBRM FULL OUTER JOIN SYSIBM.SYSSTMTSTMT ON DBRM.PLNAME = STMT.PLNAME AND DBRM.NAME = STMT.NAME WHERE (STMT.NAME IS NULL OR DBRM.NAME IS NULL) GROUP BY DBRM.PLNAME, DBRM.NAME, STMT.PLNAME, STMT.NAME;

The performance of these queries has improved quite a lot as shown in Figure 9-9.

Query	V8 Compatibility Mode				V8 New Function Mode			
	Old DSNTESQ		New DSNTESQ		Old DSNTESQ		New DSNTESQ	
	CPU	E.T	CPU	E.T	CPU	E.T	CPU	E.T
35	1 hr 1 min 39 sec	11 hr 53 min 1 sec	8 sec	24 sec	51 min 18 sec	11 hr 48 min 37 sec	9 sec	23 sec
36	1 hr 5 min 38 sec	17 hr 2 min 20 sec			30 min 25 sec	11 hr 47 min 9 sec		

Figure 9-9 Catalog consistency queries - Measurement results

With the changes to queries 31, 33, 35 and 36, the total CPU and elapsed time to execute the 65 queries of DSNTESQ are greatly improved. The new queries have been retrofitted by APAR PQ71775 respectively with PTF UQ85042 to DB2 V6 and UQ85043 to DB2 V7.

Performance tools

In this chapter, we discuss the IBM DB2 performance tools and the key DB2 V8 features they exploit.

The following IBM tools are discussed in this chapter:

- ▶ DB2 Estimator
- ▶ DB2 Performance Expert
- ▶ Tivoli OMEGAMON XE for DB2 on z/OS
- ▶ Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS
- ▶ DB2 Archive Log Accelerator
- ▶ DB2 Query Monitor
- ▶ DB2 High Performance Unload
- ▶ Data Encryption for IMS and DB2 Databases
- ▶ DB2 Data Archive Expert
- ▶ DB2 Bind Manager
- ▶ DB2 High Performance Unload

The DB2 tools may require maintenance to support DB2 V8. You may look at the following URL to see the minimum maintenance level for the DB2 tools in order to support and exploit DB2 V8:

http://www.ibm.com/support/docview.wss?rs=434&context=SSZJXP&uid=swg21162152&loc=en_US&cs=utf-8&lang=en+en

10.1 DB2 Estimator

DB2 Estimator V8 is a no charge tool provided by DB2 to help you predict the performance and cost of running DB2 applications and utilities under DB2 UDB for z/OS.

This tool can be downloaded from the URL:

<http://www.ibm.com/software/data/db2/zos/estimate/>

With DB2 Estimator you create definitions of DB2 objects, SQL statements, transactions, utilities, and system configurations. You also have the option to combine elements to form a workload which can be a representative sampling of an application. DB2 Estimator uses this information as input to provide performance estimates which you can use for tuning and also for capacity planning. You can also estimate the disk space used for tables and indexes.

DB2 Estimator calculates CPU time, I/O time, and elapsed time for utilities, SQL statements, transactions, and whole applications. The tool gives you the ability to experiment with DB2 applications and utilities on a PC without the need to use DB2. If you want to estimate DB2 V8 functions, then you need to create a new project and specify that you are using DB2 V8.

Tip: This can be done in two ways. You can create a new project or copy an existing project using “save as”. In both cases, specify you are using DB2 V8.

The DB2 Estimator formulas have been updated to reflect DB2 V8 CPU times. There is support for a limited number of new DB2 V8 features. Check the Web site for an up-to-date list.

10.2 DB2 Performance Expert

One of the premier products for performance monitoring is the DB2 Performance Expert (PE). This product has many different components providing a robust set of monitoring functions for viewing your DB2 subsystems. Here is a list of some of the most important functions DB2 PE provides as a monitoring tool:

- ▶ ISPF host online monitor
- ▶ Workstation online monitor with the capability to monitor all of your DB2 subsystems in a single view
- ▶ Exception processing which handles both event and periodic exceptions
- ▶ Batch reporting capability for all sorts of reports using DB2 trace data as input
- ▶ Performance Database which is a repository for DB2 trace data controlled by the user
- ▶ Performance Warehouse which is a repository for DB2 accounting and statistics trace data controlled by PE
- ▶ Buffer Pool Analysis to analyze your buffer pool configuration and make recommendations for improvement

For more information about this DB2 tool, see:

<http://www.ibm.com/software/data/db2imstools/>

or refer to the book, *DB2 Performance Expert for z/OS Version 2*, SG24-6867-01, for details on the tool.

The following changes to DB2 V8 have been incorporated into the batch reporting function of DB2 PE after publication of that book:

- ▶ Performance Expert PTF UQ90726 for APAR PQ88224 supports the ACCUMACC parameters in Batch Accounting reports, such that fields which are not valid for roll-up are shown as N/P or N/C (not present or not calculated). These fields are shown in the latest version of the *DB2 Performance Expert Reporting User's Guide*, SC18-7978, in Appendix D.

Note: The recent DB2 APAR PQ90547 contains some rework on the roll-up mechanism which will be reflected in a Performance Expert PTF targeted for March/April 2005.

- ▶ Performance Expert PTF UQ93207 for APAR PQ94872 supports new counters of IFCID 225 for real storage statistics and dynamic statement caching introduced recently by DB2 in PQ91101.

In addition, the PTFs provide enhanced support for V8 changes available in Record Trace, System Parameter and EXPLAIN reports.

The following new major PE functions have been delivered with PTF UQ95694 for APAR PQ95989 in December 2004:

- ▶ READS support of IFCID 225 in Workstation Online Monitor
- ▶ Enhancements in Locking Suspension Reports to accumulate additional locking data and write it into a file which can be imported into a spreadsheet for further analysis
- ▶ Enhancements in SQL Activity Reports for displaying information on host variables
- ▶ Enhancements in I/O Activity Reports supporting new ORDER functionality

Virtual storage layout

With PQ91101, DB2 introduces new counters in IFCID 225 for real storage statistics and dynamic statement caching.

DB2 Performance Expert V2.1 supports this functionality with PTF UQ93207 for APAR PQ94872. See also 4.3, "Monitoring DBM1 storage" on page 157, and Appendix B, "The DBM1 storage map" on page 393. Here is a summary of the features:

- ▶ Record Trace

The report block for IFCID 225 has been rearranged to better match the layout in the statistics reports.

- ▶ Statistics Report and Trace (long format)

The report blocks have been implemented as follows:

- DBM1 AND MVS STORAGE BELOW 2 GB

This block contains the new fields related to dynamic statement caching.

Two new fields AVERAGE THREAD FOOTPRINT (MB) and MAX NUMBER OF POSSIBLE THREADS have been added. Here are the formulas:

Average Thread Footprint: $(QW0225VR - QW0225AS) / (QW0225AT + QDSTCNAT)$

Max number of possible threads: $((QW0225RG - QW0225EL) - (200 * (1024 * 1024)) - (QW0225GM + QW0225GS + QW0225FX)) / \text{average thread footprint}$

(The block has been split up because the page size limit has been reached.)

- DBM1 STORAGE ABOVE 2 GB

The block has been extended to contain the field QW0225SJ with label STAR JOIN MEMORY POOL (MB).

- REAL AND AUXILIARY STORAGE

This is the new block containing the real storage statistics counters.

- EDM POOL

The block has been extended to contain the field QISESTMT with label STATEMENTS IN GLOBAL CACHE.

- Statistics SAVE file

The statistics SAVE file layout has been changed. The Statistics SAVE migration and convert programs shipped with the PTF must be used from now on.

- Performance Database

The new fields of IFCID 225 are supported in the Statistics General Table of the Performance Database. The member SFPESAMP(DGOSUPDB) contains ALTER statements for migration purposes.

The member SFPESAMP(DGOSQGEN) contains a query that calculates the formulas for “average thread footprint” and “max number of possible threads” within the performance database.

- Performance Warehouse

The PE Server shipped with the PTF automatically adds the new columns to the Statistics General Table of the Performance Warehouse.

- Documentation

The PE books have been republished in December 2004 and contain updates about these changes.

10.3 Tivoli OMEGAMON XE for DB2 on z/OS

IBM Tivoli OMEGAMON XE for DB2 on z/OS is the next evolution in OMEGAMON performance and availability solutions designed to help you proactively manage your DB2 mainframe environment and tune for optimal performance. Its Web interface provides a single interface at the big picture and granular levels, including interaction between DB2 and other applications. IBM Tivoli OMEGAMON XE for DB2 on z/OS helps you identify performance spikes and anomalies that might otherwise go unseen, take action in real time and automate repetitive DB2 operations.

In addition to offering leading performance and availability management for DB2, Tivoli OMEGAMON XE for DB2 integrates with other Tivoli products. You can use these integrated products to deploy true end-to-end availability management and help prevent threats to system performance before they impact service levels.

Tivoli OMEGAMON XE for DB2 helps you monitor and tune how Workload Manager allocates and balances resources to find the source of performance problems quickly and adjust resources accordingly. You can gain visibility into, and control over, enclave usage. And you can analyze transaction rate information to identify overloaded and under performing workloads. Optimize use of the coupling facility in Parallel Sysplex environments.

Because Tivoli OMEGAMON XE for DB2 gives you a single point of control over your DB2 parallel data sharing environment, you have the information necessary to detect, isolate and resolve problems. To optimize the performance of the coupling facility in the Parallel Sysplex, you can analyze coupling facility structures, participating member status and contention rates.

The ability to integrate information from Tivoli OMEGAMON XE monitors and third-party software into a single view enables you to identify and track problems across all your enterprise platforms.

10.4 Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS

As communicated with the statement of direction from December 7 Announcement 204-297 (US), IBM has converged and expanded the capabilities of DB2 Performance Expert for z/OS, V2.1 (5655-J49) and Tivoli OMEGAMON XE for DB2 on z/OS, V3.0 (5608-A67).

IBM Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS represents the effort on converging OMEGAMON XE for DB2 and DB2 Performance Expert into one product that retains the best features of each. This new tool gives you a single, comprehensive tool to help assess the efficiency of and optimize performance from your DB2 Universal Database™ in the z/OS environment. It automates the analysis of your database performance in real time and also adds expert database analysis functions to help you maximize performance and enhance productivity.

The main functions of this tool allow you to:

- ▶ Monitor, analyze, and tune the performance of IBM DB2 Universal Database and DB2 applications on z/OS
- ▶ Improve productivity with meaningful views of performance
- ▶ Quickly and easily identify performance bottlenecks using predefined rules of thumb
- ▶ Enjoy substantial breadth and depth in monitoring DB2 environments by combining batch-reporting capabilities with real-time monitoring and historical tracking functions
- ▶ Support an enterprise-wide integrated systems management strategy activated by the IBM Tivoli OMEGAMON XE family
- ▶ Store performance data and analysis tools in a performance warehouse

The software combines the sophisticated reporting, monitoring, and buffer pool analysis features of the IBM Tivoli OMEGAMON XE for DB2 Performance Monitor on z/OS and IBM DB2 Buffer Pool Analyzer products.

The book *A Deep Blue View of DB2 Performance: IBM Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS*, SG24-7224 helps you understand and install the main functions of the product, clarify the differences, and point out the advantages if you had one of the pre-existing products already in use.

10.5 DB2 Archive Log Accelerator

If a hardware failure occurs while DB2 is writing changed pages to DASD, or you have applied the wrong updates, you may need to recover the data, and recovery from the logs occurs. If necessary, the DB2 recover utility will mount archive logs to get the needed data in order to restore the corrupted tables. The DB2 archive logs can also be used in other critical recovery situations or an unplanned outage such as a DB2 subsystem restart after a failed DB2 or MVS image. The amount of time spent in such scenarios needs to be minimized to get the application back up as quickly as possible.

Another use is during back out of a long running process that issues a ROLLBACK. DB2 Archive Log Accelerator compresses the archive log as an effort to reduce the times to recover from such situations.

Preliminary measurements are listed in Table 10-1.

Table 10-1 RECOVER and Archive Log Accelerator

RECOVER from active log has ET of 335 sec.	RECOVER from archive log	With ALA	With ALA and compression
Stripes	CPU/ET (sec.)	CPU/ET (sec.)	CPU/ET (sec.)
1	183/460	177/444	176/323
2	N/A	176/439	176/330
4	N/A	176/435	176/324
non-ext ds	177/543	N/A	N/A

These measurements are for RECOVER with PTF UQ90143 for PQ88896 applied. RECOVER with ALA, striping and compression, shows an elapsed time improvement of almost 30%. Increasing the number of stripes did not improve elapsed time.

For more information about this DB2 tool, see:

<http://www.ibm.com/software/data/db2imstools/>

DB2 Archive Log Accelerator V2.1 fully exploits DB2 V8.

10.6 DB2 Query Monitor

Monitoring and tuning the SQL activity in a DB2 subsystem is important especially when problems occur that affect the response times. You may need to identify problematic SQL statements and focus on improving the response times of important transactions. DB2 Query Monitor allows you to view current and historical query activity within your DB2 subsystems and make important decisions affecting its performance.

For more information about this DB2 tool, see

<http://www.ibm.com/software/data/db2imstools/>

Here is a list of features of Query Monitor V2.1:

- ▶ Data collection and activity viewing features
- ▶ SQL statement resource consumption
- ▶ Object access statistics with granularity down to the object level
- ▶ Current active statements in DB2
- ▶ Pinpoint SQL from many different views:
 - Plan
 - Program
 - User
 - SQL statement
- ▶ Ability to exclude workloads or specific SQLCODEs
- ▶ Set exception limits and thresholds
- ▶ Define alert notification thresholds

DB2 Query Monitor V2.1 supports and exploits DB2 V8.

10.7 DB2 SQL Performance Analyzer for z/OS

DB2 SQL Performance Analyzer for z/OS V2.2 (SQL PA) allows you to analyze SQL statements without having to execute them. It provides many different types of detailed reports giving the novice developer or experienced DBAs reports of different information depending on the level of experience.

For more information about this DB2 tool, see

<http://www.ibm.com/software/data/db2imstools/>

SQL PA V2.2 has been completely redesigned and contains full support and exploits DB2 V8 in both compatibility and new-function modes.

Here is a list of new features in the tool that expand on support for DB2 V8:

- ▶ Handles DBRMs in Unicode or EBCDIC
- ▶ Supports larger SQL statements up to 2 MB
- ▶ Provides consideration of MQTs for access paths
- ▶ Supports the REFRESH statement which allows the EXPLAIN function to consider the use of MQTs when considering access path selection
- ▶ Recognizes
 - DB2 catalog and longer names
 - Columns containing data in Unicode
 - Padded/nonpadded indexes
- ▶ Includes redesigned report structure in order to handle long names
- ▶ Provides statistics collector program that supports DB2 V8 catalog
- ▶ Supports new PLAN_TABLE columns

Additionally, SQL PA reports on table and index versioning, volatile tables, MQTs, DPSIs, and other DB2 V8 enhancements.

Refer to Appendix C, “SQL PA sample reports” on page 399 for a sample of the new SQL PA report illustrating some of the DB2 V8 functionality.

10.8 Data Encryption for IMS and DB2 Databases

If you need extra protection for your company's IMS and DB2 data in order to comply with regulatory legislation, then encryption may be the thing you are looking for.

IBM Data Encryption for IMS and DB2 Databases is implemented using standard IMS and DB2 exits. The exit code invokes the zSeries and S/390® Crypto Hardware to encrypt data for storage and decrypt data for application use, thereby protecting sensitive data residing on various storage media. This tool can help you save the time and effort required to write and maintain your own encryption software for use with such exits or within your applications.

In IMS, the IBM Data Encryption tool implements encryption using the standard Segment Edit/Compression exit routine. Both IMS data and index databases can be encrypted and decrypted.

In DB2, the IBM Data Encryption tool implements encryption through the standard EDITPROC exit.

Data Encryption for IMS and DB2 Databases V1.1 supports and exploits DB2 V8.

For performance data see 4.7.1, “Performance” on page 184.

For more information about this DB2 tool, see

<http://www.ibm.com/software/data/db2imstools/>

10.9 DB2 Data Archive Expert

One of the pressing concerns for Information Technology is the increasing volumes of data; databases are growing at an exponential rate and many organizations are alarmed at the volume of data they need to keep on hand. Inactive data exists everywhere; data warehouses grow every day. Inactive data is not unusable; it just has a lower probability of access. Organizations may have governmental requirements to keep certain data and there may be a valid business need.

As databases grow in size, the percentage of inactive data also grows. Inactive data costs are high and many organizations are examining ways to archive dormant data to keep operational costs down. The Data Archive Expert tool can help you manage your archived data so that it can be quickly accessed whenever necessary.

For more information about this DB2 tool, see

<http://www.ibm.com/software/data/db2imstools/>

The DB2 Data Archive Expert for z/OS V1.1 supports and exploits DB2 V8.

10.10 DB2 Bind Manager

It might be important to know if you perform a rebind: Will the access path change? Does a plan or package need a rebind? Can we bypass a rebind because the application logic changed but not the SQL? Often, with a new release of DB2, there are changes in the access path, but it would be advantageous to know ahead of time the effect of a rebind on the access path selection by DB2. The bind process uses CPU resources and DB2 must quiesce access to the plan or package. If the bind is unnecessary, you can save much in resource consumption by avoiding a bind.

As the name implies, the DB2 Bind Manager tool manages binds in the DB2 environment and can help with all of the issues mentioned above.

For more information about this DB2 tool, see

<http://www.ibm.com/software/data/db2imstools/>

The DB2 Bind Manager Tool V2.2 supports and exploits DB2 V8.

10.11 DB2 High Performance Unload

The DB2 High Performance Unload (HPU) is a tool that allows you to unload data from your DB2 tables outside of DB2. The UNLOAD utility is provided with the DB2 Utilities Suite V7

and DB2 V8. These two products provide different unload capabilities. The major functional differences between them are outlined in Table 10-2 below.

For more information about this DB2 tool, see the following Web site:

<http://www.ibm.com/software/data/db2imstools/>

Table 10-2 DB2 HPU vs. UNLOAD utility

DB2 High Performance Unload tool	UNLOAD utility
Supports DB2 V5, V6, V7 and V8	Supports DB2 V and V8
Supports any SQL statement by passing any statements it cannot process to DB2	Supports a limited number of statements
Has 4 output formats	DB2 V7: only one output format - LOAD. DB2 V8: supports delimited format in addition to LOAD above.
Has a user exit that allows you to examine the row prior to writing to the output data set. You can also modify the row prior to writing it to the data set.	N/A
Can unload data from a dropped table. HPU requires that the dropped table be recreated in the catalog and then perform the unload from an image copy using the OBID of the original table.	If the created table has the same OBID as the original table, the UNLOAD utility does the unload. If the OBID is different, then the UNLOAD utility will not work.
HPU UNLOAD command syntax same between z/OS product and MP product.	N/A
HPU V2 supports BMC and CA UNLOAD product syntax.	N/A

Table 10-3 describes the different output formats available with the DB2 High Performance Unload tool.

Table 10-3 HPU output formats

HPU output formats	Description
REORG UNLOAD	Same format as the IBM REORG UNLOAD utility
DSNTIAUL	Output identical to DSNTIAUL format
DELIMITED	You can specify a separator and a delimiter character
VARIABLE	Treat all variable length fields as variable
USER	Completely user-defined; can customize every column differently

10.11.1 Performance measurements

Several performance measurements were made to compare the performance of different unload options. One set of tests compares the HPU V2.2 and the DB2 V8 UNLOAD utility. Another set of tests illustrate the difference between unloading data using the DB2 V8 UNLOAD utility with a load output format as opposed to using the delimited data format. The results are discussed in this section.

The utilities ran on a G7 Turbo with ESS F20 DASD using z/OS V1.5. The table space contains only one table with two indexes. The table space is a partitioned table space and so one case involves using parallelism for the UNLOAD. The table used contains 124,999 rows. The test cases are described in detail within Table 10-4.

Table 10-4 Unload test cases

Test case	Description
Case 1	Unload from a table space using an image copy data set
Case 2	Unload from an image copy data set
Case 3	UNLOAD SELECT * FROM tablename
Case 4	UNLOAD SELECT C1, C2, C3, C4, C5 FROM tablename
Case 5	UNLOAD SELECT * FROM tablename WHERE C1 <> constant C1 is a non-indexed column
Case 6	UNLOAD SELECT C1, C2, C3, C4, C5 FROM tablename WHERE C1 <> constant C1 is a non-indexed column
Case 7	UNLOAD SELECT * FROM tablename WHERE C3 = constant C3 is a non-indexed column
Case 8	UNLOAD SELECT * FROM tablename WHERE C4 <> constant C4 is an indexed column
Case 9	UNLOAD SELECT * FROM tablename WHERE C4 = constant C4 is an indexed column
Case 10	Same as Case 9 but used HPU parameter FORCE DB2
Parallel	Run unload against multiple partitions of the same table space in parallel

HPU and UNLOAD utility

The results of the performance measurement between HPU and the DB2 V8 Unload utility are summarized in Table 10-5.

Table 10-5 HPU V2.2 vs. DB2 V8 UNLOAD utility

Test case	HPU V2.2		DB2 V8 UNLOAD		% Difference	
	CPU	Elapsed	CPU	Elapsed	CPU	Elapsed
Case 1	11.89	62	27.56	58	131.79	-6.45
Case 2	9.44	61	46.86	57	396.39	-6.55
Case 3	13.57	59	N/A	N/A	N/A	N/A
Case 4	9.67	46	42.11	46	335.47	0.00
Case 5	18.00	58	33.02	57	83.44	-1.72
Case 6	13.93	46	47.10	50	238.11	8.69
Case 7	12.87	45	14.34	42	11.42	-6.66
Case 8	17.80	59	33.60	58	88.76	-1.69
Case 9	12.51	45	14.35	42	14.70	-6.66
Case 10	2.39	22	14.35	42	500.41	90.90

Test case	HPU V2.2		DB2 V8 UNLOAD		% Difference	
	CPU	Elapsed	CPU	Elapsed	CPU	Elapsed
Parallel	15.33	24	68.08	31	344.09	29.16

Even if an index is available, the UNLOAD utility does not take advantage of it.

You can see that in all cases, HPU outperforms the DB2 V8 UNLOAD utility in CPU time. However, in most cases, the UNLOAD utility elapsed time is slightly better.

In case 2, where the unload is performed using an image copy, the UNLOAD utility takes almost 4 times more CPU than HPU although the elapsed time for the UNLOAD utility is almost 7% better than HPU.

In case 4, the elapsed time for both HPU and the UNLOAD utility is the same, but HPU outperforms the UNLOAD utility using 1/4 the CPU time.

When parallelism is used, HPU outshines the DB2 UNLOAD utility. The CPU time for the UNLOAD utility is almost 3.5 times more than HPU and the elapsed time for the UNLOAD utility is 29% higher than HPU.

Case 10 is the same as Case 9 except the measurement for HPU involved the use of DB2 FORCE. This HPU parameter indicates that DB2 must be used to extract the requested rows. This is generally preferred when the SELECT statement uses a filtering predicate that is efficiently processed through SQL, and the filter factor is high.

Note: The DB2 UNLOAD utility in case 1 unloads the entire table space. Case 3 unloads the entire table; since it is the same as case 1, the Unload is not reported for case 3.

The graphical depiction of the results is in Figure 10-1 and Figure 10-2. The results are dramatic for the DB2 HPU tool.

Figure 10-1 shows Case 1 through Case 5.

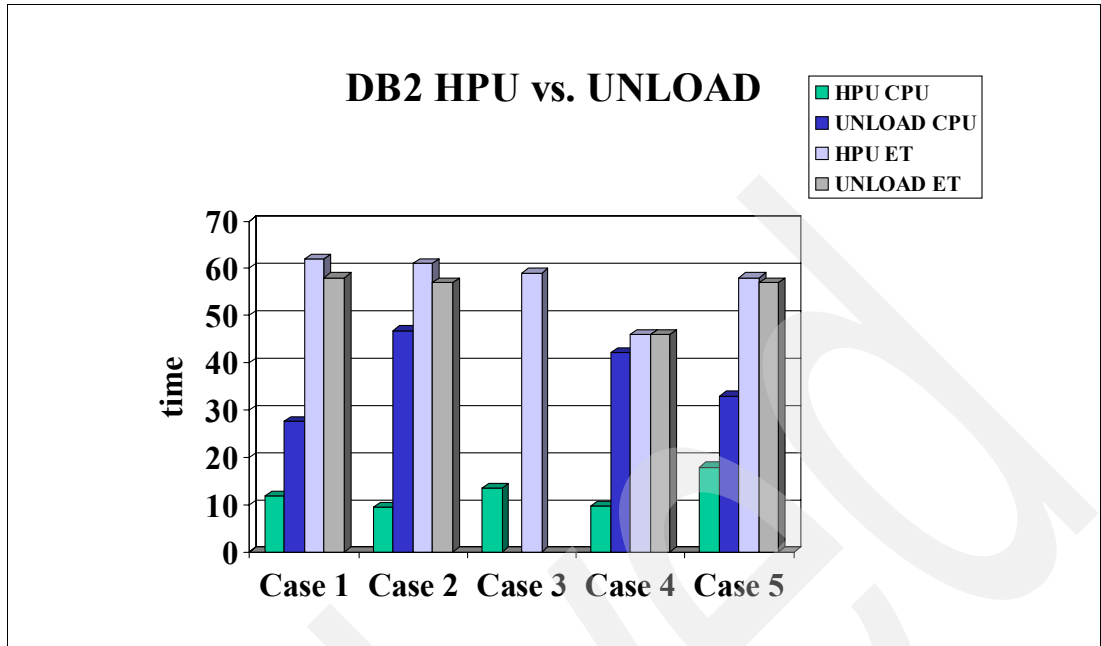


Figure 10-1 DB2 HPU vs. UNLOAD utility Case 1 through Case 5

Figure 10-2 shows Case 6 through Parallel.

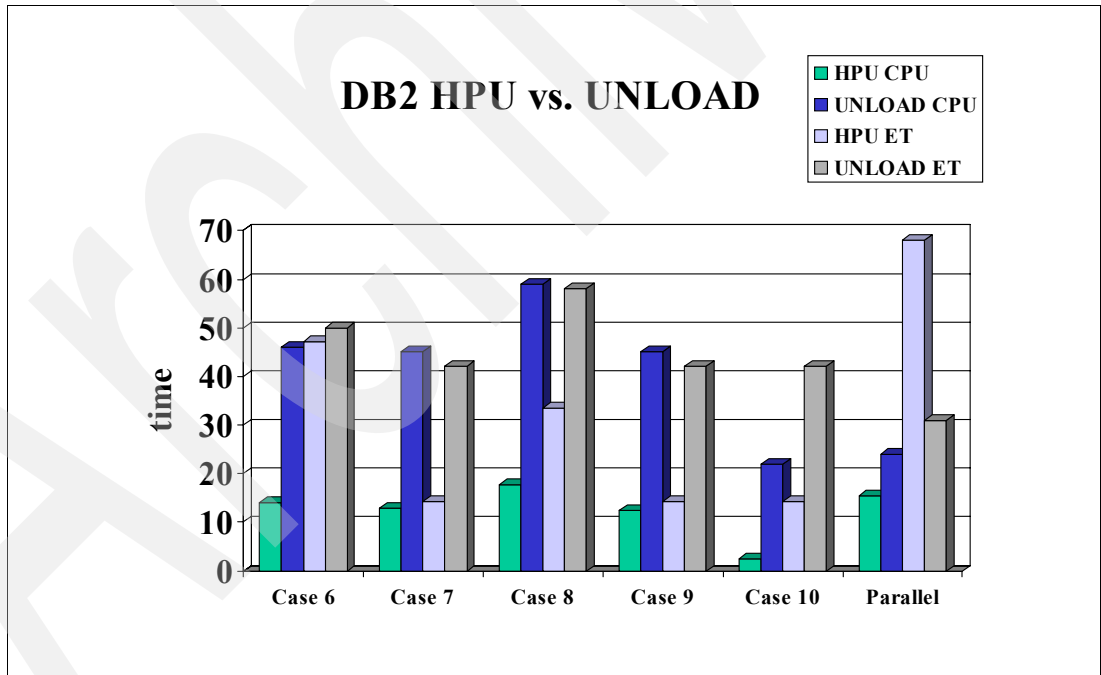


Figure 10-2 DB2 HPU vs. UNLOAD utility Case 6 through Parallel

HPU and DSNTIAUL

We show two sets of measurements, one with DSNTIAUL not using multi-row fetch, and another one using multi-row fetch. Note that DSNTIAUL in V8 always uses multi-row fetch.

DSNTIAUL without multi-row fetch

The results of the performance measurement between HPU and DB2 V8 DSNTIAUL are summarized in Table 10-6 for the case where DSNTIAUL is not using multi-row fetch.

Table 10-6 HPU V2.2 vs. DB2 V8 DSNTIAUL (no multi-row)

Test case	HPU V2.2		DB2 V8 DSNTIAUL		% Difference	
	CPU	Elapsed	CPU	Elapsed	CPU	Elapsed
Case 1	11.89	62	186.99	239	1472.66	285.48
Case 2	9.44	61	N/A	N/A	N/A	N/A
Case 3	13.57	59	186.77	238	1276.34	303.38
Case 4	9.67	46	148.46	166	1435.26	260.86
Case 5	18.00	58	184.71	236	926.16	306.89
Case 6	13.93	46	146.33	163	950.46	254.34
Case 7	12.87	45	8.23	45	-36.05	0
Case 8	17.80	59	191.34	237	974.94	301.69
Case 9	12.51	45	1.75	22	-86.01	-51.11
Case 10	2.39	22	1.75	22	-26.77	0
Parallel	15.33	24	N/A	N/A	N/A	N/A

DSNTIAUL can use an index so one would assume that DSNTIAUL may have a slight advantage over the UNLOAD utility. However, the measurement cases illustrate otherwise.

DSNTIAUL was the larger consumer of CPU time and also typically had the longer elapsed time, although two test cases showed DSNTIAUL being the better performer. However, when you compare DSNTIAUL with HPU, the results are extraordinary. HPU outperformed DSNTIAUL by using less than 10% of the CPU time.

Nearly every unload product is written with the assumption that users will unload a large percentage of the available data, usually in the 50 - 100% range. In these instances, it may be better to scan the entire table rather than using an index.

However, there are times when you may want to unload a small percentage of the rows, perhaps 10 - 20% and you know an index exists that allows access to the table data. In this case, a scan of the table may result in worse performance than if an SQL statement were executed directly by DB2; you may then see better performance with DSNTIAUL than with HPU.

If you are using HPU and you have a query where using the SQL might perform better than a table space scan, use the DB2 FORCE parameter. This causes HPU to pass the query to DB2 for execution and may result in better performance. This is evident in case 10 which is the same as case 9 except DB2 FORCE is used. The results are once again dramatic; there is a 6X improvement in CPU time with the exact same elapsed time.

Note: DSNTIAUL cannot unload from an image copy data set so the results are N/A for case 2.

DSNTIAUL with multi-row fetch

The results of the performance measurement between DB2 V8 DSNTIAUL and HPU are summarized in Table 10-7 for the case where DSNTIAUL is using multi-row fetch.

Table 10-7 HPU V2.2 vs. DB2 V8 DSNTIAUL (multi-row)

Test case	HPU V2.2		DB2 V8 DSNTIAUL		% Difference	
	CPU	Elapsed	CPU	Elapsed	CPU	Elapsed
Case 1	11.89	62	109.27	159	819	156.45
Case 2	9.44	61	N/A	N/A	N/A	N/A
Case 3	13.57	59	109.97	160	710.39	171.18
Case 4	9.67	46	57.21	74	491.62	60.86
Case 5	18.00	58	110.00	159	511.11	174.13
Case 6	13.93	46	57.54	74	313.06	60.86
Case 7	12.87	45	7.05	44	-45.22	-2.22
Case 8	17.80	59	110.81	160	522.52	171.18
Case 9	12.51	45	-	-	-	-
Case 10	2.39	22	1.44	21	-39.74	-93.45
Parallel	15.33	24	N/A	N/A	N/A	N/A

With Unload using multi-row fetch there is a general improvement in CPU time, almost half than the single row case, and the elapsed time goes down accordingly.

DSNTIAUL is still the largest consumer of CPU time and also typically has longer elapsed time when compared to HPU, although Case 7 and 10 show DSNTIAUL being the best performer. In general HPU outperforms DSNTIAUL multi-row by using about 1/3 - 1/4 of the CPU time.

You see better performance with DSNTIAUL than with HPU, in cases 7 and 10, by an even larger margin than in the previous case.

HPU vs. UNLOAD with format delimited

If the UNLOAD utility prepares the output in delimited output format, you can assume that there is more work it has to perform to add both the additional column delimited character and character string delimiter into the output data set as opposed to the external format. In spite of the extra CPU expended for this processing, the usability this feature provides is well worth it; the process is much easier since you can load the data without manually reformatting the input file for the LOAD utility.

Refer to 6.3, "LOAD and UNLOAD delimited input and output" on page 270 for the results of this performance analysis.

Summary of performance maintenance

In this appendix, we look at DB2 V7 and V8 recent maintenance that generally relates to DB2 performance and availability.

This list represents a snapshot of the current maintenance at the moment of writing, and as such, it becomes incomplete or even incorrect at the time of reading. It is here to identify areas of performance improvements.

Make sure to contact your IBM Service Representative for the most current maintenance at the time of your installation and check on RETAIN® for the applicability of these APARs to your environment, as well as to verify pre- and post-requisites.

We recommend you use the Consolidated Service Test (CST) as the base for service.

A.1 Maintenance for DB2 V8

The APARs applicable to DB2 V8 are listed in Table A-1.

They are provided here as the maintenance of interest for performance and availability functions, at the time of writing.

Make sure to contact your IBM Service Representative for the most current maintenance at the time of *your* installation.

Effective June 15, 2007, a DB2 for z/OS Version 8 SUP tape is available world-wide for new customer orders. This SUP tape integrates PTFs COR-closed through December 2006, which had also completed a Consolidated Service Test (CST) cycle. It contains CST tested PTFs which were marked "RSU 0703", which means they completed CST testing in March 2007.

This SUP build integrates a total of 964 PTFs (the delta since the previous December 2005 SUP) and will certainly help all new DB2 customers during their V8 installation.

For additional information on CST and RSU, please see:

<http://www.ibm.com/servers/eserver/zseries/zos/servicetst/mission.html>

Table A-1 DB2 V8 performance-related APARs

APAR #	Area	Text	PTF and Notes
II10817	Storage usage	Fix list part 1	
II14047	DFSORT	Description of DFSORT changes for V8 utilities	
II4309	Storage usage	Fix list part 2	
II13695	Migration/fall back	Toleration, coexistence and compatibility PTFs between V8 and V7	
PQ80631	Dependent routines	Support for dependent WLM-managed routines interface. This APAR along with WLM APAR OA04555 5/04 on z/OS1.4 allows DB2 to inform WLM when inserting work that there is an in-use WLM server TCB waiting for this work to complete. This allows WLM to be more responsive in starting server address spaces.	UQ90159 also in V7
PQ82878	Striping	Partitioned TS ran out of 123 extents with striped data set with 3 track minimum extent size because primary volume did not have sufficient free space	UQ90794 also in V7
PQ83649	RAS	-DIS THD(*) SERVICE(WAIT)	UQ87013 also in V7
PQ84160	Xloader	Xloader performance 2x improvement, now even with Unload and Load	UQ91422 also in V7
PQ86071	DSMAX	Enforce DSMAX limit by failing data set open if DSMAX exceeded and #open data set >32K, to prevent potential storage shortage	UQ87443

APAR #	Area	Text	PTF and Notes
PQ86074	DSC	Instead of reserving 2K storage for Current Path Special Register in each statement stored in DSC, allocate just enough	UQ86872
PQ86201	DSC	Reduce runtime structure in DSC	UQ86831
PQ86477	IFC	The IFCID flat file for V7 and V8 now contains current information	UQ92475 also in V7
PQ86787	Java	Abend and storage leak	UQ87049
PQ86904	IRLM	Service Roll-up	UQ87591
PQ87126	DBM1 virtual storage	Excessive DBM1 virtual storage use with KEEP DYN Y and input decimal hv	UQ88971 also in V7
PQ87129	Java	JCC driver data type switching causes skipping the DDF performance feature	UQ87633 also in V7
PQ87168	Data Sharing	Protocol 2: correct lock processing	UQ87760
PQ87381	Insert	Insert performance suffers because of exhaustive search for space to extending the page set	UK06754 also in V7
PQ87390	Access path	Bad access path when running a complex query with OPTIMIZE FOR N ROWS	UQ91022
PQ87447	Catalog query	Query 5 and 18 of the DSNTEST catalog consistency queries perform slowly	UQ90654 also in V7
PQ87509	RUNSTATS	DSTATS correction for COLGROUP FREQUENCY VALUES for aggregated statistics in partitioned table spaces	UQ91099
PQ87611	IRLM	IRLM 2.2 storage management enhancements	UK16124
PQ87756	Data sharing	Allow enablement of data sharing protocol 2	UQ87903
PQ87848	IFC	IFCID 173 for -905 occurrence	UQ90756 also in V7
PQ87917	I/O	I/O performance on ESS/2105 devices can be improved by utilizing the "bypass read/write serialization" option. It is already done for writer, this APAR adds support for reader. Support R/W I/O concurrency in same defined domain (but not same track). ESS "bypass define extent serialization" set by DB2 is disabled when XRC/PPRC/FlashCopy in progress. Extent=1 cylinder in V7, bigger in V8.	UQ88680 also in V7
PQ87969	Multi-row	Problems while processing a FETCH against a rowset cursor, after a DELETE WHERE CURRENT OF request. The problem only occurred while index RID access path was chosen by the database.	UQ92692
PQ88073	DSC	A new keyword ALL is added to EXPLAIN STMTCACHE and a new explain table DSN_STATEMENT_CACHE_TABLE is created to hold the output of IFCID 316 and 318.	UQ89372

APAR #	Area	Text	PTF and Notes
PQ88375	RUNSTATS	Correction on COLGROUP CARDINALITY for partitioned table spaces	UQ88754
PQ88582	ODBC	DB2 V8 ODBC now supports SQL statements up to 2 MB in size (was 32765) when connected to z DB2 servers running in NFM.	UQ91257
PQ88665	Space allocation	The secondary space allocation quantity in extending DB2 data set may be smaller than the one specified by the user or calculated by the sliding scale. This happens in z/OS 1.5 where extent consolidation is in effect.	UQ89517
PQ88784	Substrings	The new and changed functions allow you to specify the code unit in which the strings should be processed. The code unit determines the length in which the operation is to occur.	UQ94797
PQ88896	Archive logs	Improved performance when sequentially reading DFSMS compressed archive logs.	UQ90143
PQ89070	Locks	LOCK AVOIDANCE extended to singleton SELECT with ISO(CS) and CD(YES)	UQ91470
PQ89181	Locks	CPU reduction, benefitting multi-row by eliminating redundant lock/unlock requests	UQ90464
PQ89191	Load	Load zero rows into a DEFINE NO TABLESPACE sets the non partitioning index to RBDP, not leaving it in RW	UQ91700 also in V7
PQ89297	FLA	FLA performance enhancement with priority of Log Apply phase in RECOVER and RESTORE utility inherits that of the utility batch job instead of preempted SRB priority	UQ92067 (and in V7 for Recover)
PQ89919	Data sharing	ECSA storage is not being released by DB2 when it is obtained while running a REORG utility. Also to exclude deferred write from 180 CI limit removal.	UQ89194 also in V7
PQ90022 (and PQ93821)	Visual Explain	DSN8EXP stored procedure for usage by Visual Explain so the user is authorized to Explain without being authorized to the data	UQ92323 also in V7
PQ90075	LOBs	To reduce 15 second timer interval to milliseconds in order to avoid intermittent 15 second wait in commit when another agent is also updating GBP-dependent LOG YES LOB written at commit	UQ89852 also in V7
PQ90147	RACF	Support for 1012 secondary IDs	UQ92698
PQ90263	X Loader	Improvements for LOBs support by XLoader	UK03227 Also in V7
PQ90547	IFCID	Loss of accounting fields	UQ95032
PQ90884	RUNSTATS	DSTATS on non-indexed columns to achieve equivalent performance with V7 tool	UQ93177
PQ91009	DDF	Location name is mandatory in BSDS	UQ90701

APAR #	Area	Text	PTF and Notes
PQ91101	IFC	For V7 and V8, this APAR adds fields to IFCID 225 to record real storage statistics information. In addition, for V8 only, this APAR adds fields to IFCID 225 to record some statistics information for use with dynamic statement caching.	UQ92441 also in V7
PQ91493	Locks	CPU reduction, benefitting multi-row	UQ91466
PQ91509	EWLM	Preliminary DDF EWLM support	UK03835
PQ91898	Locks	CPU reduction, benefitting multi-row	UQ92546
PQ92072	Multi-row	ODBC array input support of multi-row Insert	UK06883
PQ92227	DPSI	It improves performance in some cases where a data-partitioned secondary index is being used to access a table. It may do this by either: 1. Reducing the overhead involved with DPSI 2. Avoiding the DPSI	UQ93972
PQ92749	Check Index	Online Check Index support (availability improvement)	UK04683
PQ93156	Restore System	FLA for Restore System utility, 4x faster with 100 MB	UQ93079
PQ93620	Multi-row	Multi-row Delete shows high CPU when number of rows in a Fetch/Delete call is high	UQ96567
PQ93821	Visual Explain	EXPLAIN sample stored procedure DSN8EXP	UQ90022
PQ94147	Prepare	High CPU consumption for dynamic prepare occurs when FREQUAL COUNT in SYSCOLDIST is greater than 100	UK00296 also V7
PQ94303	MLS	Correction	UQ96157
PQ94872	IFC	It adds fields to IFCID 225 to record some statistics information for use with dynamic statement caching.	UQ93207
PQ94822	Encryption	Exploitation of the clear key DES instructions on the CPACF and ICSF CLE, coreq is OA08172	UK00049
PQ94923	IFC	It adds fields to IFCID 225 to record some statistics information for use with dynamic statement caching.	UQ93207
PQ95164	Restore System	Corrections on RBLP, LPL, and FLA default changed to 500 MB (5x faster).	UQ95553
PQ95881	Application connectivity	JCC T4 z/OS XA Connectivity (Version 2.3)	UQ96302
PQ96189	Max open data sets	DB2 fails data set open if 65041 reached in z/OS1.6.	UQ96862

APAR #	Area	Text	PTF and Notes
PQ96628	Reorg	Online Reorg Claim/Drain lock avoidance For DB2 V7, the data-first claiming and table space level claim/drain will only be enabled if the new CLAIMDTA ZPARM is set to YES. For DB2 V8, data-first claiming and table space level claim and drain is always enabled.	UK01430 also in V7
PQ96772	DSC	Relief for large dynamic SQL cache environments	UK00991
PQ96956	Check Index	Check Index availability	UK08561, see also II14106
PQ97794	Runstats	Problem with incorrect sort parameter file size	UQ96531
PQ99524	Cancel Routines	Soft cancel recovery fixes for sp/udf code paths	UK01175 also V7
PQ99608	Logging	Additional I/O wait when FORCE for identity columns is used	UK00314
PQ99658	IFC	Availability of virtual storage diagnostic record IFCID 225 with Class 1 Statistics System parameter STATIME default time is reduced from 30 minutes to 5. IFCID 225 is added to STATISTICS trace class 1. An internal cache of IFCID 225 records is being kept for storage changes over time.	UK04394 UK04393 for V7
PQ99707	JCC	Performance improvement for JCC clients with EWLM	UK03058
PK00213 PK41182	REORG	Display claimers on an object when REORG fails to drain	UK20497 UK20496 for V7
PK01510	Runstats	NPI clusterratio	UK03490
PK01841	Storage	Large increase in stack storage consumption	UK01844
PK01855	Storage	Reducing stack storage related to partitions	UK01467
PK01911	Latches	Waits on latch 32 for RRSAB batch programs	UK04743
PK03293	RLF	Extra resource limit facility RLST scans	UK04036
PK03469	Runstats	Errors for RUNSTATS TABLESPACE TABLE with multiple COLGROUPs	UK03176
PK04076	LOAD	Sort is avoided for LOAD SHRLEVEL NONE if data is sorted and only one IX exists	UK03983
PK05360	Multi-row	Allow hybrid join for local multi-row SQL	UK9531
PK04107	Optimizer	Bidirectional indexability between Unicode and EBCDIC	UK06418
PK05644	Preformat	Pageset preformat quantity adjustment to 2 cys	UK08807
PK09158	Bind	Speed-up Bind Replace or Free by eliminating a non-linear performance problem	UK05786 also in V7
PK09268	Column processing	Improved CPU time for date, time and timestamp	UK06505

APAR #	Area	Text	PTF and Notes
PK10021	Multi-row	Support parallelism in multi-row Fetch with host variable, parameter marker, or special register	UK08497
PK10278	LOBs	File reference variable for LOAD/REORG	UK13721 also V7
PK10662	EDM	EDM LRU chain latch reduction (LC24) for large EDM pool where no LRU replacement takes place	UK07167
PK11355	Block fetch	Avoid block fetch disablement in a specific situation	UK07192
PK12389	SQL stage 1 and 2 predicates	Enabled in compatibility mode	UK10002
PK13455	Partitions	High getpage with page range Y on table-based partitioning	UK09343
PK14393	Latches	High LC32 contention from concurrent prefetch requests accessing a shared storage pool	UK09097
PK14477	Utilities	Parallel tasks in index build for LOAD and REORG	UK14058 also V7
PK15056	Data sharing	GBP-dependency not removed for pagesets after extended period	UK10819
PK15288	Multi-row FETCH	Optimize DB2 ODBC's bulk fetch to use the multi-row FETCH statement	UK26761
PK18059	SORTDATA	Make available optional operand YES/NO for SORTDATA of the REORG utility for situations of insufficient sort disk or just space reclaim.	UK12821
PK18162	DSNTRACE	CPU performance problems when DSNTRACE is used (like by CAF)	UK13636
PK18454	DRDA	DRDA over TCP/IP is OK for zIIP redirect	UK15499
PK19191	DRDA	Change to properly increment the accounting and statistics fields using block fetching for a cursor	UK12124
PK19303	WLM	Lock holder inherits WLM priority from lock waiter	OPEN
PK19769	Real storage	MVS discard and pool reset to reduce thread storage	UK12253
PK19920	Utilities	Utilities are OK for zIIP redirect	UK15814
PK20157	DDF/LOBs	Running out of DDF Virtual Storage with LOBs	UK12328
PK21237	Virtual storage DBM1	The buffer manager engines to use a single above the bar storage pool. The default maximum for the number of engines has been reduced for deferred write, castout and GBP engines	UK14283
PK21268	Virtual storage	Move CURRENT PATH storage above the bar	UK14343

APAR #	Area	Text	PTF and Notes
PK21861	Virtual storage DBM1	Dynamic Statement Caching and bind option KEEP_DYNAMIC(YES). New online ZPARM CACHEDYN_FREELOCAL indicates that a cleanup will be in effect. More instrumentation fields in the IFCID 225 for more information on the cache	UK15493
PK21892	Virtual storage DBM1	Reduce storage use for accumulation of cached stack entries of allied agents (like ND type CICS threads)	UK14540
PK22442	Virtual storage DDF	DDF storage shortage while processing long threads with KEEP_DYNAMIC=YES and hundreds of output columns causing large SQLDA	UK14634
PK22611	Locking and prefetch	For data sharing users excessive drain locks acquired. For both data sharing and non-data sharing users excessive sequential prefetches scheduled	UK14326
PK22814	Sort Merge Join	Cost estimation with correlated subqueries	UK13671
PK22887	LOBs	Improve insert or update performance for LOB data	UK15037 also in V7
PK22910	LOBs	LOAD and UNLOAD with file reference variables	UK13721 also V7
PK23743	Virtual storage DDF	DB2 storage management will monitor xxxxDIST address space storage usage. When the storage cushion for the address space is breached, DB2 will begin contraction of storage pools	UK17364
PK23495	Views	Push down predicates with outer join	UK14370
PK23523	UNION ALL	Runtime structure memory consumption	UK18547
PK24556 PK24585	DSNAIMS	DSNA317I DSNAIMS ERROR IN OTMA_OPENX.A	UK15449 UK15285, see also PK24819 (UK15224) for IMS
PK24558	DISPLAY	Poor performance for -DISPLAY DATABASE SPACENAM LOCKS command	UK15650 also V7
PK24710	Incorrect data	Incorrect output with IN list query	UK15150
PK25241	LOBs	Insert/update performance	UK18503
PK25326	Real storage	Contract PLOCK engines storage after use	UK15958
PK25427	Real storage	Better real frame accounting with MVS OA15666	UK12253
PK25742	Utilities	REORG TABLESPACE SHRLEVEL CHANGE improved building the mapping table index for BUILD or SORTBLD phases	UK15904
PK26199	DISPLAY performance	Performance improvement for -DISPLAY DATABASE SPACENAM RESTRICT or ADVISORY command	PK21371 (PK13555)

APAR #	Area	Text	PTF and Notes
PK26692	PREPARE	Short PREPARE CPU time degradation	UK18020
PK26879	Latches	High LC32 contention with a large number of package search lists (collid*)	UK16191
PK27281	SQL	Non correlated EXISTS subquery	UK17220
PK27287	LOAD	New LOAD keyword, IDENTITYOVERRIDE, which enables LOAD REPLACE to reload unloaded identity column that was created as GENERATED ALWAYS back into the same table	UK17369
PK27578	Parallel queries	Queries are OK for zIIP redirect	UK16616
PK27712	Utilities	LOAD, REORG, and REBUILD now show correct t accounting for zIIP	UK17089
PK28561	Packages	IFCID239 will no longer collect package detail information by default	UK18090
PK28637	Packages	Change improves all packages and triggers	UK18201
PK29626	Buffer pool management	Real storage management and hash chain	UK17312
PK29791	Data sharing	Option to LIGHT mode to allow the user the default V7 behavior which ignores the INDOUBT URs and terminates after freeing retained locks.	UK19282
PK29826	RECOVER	It allows RECOVER PARALLEL utility to run when cataloged image copy data set is moved by external means from one kind of tape device to another	UK19777 UK19776 for V7
PK30087	REBUILD INDEX	REBUILD INDEX of a single NPI over a partitioned table space may experience degradation in the BUILD phase with zIIP	UK18276
PK30160	INSERT	Free space is not reused correctly during space search for non-segmented table spaces	UK17805
PK34251	TEMPLATE	Three new TEMPLATE utility keywords, SUBSYS, LRECL, and RECFM, to allow dynamic allocation of a DDNAME to use MVS BATCHPIPES SUBSYSTEM	UK25290 UK25291 for V9
PK34441	REORG	Improve availability of concurrent SQL	UK21950
PK35390	REORG	REORG SHRLEVEL REFERENCE or CHANGE: reduce elapsed time during the SWITCH phase.	UK22021 UK22020 for V7
PK36717	INSERT	Insert performance suffers with segmented table spaces	UK21175
PK37354	IFC	IFCID 225 to collect buffer manager storage blocks below the bar	UK25044 more functions in V9
PK38201	INSERT	Storage leak with PK32606 (PE) causes long insert time	UK23728 UK23727 for V7
PK40057	Prepare	Avoid column level authorization checking	UK24197

APAR #	Area	Text	PTF and Notes
PK46170	Precompiler	Precompiler porting from V7	UK28485
PK47840	INSERT	Implement P-lock failure threshold of 20 to improve INSERT performance for row locking	UK27594 UK27593 for V7
PK49972	CPU reduction	Reduce GETMAIN and FREEMAIN activity during DB2 SIGNON processing for recovery coordinators such as CICS or IMS	OPEN

A.2 Maintenance for DB2 V7

The APARs applicable to DB2 V7 are listed in Table A-2.

They are provided here as the most known current maintenance of interest for performance and availability functions, at the time of writing.

Make sure to contact your IBM Service Representative for the most current maintenance at the time of *your* installation.

Table A-2 DB2 V7 performance-related APARs

APAR #	Area	Text	PTF and Notes
II13695	Migration/fall back	Toleration, coexistence and compatibility PTFs between V8 and V7	
PQ47973	IFC	A new IFCID 234 trace record is created to return user authorization information via READS	UQ57178
PQ49458	DDF	1.OPT FOR for APS and network blocking when specified 2. V7 FETCH FIRST for APS but not network blocking when no OPT FOR specified If you specify the OPTIMIZE FOR n ROWS and the FETCH FIRST m ROWS clauses, DB2 optimizes the query for n rows.	UQ79775
PQ54042	Access path	Performance improvements for SQL statements containing a BETWEEN predicate	UQ61327
PQ61458	Star join	Performance improvement for star join using sparse index on snowflake work files	UQ67433
PQ62695	DB2 Universal Driver	It provides 13 stored procedures for generating a result set corresponding to the Schema Metadata APIs documented in the JDBC and ODBC specifications	UQ72083
PQ69741	Data sharing	Code changes to determine whether an idle page set in data sharing should be physically closed for CLOSE YES or NO attribute of the table space or index.	UQ74866 Included in DB2 V8

APAR #	Area	Text	PTF and Notes
PQ69741	Data sharing	Close inactive GBP-dependent object if CLOSE Y to minimize performance disruption at the beginning of the day for example: Alter CLOSE Y from N if the old behavior is desired.	UQ74866
PQ69983	Accounting	It needs also V7 PQ74772 (DB2 PE) to fix incorrect ET and CPU time in accounting for SP, UDF, triggers	UQ77215
PQ71179	LOBs	LOB space reuse code has been changed to rely upon read-LSN processing as before, but if read-LSN processing fails then a lock is obtained on the deallocated LOB to determine whether the page owned by the deallocated LOB can safely be reused	UQ77421
PQ71766	Data spaces	Avoid multiple I/Os resulting from single prefetch or deferred write request when multiple data spaces exist	UQ75848
PQ71775	Catalog query	Performance improvement for catalog consistency query DSNTSEQ, 200x faster by query rewrite	UQ85043
PQ75064	Statistics	Avoid incremental bind count in statistics for SP call with name in hv when the match is found in cache	UQ79020
PQ78515	RDS	This fix checks every ten minutes to see if RDS OP pool is greater than 50 MB. If it is greater than 50 mb, then with fix applied it contracts/defragments only the leftover and unused storage. It does not flush any reusable parts of the RDS OP pool.	UQ83466
PQ79232	DSMAX	Increasing DSMAX and activating it via SET SYSPARM do not use new value even though trace shows new value	UQ81754
PQ80631	Dependent routines	Support for dependent WLM-managed routines interface. This APAR along with WLM APAR OA04555 5/04 on z/OS1.4 allows DB2 to inform WLM when inserting work that there is an in-use WLM server TCB waiting for this work to complete. This will allow WLM to be more responsive in starting server address spaces.	UQ90158 also in V8
PQ81904	RDS	Use of thread rather than shared pool to reduce cl32 latch contention (RDS OP pool)	UQ84631
PQ82878	Striping	Partitioned TS ran out of 123 extents with striped data sets with 3 track minimum extent size because primary volume did not have sufficient free space	UQ90793 also in V8
PQ84160	Xloader	Xloader performance 2x improvement, now even with Unload and Load	UQ91416 also in V8
PQ85764	IFI	IFCID 225 available via READS	UQ87093

APAR #	Area	Text	PTF and Notes
PQ85843	Data sharing	Castout to skip 180CI limit if serviceability IFCID 338 on to remove CPU spikes every 15 sec caused by excessive unlock castout and delete namelist. V8 PQ86071 4/04 unconditionally for all async I/O such as DASD read and castout.	UQ86458
PQ86037	Insert	Insert at end option for Member Cluster TS, with Alter TS for 0 PCTFREE and FREEPAGE to always search forward for free space in Insert into MC TS to avoid periodically reading space map pages from the beginning costing >10sec, also heavy space map latch contention, for any page size. Load Replace dummy or REORG as needed if deletes later.	UQ86868 also V8 (PQ87391) +PQ99524
PQ86049	Data sharing	Request preference for CFLEVEL 13 instead of 7 for GBP allocation to reduce high CF utilization and use read castout class	UQ87321
PQ86477	IFC	The IFCID flat file for V7 and V8 now contains current information.	UQ92474 also in V8
PQ87126	Virtual storage	Excessive DBM1 VS use with KEEPDMY Y and input decimal hv (also V8)	UQ88970
PQ87381	Insert	Insert performance suffers because of exhaustive search for space to extending the page set	UK06753 also in V8
PQ87447	Catalog query	Query 5 and 18 of the DSNTESEQ catalog consistency queries perform slowly	UQ90653 also in V8
PQ87783	Data sharing	Correct false contention counter when 1 member in LPAR. If multiple members in the same data sharing group on the same LPAR (rather rare), false contention overreported. Needs DB2 PE fix as of 8/04	UQ88734
PQ87917	I/O	I/O performance on ESS/2105 devices can be improved by utilizing the "bypass read/write serialization" option. It is already done for writer, this APAR adds support for reader. Support R/W I/O concurrency in same defined domain (but not same track). ESS "bypass define extent serialization" set by DB2 is disabled when XRC/PPRC/FlashCopy in progress.	UQ886790 also in V8
PQ88587	Load	LOAD utility with STATISTICS INDEX keywords specified at many tables present, appears to loop and completes with high elapsed time.	UQ89026
PQ89191	Load	Load zero rows into a DEFINE NO TABLESPACE sets the non partitioning index to RBDP, not leaving it in RW.	UQ91968 also in V8
PQ89297	FLA	FLA performance enhancement with priority of Log Apply phase in RECOVER and RESTORE utility inherits that of the utility batch job instead of preempted SRB priority	UQ92066 (and V8 for Restore)

APAR #	Area	Text	PTF and Notes
PQ89919	Reorg	ECSA storage is not being released by DB2 when it is obtained while running a REORG utility.	UQ89190 also in V8
PQ89919	Data Sharing	To exclude write 1. IRWW non data sharing: +2.9% ITR, -3% CPU, IRWW 2way data sharing: +4.8% ITR, -4.4% CPU -> less improvement after excluding write, 0to3% instead of 0to4%? 2. More pages read per list prefetch, reduced Unlock Castout (#async writes in GBP-dep LBP stats), reduced DBM1 SRB time	UQ89190 also in V8
PQ90022 (and PQ93821)	Visual Explain	DSN8EXP stored procedure for usage by Visual Explain so the user is authorized to Explain without being authorized to the data	UQ92322
PQ90075	LOBs	To reduce 15 second timer interval to milliseconds in order to avoid intermittent 15 second wait in commit when another agent is also updating GBP-dependent LOG YES LOB written at commit	UQ89850 also in V8
PQ90263	Xloader LOBs	>32 KB support	UK03226 also in V8
PQ91101	IFC	For V7 and V8, this APAR adds fields to IFCID 225 to record real storage statistics information. In addition, for V8 only, this APAR adds fields to IFCID 225 to record some statistics information for use with dynamic statement caching.	UQ92440 also in V8
PQ93009	Online stats	New criteria for DSNACCOR to recommend RUNSTATS	UQ93893
PQ94147	Bind	To reduce bind time by hashing instead of sequential search on SYSCOLDIST entries in DSNXOCN, -6.7x CPU	UK00295
PQ94793	Data Sharing	ABEND04E RC00D10340 in DSNJR006 attempting to read/merge the logs in a data sharing environment. Also for DB2 V7.	UQ93407
PQ96628	Reorg	Online Reorg Claim/Drain lock avoidance For DB2 V7, the data-first claiming and table space level claim/drain will only be enabled if the new CLAIMDTA ZPARM is set to YES. For DB2 V8, data-first claiming and table space level claim and drain is always enabled.	UK01429 also in V8
PQ99524	Cancel Routines	Soft cancel recovery fixes for stored procedures and UDFs code paths	UK01174 also in V8
PQ99658	Statistics	Availability of virtual storage diagnostic record IFCID 225 with Class 1 Statistics. System parameter STATIME default time is reduced from 30 minutes to 5. IFCID 225 is added to STATISTICS trace class 1. An internal cache of IFCID 225 records is being kept for storage changes over time.	UK04393 also in V8

APAR #	Area	Text	PTF and Notes
PK09158	Bind	Speed-up Bind Replace or Free by eliminating a non-linear performance problem	UK05785 also for V8
PK14477	Utilities	Parallel tasks in index build for LOAD and REORG	UK14057 also V8
PK22887	LOBs	Improve insert or update performance for LOB data	UK15036 also V8
PK24558	DISPLAY	Poor performance for -DISPLAY DATABASE SPACENAM LOCKS command	UK15649 also V8
PK47840	INSERT	Implement P-lock failure threshold of 20 to improve INSERT performance for row locking	UK27593 UK27594 for V8

A.3 Maintenance for z/OS

The APARs applicable to z/OS are listed in Table A-3.

They are provided here as the most known current maintenance of interest for performance and availability functions, at the time of writing.

Make sure to contact your IBM Service Representative for the most current maintenance at the time of *your* installation.

Table A-3 z/OS DB2 performance-related APARs

APAR #	Area	Text	PTF and Notes
OA06729	Coupling Facility	To avoid delay in getting asynchronous (heuristic synchronous to asynchronous conversion) lock request to be processed. Also fix inconsistent stats on lock request and contention.	UA11478 UA11478 UA11479 UA11480 UA11481 UA11482
OA04555	WLM	The WLM queue/server management application interfaces are changed by this SPE to allow an application to inform WLM that a work request with a dependency to other work already in progress is inserted.	UA11154 UA11155 UA11156
OA05960	Extents	The request for an extend to a specific CI or RBA number through Media Manager Services is now honored for all cases.	UA08703
OA07196	EWLM	Function	UA15188 UA15189
OA07356	EWLM	Function	UA15456
OA08172	Encryption	Enable more exploitation of the clear key DES instructions on the CPACF	UA15677 UA156780 UA15679
OA12005	EWLM	Performance improvement	UA22388

APAR #	Area	Text	PTF and Notes
OA15666	Storage	DISCARDATA request support	UA27812
OA17114	Storage	Reduce excessive amount of fixed frames above the line (but below the bar)	UA34634

Archived

The DBM1 storage map

In this appendix we describe in more details the contents of the DB2 PE Storage Statistics reports.

We provide a description for the headings in the three sections of the report and point out what is applicable for V7 or V8 of DB2.

B.1 DB2 PE Storage Statistics trace and report

This section describes the PE Storage Statistics report and all the fields contained in the report.

It is important to know that this trace and report layout is used not only for DB2 V8 but also for the other supported versions of DB2. As a result, some of the information in this report, when associated with a previous version (like a DB2 V7 subsystem) is not available (N/A) or reported differently, and therefore shown as N/A. These storage headings are designated in **bold** characters in the tables below.

The Storage Statistics trace has the following three different storage blocks:

- ▶ DBM1 AND MVS STORAGE BELOW 2 GB
- ▶ DBM1 AND MVS STORAGE ABOVE 2 GB
- ▶ REAL AND AUXILIARY STORAGE

The storage quantities are shown in MB; as a result, you need to multiply the number by 1024x1024 to get the actual number of bytes.

Table B-1 summarizes the storage areas below the 2 GB bar.

Table B-1 DBM1 AND MVS STORAGE BELOW 2 GB

Storage heading	Description
TOTAL DBM1 STORAGE BELOW 2 GB	Sum of TOTAL GETMAINED STORAGE TOTAL VARIABLE STORAGE TOTAL FIXED STORAGE TOTAL GETMAINED STACK STORAGE
TOTAL GETMAINED STORAGE	Total storage acquired by GETMAIN macro. This includes space for virtual pools, EDM pool, compression dictionaries, castout buffers, data space lookaside buffer, hiper pool control blocks, and data space buffer pool control blocks
VIRTUAL BUFFER POOLS	Total storage allocated for virtual buffer pools
VIRTUAL POOL CONTROL BLOCKS	Total storage allocated for virtual pool control blocks. Each virtual pool control block is 128 bytes resulting in 128 * (VIRTUAL BUFFER POOLS)
EDM POOL	Total storage for plans and packages in EDM pool
COMPRESSION DICTIONARY	Total storage for compression dictionaries
CASTOUT BUFFERS	Total storage for data sharing castout buffers
DATA SPACE LOOKASIDE BUFFER	Total storage for data space lookaside buffers
HIPERPOOL CONTROL BLOCKS	Total storage for hiper pool control blocks - 56 bytes for each hiper pool page
DATA SPACE BP CONTROL BLOCKS	Total storage for data space buffer pool control blocks - a control block is 128 bytes and one is allocated for each data space page
TOTAL VARIABLE STORAGE	Total storage used by all variables. This includes storage for: system agents, local agents, RID pool, pipe manager subpool, local dynamic cache storage blocks, local dynamic statement cache statement pool, buffer and data manager trace tables

Storage heading	Description
TOTAL AGENT LOCAL STORAGE	The amount of storage allocated for agent-related local storage for operations such as sort.
TOTAL AGENT SYSTEM STORAGE	Storage consumed by the various system agents, prefetch, castout, and deferred write engines.
NUMBER OF PREFETCH ENGINES	Number of engines used for sequential, list, and dynamic prefetch
NUMBER OF DEFERRED WRITE ENGINES	Number of engines used for deferred write operations
NUMBER OF CASTOUT ENGINES	Number of engines used for data sharing castout engines
NUMBER OF GBP WRITE ENGINES	Number of write engines used for group buffer pools
NUMBER OF P-LOCK/NOTIFY EXIT ENGINES	Number of engines used for data sharing P-lock or notify exit conditions
TOTAL AGENT NON-SYSTEM STORAGE	This is a derived field: TOTAL AGENT LOCAL STORAGE - TOTAL AGENT SYSTEM STORAGE
TOTAL NUMBER OF ACTIVE USER THREADS	Includes all active allied threads and the current number of active DBATs (derived from IFCID 225 and 001)
RDS OP POOL	Total storage for RDS operations pool used for sort, prepare, and other common functions. In V8 it is only used for binding (prepare)
RID POOL	Storage used by list prefetch, multiple index access processing, and hybrid joins; maximum size specified by DSNZPARM MAXRBLK
PIPE MANAGER SUB POOL	Storage allocated to pipe manager for parallel query operations
LOCAL DYNAMIC STMT CACHE CNTL BLKS	Storage for local dynamic statement cache blocks
THREAD COPIES OF CACHED SQL STMTS	Storage used for the local dynamic statement cache pool
IN USE STORAGE	The amount of storage used for thread copies in the local cache storage pool. This is a subset of the total allocated storage for THREAD COPIES OF CACHED SQL STMTS
STATEMENTS COUNT	Number of SQL statements in cached pools
HWM FOR ALLOCATED STATEMENTS	A statistics interval high water mark of allocated storage for thread copies in the local cache storage pool
STATEMENT COUNT AT HWM	The number of statements in the local cache storage pool at high storage time
DATE AT HWM	Date at high water mark for storage
TIME AT HWM	Timestamp at high water mark for storage
BUFFER & DATA MANAGER TRACE TBL	Storage used for Buffer and Data Manager trace tables
TOTAL FIXED STORAGE	Total amount of long-term page fixed storage
TOTAL GETMAINED STACK STORAGE	Total GETMAINED storage allocated for program stack use
STORAGE CUSHION	Storage reserved to allow DB2 to complete critical functions while short-on-storage. This includes the contract warning cushion, storage reserved for must-complete operations, and storage for MVS use.

Storage heading	Description
24 BIT LOW PRIVATE	Amount of private MVS storage below the 16 MB line. This storage is obtained from bottom upward, usually for unauthorized programs
24 BIT HIGH PRIVATE	Amount of private MVS storage below 16 MB line. This storage is obtained from top downward, usually for authorized programs
31 BIT EXTENDED LOW PRIVATE	Amount of private MVS storage above 16 MB line. This storage is obtained from bottom upward, usually for unauthorized programs
31 BIT EXTENDED HIGH PRIVATE	Amount of private MVS storage above 16 MB line. This storage is obtained from top downward, usually for authorized programs
EXTENDED REGION SIZE (MAX)	Maximum amount of MVS private storage available above the 16 MB line
EXTENDED CSA SIZE	The size of the common storage area (ECSA) above the 16 MB line
AVERAGE THREAD FOOTPRINT	The current average memory usage of active user threads. Derived as (TOTAL VARIABLE STORAGE - TOTAL AGENT SYSTEM STORAGE) / (ALLIED THREADS + DBATS)
MAX NUMBER OF POSSIBLE THREADS	The maximum number of possible threads dependent on storage size and average thread footprint

Table B-2 summarizes all of the storage areas above the 2 GB bar.

Table B-2 DBM1 STORAGE ABOVE 2 GB

Storage heading	Description
FIXED STORAGE	Total amount of long-term page fixed storage
GETMAINED STORAGE	Total storage acquired by GETMAIN macro. This includes space for buffer pools, EDM pool, compression dictionaries and castout buffers
COMPRESSION DICTIONARY	Total storage for compression dictionaries
CACHED DYNAMIC SQL STATEMENTS (MAX)	Maximum size of the dynamic statement cache. Since it can be increased and decreased dynamically using SET SYSPARM, this is its maximum value.
DBD CACHE (MAX)	Maximum size of the DBD cache. Since it can be increased and decreased dynamically using SET SYSPARM, this is its maximum value.
VARIABLE STORAGE	Amount of variable storage
VIRTUAL BUFFER POOLS	Total storage allocated for buffer pools
VIRTUAL POOL CONTROL BLOCKS	Total storage allocated for buffer pool control blocks
CASTOUT BUFFERS	Total storage used for castout buffers in data sharing
STAR JOIN MEMORY POOL	Total storage used for star join cache for data and keys

Table B-3 describes the amount of real and auxiliary storage in use.

Table B-3 REAL AND AUXILIARY STORAGE

Storage heading	Description
REAL STORAGE IN USE	Number of real frames (4K) in use
AUXILIARY STORAGE IN USE	Number of auxiliary slots (4k) in use

Archived

SQL PA sample reports

In this appendix we show samples of the new SQL PA reports illustrating some of the DB2 V8 functionality.

For more information on SQL PA Version 2.2, refer to the white paper *Achieving Maximum Productivity With DB2 SQL Performance Analyzer* available from:

<http://www.ibm.com/software/data/db2imstools/db2tools/db2sqlpa.html>

C.1 DB2 SQL PA additional reports

Notice that the index used has a NOT PADDED attribute and a cartesian join was detected.

Example C-1 reports the query definition and also some good news for the query.

Example: C-1 SQL PA query description

=====

SQL PA ANALYSIS FOR QUERYNO 100000001

```
SELECT COUNT(*) FROM
  SYSIBM.SYSPACKSTMT A, SYSIBM.SYSPACKDEP B
WHERE  A.LOCATION = ? AND
       B.DLOCATION = ? AND
       B.DCOLLID = ? AND
       A.COLLID <> ?
```

QUERYNO: 100000001 QBLOCKNO: 1 PLANNO: 1 MIXOPSEQ: 0
PROCESS ->

+-----+
|ANL7002I *** GUIDELINE:

|This plan step has not selected any Sequential|List Prefetch I/O.
|If the SQL processes just a few rows that is OK, but if many rows
|are involved, you can help promote Sequential Detection by both
|accessing the data in sequential order (presort?) and by binding
|with Release (Deallocate) to avoid resetting counters at Commit.
+-----+

+-----+
|ANL7005I *** GOOD NEWS:

|This SQL features a Unique Index being used to access the table
|that can avoid Distinct Sorts with Group By and Count Distinct C1.
|Also, unique index will avoid 4K "in memory table" on Correlated
|Subquery, considers special access path when used with "=" preds,
|and may convert Subquery to Join on IN, =ANY or =SOME predicates.
+-----+

+-----+
|ANL6005I *** NOTE:

|This statement contains a Built in Column Function: however, it
|is not being processed at either Stage 1 Retrieval or Sort Time.
|In general, this means poor performance, because the column(s) get
|evaluated only at the end of Stage 2 processing. This might be due
|to multiple columns used, Group By, not all Stage 1 preds, etc.
+-----+

Example C-2 provides information about the indexes.

Example: C-2 SQL PA index information

ACCESS IS VIA THE CLUSTERING (INSERT & LOAD ORDER) INDEX FOR THIS TABLE

+-----+
|ANL5018W *** ALERT:

This clustering (insert) index has a low cluster ratio (below 80) and/or is not well clustered (Clustered flag = N), meaning that inserts are following an unclustered pattern: this is not helpful. You should Reorganize your Table and Index, and re-run Runstats to update these statistics. That should result in better access paths and performance in general.

RANDOM MATCH IX SCAN

IX CREATOR: SYSIBM
INDEX NAME: DSNKSX01

VERS: 1 KEY LEN: -1 PADDED: N C-ED: N C-ING: Y CLURATIO: 10.0000
FULLKEY CARD: 25 FIRSTKEY CARD: 25
TYPE: 2 NLEAF PAGES: 34 NLEVELS: 2 UNIQUE: U DECLARE UNIQ
1 OF 5 COLUMNS ARE MATCHED CLOSE: N LOCK MODE: IS BPOOL: BPO

KEY	COLUMN NAME	ORDER	TYPE	DIST	LEN	NULL	COLCARD	DIST#
1	LOCATION	A	VARCHAR	N	128	N	-1	0
2	COLLID	A	VARCHAR	N	128	N	-1	0
3	NAME	A	VARCHAR	N	128	N	-1	0
4	CONTOKEN	A	CHAR	N	8	N	-1	0
5	SEQNO	A	SMALLINT	N	2	N	-1	0

ANL6052I *** NOTE:
This index is specified as "Not Padded", allowing storage of a varying length index key. Padded indexes use blanks to fill out their fixed length keys and are not eligible for Index Only scan. "Not Padded" indexes do not blank fill CHAR and VARCHAR columns, allowing greater flexibility and better use of storage, packing more entries into a single Leaf page. Index Only access allowed.

NOTE: NO ALTERNATIVE INDEXES AVAILABLE FOR THIS TABLE

ANL6016I *** NOTE:
Presently this query is only matching some of the indexed columns on this index key. Maximizing use of Stage 1 predicates against these index columns will improve performance, by matching and/or screening rows and therefore reducing data page I/O requirements.

ANL7034I *** GOOD NEWS:
This index is Not Padded, making it a true variable length index. "Index Only" access is now possible and DB2 will compare CHAR and VARCHAR columns of unequal length against this index during Stage 1 processing.

ANL7009I *** GUIDELINE:
On composite (multi-column) indexes, favor matching index scan of as many columns as possible, using Equals (=), Range (>, <, >=, <=, Between, Like), In (List) and Null predicates, where a Range pred

```

|is the last matching predicate. Apply left to right against index |
|columns, skipping no columns in the L-R sequence. DB2 may "screen" |
|remaining Stage 1 preds against the rid list before data access. |
+-----+

```

```

+-----+
|ANL7010I *** GUIDELINE: |
|On composite (multi-column) indexes, collect correlated key stats |
|on column pairings, as well as cardinality stats, like 2ndkeycard, |
|3rdkeycard, etc., with Runstats. These additional statistics will |
|allow DB2 to make better filter estimates for Equality and Range |
|and In List predicates, thereby selecting proper indexes and more |
|realistic estimates for rows returned from the query (QCARD). |
+-----+

```

```

+-----+
|ANL7024I *** GOOD NEWS: |
|Index Screening is now available for List Prefetch processing, |
|which allows Random indexes to "screen" additional predicates |
|after matching index scan, and before data access. |
+-----+

```

THIS IS AN "INDEX ONLY" ACCESS: NO DATA PAGES ARE READ FROM THE TABLE

```

+-----+
|ANL7015I *** GUIDELINE: |
|Collecting Non-uniform column statistics greatly enhance the DB2 |
|Optimizer accuracy when processing Equals and In (List) preds. |
+-----+

```

Example C-3 shows the analysis of the query.

Example: C-3 SQL PA query analysis

A JOIN OF 2 TABLES HAS BEEN DETECTED. THIS WAS THE FIRST TABLE ACCESS.

```

+-----+
|ANL7023I *** GOOD NEWS: |
|Equi-join predicates of unequal length may be handled at stage 1: |
|DB2 pads out CHAR and VARCHAR column types to accomplish the join. |
+-----+

```

QUERYNO: 100000001 QBLOCKNO: 1 PLANNO: 2 MIXOPSEQ: 0
PROCESS ->

```

+-----+
|ANL7002I *** GUIDELINE: |
|This plan step has not selected any Sequential|List Prefetch I/O. |
|If the SQL processes just a few rows that is OK, but if many rows |
|are involved, you can help promote Sequential Detection by both |
|accessing the data in sequential order (presort?) and by binding |
|with Release (Deallocate) to avoid resetting counters at Commit. |
+-----+

```

```

+-----+
|ANL6005I *** NOTE: |
+-----+

```

```

|This statement contains a Built in Column Function: however, it
|is not being processed at either Stage 1 Retrieval or Sort Time.
|In general, this means poor performance, because the column(s) get
|evaluated only at the end of Stage 2 processing. This might be due
|to multiple columns used, Group By, not all Stage 1 preds, etc.
+-----+

```

ACCESS IS VIA THE CLUSTERING (INSERT & LOAD ORDER) INDEX FOR THIS TABLE

```

+-----+
|ANL5018W *** ALERT:
|This clustering (insert) index has a low cluster ratio (below 80)
|and/or is not well clustered (Clustered flag = N), meaning that
|inserts are following an unclustered pattern: this is not helpful.
|You should Reorganize your Table and Index, and re-run Runstats to
|update these statistics. That should result in better access paths
|and performance in general.
+-----+

```

RANDOM MATCH IX SCAN

IX CREATOR: SYSIBM
INDEX NAME: DSNKDX01

VERS: 1 KEY LEN: -1 PADDED: N C-ED: N C-ING: Y CLURATIO: 10.0000
FULLKEY CARD: 25 FIRSTKEY CARD: 25
TYPE: 2 NLEAF PAGES: 34 NLEVELS: 2 UNIQUE: D DUPLICATE OK
2 OF 4 COLUMNS ARE MATCHED CLOSE: N LOCK MODE: IS BPOOL: BPO

KEY	COLUMN NAME	ORDER	TYPE	DIST	LEN	NULL	COLCARD	DIST#
1	DLOCATION	A	VARCHAR	N	128	N	-1	0
2	DCOLLID	A	VARCHAR	N	128	N	-1	0
3	DNAME	A	VARCHAR	N	128	N	-1	0
4	DCONTOKEN	A	CHAR	N	8	N	-1	0

```

+-----+
|ANL6052I *** NOTE:
|This index is specified as "Not Padded", allowing storage of a
|varying length index key. Padded indexes use blanks to fill out
|their fixed length keys and are not eligible for Index Only scan.
|"Not Padded" indexes do not blank fill CHAR and VARCHAR columns,
|allowing greater flexibility and better use of storage, packing
|more entries into a single Leaf page. Index Only access allowed.
+-----+

```

ALTERNATIVE INDEX

+++++

CREATOR: SYSIBM
IX NAME: DSNKDX02

VERS: 1 KEY LEN: -1 PADDED: N C-ED: N C-ING: N CLURATIO: 0.0000
FULLKEY CARD: -1 FIRSTKEY CARD: -1
TYPE: 2 NLEAF PAGES: -1 NLEVELS: -1 UNIQUE: D KEY COLS: 3
1 A -1 BQUALIFIER
2 A -1 BNAME
3 A -1 BTYPE

+-----+

```

|ANL6052I *** NOTE:
|This index is specified as "Not Padded", allowing storage of a
|varying length index key. Padded indexes use blanks to fill out
|their fixed length keys and are not eligible for Index Only scan.
|"Not Padded" indexes do not blank fill CHAR and VARCHAR columns,
|allowing greater flexibility and better use of storage, packing
|more entries into a single Leaf page. Index Only access allowed.
+-----+

```

ALTERNATIVE INDEX

```

+++++

```

```

CREATOR: SYSIBM

```

```

IX NAME: DSNKDX03

```

```

VERS: 1 KEY LEN: -1 PADDED: N C-ED: N C-ING: N CLURATIO: 0.0000
FULLKEY CARD: -1 FIRSTKEY CARD: -1
TYPE: 2 NLEAF PAGES: -1 NLEVELS: -1 UNIQUE: D KEY COLS: 4
  1 A -1 BQUALIFIER
  2 A -1 BNAME
  3 A -1 BTYPE
  4 A -1 DTYPE

```

```

|ANL6052I *** NOTE:
|This index is specified as "Not Padded", allowing storage of a
|varying length index key. Padded indexes use blanks to fill out
|their fixed length keys and are not eligible for Index Only scan.
|"Not Padded" indexes do not blank fill CHAR and VARCHAR columns,
|allowing greater flexibility and better use of storage, packing
|more entries into a single Leaf page. Index Only access allowed.
+-----+

```

```

|ANL6016I *** NOTE:
|Presently this query is only matching some of the indexed columns
|on this index key. Maximizing use of Stage 1 predicates against
|these index columns will improve performance, by matching and/or
|screening rows and therefore reducing data page I/O requirements.
+-----+

```

```

|ANL7034I *** GOOD NEWS:
|This index is Not Padded, making it a true variable length index.
|"Index Only" access is now possible and DB2 will compare CHAR and
|VARCHAR columns of unequal length against this index during Stage
|1 processing.
+-----+

```

```

|ANL7009I *** GUIDELINE:
|On composite (multi-column) indexes, favor matching index scan of
|as many columns as possible, using Equals (=), Range (>,<, =>,<=,
|Between, Like), In (List) and Null predicates, where a Range pred
|is the last matching predicate. Apply left to right against index
|columns, skipping no columns in the L-R sequence. DB2 may "screen"
|remaining Stage 1 preds against the rid list before data access.
+-----+

```

```

|ANL7010I *** GUIDELINE:
|

```

```
|On composite (multi-column) indexes, collect correlated key stats |
|on column pairings, as well as cardinality stats, like 2ndkeycard, |
|3rdkeycard, etc., with Runstats. These additional statistics will |
|allow DB2 to make better filter estimates for Equality and Range |
|and In List predicates, thereby selecting proper indexes and more |
|realistic estimates for rows returned from the query (QCARD). |
+-----+
```

```
+-----+
|ANL7024I *** GOOD NEWS: |
|Index Screening is now available for List Prefetch processing, |
|which allows Random indexes to "screen" additional predicates |
|after matching index scan, and before data access. |
+-----+
```

```
+-----+
|ANL6020I *** NOTE: |
|This index is presently designated as allowing "duplicate values". |
|Make sure this is a nonunique index that does not fall into one of |
|the following "unique" categories: explicitly unique, primary key, |
|non-primary RI parent key, unique where not null, unique column |
|constraint. Unique indexes have definite advantages whenever they |
|can be declared by the user, so be sure it contains duplicates. |
+-----+
```

THIS IS AN "INDEX ONLY" ACCESS: NO DATA PAGES ARE READ FROM THE TABLE

```
+-----+
|ANL7015I *** GUIDELINE: |
|Collecting Non-uniform column statistics greatly enhance the DB2 |
|Optimizer accuracy when processing Equals and In (List) preds. |
+-----+
```

THIS TABLE IS JOINED WITH THE PRIOR TABLE VIA THE "NESTED LOOP" METHOD.
THIS IS THE INNER TABLE IN THE NESTED LOOP JOIN.

THIS IS A CARTESIAN JOIN, ALL ROWS TO ALL ROWS, NO JOIN PREDICATE.

```
+-----+
|ANL6034I *** NOTE: |
|Only the Nested Loop join method supports the use of non-Equijoin |
|predicates (T1.C1 <= T2.C2). If you are using these predicates to |
|join tables together, you are limiting DB2 to selecting only NL. |
|Merge Scan and Hybrid require "Equijoin" preds (T1.C1 = T2.C2). |
+-----+
```

```
+-----+
|ANL7018I *** GUIDELINE: |
|If you wish to reverse the Nested Loop join table order, use some |
|redundant predicates (Tn.Cn = Tn.Cn) on the table you would like |
|to become the "outer" table in the process (reverse this logic for |
|Merge Scan joins, applying them to the "inner" table instead). |
+-----+
```

And finally, Example C-4 shows the new predicate analysis for the same query.

Example: C-4 SQL PA predicate analysis

QUERYNO: 100000001

```

QBLKNO:      1  PREDNO:      5  FILTER: 0.96000000  TYPE: EQUAL      JOIN PRED? N
STAGE 1? Y  BOOLEAN TERM? Y  INDEX KEYFIELD? N  REDUNDANT? N  AFTER JOIN? N
ADDED BY PTC? N  FOR NEGATION? Y  LITERALS: HV
LEFT SIDE --> COLLID                                TABNO:      1  BLOCKNO:      0  PREDNO:      0
RIGHT SIDE -> VALUE                                TABNO:      0  BLOCKNO:      0  PREDNO:      0
PSUEDO TEXT:
A.COLLID<=>(EXPR)

```

[illegible]

ANL5026W *** WARNING:
ESTIMATE OF 443 EXCEEDS "SERVICE UNIT" LIMIT OF 200 UNITS !

EXPLAIN and its tables

In this appendix we describe the DB2 EXPLAIN function and show the contents of the EXPLAIN tables:

- ▶ DSN_PLAN_TABLE
- ▶ DSN_STATEMNT_TABLE
- ▶ DSN_FUNCTION_TABLE

We have all the EXPLAIN tables described here, including contents that have not changed with V8, in order to make the material more complete and understandable.

D.1 DB2 EXPLAIN

EXPLAIN describes the access paths selected by DB2 to execute an SQL statement (UPDATE, SELECT, DELETE, and INSERT), obtains information about how SQL statements from a package or plan will execute, and inserts that information into the *package_owner.PLAN_TABLE* or *plan_owner.PLAN_TABLE*. For dynamically prepared SQL, the qualifier is the current SQLID. An EXPLAIN can be executed either statically from an application program, or dynamically, using QMF or SPUFI.

You can use EXPLAIN to:

- ▶ Help application program design
- ▶ Assist in database design
- ▶ Give a description of the access path chosen for a query by DB2
- ▶ Help you verify if your application needs to be rebound
- ▶ Check if index or table scan is used by a statement
- ▶ Check if DB2 plans are using parallelism

With DB2 V8 EXPLAIN has been enhanced to:

- ▶ EXPLAIN SQL in the dynamic statement cache (DSC)
- ▶ Use ALIAS with the EXPLAIN_TABLE

Before you start using EXPLAIN, you need to create a PLAN_TABLE to hold the results of the EXPLAIN. The EXPLAIN can use two other tables, FUNCTION_TABLE or STATEMENT_TABLE, but unless you need the information they provide, it is not necessary to create them to use EXPLAIN. A description of the contents of these tables follows.

With DB2 V8, the PLAN_TABLE has had 7 additional columns defined, we list them in Table D-2 on page 410. The contents of 2 columns have been changed in order store MQT related information:

- ▶ TNAME
The name of a table, *materialized query table*, created or declared temporary table, materialized view, table expression, or an intermediate result table for an outer join that is accessed in this step, blank if METHOD is 3.
- ▶ TABLE_TYPE
New type *M* for materialized query table

EXPLAIN informs you about the access paths DB2 chooses, and can tell you:

- ▶ The number of indexes used
- ▶ The I/O method used to read the pages (list prefetch or sequential prefetch)
- ▶ How many index columns are used as search criteria
- ▶ If an index access or table scan is used
- ▶ The join method
- ▶ The order in which the tables are joined
- ▶ When and why sorts are performed
- ▶ To determine if DB2 plans to use multiple concurrent I/O streams to access your data
- ▶ Select, Update and Delete statements

This information is saved in the PLAN_TABLE. This table has been evolving with each release of DB2 adding new columns in order to provide new functions. DB2 inserts rows into the PLAN_TABLE whenever a plan or a package is bound or rebound, or each time an explainable SQL statement is executed.

Note: If you want to empty the PLAN_TABLE, you must use the DELETE statement, just as you would for deleting rows from any table. You also can use the DROP TABLE statement to drop the PLAN_TABLE completely. The action of binding, rebinding or executing another SQL statement does not replace or delete rows from the PLAN_TABLE.

D.1.1 DSN_PLAN_TABLE

The table can be created by the statements contained in member name DSNTESC of the DB2 sample library. DB2 tools also create or update your PLAN_TABLE when needed.

Let us look at the evolution of the PLAN_TABLE in Table D-1. It is a way to see the evolution of DB2.

Table D-1 PLAN_TABLE columns by release

DB2 release	Columns in PLAN_TABLE
V1	25
V1R2	28
V2	30
V3	34
V4	43
V5	46
V6	49
V7	51
V8	58

Note: When you execute EXPLAIN using DB2 Performance Expert in V8 for the first time, DB2 PE advises you that your PLAN_TABLE is old and updates it for you.

Of course, the 58-column format gives you the most information. If you alter an existing PLAN_TABLE to add new columns, in general specify the columns as NOT NULL WITH DEFAULT, so that default values are included for the rows already in the table. However, as you can see in Table D-2, there are some exceptions, certain columns do allow nulls. Do *not* specify those columns as NOT NULL WITH DEFAULT.

Table D-2 describes the PLAN_TABLE, giving a small description of each column and a brief history of DB2 evolution. If you want more information about it, refer to *DB2 UDB for z/OS Version 8 Administration Guide*, SC18-7413.

Note: There are some objects for which accesses are not described by EXPLAIN. They are, for example, the access to LOB values, which are stored separately from the base table, and access to parent or dependent tables needed to enforce referential constraints, SQL for routines (triggers, functions, or stored procedures), and explicit access to SECLABELs for row level security.

Table D-2 *PLAN_TABLE* contents and brief history

Column	Type	Content
QUERYNO	INTEGER NOT NULL	A number intended to identify the statement being explained.
QBLOCKNO	SMALLINT NOT NULL	A number that identifies each query block within a query.
APPLNAME	CHAR(8) NOT NULL	The name of the application plan for the row.
PROGNAME	CHAR(8) NOT NULL	The name of the program or package containing the statement being explained.
PLANNO	SMALLINT NOT NULL	The number of the step in which the query indicated in QBLOCKNO was processed.
METHOD	SMALLINT NOT NULL	A number (0, 1, 2, 3, or 4) that indicates the join method used for the step: 0 = First table accessed, continuation of the previous table. 1= NESTED LOOP JOIN 2= MERGE SCAN JOIN 3 = Sorts needed by ORDER BY, GROUP BY, SELECT DISTINCT, UNION, 4= HYBRID JOIN
CREATOR	CHAR(8) NOT NULL	The creator of the new table accessed in this step, blank if METHOD is 3.
TNAME	CHAR(18) NOT NULL	The name of a table, materialized query table, created or declared temporary table, materialized view, or materialized table expression.
TABNO	SMALLINT NOT NULL	Values are for IBM use only.

Column	Type	Content
ACCESSTYPE	CHAR(2) NOT NULL	<p>The method of accessing the new table:</p> <p>I - index</p> <p>I1 - One-fetch index scan</p> <p>M - Multiple index scan (followed by MX, MI, or MU)</p> <p>MI - intersection of multiple indexes</p> <p>MU - union of multiple indexes</p> <p>MX - index scan on the index named in ACCESSNAME</p> <p>N - index scan when the matching predicate contains the IN keyword</p> <p>R - Table space scan</p> <p>RW - Work file scan of the result of a materialized user-defined table function</p> <p>T - Sparse index (star join work files)</p> <p>V - Buffers for an INSERT statement within a SELECT</p> <p>blank - Not applicable to the current row ACCESSTYPE</p>
MATCHCOLS	SMALLINT NOT NULL	For ACCESSTYPE I, I1, N, or MX, the number of index keys used in an index scan; otherwise, 0.
ACCESSCREATOR	CHAR(8) NOT NULL	For ACCESSTYPE I, I1, N, or MX, the creator of the index; otherwise, blank.
ACCESSNAME	CHAR (18) NOT NULL	For ACCESSTYPE I, I1, N, or MX, the name of the index; otherwise, blank.
INDEXONLY	CHAR(1) NOT NULL	Whether access to an index alone is enough to carry out the step, or whether data too must. Y = YES; N =NO
SORTN_UNIQ	CHAR(1) NOT NULL	Whether the new table is sorted to remove duplicate rows. Y=Yes; N=No.
SORTN_JOIN	CHAR(1) NOT NULL	Whether the new table is sorted for join method 2 or 4. Y=Yes; N=No.
SORTN_ORDERBY	CHAR(1) NOT NULL	Whether the new table is sorted for ORDER BY. Y=Yes; N=No.
SORTN_GROUPBY	CHAR(1) NOT NULL	Whether the new table is sorted for GROUP BY. Y=Yes; N=No.
SORTC_UNIQ	CHAR(1) NOT NULL	Whether the composite table is sorted to remove duplicate rows. Y=Yes; N=No.
SORTC_JOIN	CHAR(1) NOT NULL	Whether the composite table is sorted for join method 1, 2 or 4. Y=Yes; N=No.

Column	Type	Content
SORTC_ORDER BY	CHAR(1) NOT NULL	Whether the composite table is sorted for an ORDER BY clause or a quantified predicate. Y=Yes; N=No.
SORTC_GROUP BY	CHAR(1) NOT NULL	Whether the composite table is sorted for a GROUP BY clause. Y=Yes; N=No.
TSLOCKMODE	CHAR(3) NOT NULL	An indication of the mode of lock to be acquired on either the new table, or its table space or table space partitions. If the isolation can be determined at bind time.
TIMESTAMP	CHAR(16) NOT NULL	The time at which the row is processed, to the last .01 second. If necessary, DB2 adds .01 second to the value to ensure that rows for two successive queries have different values.
REMARKS	VARCHAR(254) NOT NULL	A field into which you can insert any character string of 762 or fewer characters.
----- 25 COLUMNS FORMAT -----Version 1 -1983--		-----25 COLUMNS FORMAT -----Version 1-1983-
PREFETCH	CHAR(1) NOT NULL WITH DEFAULT	S = pure sequential prefetch; L = prefetch through a page list; D = optimizer expects dynamic prefetch; blank = unknown or no prefetch.
COLUMN_FN_EVAL	CHAR(1) NOT NULL WITH DEFAULT	When an SQL aggregate function is evaluated. R = while the data is being read from the table or index; S = while performing a sort to satisfy a GROUP BY clause; blank = after data retrieval and after any sorts.
MIXOPSEQ	SMALLINT NOT NULL WITH DEFAULT	The sequence number of a step in a multiple index operation. 1,2,..., N For the steps of the multiple index procedure (ACCESSTYPE is MX, MI, or MU.) 0 For any other rows (ACCESSTYPE is I, I1, M, N, R, or blank.)
-----28 COLUMNS FORMAT ---Version 1 --1984---		-----28 COLUMNS FORMAT -----Version 1-1984---
VERSION	VARCHAR(64) NOT NULL WITH DEFAULT	The version identifier for the package.
COLLID	CHAR(18) NOT NULL WITH DEFAULT	The collection ID for the package.
----- 30 Columns Format -----Version 2 ---1988---		----- 30 columns format -----Version 2 ---1988-

Column	Type	Content
ACCESS_DEGREE	SMALLINT	The number of parallel tasks or operations activated by a query.
ACCESS_PGROUP_ID	SMALLINT	The identifier of the parallel group for accessing the new table.
JOIN_DEGREE	SMALLINT	The number of parallel operations or tasks used in joining the composite table with the new table.
JOIN_PGROUP_ID	SMALLINT	The identifier of the parallel group for joining the composite table with the new table.
- 34 Columns Format -Version 3 -Dec.1993-		- 34 Columns Format - Version 3 -Dec.1993-
SORTC_PGROUP_ID	SMALLINT	The parallel group identifier for the parallel sort of the composite table.
SORTN_PGROUP_ID	SMALLINT	The parallel group identifier for the parallel sort of the new table.
PARALLELISM_MODE	CHAR(1)	The kind of parallelism, if any, that is used at bind time: I Query I/O parallelism C Query CP parallelism X Sysplex query parallelism
MERGE_JOIN_COLS	SMALLINT	The number of columns that are joined during a merge scan join (Method=2).
CORRELATION_NAME	CHAR(18)	The correlation name of a table or view that is specified in the statement.
PAGE_RANGE	CHAR(1) NOT NULL WITH DEFAULT	The table qualifies for page range screening, so that plans scan only the partitions that are needed. Y = yes Blank =N
JOIN_TYPE	CHAR(1) NOT NULL WITH DEFAULT	-Type of join F, L, S, blank
GROUP_MEMBER	CHAR(8) NOT NULL WITH DEFAULT	member for EXPLAIN
IBM_SERVICE_DATA	VARCHAR(254) NULL WITH DEFAULT	IBM use only
-43 Columns Format - Version 4 - Nov. 1995-		- 43 Columns Format -Version 4 -Nov. 1995-
WHEN_OPTIMIZE	CHAR(1) NOT NULL WITH DEFAULT	-BIND, RUN
QBLOCK_TYPE	CHAR(6) NOT NULL WITH DEFAULT	For each query block, an indication of the type of SQL operation performed. SELECT, INSERT, UPDATE...
BIND_TIME	TIMESTAMP NULL WITH DEFAULT	The time at which the plan or package for this statement query block was bound.
- 46 Column Format - Version 5 - June 1997 -		- 46 Column Format - Version 5 - June 1997-

Column	Type	Content
OPTHINT	CHAR(8) NOT NULL WITH DEFAULT	A string that you use to identify this row as an optimization hint for DB2. DB2 uses this row as input when choosing an access path.
HINT_USED	CHAR(8) NOT NULL WITH DEFAULT	If DB2 used one of your optimization hints, it puts the identifier for that hint (the value in OPTHINT) in this column.
PRIMARY_ACCESSTYPE	CHAR(1) NOT NULL WITH DEFAULT	if direct row access.
- 49 Column Format -----Version 6 - June 1999		- 49 Column Format -Version 6 -June 1999-
PARENT_QBLOCKNO	SMALLINT NOT NULL WITH DEFAULT	A number that indicates the QBLOCKNO of the parent query block.
TABLE_TYPE	CHAR(1)	T table, W work, RB, Q, M, F, C, B...
----- 51 Column Format ----Version 7 --2001---		- 51 Column Format -Version 7-Mar. 2001-
TABLE_ENCODE	CHAR(1)	The encoding scheme of the table. If the table has a single CCSID set, possible values are: A ASCII E EBCDIC U Unicode M is the value of the column when the table contains multiple CCSID sets
TABLE_SCCSID	SMALLINT NOT NULL WITH DEFAULT	The SBCS (Single Byte Character Set) CCSID value of the table. If column TABLE_ENCODE is M, the value is 0.
TABLE_MCCSID	SMALLINT NOT NULL WITH DEFAULT	The mixed CCSID value of the table. Mixed and DBCS (Double Byte Character Set) CCSIDs are only available for a certain number of SBCS CCSIDs, namely, CCSIDs for Japanese, Korean, and Chinese. That is, for CCSIDS such as 273 for Germany, the mixed and DBCS CCSIDs do not exist.
TABLE_DCCSID	SMALLINT NOT NULL WITH DEFAULT	The DBCS (Double Byte Character Set) CCSID value of the table. If column TABLE_ENCODE is M, the value is 0.
ROUTINE_ID	INTEGER	Values are for IBM use only.
CTEREF	SMALLINT NOT NULL WITH DEFAULT	If the referenced table is a common table expression, the value is the top-level query block number.
STMTTOKEN	VARCHAR(240)	User specified statement token.

Column	Type	Content
-----58 Columns Format-----Version 8 ----2003-		-58 Column Format -Version 8 -Mar. 26, 2004-

D.1.2 DSN_STATEMNT_TABLE

EXPLAIN estimates the cost of executing an SQL SELECT, INSERT, UPDATE, or DELETE statement. The output appears in a table called DSN_STATEMNT_TABLE. The columns of this table and their contents are listed in Table D-3. For more information about statement tables, see “Estimating a statement’s cost” on the *DB2 UDB for z/OS Version 8 Administration Guide*, SC18-7413.

Table D-3 EXPLAIN enhancements in DSN_STATEMNT_TABLE

Column Name	Type	Content
QUERYNO	INTEGER NOT NULL WITH DEFAULT	A number that identifies the statement being explained.
APPLNAME	CHAR(8) NOT NULL WITH DEFAULT	The name of the application plan for the row, or blank.
PROGNAME	VARCHAR (128) NOT NULL WITH DEFAULT	The name of the program or package containing the statement being explained, or blank.
COLLID	VARCHAR (128) NOT NULL WITH DEFAULT	The collection ID for the package.
GROUP_MEMBER	CHAR (8) NOT NULL WITH DEFAULT	The member name of the DB2 that executed EXPLAIN, or blank.
EXPLAIN_TIME	TIMESTAMP	The time at which the statement is processed.
STMT_TYPE	CHAR (6) NOT NULL WITH DEFAULT	The type of statement being explained. SELECT, UPDATE,INSERT, or UPDATE...
COST_CATEGORY	CHAR (1) NOT NULL WITH DEFAULT	Indicates if DB2 was forced to use default values when making its estimates (B) or used statistics (A).
PROCMS	INTEGER NOT NULL WITH DEFAULT	The estimated processor cost, in milliseconds, for the SQL statement.
PROCSU	INTEGER NOT NULL WITH DEFAULT	The estimated processor cost, in service units, for the SQL statement.
REASON	VARCHAR (256) NOT NULL WITH DEFAULT	A string that indicates the reasons for putting an estimate into cost category B.
STMT_ENCODE	CHAR (1) NOT NULL WITH DEFAULT	Encoding scheme of the statement. A = ASCII E = EBCDIC U = Unicode

D.1.3 DSN_FUNCTION_TABLE

You can use DB2 EXPLAIN to obtain information about how DB2 resolves functions. DB2 stores the information in a table called DSN_FUNCTION_TABLE. DB2 inserts a row in DSN_FUNCTION_TABLE for each function that is referenced in an SQL statement when one of the following events occurs:

- ▶ You execute the SQL EXPLAIN statement on an SQL statement that contains user-defined function invocations
- ▶ You run a program whose plan is bound with EXPLAIN(YES), and the program executes an SQL statement that contains user-defined function invocations

Before you use EXPLAIN to obtain information about function resolution, you need to create the DSN_FUNCTION_TABLE. The contents are listed in Table D-4.

Table D-4 DSN_FUNCTION_TABLE extensions

Column	Type	Content
QUERYNO	INTEGER	A number that identifies the statement being explained.
QBLOCKNO	INTEGER	A number that identifies each query block within a query.
APPLNAME	VARCHAR (128)	The name of the application plan for the row.
PROGNAME	VARCHAR (128)	The name of the program or package containing the statement being explained.
COLLID	CHAR (8)	The collection ID for the package.
GROUP_MEMBER	CHAR (8)	The member name of the DB2 that executed EXPLAIN, or blank.
EXPLAIN_TIME	TIMESTAMP	Timestamp when the EXPLAIN statement was executed.
SCHEMA_NAME	VARCHAR(128) NOT NULL WITH DEFAULT	Schema name of the function that is invoked in the explained statement.
FUNCTION_NAME	VARCHAR(128) NOT NULL WITH DEFAULT	Name of the function that is invoked in the explained statement.
SPEC_FUNC_NAME	VARCHAR(128) NOT NULL WITH DEFAULT	Specific name of the function that is invoked in the explained statement.
FUNCTION_TYPE	CHAR(2) NOT NULL WITH DEFAULT	The type of function that is invoked in the explained statement. Possible values are: SU - Scalar function, TU - Table function
VIEW_CREATOR	VARCHAR(128) NOT NULL WITH DEFAULT	The creator of the view, if the function that is specified in the FUNCTION_NAME column is referenced in a view definition. Otherwise, this field is blank.

Column	Type	Content
VIEW_NAME	VARCHAR(128) NOT NULL WITH DEFAULT	The name of the view, if the function that is specified in the FUNCTION_NAME column is referenced in a view definition. Otherwise, this field is blank.
PATH	VARCHAR(2048) NOT NULL WITH DEFAULT	The value of the SQL path when DB2 resolved the function reference.
FUNCTION_TEXT	VARCHAR(1500) NOT NULL WITH DEFAULT	The text of the function reference (the function name and parameters).

D.1.4 DSN_STATEMENT_CACHE_TABLE

This new table was recently introduced by PTF UQ89372 for APAR PQ88073. With this enhancement a new keyword ALL is added to EXPLAIN STMTCACHE and a new explain table DSN_STATEMENT_CACHE_TABLE is created to hold the output of IFCID 316 and IFCID 318.

A brief description follows, as reported in the APAR text.

There are two different sets of information that can be collected from the SQL statements in the dynamic statement cache. Specifying STMTCACHE with the STMTID or STMTOKEN keywords causes the traditional access path information to be written to the PLAN_TABLE for the associated SQL statement as well as a single row written to DSN_STATEMENT_CACHE_TABLE if it exists.

However, specifying STMTCACHE with the new ALL keyword causes information to be written to only DSN_STATEMENT_CACHE_TABLE. It consists of one row per SQL statement in the dynamic statement cache for which the current authorization ID is authorized to execute.

The contents of these rows show identifying information about the cache entries, as well as an accumulation of statistics reflecting the executions of the statements by all processes that have executed the statement.

This information is nearly identical to the information returned from the IFI monitor READS API for IFCIDs 0316 and 0317.

Note that the collection and reset of the statistics in these records is controlled by starting and stopping IFCID 318. For more details, see "Controlling collection of dynamic statement cache statistics with IFCID 0318" in *DB2 UDB for z/OS Version 8 Administration Guide*, SC18-7413-01, Appendix E - Programming for the Instrumentation Facility Interface (IFI).

The new EXPLAIN option is:

```
EXPLAIN STMTCACHE ALL;
```

SQLCODE -20248 is issued if no statement is found in the cache for the auth ID that is used.

```
-20248 ATTEMPTED TO EXPLAIN A CACHED STATEMENT WITH
STMTID, STMTOKEN ID-token, OR ALL BUT THE REQUIRED
EXPLAIN INFORMATION IS NOT ACCESSIBLE.
```

Before you can use EXPLAIN STMTCACHE ALL, Statement Cache must be turned on. You must also create the table DSN_STATEMENT_CACHE_TABLE to hold the results of EXPLAIN STMTCACHE ALL. The DDL is shown in Example D-1.

Example: D-1 Creating DSN_STATEMENT_CACHE_TABLE

```
CREATE DATABASE DSNSTMT;

CREATE TABLESPACE DSNSTMTS IN DSNSTMT;

CREATE LOB TABLESPACE DSNLOBTS IN DSNSTMT
BUFFERPOOL BP32K1;

CREATE TABLE DSN_STATEMENT_CACHE_TABLE
(
  STMT_ID          INTEGER          NOT NULL,
  STMT_TOKEN       VARCHAR(240)    ,
  COLLID          VARCHAR(128) NOT NULL,
  PROGRAM_NAME     VARCHAR(128) NOT NULL,
  INV_DROPALT      CHAR(1)         NOT NULL,
  INV_REVOKE       CHAR(1)         NOT NULL,
  INV_LRU          CHAR(1)         NOT NULL,
  INV_RUNSTATS     CHAR(1)         NOT NULL,
  CACHED_TS        TIMESTAMP       NOT NULL,
  USERS            INTEGER          NOT NULL,
  COPIES           INTEGER          NOT NULL,
  LINES            INTEGER          NOT NULL,
  PRIMAUTH         VARCHAR(128) NOT NULL,
  CURSQLID         VARCHAR(128) NOT NULL,
  BIND_QUALIFIER   VARCHAR(128) NOT NULL,
  BIND_ISO         CHAR(2)         NOT NULL,
  BIND_CDATA       CHAR(1)         NOT NULL,
  BIND_DYNRL       CHAR(1)         NOT NULL,
  BIND_DEGRE       CHAR(1)         NOT NULL,
  BIND_SQLRL       CHAR(1)         NOT NULL,
  BIND_CHOLD       CHAR(1)         NOT NULL,
  STAT_TS          TIMESTAMP       NOT NULL,
  STAT_EXEC        INTEGER          NOT NULL,
  STAT_GPAG        INTEGER          NOT NULL,
  STAT_SYNR        INTEGER          NOT NULL,
  STAT_WRIT        INTEGER          NOT NULL,
  STAT_EROW        INTEGER          NOT NULL,
  STAT_PROW        INTEGER          NOT NULL,
  STAT_SORT        INTEGER          NOT NULL,
  STAT_INDX        INTEGER          NOT NULL,
  STAT_RSCN        INTEGER          NOT NULL,
  STAT_PGRP        INTEGER          NOT NULL,
  STAT_ELAP        FLOAT           NOT NULL,
  STAT_CPU         FLOAT           NOT NULL,
  STAT_SUS_SYNIO   FLOAT           NOT NULL,
  STAT_SUS_LOCK    FLOAT           NOT NULL,
  STAT_SUS_SWIT    FLOAT           NOT NULL,
  STAT_SUS_GLCK    FLOAT           NOT NULL,
  STAT_SUS_OTH     FLOAT           NOT NULL,
  STAT_SUS_OTHW    FLOAT           NOT NULL,
  STAT_RIDLIMT     INTEGER          NOT NULL,
  STAT_RIDSTOR     INTEGER          NOT NULL,
  EXPLAIN_TS       TIMESTAMP       NOT NULL,
  SCHEMA           VARCHAR(128) NOT NULL,
  STMT_TEXT        CLOB(2M)        NOT NULL,
  STMT_ROWID       ROWID           NOT NULL  GENERATED ALWAYS
)
IN DSNSTMT.DSNSTMTS CCSID EBCDIC;
```

```

CREATE TYPE 2 INDEX DSN_STATEMENT_CACHE_IDX1
ON DSN_STATEMENT_CACHE_TABLE (STMT_ID ASC) ;

CREATE TYPE 2 INDEX DSN_STATEMENT_CACHE_IDX2
ON DSN_STATEMENT_CACHE_TABLE (STMT_TOKEN ASC)
CLUSTER;

CREATE TYPE 2 INDEX DSN_STATEMENT_CACHE_IDX3
ON DSN_STATEMENT_CACHE_TABLE (EXPLAIN_TS DESC) ;

CREATE AUX TABLE DSN_STATEMENT_CACHE_AUX IN
DSNSTMTC.DSNLOBTS
STORES DSN_STATEMENT_CACHE_TABLE COLUMN STMT_TEXT;

CREATE TYPE 2 INDEX DSN_STATEMENT_CACHE_AUXINX
ON DSN_STATEMENT_CACHE_AUX;

```

The contents of this new EXPLAIN table are described in Table D-5.

Table D-5 Contents of DSN_STATEMENT_CACHE_TABLE

Column	Type	Content
STMT_ID		Statement ID, EDM unique token
STMT_TOKEN		Statement token. User-provided identification string
COLLID		Collection id value is DSNDYNAMICSQLCACHE
PROGRAM_NAME		Program name, Name of package or DBRM that performed the initial PREPARE
INV_DROPALT		Invalidated by DROP/ALTER
INV_REVOKE		Invalidated by REVOKE
INV_LRU		Removed from cache by LRU
INV_RUNSTATS		Invalidated by RUNSTATS
CACHED_TS		Timestamp when statement was cached
USERS		Number of current users of statement. These are the users that have prepared or executed the statement during their current unit of work.
COPIES		Number of copies of the statement owned by all threads in the system
LINES		Precompiler line number from the initial PREPARE
PRIMAUTH		User ID - Primary authorization ID of the user that did the initial PREPARE
CURSQLID		CURRENT SQLID of the user that did the initial prepare

Column	Type	Content
BIND_QUALIFIER		Bind Qualifier, object qualifier for unqualified table names
BIND_ISO		ISOLATION BIND option: 'UR' - Uncommitted Read 'CS' - Cursor Stability 'RS' - Read Stability 'RR' - Repeatable Read
BIND_CDATA		CURRENTDATA BIND option: 'Y' - CURRENTDATA(YES) 'N' - CURRENTDATA(NO)
BIND_DYNRL		DYNAMICRULES BIND option: 'B' - DYNAMICRULES(BIND) 'R' - DYNAMICRULES(RUN)
BIND_DEGRE		CURRENT DEGREE value: 'A' - CURRENT DEGREE = ANY '1' - CURRENT DEGREE = 1
BIND_SQLRL		CURRENT RULES value: 'D' - CURRENT RULES = DB2 'S' - CURRENT RULES = SQL
BIND_CHOLD		Cursor WITH HOLD bind option 'Y' - Initial PREPARE was done for a cursor WITH HOLD 'N' - Initial PREPARE was not done for a cursor WITH HOLD
STAT_TS		Timestamp of stats when IFCID 318 is started
STAT_EXEC		Number of executions of statement For a cursor statement, this is the number of OPENS
STAT_GPAG		Number of getpage operations performed for statement
STAT_SYNR		Number of synchronous buffer reads performed for statement
STAT_WRIT		Number of buffer write operations performed for statement
STAT_EROW		Number of rows examined for statement
STAT_PROW		Number of rows processed for statement
STAT_SORT		Number of sorts performed for statement
STAT_INDX		Number of index scans performed for statement
STAT_RSCN		Number of table space scans performed for statement

Column	Type	Content
STAT_PGRP		Number of parallel groups created for statement
STAT_ELAP		Accumulated elapsed time used for statement
STAT_CPU		Accumulated CPU time used for statement
STAT_SUS_SYNIO		Accumulated wait time for synchronous I/O
STAT_SUS_LOCK		Accumulated wait time for lock and latch request
STAT_SUS_SWIT		Accumulated wait time for synchronous execution unit switch
STAT_SUS_GLCK		Accumulated wait time for global locks
STAT_SUS_OTHM		Accumulated wait time for read activity done by another thread
STAT_SUS_OTHW		Accumulated wait time for write activity done by another thread
STAT_RIDLMT		Number of times a RID list was not used because the number of RIDs would have exceeded one or more DB2 limits
STAT_RIDSTOR		Number of times a RID list was not used because not enough storage was available to hold the list of RIDs
EXPLAIN_TS		When statement cache table is populated
SCHEMA		CURRENT SCHEMA value
STMT_TEXT		Statement text
STMT_ROWID		Statement ROWID

Archived

Abbreviations and acronyms

ACS	automatic class selection	DES	Data Encryption Standard
AIX	Advanced Interactive eXecutive from IBM	DLL	dynamic load library manipulation language
APAR	authorized program analysis report	DML	data manipulation language
ARM	automatic restart manager	DNS	domain name server
ASCII	American National Standard Code for Information Interchange	DRDA	distributed relational database architecture
BCRS	business continuity recovery services	DSC	dynamic statement cache, local or global
BLOB	binary large objects	DTT	declared temporary tables
BPA	buffer pool analysis	DWDM	dense wavelength division multiplexer
BCDS	DFSMSHsm backup control data set	DWT	deferred write threshold
BSDS	boot strap data set	EA	extended addressability
CCA	channel connection address	EBCDIC	extended binary coded decimal interchange code
CCA	client configuration assistant	ECS	enhanced catalog sharing
CCP	collect CPU parallel	ECSA	extended common storage area
CCSID	coded character set identifier	EDM	environment descriptor management
CD	compact disk	ELB	extended long busy
CEC	central electronics complex	ERP	enterprise resource planning
CF	coupling facility	ERP	error recovery procedure
CFCC	coupling facility control code	ESA	Enterprise Systems Architecture
CFRM	coupling facility resource management	ESP	Enterprise Solution Package
CLI	call level interface	ESS	Enterprise Storage Server
CLP	command line processor	ETR	external throughput rate, an elapsed time measure, focuses on system capacity
CPU	central processing unit	EWLM	enterprise workload manager
CRCR	conditional restart control record	FIFO	first in first out
CRD	collect report data	FTD	functional track directory
CSA	common storage area	FLA	fast log apply
CTT	created temporary table	FTP	File Transfer Program
DASD	direct access storage device	GB	gigabyte (1,073,741,824 bytes)
DB2 PE	DB2 Performance Expert	GBP	group buffer pool
DBAT	database access thread	GRS	global resource serialization
DBD	database descriptor	GUI	graphical user interface
DBID	database identifier	HPJ	high performance Java
DBRM	database request module	I/O	input/output
DCL	data control language	IBM	International Business Machines Corporation
DDCS	distributed database connection services		
DDF	distributed data facility		
DDL	data definition language		

ICF	integrated catalog facility	PDS	partitioned data set
ICF	integrated coupling facility	PIB	parallel index build
ICMF	integrated coupling migration facility	PPRC	Peer-to-Peer Remote Copy
CSF	Integrated Cryptographic Service Facility	PSID	pageset identifier
IFCID	instrumentation facility component identifier	PSP	preventive service planning
IFI	instrumentation facility interface	PTF	program temporary fix
IFI	Instrumentation Facility Interface	PUNC	possibly uncommitted
IGS	IBM Global Services	PWH	Performance Warehouse
IPLA	IBM Program Licence Agreement	QA	Quality Assurance
IRLM	internal resource lock manager	QMF	Query Management Facility
ISPF	interactive system productivity facility	RACF	Resource Access Control Facility
IRWW	IBM Relational Warehouse Workload	RBA	relative byte address
ISV	independent software vendor	RBLP	recovery base log point
IT	Information Technology	RECFM	record format
ITR	internal throughput rate, a processor time measure, focuses on processor capacity	RID	record identifier
ITSO	International Technical Support Organization	RR	repeatable read
IVP	installation verification process	RRS	resource recovery services
JDBC	Java Database Connectivity	RRSAF	resource recovery services attach facility
JFS	journaled file systems	RPO	recovery point objective
JNDI	Java Naming and Directory Interface	RS	read stability
JVM	Java Virtual Machine	RTO	recovery time objective
KB	kilobyte (1,024 bytes)	SCUBA	self contained underwater breathing apparatus
LOB	large object	SDM	System Data Mover
LPAR	logical partition	SMIT	System Management Interface Tool
LPL	logical page list	SPL	selective partition locking
LRECL	logical record length	SU	Service Unit
LRSN	log record sequence number	UOW	unit of work
LRU	least recently used	XRC	eXtended Remote Copy
LUW	logical unit of work	WLM	workload manager
LVM	logical volume manager	WTO	write to operator
MB	megabyte (1,048,576 bytes)		
NPI	non-partitioning index		
NVS	non volatile storage		
ODB	object descriptor in DBD		
ODBC	Open Data Base Connectivity		
OP	Online performance		
OS/390	Operating System/390®		
PAV	parallel access volume		

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 427. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *DB2 UDB for z/OS Version 8: Everything You Ever Wanted to Know, ... and More*, SG24-6079
- ▶ *Disaster Recovery with DB2 UDB for z/OS*, SG24-6370
- ▶ *DB2 Performance Expert for z/OS Version 2*, SG24-6867-01
- ▶ *A Deep Blue View of DB2 Performance: IBM Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS*, SG24-7224
- ▶ *DB2 UDB for z/OS Version 8 Technical Preview*, SG24-6871
- ▶ *DB2 for z/OS and OS/390 Version 7 Performance Topics*, SG24-6129
- ▶ *DB2 for z/OS and OS/390 Version 7 Selected Performance Topics*, SG24-6894
- ▶ *DB2 for z/OS and OS/390: Ready for Java*, SG24-6435
- ▶ *Distributed Functions of DB2 for z/OS and OS/390*, SG24-6952
- ▶ *z/OS Multilevel Security and DB2 Row-level Security Revealed*, SG24-6480
- ▶ *Ready for e-business: OS/390 Security Server Enhancements*, SG24-5158
- ▶ *XML for DB2 Information Integration*, SG24-6994
- ▶ *IBM Enterprise Workload Manager*, SG24-6350
- ▶ *IBM eServer zSeries 990 (z990) Cryptography Implementation*, SG24-7070
- ▶ *DB2 for MVS/ESA Version 4 Data Sharing Performance Topics*, SG24-4611
- ▶ *DB2 for z/OS: Data Sharing in a Nutshell*, SG24-7322
- ▶ *How does the MIDAW facility improve the performance of FICON channels using DB2 and other workloads?*, REDP4201
- ▶ *Disk storage access with DB2 for z/OS*, REDP4187

The following publications, not referenced in this book, provide other up-to-date information on related topics:

- ▶ *Data Integrity with DB2 for z/OS*, SG24-7111
- ▶ *DB2 UDB for z/OS: Design Guidelines for High Performance and Availability*, SG24-7134
- ▶ *The Business Value of DB2 UDB for z/OS*, SG24-6763
- ▶ *DB2 for z/OS Stored Procedures: Through the CALL and Beyond*, SG24-7083
- ▶ *DB2 for z/OS and WebSphere: The Perfect Couple*, SG24-6319
- ▶ *Large Objects with DB2 for z/OS and OS/390*, SG24-6571

- ▶ *DB2 UDB for z/OS V8: Through the Looking Glass and What SAP Found There*, SG24-7088
- ▶ *Moving Data Across the DB2 Family*, SG24-6905

Other publications

These recently updated publications are also relevant as further information sources:

- ▶ *z/Architecture Principles of Operation*, SA22-7832
- ▶ *DB2 UDB for z/OS Version 8 Administration Guide*, SC18-7413-01
- ▶ *DB2 UDB for z/OS Version 8 Application Programming and SQL Guide*, SC18-7415-01
- ▶ *DB2 UDB for z/OS Version 8 Application Programming Guide and Reference for Java*, SC18-7414-01
- ▶ *DB2 UDB for z/OS Version 8 Command Reference*, SC18-7416-01
- ▶ *DB2 UDB for z/OS Version 8 Data Sharing: Planning and Administration*, SC18-7417-01
- ▶ *DB2 UDB for z/OS Version 8 Installation Guide*, GC18-7418-02
- ▶ *DB2 UDB for z/OS Version 8 Messages and Codes*, GC18-7422-01
- ▶ *DB2 UDB ODBC Guide and Reference*, SC18-7423-01
- ▶ *DB2 UDB for z/OS Version 8 RACF Access Control Module Guide*, SC18-7433-01
- ▶ *DB2 UDB for z/OS Version 8 Release Planning Guide*, SC18-7425-01
- ▶ *DB2 UDB for z/OS Version 8 SQL Reference*, SC18-7426-01
- ▶ *DB2 UDB for z/OS Version 8 Utility Guide and Reference*, SC18-7427-01
- ▶ *DB2 UDB for z/OS Version 8 Program Directory*, GI10-8566-02
- ▶ *z/OS Planning for Multilevel Security and Common Criteria*, SA22-7509
- ▶ Article *Unicode Performance in DB2 for z/OS*, by Jeffrey Berger and Kyoko Amano, published in the IDUG Solutions Journal Volume 11 Number 2
- ▶ Article *Using 64-bit Virtual Storage and Large Real Storage to Improve Performance and Availability in DB2 for z/OS*, by Jeffrey Berger, The IDUG Solutions Journal Volume 11 Number 3
- ▶ Article *DB2 UDB for OS/390 Storage Management* by John Campbell and Mary Petras, The IDUG Solutions Journal Spring 2000 - Volume 7, Number 1.
- ▶ The manual *DB2 UDB Internationalization Guide* is available from the DB2 for z/OS Web site:

<http://www.ibm.com/software/data/db2/zos/v8books.html>

Online resources

These Web sites and URLs are also relevant as further information sources:

- ▶ DB2 for z/OS :
<http://ibm.com/software/db2zos/>
- ▶ The CFSIZER can be found at:
<http://www.ibm.com/servers/eserver/zseries/cfsizer/>
- ▶ DB2 Data Management Tools and DB2 for z/OS V8.1 Compatibility:

http://www.ibm.com/support/docview.wss?rs=434&context=SSZJXP&uid=swg21162152&loc=en_US&cs=utf-8&lang=en+en

- ▶ The CF levels are described at:
<http://www.ibm.com/servers/eserver/zseries/pso/coupling.html>
- ▶ DB2 Estimator:
<http://www.ibm.com/software/data/db2/zos/estimate/>

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Archived

Index

Numerics

00D70027 229
16 exabytes 3, 142
-20248 417
4096 partitions 144
64-bit
 Addressability 13, 142, 144, 157
 Virtual storage xxv, 2, 18, 127
65041 open data sets 5, 128, 213
767 5
-952 307

A

AA12005 313
Accounting enhancements 208
ACCUMACC 26, 365
Additional high water mark counters 205
Advisory reorg pending 230
Agent level workfile tracing 211
ALTER BUFFERPOOL 25, 169, 339, 345
ALTER GROUPBUFFERPOOL 25, 345
ALTER INDEX
 NOT PADDED 242
 PADDED 242
ALTER TABLE
 ADD MATERIALIZED QUERY 41
 ROTATE PARTITION FIRST TO LAST 233
ALTER TABLE ADD PARTITION 233
Application Program Interface 31
application requestor 287
application server 285
Architecture xxv, 2, 9, 12–13, 16, 130, 281, 284–285
AREO* 230, 235
Array fetch 285, 292
Array input 285
AS 40, 84
AS SECURITY LABEL 193
ASENSITIVE 94–96, 99, 103
Assign 147, 194, 258
ATOMIC 53, 87
Audit record 212
AUTHCACH 26
Automatic LPL recovery 257, 337
Automatic query rewrite 39–41, 43, 46–47, 52
automatic space management 5, 229
Automatic summary tables 39
Availability enhancements
 Other 217
AVG 109

B

BACKUP 4, 6, 26, 217, 219, 221–223, 243, 261
BACKUP SYSTEM 217–218, 220, 223–226

DATA ONLY 223–224
FULL 223

Backward index scan 7, 107
Batched index page splits 320
BLKSIZE 25
BOTH 274
BOTH, 74
BPSIZE 142
BSDS 6, 219–220, 358
Buffer pool
 Control blocks 17, 19, 141–142, 159, 162, 164, 394
Buffer pools 141–142
BUILD2 4, 248, 255
 Elimination of 248
Built-in functions 188–189
 DECRYPT_BIN 188
 DECRYPT_CHAR 188
 ENCRYPT 188–189
 ENCRYPT_TDES 188
 GETHINT 188
Built-in security labels 193
Business Information Warehouse 6
Byte addressable 142

C

CACHE 25–26, 29, 53, 59, 65, 129, 143, 147, 153, 158, 165, 198, 201, 336, 351, 366, 394–395
Cached statement ID 210–211
CACHEDYN 25
CARDF 276
CARDINALITY 7, 30, 41, 73–74, 83, 85–86, 114–115, 124, 273, 275, 277, 402, 405
CARDINALITY MULTIPLIER 83
Cardinality values 74, 273
CASTOUT 3, 12, 18, 20, 25, 142, 153, 158–160, 322–323, 333, 337, 339, 346, 394–396
Castout owner 323
CASTOUT processing 22, 138, 319, 321, 323, 334–335
CATEGORY 114, 117, 192, 194, 310, 415
CATMAINT 351, 356–357
CATMAINT UPDATE 351, 356
CCSID 174–177, 179–181, 183–184, 304, 343
 1200 177
 1208 177, 179
 UNICODE 175, 342
CCSID set 414
CF 324, 336, 339, 388
CF level 12 22, 319, 321
CF lock propagation reduction 8
CF request batching 22, 26, 132, 135, 138, 319, 321–325
Change Log Inventory utility 283–284
changeable parameters 256
Changing partition boundaries 230
character conversion 342

- character set 174
- CHECK DATA 262
- CHECK INDEX 5, 133, 240, 242, 249, 253, 262–267
- CHECK INDEX SHRLEVEL(REFERENCE) PART 265
- Child L-lock propagation 331
- CHKFR 26
- CI size 6, 199, 202
- CICS 130, 160
- CISIZE 199
- CLASSPATH 293–294
- CLASST 25, 346
- clear key 189
- CLOB 307, 309
- CLOSE NO 320
- close processing for pseudo close 320
- CLUSTER 4, 8, 115, 199, 401, 403
- cluster table 246
- Cluster tables 8
- Clustering 400, 403
- Clustering index 3–4, 247
 - Secondary index 247
- Clusters 8
- CMTSTAT 26
- Code page 22, 174, 177
- COLGROUP 73–74, 274, 276
- Collating sequence 68, 247
- Column type change 4
- COMMENT 205, 347
- Common table expression and recursive SQL 6
- Common table expressions 259
 - Recursive SQL 6
- Communications database 283
- Compatibility mode 8, 13, 133–136, 286, 343, 354
- compatibility mode
 - functions available 18
- Compression dictionaries 141
- Compression dictionary 6, 144, 158, 160, 394, 396
- Concurrent copy 6, 200, 263, 266
- CONDBAT 25, 206
- CONNECT 182, 206, 283, 285, 351
- connection concentration 299
- CONTOKEN 401
- Control Center 314
- CONTSTOR 145–146
- Conversion
 - Remote applications 176
- Coprocessor 188, 190
- COPY 112, 133, 142, 200, 202, 218, 220, 226, 263, 266, 359, 364, 371
- Copy pool 218–222, 225
- COPYPOOL 219, 223–224
- COUNT 30, 40–41, 46–47, 50–51, 69, 72–73, 108–109, 158, 164, 189, 206, 254–255, 274, 276, 279, 360, 395, 400
- Coupling Facility 257, 319–325, 327–332, 336, 339
- CPACF 188
- Create index dynamic statement invalidation 5
- CSA 158, 162, 172, 396
- CSV 270
- CT 143

- CTHREAD 25, 162, 205–206
- CURRENT MAINT TYPES 42, 352
- CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION 124, 352
- CURRENT REFRESH AGE 42, 44, 124, 352
- CURRENT SQLID 212, 259, 408
- CYCLE 223, 226, 354

D

- Data caching 6, 65
- Data Encryption for IMS and DB2 Databases 369
- Data Encryption Tool 189
- DATA INITIALLY DEFERRED 40–41
- Data Manager 158, 394–395
- data set extend 226
- data sharing 319
- Data sharing locking 319, 325–326, 328, 335
- Data source property 179, 188
- data space 141–143, 148, 158–159, 164, 394
- Data space buffer pool 141, 159
- Database name 283
- Data-partitioned secondary index 4, 248, 274
- Data-partitioned secondary indexes 4
- DATE 53, 122, 158, 175, 206, 272, 395
- DB2 Administration Tools 230
- DB2 Archive Log Accelerator 367
- DB2 authorization 260
- DB2 Bind Manager 370
- DB2 catalog changes 11, 23, 344
- DB2 CLI 285
- DB2 Connect 177–179, 187, 281, 284–285, 290, 297, 299
- DB2 Connect and DRDA 3
- DB2 Connect Personal Edition 112
- DB2 Data Archive Expert 370
- DB2 Estimator 364
- DB2 High Performance Unload 370
- DB2 performance consulting offering 167
- DB2 Query Monitor 368
- DB2 restart light 337
- DB2 SQL Performance Analyzer for z/OS 369
- DB2 tools 363
- DB2 Universal driver 292, 294–295, 303–304, 306
- DB2 Universal Driver for SQLJ and JDBC 281, 285, 293–294
- DB2 Utilities Suite 370
- DB2 V8 APARs 378, 386
- Dbalias 283
- DBCS 174, 180, 186, 342, 414
- DBD 20, 23, 25, 135, 142, 158, 243, 351, 396
- DBD01 220
- DBM1 STORAGE STATISTICS 163
- DBM1 Storage Statistics 158
- DBM1 virtual storage constraint 129
- DBRM 23, 113, 206, 361
- DDF 8, 21, 25–26, 94, 208–209, 281, 283–285, 295, 338, 351–352
- Declared temporary table 91, 408, 410
- Delayed index availability 231
- DELIMITED 371

- UNLOAD 371
- DES 188–189
- DFSMS 201, 217–218, 221, 226, 229
- DFSMSshm 4, 27, 218–220, 224
- DFSORT 26, 263, 274
- DIS GROUP 354
- DISABLE QUERY OPTIMIZATION 40, 53
- Disaster recovery 4, 221
- Disjoint 194
- DIST address space 152, 176–177, 290–291
- Distribution statistics 7, 72, 76, 273–278, 280
- DNAME 403
- Dominance 193–194, 196
- Dominate 194–195, 197
- Double Byte Character Set 174, 342, 414
- DPSI 92, 247–256
 - BUILD2 250–251
 - CHECK INDEX 252
 - Data sharing 255
 - LOAD 248
- DRAIN ALL 264, 337
- DRDA 3, 15, 18, 21–22, 130, 132, 136, 177, 179, 181, 206, 208, 280–281, 284–290, 293, 295–296, 303, 354
- DSC 7, 152
- DSC statement ID in Explain 7
- DSMAX 26, 162, 213, 338
- DSN6ARV 25
- DSN6FAC 25–26
- DSN6SPRM 25–26
- DSN6SYSP 25–26, 352
- DSN8EXP 259
- DSNAIMS 384
- DSNB250E 337
- DSNB406I 169
- DSNB508I 153
- DSNB536I 153, 157
- DSNB543I 169
- DSNB610I 153, 157
- DSNDB06.SYSDDF 345
- DSNDB06.SYSSTR 345
- DSNHDECP 175, 350–351, 353
- DSNHMCID 350
- DSNI031I 207
- DSNJ031I 258
- DSNJU003 220
- DSNR031I 207, 256
- DSNR035I 258
- DSNTEJ3M 123
- DSNTEJXP 259
- DSNTEP2 346
- DSNTEP4 29, 136, 346–347
 - MULT_FETCH 347
- DSNTESQ 359, 362
- DSNTIAUL 136, 189, 346, 348–350, 371, 374–375
- DSNTIJNE 353, 356, 358–359
- DSNTIJNF 353
- DSNTIJNH 353, 358
- DSNTIJTC 351
- DSNTIP5 25
- DSNTIP6 42

- DSNTIP7 25, 351
- DSNTIP8 352
- DSNTIPC 25, 351
- DSNTIPE 25, 351
- DSNTIPF 351
- DSNTIPN 26, 351
- DSNTIPP 26, 351
- DSNTIPR 26, 351
- DSNU1129I 245
- DSNUTILB 263
- DSNUTILS 111, 117
- DSNUTILS stored procedure 111
- DSNWZP 111
- DSNX@XAC 195
- DSNZPARM 9, 14, 17, 20, 42, 64–65, 111, 128, 136, 138, 143–144, 147, 160, 173, 199, 205, 207, 209, 213, 220, 225–227, 232, 246, 256–257, 332, 395
 - CACHEDYN 20, 143
 - CMTSTAT 209
 - NPGETHRSH 246
 - UIFICIDS 205
- DSNZPARM default values 24–25
- DSSIZE 4, 229
- DSTATS 275–278, 280
- DSVCI 199
- DTT 91, 93
- DTYPE 404
- DWQT 25, 345
- Dynamic allocations 213
- Dynamic scrollable cursor
 - Example 107
- Dynamic scrollable cursors 91, 94–95, 97, 107
 - Considerations 95
 - Locking 94
 - UPDATE 94
- Dynamic statement cache 3, 5, 7, 12, 20, 143, 146, 148, 152, 160, 211, 236, 304, 352, 394–396
- DYNAMICRULES 259

E

- ECSA 21, 127, 162, 172–174, 396
- EDITPROC 189
- EDM pool 3, 16, 20, 25, 139, 141–143, 148–149, 153, 158–159, 162, 209, 331, 352, 366, 394, 396
- EDMDBDC 25, 143
- EDMDSPAC 143
- EDMPOOL 25, 143, 160, 351
- EDMSTMTC 25, 143
- ENABLE QUERY OPTIMIZATION 40
- Enabling-new-function mode 8
- ENCODING 174, 205, 342, 351
- Encoding scheme 11, 174, 176, 304, 342, 344, 353, 414–415
- encryption 188
- Encryption functions 188
- encryption keys 190
- ENDING AT 233
- ENFM 8, 13–14, 24, 135, 345, 353, 357
- ENFORCED 21, 41, 194, 268, 343
- Enterprise Workload Manager 312

- Equi-join predicates 54, 402
- Equivalence 51, 194
- ESS FlashCopy 218
- EWLM 312
- EXISTS 360
- Expanded storage 13, 17, 19, 153
- Explain 7, 42, 47, 50, 72, 111, 115, 124, 127, 161, 211, 258–259, 275, 365, 369, 407–408, 413, 415–416
- Explicit clustering 248
- Explicit clustering index 247
- External access control 192
- EXTSEQ 26

F

- FACILITY class 194
- Fact table 54–55, 58–59
- Fallback 8, 14, 18, 135, 138, 176, 343, 353–354
- False contention 8, 328–329, 331
- Fast cached SQL statement invalidation 7
- fast column processing 22
- Fast log apply 21, 220, 225–226, 352
- Fast Replication 27
- Fast replication services 217
- FETCH 7, 15, 31, 35–36, 38, 81–82, 85–86, 91, 93, 95–97, 99–100, 102–104, 115, 124, 131, 139, 190, 196, 208, 234, 237–238, 285–286, 292, 323, 347–349, 411
- FETCH CURRENT 96
- FETCH RELATIVE 96
- FETCH SENSITIVE 93, 100–101, 105
- Fetch/Delete 35
- Fetch/Updat 35
- Filter factor estimation 74
- Fixed pools 141
- FLA 21, 220–221, 225, 388
- FlashCopy 200, 218, 224, 261, 263, 265–266
- FOR BIT DATA 189, 345
- FOR n ROWS 284
- FOR ROW n OF ROWSET 34
- FOR UPDATE OF 97
- FORMAT DELIMITED 376
- FRBACKUP 219
- FRDELETE 219
- FREEMAIN 141, 162
- FREQVAL 73, 274, 276, 279
- FRRECOV 219, 224
- Full system backup 220

G

- GBP checkpoint 25
- GBPCHKPT 25, 346
- GBP-dependent objects 325
- GBPOOLT 25, 346
- GET DIAGNOSTICS 346
 - Multi-row insert 33
- GETMAIN 141, 394, 396
- Getmained Block Storage 141
- GETVARIABLE 199
- Global lock contention 131, 319, 325–326, 331
- global lock contention 136

- GMB 141
- Granularity 87, 174, 191, 195, 326, 368
- GRAPHIC 68–69, 72, 124, 175, 177–178, 180–181, 183, 185, 187, 231, 343
- Group restart 327

H

- Hardware compression 144
- HC3 184
- Hierarchical security level 192
- Hiperpool 19, 141, 158–159, 352, 394
- Host variable array 32, 38, 289
- host variable array 32
- Host variables impact on access paths 7
- Host-variable 33, 256
- HPSIZE 142
- HVA 32
- Hybrid join 54, 75

I

- I/O CI limit removed 319
- ICF catalog 219
- ICSF 188–190
- IDBACK 25, 205–206
- IDFORE 25, 205–206
- IDTHTOIN 25
- IFCID
 - 0335 256
 - 124 211
 - 140 212
 - 142 212
 - 148 321
 - 172 206
 - 217 205
 - 234 212
 - 239 209
 - 3 207, 209, 212, 256, 258, 321
 - 313 207, 258
 - 317 211
 - 335 212
 - 337 207
 - 342 211
 - 350 211
 - 63 211
- IFCID 173 210
- IFCID 225 145, 159, 365–366
- IFCID 316 417
- IFCID 318 417
- IFCIDs 205, 210
- II10817 378
- II13695 378, 386
- II14047 269, 378
- II14106 382
- II4309 378
- image copy 375
- Immediate index availability 231
- IMMEDWRI 332
- IMMEDWRITE 137, 319, 332

IMMEDIATE NO 333
 Implicit clustering 248
 Improved LPL recovery 257, 320, 336
 IMS 171, 188–189
 Inactive connection 353
 Inactive DBAT 353
 Index only access for varchar 7
 Index-controlled partitioning 247
 Indoubt units of recovery 320, 338
 indoubt URs 320, 338
 informational referential constraints 45
 Inherit WLM priority from lock waiter 258
 In-memory workfiles 6, 56
 INSENSITIVE 92–96, 98, 103, 107
 INSERT 7, 16, 32, 35, 38, 40–41, 86, 102, 111, 124, 131, 136, 139, 179, 181, 183, 188, 196–197, 208, 227, 229, 232, 234–236, 248–249, 251–252, 259, 270, 281, 285, 288–292, 310–311, 319, 321, 332, 359, 400, 403, 408, 411–413, 415
 Integrated Cryptographic Service Facility 188–189
 IPCS 165
 IRLM 9, 12, 127, 134–137, 140, 153–154, 172–173, 324, 327–331
 IRLM V2.2 21, 127, 153, 171–173
 IRR.WRITEDOWN.BYUSER profile 194
 IS NULL 360–361
 ISO 99, 101, 103, 105, 215
 IVP sample programs 346
 IXQTY 228

J

Java xxv, 3, 66, 112, 175–177, 179–182, 281, 285, 292–293, 295, 304, 312
 Java Common Connectivity 293
 JCC 292–295, 304
 JDBC 2.0 294, 306
 JDBC 3.0 293
 JDBC-ODBC bridge 292
 JOIN_TYPE 54, 413

K

KEEPDYNAMIC 147–148, 209, 304–305

L

LARGE 229
 large address space 3
 large catalog/directory 359
 large CIs 200
 large pages 202
 large secondary allocations 228
 large strings 186
 large table space scan 6
 Larger buffer pools 17, 139, 141, 171
 LEAST 40, 74, 106, 143, 187, 194, 256, 274, 276, 284, 326, 354
 LEAST frequently occurring values 74, 273
 Legacy driver 177, 179, 294
 LIST 219

List prefetch 12, 137, 160, 221, 225, 319, 321, 333, 335, 395, 400, 402, 405
 list prefetch 335, 408
 LISTDEF 264
 L-locks 8, 319, 325–329, 331
 LOAD 132, 271
 FORMAT DELIMITED 271
 LOAD table PART n REPLACE 243
 LOB 72, 145, 228, 305, 307, 351, 380, 389, 409
 LOB ROWID transparency 5
 LOBVALA 25, 145
 LOBVALS 145
 Local SQL cache issues and short prepare 7
 Location alias 284
 Location name 206, 219, 283–284, 293
 Lock contention 96, 131, 174, 319, 325–326, 331
 Lock escalation 33, 173–174, 207–208
 Lock escalation trace record 8
 Locking 7, 23, 93, 134–136, 138, 173–174, 208, 319, 325–328, 330–331, 335, 337, 365
 log copy 6
 Log data sets 128, 202, 213
 LOG NO events 218
 Log truncation point 220
 LOGAPSTG 26
 Logical partition 233, 245, 255, 264
 Logical partitions 250–251
 Long index keys 129
 Long names 5, 33, 129, 135, 152, 205, 261, 344–345, 353
 long names 164
 Long running UR backout 256
 long term page fix 320
 Longer SQL statements xxv
 Longer table and column names 5, 346
 Long-term page fixing 169
 Lookaside pool 17
 LOOP 48, 50, 53–54, 58, 405, 410
 LPL 337
 LPL recovery 257, 320, 336
 LRDRTHLD 207, 258
 LRSN 358
 LRU 13, 171

M

Management Client Package 112
 Materialized query table 39–40, 42–43, 53, 124, 408, 410
 materialized query tables 6, 23, 39, 41, 43, 124, 352
 Materialized snowflakes 53
 MAXCSA 172–173
 MAXDBAT 25, 206
 Maximum number of partitions 4
 MAXKEEPD 145–147, 160
 MEMLIMIT 172, 174
 Memory usage 152, 396
 MGEXTSZ 227, 229
 Migration 8, 11, 14, 19, 113, 148, 199, 283, 341, 350–351, 353–354, 356–359, 366
 Migration process 8, 343, 353
 Minimize impact of creating deferred indexes 5

- MINSTOR 145–146
- Mismatched data type 67, 77
- MLMT 172
- MLS 191, 194–195, 198
- MODIFY 139, 172, 226, 233, 354, 359, 371
- Monitor system checkpoints 212, 256
- Monitoring
 - Log offload activity 212
- More log data sets xxv, 6
- More partitions 4, 256
- MOST 74, 274
- MQT 39–43, 45, 48, 50–53, 123–124, 408
 - Considerations 52
 - Creation 40
 - CURRENT REFRESH AGE 123
 - Summary 47
 - System-maintained 40
 - User-maintained 40
- MSTR 154
- MSTR address space 137, 153, 155, 332–333
- MSTR SRB 333
- Multilevel security 8, 26, 191, 193–194, 197
 - Requirements 27
 - Utilities 197
- Multilevel security at the object level 192
- Multilevel security with row-level granularity 192
- Multiple DISTINCT clauses in SQL statements 5
- Multiple IN values 7
- Multiple physical partitions 247
- Multi-row fetch 21, 31–33, 35, 37–38, 103–106, 136, 281, 285, 287–291, 346, 348
 - Positioned DELETE 33
 - Positioned UPDATE 33
 - Rowset 106
- multi-row fetch 31
- Multi-row FETCH and INSERT 102, 285
- Multi-row INSERT 16, 32, 36, 289, 291
 - ATOMIC 33
 - GET DIAGNOSTICS 33
- multi-row INSERT and FETCH 6
- MVS system failure 325, 337

N

- nested loop join 69, 112
- Net driver 293, 299
- New Function Mode 289
- New table spaces 345
- New tables 345
- New-function mode 8, 218
- NO SCROLL 94
- Non-clustering index 247
- Non-GBP-dependent 326, 338
- Non-partitioned index 247, 255, 265, 267, 275
- Non-partitioned indexes 248, 255
- NOT CLUSTER 248
- NOT EXISTS 113
- NOT NULL WITH DEFAULT 409
- NOT PADDED 217, 237–239, 242, 358, 400–401, 403–404
- NPAGES 246

- NPGETHRSH 246
- NPI 137, 243–244, 249–255, 265
- NPSI 248
- NUMLKTS 174
- NUMLKUS 174

O

- OA04555 390
- OA05960 390
- OA06729 390
- OA07196 xxii, 313, 390
- OA07356 xxii, 313, 390
- OA08172 190, 390
- OA10984 202
- OA12005 xxii, 390
- OA15666 384, 391
- OA17114 391
- Object 11, 111, 176, 181, 185, 192, 195, 205, 207, 230, 258, 263, 284, 323, 336, 338
- ODBC 15, 154, 281, 285–286, 290–292, 306
- Online DSNZPARMs 256
- Online partitioning changes 4
- Online REORG 5, 133, 248, 255, 258, 264, 359
- Online schema evolution 3–4, 217, 230, 261
- Open Group's DRDA 281
- optimistic locking 107
- optimizer enhancements 18
- ORDER BY 23, 82, 86, 99, 410
- OS/390 V2R10 190
- OW54685 336

P

- Package level accounting 8, 208
- PADDED 137, 183, 217, 232, 237–239, 241–242, 264, 358, 401, 403–404
 - VARCHAR 11, 137, 232, 237–238, 345, 401, 403–404
- page set 323
- Page set P-locks 326, 331
- Panel
 - DSNTIP4 42, 147
 - DSNTIP7 199, 352
 - DSNTIPP 352
- Parallel Access Volume 30
- Parallel index build 268–269
- parallelism 254
- Parallelism details 111
- Parameter marker 256
- Parent lock contention 326
- PARTITION 4, 59, 111, 137, 144, 160, 230, 233, 243, 245, 248–253, 255, 257, 262, 264–265, 267–268, 275, 327, 336
- PARTITION BY 247
- Partition management 233
- Partition pruning 247, 255
- PARTITIONED 4, 34, 36, 38, 160, 217, 233, 237, 246–248, 254–257, 263, 268, 270, 274, 327, 346, 372
- Partitioned index 245, 247, 255, 265, 267, 275
- Partitioned secondary indexes 4, 247, 264

Partitioning 4, 217, 233, 247–248, 255–257, 261
 Partitioning index 3–4, 217, 246–248, 257
 Partitioning indexes 257
 Partitioning key update 257
 PARTKEYU 257
 PC=NO 127, 172–173
 PC=YES 136, 172–173
 Performance 6
 Multi-row operations 37, 289
 PERMIT 107, 193
 PGFIX 21, 169–171, 339
 GPROT 172
 Physical partitions 245, 247, 250–251, 255
 Physically partitioned 247–248
 PIECESIZE 229
 PK00213 382
 PK01510 xxii, 382
 PK01841 382
 PK01855 382
 PK01911 382
 PK03293 382
 PK03469 xxii, 382
 PK04076 xxii, 269, 382
 PK04107 382
 PK05360 xxii, 382
 PK0564 230
 PK05644 xxii, 382
 PK09158 382, 390
 PK09268 xxii, 382
 PK10021 383
 PK10278 383
 PK10662 383
 PK11355 383
 PK12389 xxii, 383
 PK13455 383
 PK14393 383
 PK14477 383, 390
 PK15056 383
 PK15288 383
 PK18059 383
 PK18162 383
 PK18454 383
 PK19191 383
 PK19303 383
 PK19769 383
 PK19920 383
 PK20157 383
 PK21237 150, 383
 PK21268 383
 PK21371 384
 PK21861 384
 PK21892 384
 PK22442 384
 PK22611 384
 PK22814 384
 PK22887 384, 390
 PK22910 384
 PK23495 384
 PK23523 384
 PK23743 384
 PK24556 384
 PK24558 384, 390
 PK24585 384
 PK24710 384
 PK24819 384
 PK25241 384
 PK25326 384
 PK25427 384
 PK25742 384
 PK26199 384
 PK26692 385
 PK26879 385
 PK27281 385
 PK27287 385
 PK27578 385
 PK27712 385
 PK28561 385
 PK28637 385
 PK29626 385
 PK29791 385
 PK29826 385
 PK30087 385
 PK30160 385
 PK32606 385
 PK34251 385
 PK34441 385
 PK35390 385
 PK36717 385
 PK37354 385
 PK38201 385
 PK40057 385
 PK41182 382
 PK46170 386
 PK47840 386, 390
 PK49972 386
 PLAN_TABLE access via a DB2 alias 259
 P-locks 326, 331
 PMDB2 166
 point-in-time 4, 27, 218–219
 PORT 112, 293, 304
 POSITION 91, 270, 281
 PQ47973 212, 386
 PQ49458 386
 PQ54042 67, 84, 386
 PQ61458 58–59, 65, 386
 PQ62695 112, 386
 PQ69741 320, 338, 386–387
 PQ69983 387
 PQ71179 387
 PQ71766 387
 PQ71775 362, 387
 PQ74772 387
 PQ75064 387
 PQ78515 387
 PQ79232 387
 PQ80631 378, 387
 PQ81904 387
 PQ82878 378, 387
 PQ83649 378
 PQ8416 280

PQ84160 378, 387
 PQ85764 387
 PQ85843 388
 PQ86037 388
 PQ86049 388
 PQ86071 335, 378
 PQ86074 379
 PQ86201 379
 PQ86477 204, 379, 388
 PQ86787 305, 379
 PQ86904 327, 379
 PQ87126 305, 379, 388
 PQ87129 305, 379
 PQ87168 327, 379
 PQ87381 379, 388
 PQ87390 379
 PQ87447 379, 388
 PQ87509 275, 379
 PQ87611 172, 379
 PQ87756 327, 379
 PQ87764 157
 PQ87783 388
 PQ87848 210, 379
 PQ87917 379, 388
 PQ87969 39, 379
 PQ88073 379, 417
 PQ88224 365
 PQ88375 275, 380
 PQ8841 179
 PQ88582 307, 380
 PQ88587 388
 PQ88665 229, 380
 PQ88784 24
 PQ88896 368, 380
 PQ89070 215, 380
 PQ89181 39
 PQ89297 380, 388
 PQ89919 335, 380, 389
 PQ90022 111, 259, 380, 389
 PQ90075 380, 389
 PQ90147 260, 380
 PQ90263 380, 389
 PQ90547 365, 380
 PQ90884 275, 277, 280, 380
 PQ91009 353, 380
 PQ91101 158, 163, 365, 381, 389
 PQ91493 381
 PQ91509 313, 381
 PQ91898 39, 381
 PQ92072 381
 PQ92227 254, 381
 PQ92749 262, 268, 381
 PQ93009 389
 PQ93156 381
 PQ93458 285
 PQ93620 39, 381
 PQ93821 111, 259, 381, 389
 PQ93943 190
 PQ9414 381
 PQ94147 389

PQ94303 381
 PQ94793 389
 PQ94822 190, 381
 PQ94872 158, 163, 365, 381
 PQ94923 381
 PQ95164 221, 225, 381
 PQ95881 304, 381
 PQ95989 365
 PQ96189 213, 381
 PQ96628 382, 389
 PQ96772 143, 152, 382
 PQ96956 262, 268, 382
 PQ97794 382
 PQ99181 380
 PQ99482 39
 PQ99524 382, 389
 PQ99608 382
 PQ99658 xxii, 382, 389
 PQ99707 xxii, 313, 382
 PQTY 228
 PRECISION 68
 Precompiler 23, 354
 NEWFUN 23, 354
 PREFETCH 12, 158, 160, 170, 221, 225, 229, 319, 321,
 324, 333, 395, 400, 402, 405, 412
 Print Log Map utility 283
 PRIQTY 226, 228–229
 Private Protocol 284
 PT 143

Q

Q3STHWCT 205
 Q3STHWIB 205
 Q3STHWIF 205
 QMF 212, 408
 QMF for TSO/CICS 8.1 39
 Qualified partitions 256
 QUALIFIER 259, 408
 QUERY 219
 Query parallelism 29, 78, 254, 413
 QUERYNO 50, 400, 402, 405, 410, 415–416
 QUIESCE 263, 327, 331, 370
 Quotation mark 272

R

RACF 26, 139, 191–198
 RACROUTE 26
 RBA 256, 358
 RBDP 231–232, 236
 RBLP 220
 RDS 158, 160, 164, 395
 RDS sort 144
 Read For Castout Multiple 321
 READS interface 211
 Read-up 194
 Real contention 329
 Real memory 18, 153, 165
 Real storage 3, 9, 11, 13, 17, 19, 21, 65, 97, 127, 129,
 138, 142, 148, 153–154, 156–158, 165, 169–171, 174,

201, 320, 339, 365, 397
 REBALANCE 245
 Rebalance partitions 3, 233
 REBUILD INDEX 132, 202, 235, 237–238, 242, 249, 252, 263
 Rebuild pending 231–232
 RECOVER 4, 21, 133, 183, 217–221, 233, 257, 320, 336, 367
 Recovery base log point 220
 Recursive SQL 6
 Common table expressions 6
 Redbooks Web site 427
 Contact us xxviii
 Reduced lock contention 7
 REFRESH DEFERRED 40
 refresh MQTs 41
 REFRESH TABLE 40–42, 53, 124
 REGION 142, 158, 161, 173, 396
 REMARKS 412
 REORG 4, 8, 11, 132, 197, 202, 226, 230–231, 233–235, 240, 242, 244–245, 247–248, 255, 258, 264, 268–269, 354, 356, 359, 371
 Clustering order 268
 DISCARD 5, 197
 Implicit clustering index 247
 REBALANCE 244–245
 REORG INDEX 133
 REORG TABLESPACE 197, 233
 REBALANCE 233
 REORG utility enhancements 5
 REORP 233
 REPORT 7, 73–74, 116, 118, 122–123, 149, 162, 164, 166, 207, 213, 258, 276, 328–330, 334, 365, 394
 Requester database ALIAS 281, 283
 RESET 16, 209, 236
 Residual predicates 66
 Resource
 Security label 192–193
 RESTART 218, 221, 256, 326–327, 331, 337
 Restart light enhancements 320, 337
 RESTART WITH 221
 RESTORE 4, 261, 367
 RESTORE SYSTEM 4, 21, 217–218, 220–221, 224–226
 RESTRICT 94
 Retained locks 172, 325, 337
 RETVLCFK 232
 Reverse dominance 194
 RFCOM 321–322
 RID 402, 404
 RID Pool 3, 17, 140, 144, 148–149, 153, 158, 160, 394–395
 RIDBLOCK 145
 RIDLIST 145
 RIDLISTs 144
 RIDMAP 145
 RIDs 73, 115, 145
 RMF 26, 148, 166, 322, 324, 328–329
 ROLLBACK 206, 332, 368
 ROWID 5
 Rowset 29, 31, 33, 92, 94, 103, 105, 288–290, 292

Rowsets 31, 33, 288–289
 RQRIOLBK 285
 RRSF 8, 208, 296
 RUNSTATS 7, 53, 71, 73, 76, 110–111, 115–117, 122–124, 132, 231, 238, 240, 242, 246, 248, 253, 273, 275–278, 280, 401–403, 405
 RUNSTATS enhancement 272

S

SBCS 22, 174, 180, 185–186, 342–343, 414
 SCA 220, 320, 336
 Scalability xxv, 3, 9, 12, 20, 128–129, 212
 SCHEMA 54, 65, 217, 230, 233, 259, 353, 416
 Schema changes 230
 Schema evolution xxv, 3–4, 217, 230, 261
 SECDATA 192
 SECLABEL 192–193, 195–198, 212
 SECLEVEL 192, 198
 secondary AUTHIDs 260
 Secondary authorization ID information 212
 Secondary index 4, 247–248, 274
 Clustering index 247
 SECQTY 226, 229
 secure key 189
 Security
 Object 192
 Security category 192
 Security label 192–194, 198, 212
 Security labels 192–194, 198
 Assigning 193
 Defining 192
 Security level 192, 194
 Security policy 191
 Security Server 191–192, 212
 SENSITIVE DYNAMIC 94, 96, 101, 105
 SENSITIVE STATIC 93–94, 100, 104
 Separation of partitioning and clustering 4
 Serialized profile 300
 Server location ALIAS 281, 283
 service time 9
 SET 34, 172, 226
 SET clause in the SQL UPDATE 199
 SET CURRENT REFRESH AGE ANY 42
 SET DPTDPTSZ 87
 SET ENCRYPTION PASSWORD 188
 SET QUERYNO 50
 -SET SYSPARM 256
 Shift-in 185, 344
 Shift-out 185, 344
 short prepare 146
 SIGNON 206, 260
 SJMXPOOL 54, 57, 65
 SKCT 143
 SKPT 143, 357
 SMF 26, 128, 136, 213, 351
 SMF89 213–214
 SMFSTST 26
 SMS-managed data sets 218
 Sort pool 3, 17, 144, 148–149, 153, 156
 SORT pools 144

- Sort tree nodes 144
- SORTDATA 268
- SORTDEVT 74, 263, 274
- SORTKEYS 261, 268–269
- SORTNUM 74, 263, 274
- Sparse index for star join 6, 59
- Special register 42, 189, 256, 352
- Special registers 41–42, 124, 352
- SPRMCTH 146
- SPRMINT 25
- SPRMSTH 146
- SPUFI 359, 408
- SQL statement 2 MB long 5
- SQL statement text 8, 211
- SQL statements 2 MB long 5
- SQL/XML publishing functions 281, 307
- SQLCA 33
- SQLCODE 96, 209, 229, 259
- SQLDA 343
- SQLExtendedFetch 285, 291
- SQLJ 150–151, 154, 179, 181, 292, 297–298
 - Static SQL 181, 297–298, 300
- SQLJ applications 293
- SQLSTATE 259
- SRB time 16
- SRTPOOL 144
- Stack Storage 141
- Stage 1 65–67, 69–71, 115, 195, 304, 400–402, 404, 406
- Stage 2 29, 65–67, 69, 115, 400, 403
- Star join
 - in memory workfile 56
- Star schema 6, 53–55, 65
 - Data model 53
- STARJOIN 54
- START DATABASE 257, 336
- START WITH 229, 247
- Static scrollable cursors 30, 92–93, 95, 97, 100, 107
- STMT 158, 161, 361, 395
- STMTTOKEN 414
- STOGROUP 227
- storage per thread 168
- Striping 6, 200
- Subject 87, 193
- SUBSTRING 24
- SUM 109
- SYSADM 40, 259, 304
- SYSALTER 345
- SYSCOLDIST 280, 355
- SYSCOLDISTSTATS 355
- SYSCOLUMNS 355
- SYSCOPY 233, 354–355, 359
- SYSCTRL 40
- SYSHIGH 193
- SYSIBM.IPLIST 283, 345
- SYSIBM.SQLCAMESSAGE 112
- SYSIBM.SYSCOLDIST 279
- SYSIBM.SYSCOLUMNS 360
- SYSIBM.SYSDUMMY1 345
- SYSIBM.SYSOBDS 345
- SYSIBM.SYSPACKSTMT 400

- SYSIBM.SYSROUTINES 83
- SYSIBM.SYSSTMT 361
- SYSIBM.SYSSTRINGS 175, 184–185, 342
- SYSIBM.SYSTABLES 72–73, 360
- SYSIBM.SYSTABLESPACE 72–73, 360
- SYSIN 264, 276, 347
- SYSLGRNX 233, 354, 359
- SYSLOW 193
- SYSMULTI 193
- SYSNONE 193
- SYSOBDS 345
- SYSPACKSTMT 355, 400
- SYSPITR CRCR 220
- sysplex workload balancing 299
- SYSPLDGD 26
- SYSPRINT 263, 347
- SYSSTMT 355, 361
- SYSTABLEPART
 - LOGICAL_PART 233
- System checkpoints 212, 256, 258
- System level point-in-time recovery 4, 219
- System-generated DDNames 213

T

- Table UDF
 - Block fetch 84
- Table UDF cardinality 6, 86
- Table-based partitioning 233
- Table-controlled partitioning 246–247
 - Clustering 246–247
- TCB time 16, 333
- TCP/IP 25, 27, 112, 281, 283, 292, 295
- TCPKPALV 25
- TEMP database 211
- TEXT 8, 38, 69, 72, 84, 175, 183, 187, 211, 259, 271, 342, 360, 378, 386, 390, 406, 417
- Text extender 84
- Thanks xxvii
- Thread-related storage 12, 129, 144
- thread-related storage 167
- TIME 218
- TIMESTAMP 41–42, 115, 395, 412–413, 415–416
- Tivoli OMEGAMON XE for DB2 on z/OS 366
- Transient data type 308
- Transition variables 86
- translation instructions 180
- Trigger
 - WHEN condition 30
- Trigger body 87
- TROO 186
- TROT 186
- TRTO 186
- TRTT 186
- TSQTY 228
- Two-phase commit 303
- Type 1 292, 353
- Type 2 7, 179–182, 285, 292–294, 296, 299, 303, 305, 353
- Type 3 292, 294, 299
- Type 4 179, 181, 187, 285, 293–294, 296–299

U

- UA05789 184, 186
- UA08703 390
- UA11154 390
- UA11478 390
- UA15188 390
- UA15189 313, 390
- UA15456 313, 390
- UA15677 390
- UA22388 390
- UA27812 391
- UA34634 391
- UK00049 381
- UK00295 389
- UK00296 279, 381
- UK00314 382
- UK00991 152, 382
- UK01174 389
- UK01175 382
- UK01429 382, 389
- UK01467 382
- UK01844 382
- UK03058 313, 382
- UK03176 382
- UK03226 389
- UK03227 380
- UK03490 382
- UK03835 313, 381
- UK03983 269, 382
- UK04036 382
- UK04393 382, 389
- UK04394 382
- UK04683 262, 381
- UK04743 382
- UK05785 390
- UK05786 382
- UK06418 382
- UK06505 382
- UK06754 379, 388
- UK06848 39
- UK06883 381
- UK07167 383
- UK07192 383
- UK08497 383
- UK08561 382
- UK08807 230, 382
- UK09097 383
- UK09343 383
- UK10002 383
- UK10819 383
- UK12124 383
- UK12253 383–384
- UK12328 383
- UK12821 383
- UK13636 383
- UK13671 384
- UK13721 383–384
- UK14058 383, 390
- UK14283 150, 383
- UK14326 384
- UK14343 383
- UK14370 384
- UK14540 384
- UK14634 384
- UK15036 390
- UK15037 384
- UK15150 384
- UK15285 384
- UK15493 384
- UK15499 383
- UK15650 384, 390
- UK15814 383
- UK15904 384
- UK15958 384
- UK16124 379
- UK16191 385
- UK16616 385
- UK17089 385
- UK17220 385
- UK17312 385
- UK17364 384
- UK17369 385
- UK17805 385
- UK18020 385
- UK18090 385
- UK18201 385
- UK18276 385
- UK18503 384
- UK18547 384
- UK19282 385
- UK19776 385
- UK19777 385
- UK20497 382
- UK21175 385
- UK21950 385
- UK22021 385
- UK23708 385
- UK24197 385
- UK25044 385
- UK25290 385
- UK26761 383
- UK27593 386, 390
- UK27594 386, 390
- UK28485 386
- UK9531 382
- Unicode xxv, 3, 8, 11, 13, 23, 127, 129, 135, 144, 152, 175–180, 183–187, 205, 261, 304, 306, 342–345, 350–351, 353, 357–358, 369, 414–415
 - Catalog and directory 11
 - Collating sequence 23, 183
 - DBRMs 369
 - Moving to 184, 351
 - Precompiler 23
- Unicode parser 22
- UNION ALL 95, 99
- UNIQUE 52, 107, 137, 219, 231–232, 236, 247, 255, 334, 354, 357, 400–401, 403–405
- UNIQUE WHERE NOT NULL 231, 248, 405
- Universal Client 305
- Universal Driver 179, 181, 187, 292, 294–295, 300, 304,

306
 Universal Driver for SQLJ and JDBC 3, 281, 285,
 293–294
 UNLOAD 133, 371–372
 DELIMITED 271–272, 371
 Delimited output 376
 UNLOAD DELIMITED 270–272
 UPDATE NONE 7
 UQ57177 212
 UQ57178 212, 386
 UQ61237 67
 UQ61327 386
 UQ67433 386
 UQ72083 386
 UQ74866 338, 386–387
 UQ75848 387
 UQ77215 387
 UQ77421 387
 UQ79020 387
 UQ79775 386
 UQ81754 387
 UQ83466 387
 UQ84631 387
 UQ85042 362
 UQ85043 362, 387
 UQ85607 285
 UQ86458 388
 UQ86831 379
 UQ86868 388
 UQ86872 379
 UQ87013 378
 UQ87049 305, 379
 UQ87093 387
 UQ87321 388
 UQ87443 378
 UQ87591 379
 UQ87633 305, 379
 UQ87760 379
 UQ87903 379
 UQ886790 388
 UQ88680 379
 UQ88734 388
 UQ88754 275, 380
 UQ88970 388
 UQ88971 305, 379
 UQ89026 388
 UQ89190 389
 UQ89194 380
 UQ89372 379, 417
 UQ89517 229, 380
 UQ89850 389
 UQ89852 380
 UQ90022 381
 UQ90143 368, 380
 UQ90158 387
 UQ90159 378
 UQ90464 39, 380–381, 389
 UQ90653 388
 UQ90654 379
 UQ90701 353, 380

UQ90726 365
 UQ90755 210
 UQ90756 210, 379
 UQ90793 387
 UQ90794 378
 UQ91022 379
 UQ91099 275, 379
 UQ91257 307, 380
 UQ91416 280, 387
 UQ91422 280, 378
 UQ91466 381
 UQ91470 380
 UQ91968 380, 388
 UQ92066 388
 UQ92067 380
 UQ92322 259, 389
 UQ92323 380
 UQ92475 379, 388
 UQ92546 39, 381
 UQ92692 39, 379
 UQ92698 260, 380
 UQ93079 381
 UQ93177 275, 380
 UQ93207 365
 UQ93407 389
 UQ93893 389
 UQ93972 254, 381
 UQ94429 285
 UQ94430 285
 UQ94797 380
 UQ95032 380
 UQ95553 381
 UQ95694 365
 UQ96157 381
 UQ96302 381
 UQ96531 382
 UQ96567 39, 381
 UQ96862 213, 381
 URL 304, 363, 371
 URL syntax 293
 USER 371
 User-defined functions 89, 307
 User-defined objects 199
 USING VCAT 199
 UTF-16 175, 177, 179–180, 182, 185–187, 343–344
 UTF-8 22, 175, 179, 181–182, 184–187, 343–344
 Utility
 RESTART 21, 220, 262, 367
 SORTDATA 268
 SORTKEYS 268

V

VALUES 6–7, 24–25, 32, 42, 44, 68, 73–74, 93, 172,
 195, 198, 217, 220, 226, 228–229, 233, 257, 273–275,
 304, 308, 322, 345, 352, 405–406, 410, 412, 414–416
 VARCHAR
 NOT PADDED 11, 14, 232, 237, 345, 401
 VARGRAPHIC 68–69, 183, 231
 VARIABLE 371
 Variable pools 141

VDWQT 25, 345
 VE 117, 124
 VERSION 412
 Versioning 369
 Virtual storage constraint 9, 12, 17, 19, 127, 129,
 139–140, 145, 156, 213
 Visual Explain 72–73, 77, 110, 112, 114–115, 123–125,
 259
 Access plan 111
 Maintain 111
 Tune SQL 111, 123
 VOLATILE 8
 Volatile table 217, 246
 volatile tables 7, 246, 369
 Volume-level backups 218
 VPSIZE 142
 VSAM
 Striping 201–202
 VSAM CI size 199–201

W

WARM 321–322
 WebSphere 26, 172, 174, 281–282, 294, 297–299, 304,
 316
 WebSphere Application Server 295, 303, 314
 WLM 26, 117, 209, 308, 312, 350
 WLM enclave 209
 WLM priority 258
 workfile 30, 56, 58–59, 65, 70, 77, 84, 86, 90, 115, 211,
 255, 268
 Write And Register Multiple 321
 WRITE CLAIM 264, 337
 Write down 191, 194
 write-down 194, 197–198
 Write-down control 196
 Write-down privilege 196–197

X

XA 303
 XES 319, 325–327, 329, 331
 XES contention 173, 325, 328–330, 332
 XML 111, 281, 307–309
 Built-in functions 281, 308
 XML composition 311
 XML data type 308–309
 XML documents 307, 311–312
 XML extender 281, 307–309, 311–312
 XML file 111
 XML publishing functions 281, 307–309, 311–312
 XML2CLOB 309–310
 XMLAGG 309–310
 XMLATTRIBUTES 309–310
 XMLCONCAT 309–310
 XMLELEMENT 309–310
 XMLFOREST 309–310
 XMLNAMESPACES 309

Z

z/Architecture 13, 139, 153, 157, 171
 z/OS APARs 390
 z800 27
 z900 27, 129, 188, 284
 z990 26–27, 129, 185, 187–188, 190, 308
 zSeries xxv, 2, 9, 13, 27, 97, 174, 178, 180, 183–186,
 189, 213, 221, 233, 248, 309, 342, 344

Archived



DB2 UDB for z/OS Version 8 Performance Topics

(1.0" spine)

0.875" x 1.498"

460 <-> 788 pages



DB2 UDB for z/OS Version 8 Performance Topics



Redbooks

Evaluate the performance impact of major new functions

Find out the compatibility mode performance implications

Understand performance and availability functions

IBM DATABASE 2 Universal Database Server for z/OS Version 8 (DB2 V8 or V8 throughout this book) is the twelfth and largest release of DB2 for MVS. It brings synergy with the zSeries hardware and exploits the z/OS 64-bit virtual addressing capabilities. DB2 V8 offers data support, application development, and query functionality enhancements for e-business, while building upon the traditional characteristics of availability, exceptional scalability, and performance for the enterprise of choice.

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

This book introduces the major performance and availability changes as well as the performance characteristics of many new functions. It helps you understand the performance implications of migrating to DB2 V8 with considerations based on laboratory measurements. It provides the type of information needed to start evaluating the performance impact of DB2 V8 and its capacity planning needs.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-6465-00

ISBN 0738492493